PRAW

Release 7.0.0.dev0

Feb 19, 2020

Getting Started

PRAW's documentation is organized into the following sections:

- *Getting Started*
- *Code Overview*
- *Tutorials*
- *Package Info*

CHAPTER 1

Documentation Conventions

Unless otherwise mentioned, all examples in this document assume the use of a **script** application. See *Authenticating via OAuth* for information on using **installed** applications and **web** applications.

1.1 Quick Start

In this section, we go over everything you need to know to start building scripts or bots using PRAW, the Python Reddit API Wrapper. It's fun and easy. Let's get started.

1.1.1 Prerequisites

Python Knowledge You need to know at least a little Python to use PRAW; it's a Python wrapper after all. PRAW supports Python 3.5+. If you are stuck on a problem, r/learnpython is a great place to ask for help.

Reddit Knowledge A basic understanding of how Reddit works is a must. In the event you are not already familiar with Reddit start at Reddit Help.

Reddit Account A Reddit account is required to access Reddit's API. Create one at reddit.com.

Client ID & Client Secret These two values are needed to access Reddit's API as a **script** application (see *Authenticating via OAuth* for other application types). If you don't already have a client ID and client secret, follow Reddit's First Steps Guide to create them.

User Agent A user agent is a unique identifier that helps Reddit determine the source of network requests. To use Reddit's API, you need a unique and descriptive user agent. The recommended format is `<platform>:<app ID>:<version string> (by u/<Reddit username>)`. For example, `android:com.example.myredditapp:v1.2.3 (by u/kemitche)`. Read more about user agents at Reddit's API wiki page.

With these prerequisites satisfied, you are ready to learn how to do some of the most common tasks with Reddit's API.

1.1.2 Common Tasks

Obtain a `Reddit` Instance

Warning: For the sake of brevity, the following examples pass authentication information via arguments to *praw.Reddit()*. If you do this, you need to be careful not to reveal this information to the outside world if you share your code. It is recommended to use a *praw.ini file* in order to keep your authentication information separate from your code.

You need an instance of the *Reddit* class to do *anything* with PRAW. There are two distinct states a *Reddit* instance can be in: *read-only*, and *authorized*.

Read-only `Reddit` Instances

To create a read-only *Reddit* instance, you need three pieces of information:

1) Client ID
2) Client secret
3) User agent

You may choose to provide these by passing in three keyword arguments when calling the initializer of the *Reddit* class: `client_id`, `client_secret`, `user_agent` (see *Configuring PRAW* for other methods of providing this information). For example:

```
import praw

reddit = praw.Reddit(client_id='my client id',
                     client_secret='my client secret',
                     user_agent='my user agent')
```

Just like that, you now have a read-only *Reddit* instance.

```
print(reddit.read_only)  # Output: True
```

With a read-only instance, you can do something like obtaining 10 'hot' submissions from `r/learnpython`:

```
# continued from code above

for submission in reddit.subreddit('learnpython').hot(limit=10):
    print(submission.title)

# Output: 10 submissions
```

If you want to do more than retrieve public information from Reddit, then you need an authorized *Reddit* instance.

Note: In the above example we are limiting the results to 10. Without the `limit` parameter PRAW should yield as many results as it can with a single request. For most endpoints this results in 100 items per request. If you want to retrieve as many as possible pass in `limit=None`.

Authorized `Reddit` Instances

In order to create an authorized *`Reddit`* instance, two additional pieces of information are required for **script** applications (see *Authenticating via OAuth* for other application types):

4) Your Reddit username, and
5) Your Reddit password

Again, you may choose to provide these by passing in keyword arguments `username` and `password` when you call the *`Reddit`* initializer, like the following:

```
import praw

reddit = praw.Reddit(client_id='my client id',
                     client_secret='my client secret',
                     user_agent='my user agent',
                     username='my username',
                     password='my password')

print(reddit.read_only)  # Output: False
```

Now you can do whatever your Reddit account is authorized to do. And you can switch back to read-only mode whenever you want:

```
# continued from code above
reddit.read_only = True
```

Note: If you are uncomfortable hard-coding your credentials into your program, there are some options available to you. Please see: *Configuring PRAW*.

Obtain a `Subreddit`

To obtain a *`Subreddit`* instance, pass the subreddit's name when calling `subreddit` on your *`Reddit`* instance. For example:

```
# assume you have a Reddit instance bound to variable `reddit`
subreddit = reddit.subreddit('redditdev')

print(subreddit.display_name)  # Output: redditdev
print(subreddit.title)         # Output: reddit Development
print(subreddit.description)   # Output: A subreddit for discussion of ...
```

Obtain `Submission` Instances from a `Subreddit`

Now that you have a *`Subreddit`* instance, you can iterate through some of its submissions, each bound to an instance of *`Submission`*. There are several sorts that you can iterate through:

- controversial
- gilded
- hot
- new

- rising
- top

Each of these methods will immediately return a *ListingGenerator*, which is to be iterated through. For example, to iterate through the first 10 submissions based on the `hot` sort for a given subreddit try:

```
# assume you have a Subreddit instance bound to variable `subreddit`
for submission in subreddit.hot(limit=10):
    print(submission.title)  # Output: the submission's title
    print(submission.score)  # Output: the submission's score
    print(submission.id)     # Output: the submission's ID
    print(submission.url)    # Output: the URL the submission points to
                             # or the submission's URL if it's a self post
```

Note: The act of calling a method that returns a *ListingGenerator* does not result in any network requests until you begin to iterate through the *ListingGenerator*.

You can create *Submission* instances in other ways too:

```
# assume you have a Reddit instance bound to variable `reddit`
submission = reddit.submission(id='39zje0')
print(submission.title)  # Output: reddit will soon only be available ...

# or
submission = reddit.submission(url='https://www.reddit.com/...')
```

Obtain `Redditor` Instances

There are several ways to obtain a redditor (a *Redditor* instance). Two of the most common ones are:

- via the `author` attribute of a *Submission* or *Comment* instance
- via the *redditor()* method of *Reddit*

For example:

```
  # assume you have a Submission instance bound to variable `submission`
 redditor1 = submission.author
 print(redditor1.name)  # Output: name of the redditor

# assume you have a Reddit instance bound to variable `reddit`
 redditor2 = reddit.redditor('bboe')
 print(redditor2.link_karma)  # Output: u/bboe's karma
```

Obtain `Comment` Instances

Submissions have a `comments` attribute that is a *CommentForest* instance. That instance is iterable and represents the top-level comments of the submission by the default comment sort (`best`). If you instead want to iterate over *all* comments as a flattened list you can call the *list()* method on a *CommentForest* instance. For example:

```
# assume you have a Reddit instance bound to variable `reddit`
top_level_comments = list(submission.comments)
all_comments = submission.comments.list()
```

Note: The comment sort order can be changed by updating the value of `comment_sort` on the *Submission* instance prior to accessing `comments` (see: /api/set_suggested_sort for possible values). For example to have comments sorted by `new` try something like:

```
# assume you have a Reddit instance bound to variable `reddit`
submission = reddit.submission(id='39zje0')
submission.comment_sort = 'new'
top_level_comments = list(submission.comments)
```

As you may be aware there will periodically be *MoreComments* instances scattered throughout the forest. Replace those *MoreComments* instances at any time by calling *replace_more()* on a *CommentForest* instance. Calling *replace_more()* access `comments`, and so must be done after `comment_sort` is updated. See *Extracting comments with PRAW* for an example.

Determine Available Attributes of an Object

If you have a PRAW object, e.g., *Comment*, *Message*, *Redditor*, or *Submission*, and you want to see what attributes are available along with their values, use the built-in `vars()` function of python. For example:

```
import pprint

# assume you have a Reddit instance bound to variable `reddit`
submission = reddit.submission(id='39zje0')
print(submission.title) # to make it non-lazy
pprint.pprint(vars(submission))
```

Note the line where we print the title. PRAW uses lazy objects so that network requests to Reddit's API are only issued when information is needed. Here, before the print line, `submission` points to a lazy *Submission* object. When we try to print its title, additional information is needed, thus a network request is made, and the instances ceases to be lazy. Outputting all the attributes of a lazy object will result in fewer attributes than expected.

1.2 Installing PRAW

PRAW supports Python 3.5+. The recommended way to install PRAW is via `pip`.

```
pip install praw
```

Note: Depending on your system, you may need to use `pip3` to install packages for Python 3.

Warning: Avoid using `sudo` to install packages. Do you *really* trust this package?

For instructions on installing Python and pip see "The Hitchhiker's Guide to Python" Installation Guides.

1.2.1 Updating PRAW

PRAW can be updated by running:

```
pip install --upgrade praw
```

1.2.2 Installing Older Versions

Older versions of PRAW can be installed by specifying the version number as part of the installation command:

```
pip install praw==3.6.0
```

1.2.3 Installing the Latest Development Version

Is there a feature that was recently merged into PRAW that you cannot wait to take advantage of? If so, you can install PRAW directly from GitHub like so:

```
pip install --upgrade https://github.com/praw-dev/praw/archive/master.zip
```

1.3 Authenticating via OAuth

PRAW supports the three types of applications that can be registered on Reddit. Those are:

- Web Applications
- Installed Applications
- Script Applications

Before you can use any one of these with PRAW, you must first register an application of the appropriate type on Reddit.

If your app does not require a user context, it is *read-only*.

PRAW supports the flows that each of these applications can use. The following table defines which tables can use which flows:

<table>
<tr><th>Application Type</th><th>Script</th><th>Web</th><th>Installed</th></tr>
<tr><td>Default Flow</td><td>Password</td><td colspan="2">Code</td></tr>
<tr><td rowspan="3">Alternative Flows</td><td>Code</td><td rowspan="2">Application-Only (Client Credentials)</td><td rowspan="2">Implicit</td></tr>
<tr><td>Application-Only (Client Credentials)</td></tr>
<tr><td colspan="3">Application-Only (Installed Client)</td></tr>
</table>

1.3.1 Password Flow

Password Flow is the simplest type of authentication flow to work with because no callback process is involved in obtaining an `access_token`.

While **password flow** applications do not involve a redirect URI, Reddit still requires that you provide one when registering your script application – `http://localhost:8080` is a simple one to use.

In order to use a **password flow** application with PRAW you need four pieces of information:

client_id The client ID is the 14-character string listed just under "personal use script" for the desired developed application

client_secret The client secret is the 27-character string listed adjacent to `secret` for the application.

password The password for the Reddit account used to register the application.

username The username of the Reddit account used to register the application.

With this information authorizing as `username` using a **password flow** app is as simple as:

```
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     password='1guiwevlfo00esyy',
                     user_agent='testscript by /u/fakebot3',
                     username='fakebot3')
```

To verify that you are authenticated as the correct user run:

```
print(reddit.user.me())
```

The output should contain the same name as you entered for `username`.

Note: If the following exception is raised, double-check your credentials, and ensure that that the username and password you are using are for the same user with which the application is associated:

```
OAuthException: invalid_grant error processing request
```

Two-Factor Authentication

A 2FA token can be used by joining it to the password with a colon:

```
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     password='1guiwevlfo00esyy:955413',
                     user_agent='testscript by /u/fakebot3',
                     username='fakebot3')
```

However, for such an app there is little benefit to using 2FA. The token must be refreshed after one hour; therefore, the 2FA secret would have to be stored along with the rest of the credentials in order to generate the token, which defeats the point of having an extra credential beyond the password.

If you do choose to use 2FA, you must handle the `prawcore.OAuthException` that will be raised by API calls after one hour.

1.3.2 Code Flow

A **code flow** application is useful for two primary purposes:

- You have an application and want to be able to access Reddit from your users' accounts.
- You have a personal-use script application and you either want to
  - limit the access one of your PRAW-based programs has to Reddit
  - avoid the hassle of 2FA (described above)
  - not pass your username and password to PRAW (and thus not keep it in memory)

When registering your application you must provide a valid redirect URI. If you are running a website you will want to enter the appropriate callback URL and configure that endpoint to complete the code flow.

If you aren't actually running a website, you can use the *Obtaining a Refresh Token* script to obtain `refresh_tokens`. Enter `http://localhost:8080` as the redirect URI when using this script.

Whether or not you use the script there are two processes involved in obtaining access or refresh tokens.

Obtain the Authorization URL

The first step to completing the **code flow** is to obtain the authorization URL. You can do that as follows:

```
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     redirect_uri='http://localhost:8080',
                     user_agent='testscript by /u/fakebot3')
print(reddit.auth.url(['identity'], '...', 'permanent'))
```

The above will output an authorization URL for a permanent token that has only the *identity* scope. See *url()* for more information on these parameters.

This URL should be accessed by the account that desires to authorize their Reddit access to your application. On completion of that flow, the user's browser will be redirected to the specified `redirect_uri`. After extracting verifying the `state` and extracting the `code` you can obtain the refresh token via:

```
print(reddit.auth.authorize(code))
print(reddit.user.me())
```

The first line of output is the `refresh_token`. You can save this for later use (see *Using a Saved Refresh Token*).

The second line of output reveals the name of the Redditor that completed the code flow. It also indicates that the `reddit` instance is now associated with that account.

The code flow can be used with an **installed** application just as described above with one change: set the value of `client_secret` to `None` when initializing *Reddit*.

1.3.3 Implicit Flow

The **implicit flow** requires a similar instantiation of the *Reddit* class as done in *Code Flow*, however, the token is returned directly as part of the redirect. For the implicit flow call *url()* like so:

```
print(reddit.auth.url(['identity'], '...', implicit=True)
```

Then use *implicit()* to provide the authorization to the *Reddit* instance.

1.3.4 Read-Only Mode

All application types support a read-only mode. Read-only mode provides access to Reddit like a logged out user would see including the default Subreddits in the `reddit.front` listings.

In the absence of a `refresh_token` both *Code Flow* and *Implicit Flow* applications start in the **read-only** mode. With such applications **read-only** mode is disabled when *authorize()*, or *implicit()* are successfully called. *Password Flow* applications start up with **read-only** mode disabled.

Read-only mode can be toggled via:

```
# Enable read-only mode
reddit.read_only = True

# Disable read-only mode (must have a valid authorization)
reddit.read_only = False
```

Application-Only Flows

The following flows are the **read-only mode** flows for Reddit applications

Application-Only (Client Credentials)

This is the default flow for **read-only mode** in script and web applications. The idea behind this is that Reddit *can* trust these applications as coming from a given developer, however the application requires no logged-in user context.

An installed application *cannot* use this flow, because Reddit requires a `client_secret` to be given it this flow is being used. In other words, installed applications are not considered confidential clients.

Application-Only (Installed Client)

This is the default flow for **read-only mode** in installed applications. The idea behind this is that Reddit *might not be able* to trust these applications as coming from a given developer. This would be able to happen if someone other than the developer can potentially replicate the client information and then pretend to be the application, such as in installed applications where the end user could retrieve the `client_id`.

Note: No benefit is really gained from this in script or web apps. The one exception is for when a script or web app has multiple end users, this will allow you to give Reddit the information needed in order to distinguish different users of your app from each other (as the supplied device id *should* be a unique string per both device (in the case of a web app, server) and user (in the case of a web app, browser session).

1.3.5 Using a Saved Refresh Token

A saved refresh token can be used to immediately obtain an authorized instance of *Reddit* like so:

```
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     refresh_token='WeheY7PwgeCZj4S3QgUcLhKE5S2s4eAYdxM',
                     user_agent='testscript by u/fakebot3')
print(reddit.auth.scopes())
```

The output from the above code displays which scopes are available on the *Reddit* instance.

Note: Observe that `redirect_uri` does not need to be provided in such cases. It is only needed when *url()* is used.

1.4 Configuring PRAW

1.4.1 Configuration Options

PRAW's configuration options are broken down into the following categories:

- *Basic Configuration Options*
- *OAuth Configuration Options*
- *Reddit Site Configuration Options*
- *Custom Configuration Options*

All of these options can be provided in any of the ways mentioned in *Configuring PRAW*.

Basic Configuration Options

check_for_updates When `true`, check for new versions of PRAW. When a newer version of PRAW is available a message is reported via standard out (default: `true`).

user_agent (Required) A unique description of your application. The following format is recommended according to Reddit's API Rules: `<platform>:<app ID>:<version string> (by /u/<reddit username>)`.

OAuth Configuration Options

client_id (Required) The OAuth client id associated with your registered Reddit application. See *Authenticating via OAuth* for instructions on registering a Reddit application.

client_secret The OAuth client secret associated with your registered Reddit application. This option is required for all application types, however, the value must be set to `None` for **installed** applications.

refresh_token For either **web** applications, or **installed** applications using the code flow, you can directly provide a previously obtained refresh token. Using a **web** application in conjunction with this option is useful, for example, if you prefer to not have your username and password available to your program, as required for a **script** application. See: *Obtaining a Refresh Token* and *Using a Saved Refresh Token*

redirect_uri The redirect URI associated with your registered Reddit application. This field is unused for **script** applications and is only needed for both **web** applications, and **installed** applications when the *`url()`* method is used.

password The password of the Reddit account associated with your registered Reddit **script** application. This field is required for **script** applications, and PRAW assumes it is working with a **script** application by its presence.

username The username of the Reddit account associated with your registered Reddit **script** application. This field is required for **script** applications, and PRAW assumes it is working with a **script** application by its presence.

Reddit Site Configuration Options

PRAW can be configured to work with instances of Reddit which are not hosted at reddit.com. The following options may need to be updated in order to successfully access a third-party Reddit site:

comment_kind The type prefix for comments on the Reddit instance (default: `t1_`).

message_kind The type prefix for messages on the Reddit instance (default: `t4_`).

oauth_url The URL used to access the Reddit instance's API (default: https://oauth.reddit.com).

reddit_url The URL used to access the Reddit instance. PRAW assumes the endpoints for establishing OAuth authorization are accessible under this URL (default: https://www.reddit.com).

redditor_kind The type prefix for redditors on the Reddit instance (default: `t2_`).

short_url The URL used to generate short links on the Reddit instance (default: https://redd.it).

submission_kind The type prefix for submissions on the Reddit instance (default: `t3_`).

subreddit_kind The type prefix for subreddits on the Reddit instance (default: `t5_`).

Custom Configuration Options

Your application can utilize PRAW's configuration system in order to provide its own custom settings.

For instance you might want to add an `app_debugging: true` option to your application's `praw.ini` file. To retrieve the value of this custom option from an instance of *Reddit* you can execute:

```
reddit.config.custom['app_debugging']
```

Note: Custom PRAW configuration environment variables are not supported. You can directly access environment variables via `os.getenv`.

Configuration options can be provided to PRAW in one of three ways:

1.4.2 praw.ini Files

PRAW comes with a `praw.ini` file in the package directory, and looks for user defined `praw.ini` files in a few other locations:

1. In the current working directory at the time *Reddit* is initialized.
2. In the launching user's config directory. This directory, if available, is detected in order as one of the following:
    1. In the directory specified by the `XDG_CONFIG_HOME` environment variable on operating systems that define such an environment variable (some modern Linux distributions).
    2. In the directory specified by `$HOME/.config` if the `HOME` environment variable is defined (Linux and Mac OS systems).
    3. In the directory specified by the `APPDATA` environment variable (Windows).

Format of praw.ini

`praw.ini` uses the INI file format, which can contain multiple groups of settings separated into sections. PRAW refers to each section as a `site`. The default site, `DEFAULT`, is provided in the package's `praw.ini` file. This site defines the default settings for interaction with Reddit. The contents of the package's `praw.ini` file are:

```
[DEFAULT]
# A boolean to indicate whether or not to check for package updates.
check_for_updates=True
```

(continues on next page)

(continued from previous page)

```
# Object to kind mappings
comment_kind=t1
message_kind=t4
redditor_kind=t2
submission_kind=t3
subreddit_kind=t5
trophy_kind=t6

# The URL prefix for OAuth-related requests.
oauth_url=https://oauth.reddit.com

# The URL prefix for regular requests.
reddit_url=https://www.reddit.com

# The URL prefix for short URLs.
short_url=https://redd.it
```

Warning: Avoid modifying the package's `praw.ini` file. Prefer instead to override its values in your own `praw.ini` file. You can even override settings of the `DEFAULT` site in user defined `praw.ini` files.

Defining Additional Sites

In addition to the `DEFAULT` site, additional sites can be configured in user defined `praw.ini` files. All sites inherit settings from the `DEFAULT` site and can override whichever settings desired.

Defining additional sites is a convenient way to store *OAuth credentials* for various accounts, or distinct OAuth applications. For example if you have three separate bots, you might create a site for each:

```
[bot1]
client_id=Y4PJOclpDQy3xZ
client_secret=UkGLTe6oqsMk5nHCJTHLrwgvHpr
password=pni9ubeht4wd50gk
username=fakebot1

[bot2]
client_id=6abrJJdcIqbclb
client_secret=Kcn6Bj8CClyu4FjVO77MYlTynfj
password=mi1ky2qzpiq8s59j
username=fakebot2

[bot3]
client_id=SI8pN3DSbt0zor
client_secret=xaxkj7HNh8kwg8e5t4m6KvSrbTI
password=1guiwevlfo00esyy
username=fakebot3
```

Choosing a Site

Site selection is done via the `site_name` parameter to *Reddit*. For example, to use the settings defined for `bot2` as shown above, initialize *Reddit* like so:

```
reddit = praw.Reddit('bot2', user_agent='bot2 user agent')
```

Note: In the above example you can obviate passing `user_agent` if you add the setting `user_agent=...` in the `[bot2]` site definition.

A site can also be selected via a `praw_site` environment variable. This approach has precedence over the `site_name` parameter described above.

Using Interpolation

By default PRAW doesn't apply any interpolation on the config file but this can be changed with the `config_interpolation` parameter which can be set to "basic" or "extended".

This can be useful to separate the components of the `user_agent` into individual variables, for example:

```
[bot1]
bot_name=MyBot
bot_version=1.2.3
bot_author=MyUser
user_agent=script:%(bot_name)s:v%(bot_version)s (by /u/%(bot_author)s)
```

This uses basic interpolation thus *Reddit* need to be initialized as follows:

```
reddit = praw.Reddit('bot1', config_interpolation='basic')
```

Then the value of `reddit.config.user_agent` will be `script:MyBot:v1.2.3 (by /u/MyUser)`.

See Interpolation of values for details.

Warning: The ConfigParser instance is cached internally at the class level, it is shared across all instances of *Reddit* and once set it's not overridden by future invocations.

1.4.3 Keyword Arguments to `Reddit`

Most of PRAW's documentation will demonstrate configuring PRAW through the use of keyword arguments when initializing instances of *Reddit*. All of the *Configuration Options* can be specified using a keyword argument of the same name.

For example, if we wanted to explicitly pass the information for `bot3` defined in *the praw.ini custom site example* without using the `bot3` site, we would initialize *Reddit* as:

```
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     password='1guiwevlfo00esyy',
                     user_agent='testscript by /u/fakebot3',
                     username='fakebot3')
```

1.4.4 PRAW Environment Variables

The highest priority configuration options can be passed to a program via environment variables prefixed with `praw_`.

For example, you can invoke your script as follows:

```
praw_username=bboe praw_password=not_my_password python my_script.py
```

The `username` and `password` provided via environment variables will override any such values passed directly when initializing an instance of *`Reddit`*, as well as any such values contained in a `praw.ini` file.

All *Configuration Options* can be provided in this manner, except for custom options.

Environment variables have the highest priority, followed by keyword arguments to *`Reddit`*, and finally settings in `praw.ini` files.

1.4.5 Using an HTTP or HTTPS proxy with PRAW

PRAW internally relies upon the requests package to handle HTTP requests. Requests supports use of `HTTP_PROXY` and `HTTPS_PROXY` environment variables in order to proxy HTTP and HTTPS requests respectively [ref].

Given that PRAW exclusively communicates with Reddit via HTTPS, only the `HTTPS_PROXY` option should be required.

For example, if you have a script named `prawbot.py`, the `HTTPS_PROXY` environment variable can be provided on the command line like so:

```
HTTPS_PROXY=https://localhost:3128 ./prawbot.py
```

1.4.6 Configuring a custom requests Session

PRAW uses requests to handle networking. If your use-case requires custom configuration, it is possible to configure a Session and then use it with PRAW.

For example, some networks use self-signed SSL certificates when connecting to HTTPS sites. By default, this would raise an exception in Requests. To use a self-signed SSL certificate without an exception from Requests, first export the certificate as a `.pem` file. Then configure PRAW like so:

```
import praw
from requests import Session


session = Session()
session.verify = '/path/to/certfile.pem'
reddit = praw.Reddit(client_id='SI8pN3DSbt0zor',
                     client_secret='xaxkj7HNh8kwg8e5t4m6KvSrbTI',
                     password='1guiwevlfo00esyy',
                     requestor_kwargs={'session': session},  # pass Session
                     user_agent='testscript by /u/fakebot3',
                     username='fakebot3')
```

The code above creates a Session and configures it to use a custom certificate, then passes it as a parameter when creating the *`Reddit`* instance. Note that the example above uses a *Password Flow* authentication type, but this method will work for any authentication type.

1.5 Running Multiple Instances of PRAW

PRAW, as of version 4, performs rate limiting dynamically based on the HTTP response headers from Reddit. As a result you can safely run a handful of PRAW instances without any additional configuration.

Note: Running more than a dozen or so instances of PRAW concurrently may occasionally result in exceeding Reddit's rate limits as each instance can only guess how many other instances are running.

If you are authorized on other users' behalf, each authorization should have its own rate limit, even when running from a single IP address.

1.5.1 Multiple Programs

The recommended way to run multiple instances of PRAW is to simply write separate independent Python programs. With this approach one program can monitor a comment stream and reply as needed, and another program can monitor a submission stream, for example.

If these programs need to share data consider using a third-party system such as a database or queuing system.

1.5.2 Multiple Threads

Warning: PRAW is not thread safe.

In a nutshell, instances of *Reddit* are not thread-safe for a number of reasons in its own code and each instance depends on an instance of `requests.Session`, which is not thread-safe [ref].

In theory having a unique *Reddit* instance for each thread should work. However, until someone perpetually volunteers to be PRAW's thread safety instructor, little to no support will go toward any PRAW issues that could be affected by the use of multiple threads. Consider using multiple processes instead.

Please see this discussion for more information.

1.6 Logging in PRAW

It is occasionally useful to observe the HTTP requests that PRAW is issuing. To do so you have to configure and enable logging.

Add the following to your code to log everything available:

```
import logging

handler = logging.StreamHandler()
handler.setLevel(logging.DEBUG)
logger = logging.getLogger('prawcore')
logger.setLevel(logging.DEBUG)
logger.addHandler(handler)
```

When properly configured, HTTP requests that are issued should produce output similar to the following:

```
Fetching: GET https://oauth.reddit.com/api/v1/me
Data: None
Params: {'raw_json': 1}
Response: 200 (876 bytes)
```

For more information on logging, see `logging.Logger`.

1.7 The Reddit Instance

class praw.**Reddit**(*site_name: str = None, config_interpolation: Optional[str] = None, requestor_class: Optional[Type[prawcore.requestor.Requestor]] = None, requestor_kwargs: Dict[str, Any] = None, **config_settings*)

The Reddit class provides convenient access to Reddit's API.

Instances of this class are the gateway to interacting with Reddit's API through PRAW. The canonical way to obtain an instance of this class is via:

```
import praw
reddit = praw.Reddit(client_id='CLIENT_ID',
                     client_secret="CLIENT_SECRET", password='PASSWORD',
                     user_agent='USERAGENT', username='USERNAME')
```

__init__(*site_name: str = None, config_interpolation: Optional[str] = None, requestor_class: Optional[Type[prawcore.requestor.Requestor]] = None, requestor_kwargs: Dict[str, Any] = None, **config_settings*)

Initialize a Reddit instance.

Parameters

- **site_name** – The name of a section in your `praw.ini` file from which to load settings from. This parameter, in tandem with an appropriately configured `praw.ini`, file is useful if you wish to easily save credentials for different applications, or communicate with other servers running Reddit. If `site_name` is `None`, then the site name will be looked for in the environment variable praw_site. If it is not found there, the DEFAULT site will be used.
- **requestor_class** – A class that will be used to create a requestor. If not set, use `prawcore.Requestor` (default: None).
- **requestor_kwargs** – Dictionary with additional keyword arguments used to initialize the requestor (default: None).

Additional keyword arguments will be used to initialize the *Config* object. This can be used to specify configuration settings during instantiation of the *Reddit* instance. For more details please see *Configuring PRAW*.

Required settings are:

- client_id
- client_secret (for installed applications set this value to `None`)
- user_agent

The `requestor_class` and `requestor_kwargs` allow for customization of the requestor *Reddit* will use. This allows, e.g., easily adding behavior to the requestor or wrapping its `Session` in a caching layer. Example usage:

```
import json, betamax, requests

class JSONDebugRequestor(Requestor):
    def request(self, *args, **kwargs):
        response = super().request(*args, **kwargs)
        print(json.dumps(response.json(), indent=4))
        return response

my_session = betamax.Betamax(requests.Session())
```

(continues on next page)

(continued from previous page)

```
reddit = Reddit(..., requestor_class=JSONDebugRequestor,
                requestor_kwargs={'session': my_session})
```

auth = None
An instance of *Auth*.

Provides the interface for interacting with installed and web applications. See *Obtain the Authorization URL*

comment(*id: Optional[str] = None*, *url: Optional[str] = None*)
Return a lazy instance of *Comment* for `id`.

Parameters

- **id** – The ID of the comment.
- **url** – A permalink pointing to the comment.

Note: If you want to obtain the comment's replies, you will need to call *refresh()* on the returned *Comment*.

---

domain(*domain: str*)
Return an instance of *DomainListing*.

Parameters **domain** – The domain to obtain submission listings for.

front = None
An instance of *Front*.

Provides the interface for interacting with front page listings. For example:

```
for submission in reddit.front.hot():
    print(submission)
```

get(*path: str*, *params: Union[str, Dict[str, str], None] = None*)
Return parsed objects returned from a GET request to `path`.

Parameters

- **path** – The path to fetch.
- **params** – The query parameters to add to the request (default: None).

inbox = None
An instance of *Inbox*.

Provides the interface to a user's inbox which produces *Message*, *Comment*, and *Submission* instances. For example to iterate through comments which mention the authorized user run:

```
for comment in reddit.inbox.mentions():
    print(comment)
```

info(*fullnames: Optional[Iterable[str]] = None*, *url: Optional[str] = None*) → Generator[[Union[praw.models.reddit.subreddit.Subreddit, praw.models.reddit.comment.Comment, praw.models.reddit.submission.Submission], None], None]
Fetch information about each item in `fullnames` or from `url`.

Parameters

- **fullnames** – A list of fullnames for comments, submissions, and/or subreddits.

- **url** – A url (as a string) to retrieve lists of link submissions from.

Returns A generator that yields found items in their relative order.

Items that cannot be matched will not be generated. Requests will be issued in batches for each 100 fullnames.

Note: For comments that are retrieved via this method, if you want to obtain its replies, you will need to call *refresh()* on the yielded *Comment*.

Note: When using the URL option, it is important to be aware that URLs are treated literally by Reddit's API. As such, the URLs "youtube.com" and "https://www.youtube.com" will provide a different set of submissions.

---

live = None

An instance of *LiveHelper*.

Provides the interface for working with *LiveThread* instances. At present only new LiveThreads can be created.

```
reddit.live.create('title', 'description')
```

multireddit = None

An instance of *MultiredditHelper*.

Provides the interface to working with *Multireddit* instances. For example you can obtain a *Multireddit* instance via:

```
reddit.multireddit('samuraisam', 'programming')
```

patch(*path: str, data: Union[Dict[str, Union[str, Any]], bytes, IO, str, None] = None*) → Any

Return parsed objects returned from a PATCH request to `path`.

Parameters

- **path** – The path to fetch.
- **data** – Dictionary, bytes, or file-like object to send in the body of the request (default: None).

post(*path: str, data: Union[Dict[str, Union[str, Any]], bytes, IO, str, None] = None, files: Optional[Dict[str, IO]] = None, params: Union[str, Dict[str, str], None] = None*) → Any

Return parsed objects returned from a POST request to `path`.

Parameters

- **path** – The path to fetch.
- **data** – Dictionary, bytes, or file-like object to send in the body of the request (default: None).
- **files** – Dictionary, filename to file (like) object mapping (default: None).
- **params** – The query parameters to add to the request (default: None).

put(*path: str, data: Union[Dict[str, Union[str, Any]], bytes, IO, str, None] = None*)

Return parsed objects returned from a PUT request to `path`.

Parameters

- **path** – The path to fetch.
- **data** – Dictionary, bytes, or file-like object to send in the body of the request (default: None).

random_subreddit(*nsfw: bool = False*) → praw.models.reddit.subreddit.Subreddit
Return a random lazy instance of *Subreddit*.

Parameters **nsfw** – Return a random NSFW (not safe for work) subreddit (default: False).

read_only
Return True when using the ReadOnlyAuthorizer.

redditor(*name: Optional[str] = None, fullname: Optional[str] = None*) → praw.models.reddit.redditor.Redditor
Return a lazy instance of *Redditor*.

Parameters

- **name** – The name of the redditor.
- **fullname** – The fullname of the redditor, starting with `t2_`.

Either `name` or `fullname` can be provided, but not both.

redditors = None
An instance of *Redditors*.

Provides the interface for Redditor discovery. For example to iterate over the newest Redditors, run:

```
for redditor in reddit.redditors.new(limit=None):
    print(redditor)
```

request(*method: str, path: str, params: Union[str, Dict[str, str], None] = None, data: Union[Dict[str, Union[str, Any]], bytes, IO, str, None] = None, files: Optional[Dict[str, IO]] = None*) → Any
Return the parsed JSON data returned from a request to URL.

Parameters

- **method** – The HTTP method (e.g., GET, POST, PUT, DELETE).
- **path** – The path to fetch.
- **params** – The query parameters to add to the request (default: None).
- **data** – Dictionary, bytes, or file-like object to send in the body of the request (default: None).
- **files** – Dictionary, filename to file (like) object mapping (default: None).

submission(*id: Optional[str] = None, url: Optional[str] = None*) → praw.models.reddit.submission.Submission
Return a lazy instance of *Submission*.

Parameters

- **id** – A Reddit base36 submission ID, e.g., `2gmzqe`.
- **url** – A URL supported by *id_from_url()*.‘.

Either `id` or `url` can be provided, but not both.

subreddit = None
An instance of *SubredditHelper*.

Provides the interface to working with *Subreddit* instances. For example to create a Subreddit run:

```
reddit.subreddit.create('coolnewsubname')
```

To obtain a lazy a *Subreddit* instance run:

```
reddit.subreddit('redditdev')
```

Note that multiple subreddits can be combined and filtered views of r/all can also be used just like a subreddit:

```
reddit.subreddit('redditdev+learnpython+botwatch')
reddit.subreddit('all-redditdev-learnpython')
```

subreddits = None
An instance of *Subreddits*.

Provides the interface for *Subreddit* discovery. For example to iterate over the set of default subreddits run:

```
for subreddit in reddit.subreddits.default(limit=None):
    print(subreddit)
```

user = None
An instance of *User*.

Provides the interface to the currently authorized *Redditor*. For example to get the name of the current user run:

```
print(reddit.user.me())
```

1.7.1 reddit.front

class praw.models.**Front**(*reddit: Reddit*)
Front is a Listing class that represents the front page.

__init__(*reddit: Reddit*)
Initialize a Front instance.

best(***generator_kwargs*) → Generator[[praw.models.reddit.submission.Submission, None], None]
Return a *ListingGenerator* for best items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

comments
Provide an instance of *CommentHelper*.

For example, to output the author of the 25 most recent comments of /r/redditdev execute:

```
for comment in reddit.subreddit('redditdev').comments(limit=25):
    print(comment.author)
```

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise ValueError if time_filter is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

gilded(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for gilded items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get gilded items in subreddit `r/test`:

```
for item in reddit.subreddit('test').gilded():
    print(item.id)
```

hot(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

new(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

random_rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for random rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get random rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').random_rising():
    print(submission.title)
```

rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').rising():
    print(submission.title)
```

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

1.7.2 reddit.inbox

class praw.models.**Inbox**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Inbox is a Listing class that represents the Inbox.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

all(***generator_kwargs*) → Generator[[Union[Message, Comment], None], None]
Return a *ListingGenerator* for all inbox comments and messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To output the type and ID of all items available via this listing do:

```
for item in reddit.inbox.all(limit=None):
    print(repr(item))
```

collapse(*items: List[Message]*)
Mark an inbox message as collapsed.

Parameters **items** – A list containing instances of *Message*.

Requests are batched at 25 items (reddit limit).

For example, to collapse all unread Messages, try:

```
from praw.models import Message
unread_messages = []
for item in reddit.inbox.unread(limit=None):
    if isinstance(item, Message):
        unread_messages.append(item)
reddit.inbox.collapse(unread_messages)
```

See also:

Message.uncollapse()

comment_replies(***generator_kwargs*) → Generator[[Comment, None], None]
Return a *ListingGenerator* for comment replies.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To output the author of one request worth of comment replies try:

```
for reply in reddit.inbox.comment_replies():
    print(reply.author)
```

mark_read(*items: List[Union[Comment, Message]]*)
Mark Comments or Messages as read.

Parameters **items** – A list containing instances of *Comment* and/or *Message* to be be marked as read relative to the authorized user's inbox.

Requests are batched at 25 items (reddit limit).

For example, to mark all unread Messages as read, try:

```
from praw.models import Message
unread_messages = []
for item in reddit.inbox.unread(limit=None):
    if isinstance(item, Message):
        unread_messages.append(item)
reddit.inbox.mark_read(unread_messages)
```

See also:

Comment.mark_read() and *Message.mark_read()*

mark_unread(*items: List[Union[Comment, Message]]*)
Unmark Comments or Messages as read.

Parameters **items** – A list containing instances of *Comment* and/or *Message* to be be marked as unread relative to the authorized user's inbox.

Requests are batched at 25 items (reddit limit).

For example, to mark the first 10 items as unread try:

```
to_unread = list(reddit.inbox.all(limit=10))
reddit.inbox.mark_unread(to_unread)
```

See also:

Comment.mark_unread() and *Message.mark_unread()*

mentions(***generator_kwargs*) → Generator[[Comment, None], None]
Return a *ListingGenerator* for mentions.

A mention is *Comment* in which the authorized redditor is named in its body like `/u/redditor_name`.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to output the author and body of the first 25 mentions try:

```
for mention in reddit.inbox.mentions(limit=25):
    print('{}\n{}\n'.format(mention.author, mention.body))
```

message (*message_id: str*) → Message
Return a Message corresponding to `message_id`.

Parameters **message_id** – The base36 id of a message.

For example:

```
message = reddit.inbox.message('7bn1gu')
```

messages (***generator_kwargs*) → Generator[[Message, None], None]
Return a *ListingGenerator* for inbox messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to output the subject of the most recent 5 messages try:

```
for message in reddit.inbox.messages(limit=5):
    print(message.subject)
```

classmethod parse (*data: Dict[str, Any], reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

sent (***generator_kwargs*) → Generator[[Message, None], None]
Return a *ListingGenerator* for sent messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to output the recipient of the most recent 15 messages try:

```
for message in reddit.inbox.sent(limit=15):
    print(message.dest)
```

stream (***stream_options*) → Generator[[Union[Message, Comment], None], None]
Yield new inbox items as they become available.

Items are yielded oldest first. Up to 100 historical items will initially be returned.

Keyword arguments are passed to *stream_generator()*.

For example, to retrieve all new inbox items, try:

```
for item in reddit.inbox.stream():
    print(item)
```

submission_replies (***generator_kwargs*) → Generator[[Comment, None], None]
Return a *ListingGenerator* for submission replies.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To output the author of one request worth of submission replies try:

```
for reply in reddit.inbox.submission_replies():
    print(reply.author)
```

uncollapse(*items: List[Message]*)
Mark an inbox message as uncollapsed.

Parameters **items** – A list containing instances of *Message*.

Requests are batched at 25 items (reddit limit).

For example, to uncollapse all unread Messages, try:

```
from praw.models import Message
unread_messages = []
for item in reddit.inbox.unread(limit=None):
    if isinstance(item, Message):
        unread_messages.append(item)
reddit.inbox.uncollapse(unread_messages)
```

See also:

Message.collapse()

unread(*mark_read: bool = False*, ***generator_kwargs*) → Generator[[Union[Message, Comment], None], None]
Return a *ListingGenerator* for unread comments and messages.

Parameters **mark_read** – Marks the inbox as read (default: False).

Note: This only marks the inbox as read not the messages. Use *Inbox.mark_read()* to mark the messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to output the author of unread comments try:

```
from praw.models import Comment
for item in reddit.inbox.unread(limit=None):
    if isinstance(item, Comment):
        print(item.author)
```

1.7.3 reddit.live

class praw.models.**LiveHelper**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Provide a set of functions to interact with LiveThreads.

__call__(*id: str*) → praw.models.reddit.live.LiveThread
Return a new lazy instance of *LiveThread*.

This method is intended to be used as:

```
livethread = reddit.live('ukaeu1ik4sw5')
```

Parameters **id** – A live thread ID, e.g., `ukaeu1ik4sw5`.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

create(*title: str*, *description: Optional[str] = None*, *nsfw: bool = False*, *resources: str = None*) → praw.models.reddit.live.LiveThread

Create a new LiveThread.

Parameters

- **title** – The title of the new LiveThread.
- **description** – (Optional) The new LiveThread's description.
- **nsfw** – (boolean) Indicate whether this thread is not safe for work (default: False).
- **resources** – (Optional) Markdown formatted information that is useful for the LiveThread.

Returns The new LiveThread object.

info(*ids: List[str]*) → Generator[[praw.models.reddit.live.LiveThread, None], None]

Fetch information about each live thread in `ids`.

Parameters **ids** – A list of IDs for a live thread.

Returns A generator that yields *LiveThread* instances.

Live threads that cannot be matched will not be generated. Requests will be issued in batches for each 100 IDs.

Note: This method doesn't support IDs for live updates.

Usage:

```
ids = ['3rgnbke2rai6hen7ciytwcxadi',
       'sw7bubeycai6hey4ciytwamw3a',
       't8jnufucss07']
for thread in reddit.live.info(ids)
    print(thread.title)
```

now() → Optional[praw.models.reddit.live.LiveThread]

Get the currently featured live thread.

Returns The *LiveThread* object, or `None` if there is no currently featured live thread.

Usage:

```

```

thread = reddit.live.now() # LiveThread object or None

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.7.4 reddit.multireddit

class praw.models.**MultiredditHelper**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Provide a set of functions to interact with Multireddits.

__call__(*redditor: Union[str, praw.models.reddit.redditor.Redditor], name: str*) → praw.models.reddit.multi.Multireddit
Return a lazy instance of *Multireddit*.

Parameters

- **redditor** – A redditor name (e.g., 'spez') or *Redditor* instance who owns the multireddit.
- **name** – The name of the multireddit.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

create(*display_name: str, subreddits: Union[str, praw.models.reddit.subreddit.Subreddit], description_md: Optional[str] = None, icon_name: Optional[str] = None, key_color: Optional[str] = None, visibility: str = 'private', weighting_scheme: str = 'classic'*) → praw.models.reddit.multi.Multireddit
Create a new multireddit.

Parameters

- **display_name** – The display name for the new multireddit.
- **subreddits** – Subreddits to add to the new multireddit.
- **description_md** – (Optional) Description for the new multireddit, formatted in markdown.
- **icon_name** – (Optional) Can be one of: art and design, ask, books, business, cars, comics, cute animals, diy, entertainment, food and drink, funny, games, grooming, health, life advice, military, models pinup, music, news, philosophy, pictures and gifs, science, shopping, sports, style, tech, travel, unusual stories, video, or None.
- **key_color** – (Optional) RGB hex color code of the form '#FFFFFF'.
- **visibility** – (Optional) Can be one of: hidden, private, public (default: private).
- **weighting_scheme** – (Optional) Can be one of: classic, fresh (default: classic).

Returns The new Multireddit object.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any
Return an instance of cls from data.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.7.5 reddit.redditors

class praw.models.**Redditors**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Redditors is a Listing class that provides various Redditor lists.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

new(***generator_kwargs*) → Generator[[Subreddit, None], None]
Return a *ListingGenerator* for new Redditors.

:returns Redditor profiles, which are a type of *Subreddit*.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

partial_redditors(*ids: Iterable[str]*) → Generator[[praw.models.redditors.PartialRedditor, None], None]
Get user summary data by redditor IDs.

Parameters **ids** – An iterable of redditor fullname IDs.

Returns A iterator producing types.SimpleNamespace objects.

Each ID must be prefixed with "*t2_*".

Invalid IDs are ignored by the server.

popular(***generator_kwargs*) → Generator[[Subreddit, None], None]
Return a *ListingGenerator* for popular Redditors.

:returns Redditor profiles, which are a type of *Subreddit*.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

search(*query: str, **generator_kwargs*) → Generator[[Subreddit, None], None]
Return a *ListingGenerator* of Redditors for `query`.

Parameters **query** – The query string to filter Redditors by.

:returns *Redditor*s.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

stream(***stream_options*) → Generator[[Subreddit, None], None]
Yield new Redditors as they are created.

Redditors are yielded oldest first. Up to 100 historical Redditors will initially be returned.

Keyword arguments are passed to *stream_generator()*.

Returns Redditor profiles, which are a type of *Subreddit*.

1.7.6 reddit.subreddit

class praw.models.**SubredditHelper**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Provide a set of functions to interact with Subreddits.

__call__(*display_name: str*) → praw.models.reddit.subreddit.Subreddit
Return a lazy instance of *Subreddit*.

Parameters **display_name** – The name of the subreddit.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

create(*name: str*, *title: Optional[str] = None*, *link_type: str = 'any'*, *subreddit_type: str = 'public'*, *wikimode: str = 'disabled'*, ***other_settings*) → praw.models.reddit.subreddit.Subreddit
Create a new subreddit.

Parameters

- **name** – The name for the new subreddit.
- **title** – The title of the subreddit. When `None` or `''` use the value of `name`.
- **link_type** – The types of submissions users can make. One of `any`, `link`, `self` (default: any).
- **subreddit_type** – One of `archived`, `employees_only`, `gold_only`, `gold_restricted`, `private`, `public`, `restricted` (default: public).
- **wikimode** – One of `anyone`, `disabled`, `modonly`.

See *update()* for documentation of other available settings.

Any keyword parameters not provided, or set explicitly to None, will take on a default value assigned by the Reddit server.

classmethod **parse**(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.7.7 reddit.subreddits

class praw.models.**Subreddits**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Subreddits is a Listing class that provides various subreddit lists.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

default(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* for default subreddits.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

gold(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* for gold subreddits.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

new(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* for new subreddits.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

popular(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* for popular subreddits.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

recommended(*subreddits: List[Union[str, praw.models.reddit.subreddit.Subreddit]]*, *omit_subreddits: Optional[List[Union[str, praw.models.reddit.subreddit.Subreddit]]] = None*) → List[praw.models.reddit.subreddit.Subreddit]
Return subreddits recommended for the given list of subreddits.

Parameters

- **subreddits** – A list of Subreddit instances and/or subreddit names.
- **omit_subreddits** – A list of Subreddit instances and/or subreddit names to exclude from the results (Reddit's end may not work as expected).

search(*query: str*, ***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* of subreddits matching `query`.

Subreddits are searched by both their title and description. To search names only see `search_by_name`.

Parameters **query** – The query string to filter subreddits by.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

search_by_name(*query: str*, *include_nsfw: bool = True*, *exact: bool = False*) → List[praw.models.reddit.subreddit.Subreddit]
Return list of Subreddits whose names begin with `query`.

Parameters

- **query** – Search for subreddits beginning with this string.
- **include_nsfw** – Include subreddits labeled NSFW (default: True).
- **exact** – Return only exact matches to `query` (default: False).

search_by_topic(*query: str*) → List[praw.models.reddit.subreddit.Subreddit]
Return list of Subreddits whose topics match `query`.

Parameters **query** – Search for subreddits relevant to the search topic.

stream(***stream_options*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Yield new subreddits as they are created.

Subreddits are yielded oldest first. Up to 100 historical subreddits will initially be returned.

Keyword arguments are passed to *stream_generator()*.

1.7.8 reddit.user

class praw.models.**User**(*reddit: Reddit*)

The user class provides methods for the currently authenticated user.

__init__(*reddit: Reddit*)

Initialize a User instance.

This class is intended to be interfaced with through `reddit.user`.

blocked() → List[praw.models.reddit.redditor.Redditor]

Return a RedditorList of blocked Redditors.

contributor_subreddits(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]

Return a *ListingGenerator* of subreddits user is a contributor of.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

friends(*user: Union[str, praw.models.reddit.redditor.Redditor, None] = None*) → Union[List[praw.models.reddit.redditor.Redditor], praw.models.reddit.redditor.Redditor]

Return a RedditorList of friends or a Redditor in the friends list.

Parameters **user** – Checks to see if you are friends with the Redditor. Either an instance of *Redditor* or a string can be given.

Returns A list of Redditors, or a Redditor if you are friends with the given Redditor. The Redditor also has friend attributes.

Raises An instance of `prawcore.exceptions.BadRequest` if you are not friends with the specified Redditor.

karma() → Dict[praw.models.reddit.subreddit.Subreddit, Dict[str, int]]

Return a dictionary mapping subreddits to their karma.

The returned dict contains subreddits as keys. Each subreddit key contains a sub-dict that have keys for `comment_karma` and `link_karma`. The dict is sorted in descending karma order.

Note: Each key of the main dict is an instance of *Subreddit*. It is recommended to iterate over the dict in order to retrieve the values, preferably through `dict.items()`.

---

me(*use_cache: bool = True*) → Optional[praw.models.reddit.redditor.Redditor]

Return a *Redditor* instance for the authenticated user.

In *read_only* mode, this method returns `None`.

Parameters **use_cache** – When true, and if this function has been previously called, returned the cached version (default: True).

Note: If you change the Reddit instance's authorization, you might want to refresh the cached value. Prefer using separate Reddit instances, however, for distinct authorizations.

---

moderator_subreddits(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]

Return a *ListingGenerator* of moderated subreddits.

..warning:: **(Deprecated) This method will be removed in the next major** version of PRAW. Please use *Redditor.moderated()* instead.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

Note: This method will return a maximum of 100 moderated subreddits, ordered by subscriber count. To retrieve more than 100 moderated subreddits, please see *Redditor.moderated()*.

Usage:

```
for subreddit in reddit.user.moderator_subreddits():
    print(subreddit.display_name)
```

multireddits() → List[Multireddit]
Return a list of multireddits belonging to the user.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

preferences
Get an instance of *Preferences*.

The preferences can be accessed as a `dict` like so:

```
preferences = reddit.user.preferences()
print(preferences['show_link_flair'])
```

Preferences can be updated via:

```
reddit.user.preferences.update(show_link_flair=True)
```

The *Preferences.update()* method returns the new state of the preferences as a `dict`, which can be used to check whether a change went through. Changes with invalid types or parameter names fail silently.

```
original_preferences = reddit.user.preferences()
new_preferences = reddit.user.preferences.update(invalid_param=123)
print(original_preferences == new_preferences)  # True, no change
```

subreddits(***generator_kwargs*) → Generator[[praw.models.reddit.subreddit.Subreddit, None], None]
Return a *ListingGenerator* of subreddits the user is subscribed to.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

1.8 Working with PRAW's Models

1.8.1 Comment

class praw.models.**Comment**(*reddit: Reddit*, *id: Optional[str] = None*, *url: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

A class that represents a reddit comments.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`author`	Provides an instance of *`Redditor`*.
`body`	The body of the comment.
`created_utc`	Time the comment was created, represented in Unix Time.
`distinguished`	Whether or not the comment is distinguished.
`edited`	Whether or not the comment has been edited.
`id`	The ID of the comment.
`is_submitter`	Whether or not the comment author is also the author of the submission.
`link_id`	The submission ID that the comment belongs to.
`parent_id`	The ID of the parent comment. If it is a top-level comment, this returns the submission ID instead (prefixed with 't3').
`permalink`	A permalink for the comment. Comment objects from the inbox have a `context` attribute instead.
`replies`	Provides an instance of *`CommentForest`*.
`score`	The number of upvotes for the comment.
`stickied`	Whether or not the comment is stickied.
`submission`	Provides an instance of *`Submission`*. The submission that the comment belongs to.
`subreddit`	Provides an instance of *`Subreddit`*. The subreddit that the comment belongs to.
`subreddit_id`	The subreddit ID that the comment belongs to.

__init__(*reddit: Reddit*, *id: Optional[str] = None*, *url: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Construct an instance of the Comment object.

block()

Block the user who sent the item.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.block()

# or, identically:

comment.author.block()
```

clear_vote()

Clear the authenticated user's vote on the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK, but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.clear_vote()

comment = reddit.comment(id='dxolpyc')
comment.clear_vote()
```

collapse()
Mark the item as collapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and collapse it
message = next(inbox)
message.collapse()
```

See also *uncollapse()*

delete()
Delete the object.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.delete()

submission = reddit.submission('8dmv8z')
submission.delete()
```

disable_inbox_replies()
Disable inbox replies for the item.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.disable_inbox_replies()

submission = reddit.submission('8dmv8z')
submission.disable_inbox_replies()
```

See also *enable_inbox_replies()*

downvote()
Downvote the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK,

but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.downvote()

comment = reddit.comment(id='dxolpyc')
comment.downvote()
```

See also *upvote()*

edit(*body*)

Replace the body of the object with `body`.

Parameters **body** – The Markdown formatted content for the updated object.

Returns The current instance after updating its attributes.

Example usage:

```
comment = reddit.comment('dkk4qjd')

# construct the text of an edited comment
# by appending to the old body:
edited_body = comment.body + "Edit: thanks for the gold!"
comment.edit(edited_body)
```

enable_inbox_replies()

Enable inbox replies for the item.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.enable_inbox_replies()

submission = reddit.submission('8dmv8z')
submission.enable_inbox_replies()
```

See also *disable_inbox_replies()*

fullname

Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

gild()

Gild the author of the item.

Note: Requires the authenticated user to own Reddit Coins. Calling this method will consume Reddit Coins.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.gild()
```

(continues on next page)

(continued from previous page)

```
submission = reddit.submission('8dmv8z')
submission.gild()
```

static id_from_url(*url: str*) → str
Get the ID of a comment from the full URL.

is_root
Return True when the comment is a top level comment.

mark_read()
Mark a single inbox item as read.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox.unread()

for message in inbox:
    # process unread messages
```

See also *mark_unread()*

To mark the whole inbox as read with a single network request, use *praw.models.Inbox.mark_read()*

mark_unread()
Mark the item as unread.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox(limit=10)

for message in inbox:
    # process messages
```

See also *mark_read()*

mod
Provide an instance of *CommentModeration*.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.mod.approve()
```

parent() → Union[_Comment, Submission]
Return the parent of the comment.

The returned parent will be an instance of either *Comment*, or *Submission*.

If this comment was obtained through a *Submission*, then its entire ancestry should be immediately available, requiring no extra network requests. However, if this comment was obtained through other

means, e.g., `reddit.comment('COMMENT_ID')`, or `reddit.inbox.comment_replies`, then the returned parent may be a lazy instance of either *Comment*, or *Submission*.

Lazy comment example:

```
comment = reddit.comment('cklhv0f')
parent = comment.parent()
# `replies` is empty until the comment is refreshed
print(parent.replies)  # Output: []
parent.refresh()
print(parent.replies)  # Output is at least: [Comment(id='cklhv0f')]
```

Warning: Successive calls to *parent()* may result in a network request per call when the comment is not obtained through a *Submission*. See below for an example of how to minimize requests.

If you have a deeply nested comment and wish to most efficiently discover its top-most *Comment* ancestor you can chain successive calls to *parent()* with calls to *refresh()* at every 9 levels. For example:

```
comment = reddit.comment('dkk4qjd')
ancestor = comment
refresh_counter = 0
while not ancestor.is_root:
    ancestor = ancestor.parent()
    if refresh_counter % 9 == 0:
        ancestor.refresh()
    refresh_counter += 1
print('Top-most Ancestor: {}'.format(ancestor))
```

The above code should result in 5 network requests to Reddit. Without the calls to *refresh()* it would make at least 31 network requests.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

refresh()

Refresh the comment's attributes.

If using *Reddit.comment()* this method must be called in order to obtain the comment's replies.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.refresh()
```

replies

Provide an instance of *CommentForest*.

This property may return an empty list if the comment has not been refreshed with *refresh()*

Sort order and reply limit can be set with the `reply_sort` and `reply_limit` attributes before replies are fetched, including any call to *refresh()*:

```
comment.reply_sort = 'new'
comment.refresh()
replies = comment.replies
```

Note: The appropriate values for `reply_sort` include `best`, `top`, `new`, `controversial`, `old` and `q&a`.

reply(*body*)

Reply to the object.

Parameters **body** – The Markdown formatted content for a comment.

Returns A *Comment* object for the newly created comment or `None` if Reddit doesn't provide one.

A `None` value can be returned if the target is a comment or submission in a quarantined subreddit and the authenticated user has not opt-ed in to viewing the content. When this happens the comment will be sucessfully created on Reddit and can be retried by drawing the comment from the user's comment history.

Note: Some items, such as locked submissions/comments or non-replyable messages will throw `prawcore.exceptions.Forbidden` when attempting to reply to them.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.reply('reply')

comment = reddit.comment(id='dxolpyc')
comment.reply('reply')
```

report(*reason*)

Report this object to the moderators of its subreddit.

Parameters **reason** – The reason for reporting.

Raises *APIException* if `reason` is longer than 100 characters.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.report('report reason')

comment = reddit.comment(id='dxolpyc')
comment.report('report reason')
```

save(*category=None*)

Save the object.

Parameters **category** – (Premium) The category to save to. If your user does not have Reddit Premium this value is ignored by Reddit (default: `None`).

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.save(category="view later")
```

(continues on next page)

(continued from previous page)

```
comment = reddit.comment(id='dxolpyc')
comment.save()
```

See also *unsave()*

submission
Return the Submission object this comment belongs to.

uncollapse()
Mark the item as uncollapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and uncollapse it
message = next(inbox)
message.uncollapse()
```

See also *collapse()*

unsave()
Unsave the object.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.unsave()

comment = reddit.comment(id='dxolpyc')
comment.unsave()
```

See also *save()*

upvote()
Upvote the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK, but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.upvote()

comment = reddit.comment(id='dxolpyc')
comment.upvote()
```

See also *downvote()*

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other

than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.2 LiveThread

class praw.models.**LiveThread**(*reddit: Reddit*, *id: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

An individual LiveThread object.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
`created_utc`	The creation time of the live thread, in Unix Time.
`description`	Description of the live thread, as Markdown.
`description_html`	Description of the live thread, as HTML.
`id`	The ID of the live thread.
`nsfw`	A `bool` representing whether or not the live thread is marked as NSFW.

__getitem__(*update_id: str*) → _LiveUpdate

Return a lazy *LiveUpdate* instance.

Parameters **update_id** – A live update ID, e.g., `'7827987a-c998-11e4-a0b9-22000b6a88d2'`.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
update = thread['7827987a-c998-11e4-a0b9-22000b6a88d2']
update.thread     # LiveThread(id='ukaeu1ik4sw5')
update.id         # '7827987a-c998-11e4-a0b9-22000b6a88d2'
update.author     # 'umbrae'
```

__init__(*reddit: Reddit*, *id: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Initialize a lazy *LiveThread* instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **id** – A live thread ID, e.g., `'ukaeu1ik4sw5'`

contrib

Provide an instance of *LiveThreadContribution*.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
thread.contrib.add('### update')
```

contributor

Provide an instance of *LiveContributorRelationship*.

You can call the instance to get a list of contributors which is represented as *RedditorList* instance consists of *Redditor* instances. Those Redditor instances have `permissions` attributes as contributors:

```
thread = reddit.live('ukaeu1ik4sw5')
for contributor in thread.contributor():
    # prints `(Redditor(name='Acidtwist'), [u'all'])`
    print(contributor, contributor.permissions)
```

discussions(***generator_kwargs*) → Generator[[Submission, None], None]
Get submissions linking to the thread.

Parameters **generator_kwargs** – keyword arguments passed to *ListingGenerator* constructor.

Returns A *ListingGenerator* object which yields *Submission* object.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
for submission in thread.discussions(limit=None):
    print(submission.title)
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

report(*type: str*)
Report the thread violating the Reddit rules.

Parameters **type** – One of `'spam'`, `'vote-manipulation'`, `'personal-information'`, `'sexualizing-minors'`, `'site-breaking'`.

Usage:

```
thread = reddit.live('xyu8kmjvfrww')
thread.report('spam')
```

updates(***generator_kwargs*) → Generator[[_LiveUpdate, None], None]
Return a *ListingGenerator* yields *LiveUpdate* s.

Parameters **generator_kwargs** – keyword arguments passed to *ListingGenerator* constructor.

Returns A *ListingGenerator* object which yields *LiveUpdate* object.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
after = 'LiveUpdate_fefb3dae-7534-11e6-b259-0ef8c7233633'
for submission in thread.updates(limit=5, params={'after': after}):
    print(submission.body)
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other

than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.3 LiveUpdate

class praw.models.**LiveUpdate**(*reddit: Reddit*, *thread_id: Optional[str] = None*, *update_id: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

An individual `LiveUpdate` object.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
`author`	The `Redditor` who made the update.
`body`	Body of the update, as Markdown.
`body_html`	Body of the update, as HTML.
`created_utc`	The time the update was created, as Unix Time.
`stricken`	A `bool` representing whether or not the update was stricken (see `strike()`).

__init__(*reddit: Reddit*, *thread_id: Optional[str] = None*, *update_id: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Initialize a lazy `LiveUpdate` instance.

Either `thread_id` and `update_id`, or `_data` must be provided.

Parameters

- **reddit** – An instance of `Reddit`.
- **thread_id** – A live thread ID, e.g., `'ukaeu1ik4sw5'`.
- **update_id** – A live update ID, e.g., `'7827987a-c998-11e4-a0b9-22000b6a88d2'`.

Usage:

```
update = LiveUpdate(reddit, 'ukaeu1ik4sw5',
                    '7827987a-c998-11e4-a0b9-22000b6a88d2')
update.thread     # LiveThread(id='ukaeu1ik4sw5')
update.id         # '7827987a-c998-11e4-a0b9-22000b6a88d2'
update.author     # 'umbrae'
```

contrib

Provide an instance of `LiveUpdateContribution`.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
update = thread['7827987a-c998-11e4-a0b9-22000b6a88d2']
update.contrib  # LiveUpdateContribution instance
```

fullname

Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

thread
Return *LiveThread* object the update object belongs to.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.4 Message

class praw.models.**Message**(*reddit: Reddit*, *_data: Dict[str, Any]*)
A class for private messages.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`author`	Provides an instance of *Redditor*.
`body`	The body of the message.
`created_utc`	Time the message was created, represented in Unix Time.
`dest`	Provides an instance of *Redditor*. The recipient of the message.
`id`	The ID of the message.
`name`	The full ID of the message, prefixed with 't4'.
`subject`	The subject of the message.
`was_comment`	Whether or not the message was a comment reply.

__init__(*reddit: Reddit*, *_data: Dict[str, Any]*)
Construct an instance of the Message object.

block()
Block the user who sent the item.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.block()

# or, identically:

comment.author.block()
```

collapse()
Mark the item as collapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and collapse it
message = next(inbox)
message.collapse()
```

See also *uncollapse()*

delete()
Delete the message.

Note: Reddit does not return an indication of whether or not the message was successfully deleted.

For example, to delete the most recent message in your inbox:

```
next(reddit.inbox.all()).delete()
```

fullname
Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

mark_read()
Mark a single inbox item as read.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox.unread()

for message in inbox:
    # process unread messages
```

See also *mark_unread()*

To mark the whole inbox as read with a single network request, use *praw.models.Inbox.mark_read()*

mark_unread()
Mark the item as unread.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox(limit=10)

for message in inbox:
    # process messages
```

See also *mark_read()*

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*)
Return an instance of Message or SubredditMessage from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

reply(*body*)
Reply to the object.

Parameters **body** – The Markdown formatted content for a comment.

Returns A *Comment* object for the newly created comment or `None` if Reddit doesn't provide one.

A `None` value can be returned if the target is a comment or submission in a quarantined subreddit and the authenticated user has not opt-ed in to viewing the content. When this happens the comment will be sucessfully created on Reddit and can be retried by drawing the comment from the user's comment history.

Note: Some items, such as locked submissions/comments or non-replyable messages will throw `prawcore.exceptions.Forbidden` when attempting to reply to them.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.reply('reply')

comment = reddit.comment(id='dxolpyc')
comment.reply('reply')
```

uncollapse()
Mark the item as uncollapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and uncollapse it
message = next(inbox)
message.uncollapse()
```

See also *collapse()*

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other

than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.5 ModmailConversation

class praw.models.**ModmailConversation**(*reddit: Reddit*, *id: Optional[str] = None*, *mark_read: bool = False*, *_data: Optional[Dict[str, Any]] = None*)

A class for modmail conversations.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`authors`	Provides an ordered list of *`Redditor`* instances. The authors of each message in the modmail conversation.
`id`	The ID of the ModmailConversation.
`is_highlighted`	Whether or not the ModmailConversation is highlighted.
`is_internal`	Whether or not the ModmailConversation is a private mod conversation.
`last_mod_update`	Time of the last mod message reply, represented in the ISO 8601 standard with timezone.
`last_updated`	Time of the last message reply, represented in the ISO 8601 standard with timezone.
`last_user_update`	Time of the last user message reply, represented in the ISO 8601 standard with timezone.
`num_messages`	The number of messages in the ModmailConversation.
`obj_ids`	Provides a list of dictionaries representing mod actions on the ModmailConversation. Each dict contains attributes of 'key' and 'id'. The key can be either 'messages' or 'ModAction'. ModAction represents archiving/highlighting etc.
`owner`	Provides an instance of *`Subreddit`*. The subreddit that the ModmailConversation belongs to.
`participant`	Provides an instance of *`Redditor`*. The participating user in the ModmailConversation.
`subject`	The subject of the ModmailConversation.

__init__(*reddit: Reddit*, *id: Optional[str] = None*, *mark_read: bool = False*, *_data: Optional[Dict[str, Any]] = None*)

Construct an instance of the ModmailConversation object.

Parameters **mark_read** – If True, conversation is marked as read (default: False).

archive()

Archive the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').archive()
```

highlight()

Highlight the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').highlight()
```

mute()

Mute the non-mod user associated with the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').mute()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*, *convert_objects: bool = True*)
Return an instance of ModmailConversation from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.
- **convert_objects** – If True, convert message and mod action data into objects (default: True).

read(*other_conversations: Optional[List[_ModmailConversation]] = None*)
Mark the conversation(s) as read.

Parameters **other_conversations** – A list of other conversations to mark (default: None).

For example, to mark the conversation as read along with other recent conversations from the same user:

```
subreddit = reddit.subreddit('redditdev')
conversation = subreddit.modmail.conversation('2gmz')
conversation.read(
                other_conversations=conversation.user.recent_convos)
```

reply(*body: str*, *author_hidden: bool = False*, *internal: bool = False*)
Reply to the conversation.

Parameters

- **body** – The Markdown formatted content for a message.
- **author_hidden** – When True, author is hidden from non-moderators (default: False).
- **internal** – When True, message is a private moderator note, hidden from non-moderators (default: False).

Returns A *ModmailMessage* object for the newly created message.

For example, to reply to the non-mod user while hiding your username:

```
conversation = reddit.subreddit('redditdev').modmail('2gmz')
conversation.reply('Message body', author_hidden=True)
```

To create a private moderator note on the conversation:

```
conversation.reply('Message body', internal=True)
```

unarchive()
Unarchive the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').unarchive()
```

unhighlight()
Un-highlight the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').unhighlight()
```

unmute()
Unmute the non-mod user associated with the conversation.

For example:

```
reddit.subreddit('redditdev').modmail('2gmz').unmute()
```

unread(*other_conversations: Optional[List[_ModmailConversation]] = None*)
Mark the conversation(s) as unread.

Parameters **other_conversations** – A list of other conversations to mark (default: None).

For example, to mark the conversation as unread along with other recent conversations from the same user:

```
subreddit = reddit.subreddit('redditdev')
conversation = subreddit.modmail.conversation('2gmz')
conversation.unread(other_conversations=conversation.user.recent_convos)
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.6 MoreComments

class praw.models.**MoreComments**(*reddit: Reddit, _data: Dict[str, Any]*)
A class indicating there are more comments.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)
Construct an instance of the MoreComments object.

comments(*update: bool = True*) → List[Comment]
Fetch and return the comments for a single MoreComments object.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.7 Multireddit

class praw.models.**Multireddit**(*reddit: Reddit, _data: Dict[str, Any]*)

A class for users' Multireddits.

This is referred to as a Custom Feed on the Reddit UI.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
`can_edit`	A `bool` representing whether or not the authenticated user may edit the multireddit.
`copied_from`	The multireddit that the multireddit was copied from, if it exists, otherwise `None`.
`created_utc`	When the multireddit was created, in Unix Time.
`description_html`	The description of the multireddit, as HTML.
`description_md`	The description of the multireddit, as Markdown.
`display_name`	The display name of the multireddit.
`name`	The name of the multireddit.
`over_18`	A `bool` representing whether or not the multireddit is restricted for users over 18.
`subreddits`	A `list` of *Subreddit*s that make up the multireddit.
`visibility`	The visibility of the multireddit, either `private`, `public`, or `hidden`.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)

Construct an instance of the Multireddit object.

add(*subreddit: praw.models.reddit.subreddit.Subreddit*)

Add a subreddit to this multireddit.

Parameters **subreddit** – The subreddit to add to this multi.

For example, to add subreddit `r/test` to multireddit `bboe/test`:

```
subreddit=reddit.subreddit('test')
reddit.multireddit('bboe', 'test').add(subreddit)
```

comments

Provide an instance of *CommentHelper*.

For example, to output the author of the 25 most recent comments of `/r/redditdev` execute:

```
for comment in reddit.subreddit('redditdev').comments(limit=25):
    print(comment.author)
```

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

copy(*display_name: Optional[str] = None*) → _Multireddit
Copy this multireddit and return the new multireddit.

Parameters **display_name** – (optional) The display name for the copied multireddit. Reddit will generate the `name` field from this display name. When not provided the copy will use the same display name and name as this multireddit.

To copy the multireddit `bboe/test` with a name of `testing`: .. code-block:: python

reddit.multireddit('bboe', 'test').copy('testing')

delete()
Delete this multireddit.

For example, to delete multireddit``bboe/test``:

```
reddit.multireddit('bboe', 'test').delete()
```

gilded(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for gilded items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get gilded items in subreddit `r/test`:

```
for item in reddit.subreddit('test').gilded():
    print(item.id)
```

hot(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

new(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
```

(continues on next page)

(continued from previous page)

```
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

random_rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for random rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get random rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').random_rising():
    print(submission.title)
```

remove(*subreddit: praw.models.reddit.subreddit.Subreddit*)
Remove a subreddit from this multireddit.

Parameters **subreddit** – The subreddit to remove from this multi.

For example, to remove subreddit `r/test` from multireddit `bboe/test`:

```
subreddit=reddit.subreddit('test')
reddit.multireddit('bboe', 'test').remove(subreddit)
```

rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').rising():
    print(submission.title)
```

static sluggify(*title: str*)
Return a slug version of the title.

Parameters **title** – The title to make a slug of.

Adapted from reddit's utils.py.

stream
Provide an instance of *SubredditStream*.

Streams can be used to indefinitely retrieve new comments made to a multireddit, like:

```
for comment in reddit.multireddit('spez', 'fun').stream.comments():
    print(comment)
```

Additionally, new submissions can be retrieved via the stream. In the following example all new submissions to the multireddit are fetched:

```
for submission in reddit.multireddit('bboe',
                                     'games').stream.submissions():
    print(submission)
```

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise ValueError if time_filter is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

update(***updated_settings*)
Update this multireddit.

Keyword arguments are passed for settings that should be updated. They can any of:

Parameters

- **display_name** – The display name for this multireddit. Must be no longer than 50 characters.
- **subreddits** – Subreddits for this multireddit.
- **description_md** – Description for this multireddit, formatted in Markdown.
- **icon_name** – Can be one of: art and design, ask, books, business, cars, comics, cute animals, diy, entertainment, food and drink, funny, games, grooming, health, life advice, military, models pinup, music, news, philosophy, pictures and gifs, science, shopping, sports, style, tech, travel, unusual stories, video, or None.
- **key_color** – RGB hex color code of the form '#FFFFFF'.
- **visibility** – Can be one of: hidden, private, public.
- **weighting_scheme** – Can be one of: classic, fresh.

For example, to rename multireddit bboe/test to bboe/testing:

```
reddit.multireddit('bboe', 'test').update(display_name='testing')
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.8 Redditor

class praw.models.**Redditor**(*reddit: Reddit*, *name: Optional[str] = None*, *fullname: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

A class representing the users of reddit.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Note: Shadowbanned accounts are treated the same as non-existent accounts, meaning that they will not have any attributes.

Note: Suspended/banned accounts will only return the `name` and `is_suspended` attributes.

Attribute	Description
`comment_karma`	The comment karma for the Redditor.
`comments`	Provide an instance of *`SubListing`* for comment access.
`created_utc`	Time the account was created, represented in Unix Time.
`has_verified_email`	Whether or not the Redditor has verified their email.
`icon_img`	The url of the Redditors' avatar.
`id`	The ID of the Redditor.
`is_employee`	Whether or not the Redditor is a Reddit employee.
`is_friend`	Whether or not the Redditor is friends with the authenticated user.
`is_mod`	Whether or not the Redditor mods any subreddits.
`is_gold`	Whether or not the Redditor has active Reddit Premium status.
`is_suspended`	Whether or not the Redditor is currently suspended.
`link_karma`	The link karma for the Redditor.
`name`	The Redditor's username.
`subreddit`	If the Redditor has created a user-subreddit, provides a dictionary of additional attributes. See below.
`subreddit['banner_img']`	The URL of the user-subreddit banner.
`subreddit['name']`	The fullname of the user-subreddit.
`subreddit['over_18']`	Whether or not the user-subreddit is NSFW.
`subreddit['public_description']`	The public description of the user- subreddit.
`subreddit['subscribers']`	The number of users subscribed to the user-subreddit.
`subreddit['title']`	The title of the user-subreddit.

__init__(*reddit: Reddit*, *name: Optional[str] = None*, *fullname: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Initialize a Redditor instance.

Parameters

- **reddit** – An instance of *`Reddit`*.
- **name** – The name of the redditor.
- **fullname** – The fullname of the redditor, starting with `t2_`.

Exactly one of `name`, `fullname` or `_data` must be provided.

block()
Block the Redditor.

For example, to block Redditor `spez`:

```
reddit.redditor("spez").block()
```

comments
Provide an instance of *SubListing* for comment access.

For example, to output the first line of all new comments by `/u/spez` try:

```
for comment in reddit.redditor('spez').comments.new(limit=None):
    print(comment.body.split('\n', 1)[0][:79])
```

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

downvoted(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for items the user has downvoted.

May raise `prawcore.Forbidden` after issuing the request if the user is not authorized to access the list. Note that because this function returns a *ListingGenerator* the exception may not occur until sometime after this function has returned.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get all downvoted items of the authenticated user:

```
for item in reddit.user.me().downvoted():
    print(item.id)
```

friend(*note: str = None*)
Friend the Redditor.

Parameters **note** – A note to save along with the relationship. Requires Reddit Premium (default: None).

Calling this method subsequent times will update the note.

For example, to friend Redditor `spez`:

```
reddit.redditor("spez").friend()
```

To add a note to the friendship (requires Reddit Premium):

```
reddit.redditor("spez").friend(note="My favorite admin")
```

friend_info() → _Redditor
Return a Redditor instance with specific friend-related attributes.

Returns A *Redditor* instance with fields `date`, `id`, and possibly `note` if the authenticated user has Reddit Premium.

For example, to get the friendship information of Redditor `spez`:

```
info = reddit.redditor("spez").friend_info
friend_data = info.date
```

classmethod from_data(*reddit*, *data*)
Return an instance of Redditor, or None from `data`.

fullname
Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

gild(*months: int = 1*)
Gild the Redditor.

Parameters **months** – Specifies the number of months to gild up to 36 (default: 1).

For example, to gild Redditor `spez` for 1 month:

```
reddit.redditor("spez").gild(months=1)
```

gilded(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for gilded items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get gilded items in subreddit `r/test`:

```
for item in reddit.subreddit('test').gilded():
    print(item.id)
```

gildings(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for items the user has gilded.

May raise `prawcore.Forbidden` after issuing the request if the user is not authorized to access the list. Note that because this function returns a *ListingGenerator* the exception may not occur until sometime after this function has returned.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get all gilded items of the authenticated user:

```
for item in reddit.user.me().gildings():
    print(item.id)
```

hidden(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for items the user has hidden.

May raise `prawcore.Forbidden` after issuing the request if the user is not authorized to access the list. Note that because this function returns a *ListingGenerator* the exception may not occur until sometime after this function has returned.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get all hidden items of the authenticated user:

```
for item in reddit.user.me().hidden():
    print(item.id)
```

hot(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

message(*subject*, *message*, *from_subreddit=None*)
Send a message to a redditor or a subreddit's moderators (mod mail).

Parameters

- **subject** – The subject of the message.
- **message** – The message content.
- **from_subreddit** – A *Subreddit* instance or string to send the message from. When provided, messages are sent from the subreddit rather than from the authenticated user. Note that the authenticated user must be a moderator of the subreddit and have the `mail` moderator permission.

For example, to send a private message to `u/spez`, try:

```
reddit.redditor('spez').message('TEST', 'test message from PRAW')
```

To send a message to `u/spez` from the moderators of `r/test` try:

```
reddit.redditor('spez').message('TEST', 'test message from r/test',
                                from_subreddit='test')
```

To send a message to the moderators of `r/test`, try:

```
reddit.subreddit('test').message('TEST', 'test PM from PRAW')
```

moderated() → List[Subreddit]
Return a list of the redditor's moderated subreddits.

Returns A `list` of *Subreddit* objects. Return `[]` if the redditor has no moderated subreddits.

Note: The redditor's own user profile subreddit will not be returned, but other user profile subreddits they moderate will be returned.

Usage:

```
for subreddit in reddit.redditor('spez').moderated():
    print(subreddit.display_name)
    print(subreddit.title)
```

multireddits() → List[Multireddit]
Return a list of the redditor's public multireddits.

For example, to to get Redditor `spez`'s multireddits:

```
multireddits = reddit.redditor("spez").multireddits()
```

new(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

saved(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for items the user has saved.

May raise `prawcore.Forbidden` after issuing the request if the user is not authorized to access the list. Note that because this function returns a *ListingGenerator* the exception may not occur until sometime after this function has returned.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get all saved items of the authenticated user:

```
for item in reddit.user.me().saved():
    print(item.id)
```

stream
Provide an instance of *RedditorStream*.

Streams can be used to indefinitely retrieve new comments made by a redditor, like:

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

Additionally, new submissions can be retrieved via the stream. In the following example all submissions are fetched via the redditor `spez`:

```
for submission in reddit.redditor('spez').stream.submissions():
    print(submission)
```

submissions
Provide an instance of *SubListing* for submission access.

For example, to output the title's of top 100 of all time submissions for `/u/spez` try:

```
for submission in reddit.redditor('spez').submissions.top('all'):
    print(submission.title)
```

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

trophies() → List[Trophy]
Return a list of the redditor's trophies.

Returns A `list` of *Trophy* objects. Return `[]` if the redditor has no trophy.

Raise `prawcore.exceptions.BadRequest` if the redditor doesn't exist.

Usage:

```
for trophy in reddit.redditor('spez').trophies():
    print(trophy.name)
    print(trophy.description)
```

unblock()
Unblock the Redditor.

For example, to unblock Redditor `spez`:

```
reddit.redditor("spez").unblock()
```

unfriend()
Unfriend the Redditor.

For example, to unfriend Redditor `spez`:

```
reddit.redditor("spez").unfriend()
```

upvoted(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for items the user has upvoted.

May raise `prawcore.Forbidden` after issuing the request if the user is not authorized to access the list. Note that because this function returns a *ListingGenerator* the exception may not occur until sometime after this function has returned.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get all upvoted items of the authenticated user:

```
for item in reddit.user.me().upvoted():
    print(item.id)
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.9 Submission

class praw.models.**Submission**(*reddit: Reddit*, *id: Optional[str] = None*, *url: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

A class for submissions to reddit.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
author	Provides an instance of *Redditor*.
clicked	Whether or not the submission has been clicked by the client.
comments	Provides an instance of *CommentForest*.
created_utc	Time the submission was created, represented in Unix Time.
distinguished	Whether or not the submission is distinguished.
edited	Whether or not the submission has been edited.
id	ID of the submission.
is_original_content	Whether or not the submission has been set as original content.
is_self	Whether or not the submission is a selfpost (text-only).
link_flair_template_id	The link flair's ID, or None if not flaired.
link_flair_text	The link flair's text content, or None if not flaired.
locked	Whether or not the submission has been locked.
name	Fullname of the submission.
num_comments	The number of comments on the submission.
over_18	Whether or not the submission has been marked as NSFW.
permalink	A permalink for the submission.
score	The number of upvotes for the submission.
selftext	The submissions' selftext - an empty string if a link post.
spoiler	Whether or not the submission has been marked as a spoiler.
stickied	Whether or not the submission is stickied.
subreddit	Provides an instance of *Subreddit*.
title	The title of the submission.
upvote_ratio	The percentage of upvotes from all votes on the submission.
url	The URL the submission links to, or the permalink if a selfpost.

__init__(*reddit: Reddit*, *id: Optional[str] = None*, *url: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Initialize a Submission instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **id** – A reddit base36 submission ID, e.g., `2gmzqe`.
- **url** – A URL supported by *id_from_url()*.

Either `id` or `url` can be provided, but not both.

clear_vote()
Clear the authenticated user's vote on the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK, but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.clear_vote()

comment = reddit.comment(id='dxolpyc')
comment.clear_vote()
```

comments
Provide an instance of *CommentForest*.

This attribute can used, for example, to obtain a flat list of comments, with any *MoreComments* removed:

```
submission.comments.replace_more(limit=0)
comments = submission.comments.list()
```

Sort order and comment limit can be set with the `comment_sort` and `comment_limit` attributes before comments are fetched, including any call to *replace_more()*:

```
submission.comment_sort = 'new'
comments = submission.comments.list()
```

Note: The appropriate values for `comment_sort` include `best`, `top`, `new`, `controversial`, `old` and `q&a`.

See *Extracting comments with PRAW* for more on working with a *CommentForest*.

crosspost(*subreddit: praw.models.reddit.subreddit.Subreddit*, *title: Optional[str] = None*, *send_replies: bool = True*, *flair_id: Optional[str] = None*, *flair_text: Optional[str] = None*, *nsfw: bool = False*, *spoiler: bool = False*) → _Submission
Crosspost the submission to a subreddit.

Note: Be aware you have to be subscribed to the target subreddit.

Parameters

- **subreddit** – Name of the subreddit or *Subreddit* object to crosspost into.
- **title** – Title of the submission. Will use this submission's title if *None* (default: None).

- **flair_id** – The flair template to select (default: None).
- **flair_text** – If the template's `flair_text_editable` value is True, this value will set a custom text (default: None).
- **send_replies** – When True, messages will be sent to the submission author when comments are made to the submission (default: True).
- **nsfw** – Whether or not the submission should be marked NSFW (default: False).
- **spoiler** – Whether or not the submission should be marked as a spoiler (default: False).

Returns A *Submission* object for the newly created submission.

Example usage:

```
submission = reddit.submission(id='5or86n')
cross_post = submission.crosspost(subreddit="learnprogramming",
                                  send_replies=False)
```

See also *hide()*

delete()
Delete the object.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.delete()

submission = reddit.submission('8dmv8z')
submission.delete()
```

disable_inbox_replies()
Disable inbox replies for the item.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.disable_inbox_replies()

submission = reddit.submission('8dmv8z')
submission.disable_inbox_replies()
```

See also *enable_inbox_replies()*

downvote()
Downvote the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK, but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.downvote()
```

(continues on next page)

(continued from previous page)

```
comment = reddit.comment(id='dxolpyc')
comment.downvote()
```

See also *upvote()*

duplicates(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for the submission's duplicates.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

Example usage:

```
submission = reddit.submission(id='5or86n')

for duplicate in submission.duplicates():
    # process each duplicate
```

See also *upvote()*

edit(*body*)
Replace the body of the object with `body`.

Parameters **body** – The Markdown formatted content for the updated object.

Returns The current instance after updating its attributes.

Example usage:

```
comment = reddit.comment('dkk4qjd')

# construct the text of an edited comment
# by appending to the old body:
edited_body = comment.body + "Edit: thanks for the gold!"
comment.edit(edited_body)
```

enable_inbox_replies()
Enable inbox replies for the item.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.enable_inbox_replies()

submission = reddit.submission('8dmv8z')
submission.enable_inbox_replies()
```

See also *disable_inbox_replies()*

flair
Provide an instance of *SubmissionFlair*.

This attribute is used to work with flair as a regular user of the subreddit the submission belongs to. Moderators can directly use *flair()*.

For example, to select an arbitrary editable flair text (assuming there is one) and set a custom value try:

```
choices = submission.flair.choices()
template_id = next(x for x in choices
                   if x['flair_text_editable'])['flair_template_id']
submission.flair.select(template_id, 'my custom value')
```

fullname
Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

gild()
Gild the author of the item.

Note: Requires the authenticated user to own Reddit Coins. Calling this method will consume Reddit Coins.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.gild()

submission = reddit.submission('8dmv8z')
submission.gild()
```

hide(*other_submissions: Optional[List[_Submission]] = None*)
Hide Submission.

Parameters **other_submissions** – When provided, additionally hide this list of *Submission* instances as part of a single request (default: None).

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.hide()
```

See also *unhide()*

static id_from_url(*url: str*) → str
Return the ID contained within a submission URL.

Parameters **url** – A url to a submission in one of the following formats (http urls will also work): * https://redd.it/2gmzqe * https://reddit.com/comments/2gmzqe/ * https://www.reddit.com/r/redditdev/comments/2gmzqe/praw_https/

Raise *InvalidURL* if URL is not a valid submission URL.

mark_visited()
Mark submission as visited.

This method requires a subscription to reddit premium.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.mark_visited()
```

mod
Provide an instance of *SubmissionModeration*.

Example usage:

```
submission = reddit.submission(id="8dmv8z")
submission.mod.approve()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

reply(*body*)
Reply to the object.

Parameters **body** – The Markdown formatted content for a comment.

Returns A *Comment* object for the newly created comment or `None` if Reddit doesn't provide one.

A `None` value can be returned if the target is a comment or submission in a quarantined subreddit and the authenticated user has not opt-ed in to viewing the content. When this happens the comment will be sucessfully created on Reddit and can be retried by drawing the comment from the user's comment history.

Note: Some items, such as locked submissions/comments or non-replyable messages will throw `prawcore.exceptions.Forbidden` when attempting to reply to them.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.reply('reply')

comment = reddit.comment(id='dxolpyc')
comment.reply('reply')
```

report(*reason*)
Report this object to the moderators of its subreddit.

Parameters **reason** – The reason for reporting.

Raises *APIException* if `reason` is longer than 100 characters.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.report('report reason')

comment = reddit.comment(id='dxolpyc')
comment.report('report reason')
```

save(*category=None*)
Save the object.

Parameters **category** – (Premium) The category to save to. If your user does not have Reddit Premium this value is ignored by Reddit (default: `None`).

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.save(category="view later")

comment = reddit.comment(id='dxolpyc')
comment.save()
```

See also *unsave()*

shortlink
Return a shortlink to the submission.

For example http://redd.it/eorhm is a shortlink for https://www.reddit.com/r/announcements/comments/eorhm/reddit_30_less_typing/.

unhide(*other_submissions: Optional[List[_Submission]] = None*)
Unhide Submission.

Parameters **other_submissions** – When provided, additionally unhide this list of *Submission* instances as part of a single request (default: None).

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.unhide()
```

See also *hide()*

unsave()
Unsave the object.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.unsave()

comment = reddit.comment(id='dxolpyc')
comment.unsave()
```

See also *save()*

upvote()
Upvote the object.

Note: Votes must be cast by humans. That is, API clients proxying a human's action one-for-one are OK, but bots deciding how to vote on content or amplifying a human's vote are not. See the reddit rules for more details on what constitutes vote cheating. [Ref]

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.upvote()

comment = reddit.comment(id='dxolpyc')
comment.upvote()
```

See also *downvote()*

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.10 Subreddit

class praw.models.**Subreddit**(*reddit*, *display_name=None*, *_data=None*)

A class for Subreddits.

To obtain an instance of this class for subreddit `r/redditdev` execute:

```
subreddit = reddit.subreddit('redditdev')
```

While `r/all` is not a real subreddit, it can still be treated like one. The following outputs the titles of the 25 hottest submissions in `r/all`:

```
for submission in reddit.subreddit('all').hot(limit=25):
    print(submission.title)
```

Multiple subreddits can be combined with a + like so:

```
for submission in reddit.subreddit('redditdev+learnpython').top('all'):
    print(submission)
```

Subreddits can be filtered from combined listings as follows. Note that these filters are ignored by certain methods, including *comments*, *gilded()*, and *SubredditStream.comments()*.

```
for submission in reddit.subreddit('all-redditdev').new():
    print(submission)
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`can_assign_link_flair`	Whether users can assign their own link flair.
`can_assign_user_flair`	Whether users can assign their own user flair.
`created_utc`	Time the subreddit was created, represented in Unix Time.
`description`	Subreddit description, in Markdown.
`description_html`	Subreddit description, in HTML.
`display_name`	Name of the subreddit.
`id`	ID of the subreddit.
`name`	Fullname of the subreddit.
`over18`	Whether the subreddit is NSFW.
`public_description`	Description of the subreddit, shown in searches and on the "You must be invited to visit this community" page (if applicable).
`spoilers_enabled`	Whether the spoiler tag feature is enabled.
`subscribers`	Count of subscribers.
`user_is_banned`	Whether the authenticated user is banned.
`user_is_moderator`	Whether the authenticated user is a moderator.
`user_is_subscriber`	Whether the authenticated user is subscribed.

__init__(*reddit*, *display_name=None*, *_data=None*)

Initialize a Subreddit instance.

Parameters

- **reddit** – An instance of *Reddit*.

- **display_name** – The name of the subreddit.

Note: This class should not be initialized directly. Instead obtain an instance via: `reddit.subreddit('subreddit_name')`

---

banned

Provide an instance of *SubredditRelationship*.

For example to ban a user try:

```
reddit.subreddit('SUBREDDIT').banned.add('NAME', ban_reason='...')
```

To list the banned users along with any notes, try:

```
for ban in reddit.subreddit('SUBREDDIT').banned():
    print('{}: {}'.format(ban, ban.note))
```

collections

Provide an instance of *SubredditCollections*.

To see the permalinks of all *Collection*s that belong to a subreddit, try:

```
for collection in reddit.subreddit('SUBREDDIT').collections:
    print(collection.permalink)
```

To get a specific *Collection* by its UUID or permalink, use one of the following:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection = reddit.subreddit('SUBREDDIT').collections(
    permalink='https://reddit.com/r/SUBREDDIT/collection/some_uuid')
```

comments

Provide an instance of *CommentHelper*.

For example, to output the author of the 25 most recent comments of `/r/redditdev` execute:

```
for comment in reddit.subreddit('redditdev').comments(limit=25):
    print(comment.author)
```

contributor

Provide an instance of *ContributorRelationship*.

Contributors are also known as approved submitters.

To add a contributor try:

```
reddit.subreddit('SUBREDDIT').contributor.add('NAME')
```

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

emoji
Provide an instance of *SubredditEmoji*.

This attribute can be used to discover all emoji for a subreddit:

```
for emoji in reddit.subreddit('iama').emoji:
    print(emoji)
```

A single emoji can be lazily retrieved via:

```
reddit.subreddit('blah').emoji['emoji_name']
```

Note: Attempting to access attributes of an nonexistent emoji will result in a *ClientException*.

filters
Provide an instance of *SubredditFilters*.

For example, to add a filter, run:

```
reddit.subreddit('all').filters.add('subreddit_name')
```

flair
Provide an instance of *SubredditFlair*.

Use this attribute for interacting with a subreddit's flair. For example to list all the flair for a subreddit which you have the `flair` moderator permission on try:

```
for flair in reddit.subreddit('NAME').flair():
    print(flair)
```

Flair templates can be interacted with through this attribute via:

```
for template in reddit.subreddit('NAME').flair.templates:
    print(template)
```

fullname
Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

gilded(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for gilded items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get gilded items in subreddit `r/test`:

```
for item in reddit.subreddit('test').gilded():
    print(item.id)
```

hot (***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

message (*subject*, *message*, *from_subreddit=None*)
Send a message to a redditor or a subreddit's moderators (mod mail).

Parameters

- **subject** – The subject of the message.
- **message** – The message content.
- **from_subreddit** – A *Subreddit* instance or string to send the message from. When provided, messages are sent from the subreddit rather than from the authenticated user. Note that the authenticated user must be a moderator of the subreddit and have the `mail` moderator permission.

For example, to send a private message to `u/spez`, try:

```
reddit.redditor('spez').message('TEST', 'test message from PRAW')
```

To send a message to `u/spez` from the moderators of `r/test` try:

```
reddit.redditor('spez').message('TEST', 'test message from r/test',
                                from_subreddit='test')
```

To send a message to the moderators of `r/test`, try:

```
reddit.subreddit('test').message('TEST', 'test PM from PRAW')
```

mod
Provide an instance of *SubredditModeration*.

For example, to accept a moderation invite from subreddit `r/test`:

```
reddit.subreddit('test').mod.accept_invite()
```

moderator
Provide an instance of *ModeratorRelationship*.

For example to add a moderator try:

```
reddit.subreddit('SUBREDDIT').moderator.add('NAME')
```

To list the moderators along with their permissions try:

```
for moderator in reddit.subreddit('SUBREDDIT').moderator():
    print('{}: {}'.format(moderator, moderator.mod_permissions))
```

modmail
Provide an instance of *Modmail*.

For example, to send a new modmail from the subreddit `r/test` to user `u/spez` with the subject `test` along with a message body of `hello`:

```
reddit.subreddit('test').modmail.create('test', 'hello', 'spez')
```

muted
Provide an instance of *SubredditRelationship*.

For example, muted users can be iterated through like so:

```
for mute in reddit.subreddit('redditdev').muted():
    print('{}: {}'.format(mute, mute.note))
```

new(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

quaran
Provide an instance of *SubredditQuarantine*.

This property is named `quaran` because `quarantine` is a Subreddit attribute returned by Reddit to indicate whether or not a Subreddit is quarantined.

To opt-in into a quarantined subreddit:

```
reddit.subreddit('test').quaran.opt_in()
```

random()
Return a random Submission.

Returns `None` on subreddits that do not support the random feature. One example, at the time of writing, is `r/wallpapers`.

For example, to get a random submission off of `r/AskReddit`:

```
submission = reddit.subreddit("AskReddit").random()
print(submission.title)
```

random_rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for random rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get random rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').random_rising():
    print(submission.title)
```

rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').rising():
    print(submission.title)
```

rules()
Return rules for the subreddit.

For example to show the rules of `r/redditdev` try:

```
reddit.subreddit('redditdev').rules()
```

search(*query*, *sort='relevance'*, *syntax='lucene'*, *time_filter='all'*, ***generator_kwargs*)
Return a *ListingGenerator* for items that match `query`.

Parameters

- **query** – The query string to search for.
- **sort** – Can be one of: relevance, hot, top, new, comments. (default: relevance).
- **syntax** – Can be one of: cloudsearch, lucene, plain (default: lucene).
- **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

For more information on building a search query see: https://www.reddit.com/wiki/search

For example to search all subreddits for `praw` try:

```
for submission in reddit.subreddit('all').search('praw'):
    print(submission.title)
```

sticky(*number=1*)
Return a Submission object for a sticky of the subreddit.

Parameters **number** – Specify which sticky to return. 1 appears at the top (default: 1).

Raises `prawcore.NotFound` if the sticky does not exist.

For example, to get the stickied post on the subreddit `r/test`:

```
reddit.subreddit("test").sticky()
```

stream
Provide an instance of *SubredditStream*.

Streams can be used to indefinitely retrieve new comments made to a subreddit, like:

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

Additionally, new submissions can be retrieved via the stream. In the following example all submissions are fetched via the special subreddit `r/all`:

```
for submission in reddit.subreddit('all').stream.submissions():
    print(submission)
```

stylesheet

Provide an instance of *SubredditStylesheet*.

For example, to add the css data `.test{color:blue}` to the existing stylesheet:

```
subreddit = reddit.subreddit('SUBREDDIT')
stylesheet = subreddit.stylesheet()
stylesheet += ".test{color:blue}"
subreddit.stylesheet.update(stylesheet)
```

submit(*title*, *selftext=None*, *url=None*, *flair_id=None*, *flair_text=None*, *resubmit=True*, *send_replies=True*, *nsfw=False*, *spoiler=False*, *collection_id=None*)

Add a submission to the subreddit.

Parameters

- **title** – The title of the submission.
- **selftext** – The Markdown formatted content for a `text` submission. Use an empty string, `''`, to make a title-only submission.
- **url** – The URL for a `link` submission.
- **collection_id** – The UUID of a *Collection* to add the newly-submitted post to.
- **flair_id** – The flair template to select (default: None).
- **flair_text** – If the template's `flair_text_editable` value is True, this value will set a custom text (default: None).
- **resubmit** – When False, an error will occur if the URL has already been submitted (default: True).
- **send_replies** – When True, messages will be sent to the submission author when comments are made to the submission (default: True).
- **nsfw** – Whether or not the submission should be marked NSFW (default: False).
- **spoiler** – Whether or not the submission should be marked as a spoiler (default: False).

Returns A *Submission* object for the newly created submission.

Either `selftext` or `url` can be provided, but not both.

For example to submit a URL to `r/reddit_api_test` do:

```
title = 'PRAW documentation'
url = 'https://praw.readthedocs.io'
reddit.subreddit('reddit_api_test').submit(title, url=url)
```

Note: For submitting images, videos, and videogifs, see *submit_image()* and *submit_video()*.

---

submit_image(*title*, *image_path*, *flair_id=None*, *flair_text=None*, *resubmit=True*, *send_replies=True*, *nsfw=False*, *spoiler=False*, *timeout=10*, *collection_id=None*, *without_websockets=False*)

Add an image submission to the subreddit.

Parameters

- **title** – The title of the submission.
- **image_path** – The path to an image, to upload and post.
- **collection_id** – The UUID of a *Collection* to add the newly-submitted post to.
- **flair_id** – The flair template to select (default: None).
- **flair_text** – If the template's `flair_text_editable` value is True, this value will set a custom text (default: None).
- **resubmit** – When False, an error will occur if the URL has already been submitted (default: True).
- **send_replies** – When True, messages will be sent to the submission author when comments are made to the submission (default: True).
- **nsfw** – Whether or not the submission should be marked NSFW (default: False).
- **spoiler** – Whether or not the submission should be marked as a spoiler (default: False).
- **timeout** – Specifies a particular timeout, in seconds. Use to avoid "Websocket error" exceptions (default: 10).
- **without_websockets** – Set to `True` to disable use of WebSockets (see note below for an explanation). If `True`, this method doesn't return anything. (default: `False`).

Returns A *Submission* object for the newly created submission, unless `without_websockets` is `True`.

If `image_path` refers to a file that is not an image, PRAW will raise a *ClientException*.

Note: Reddit's API uses WebSockets to respond with the link of the newly created post. If this fails, the method will raise *WebSocketException*. Occasionally, the Reddit post will still be created. More often, there is an error with the image file. If you frequently get exceptions but successfully created posts, try setting the `timeout` parameter to a value above 10.

To disable the use of WebSockets, set `without_websockets=True`. This will make the method return `None`, though the post will still be created. You may wish to do this if you are running your program in a restricted network environment, or using a proxy that doesn't support WebSockets connections.

For example to submit an image to `r/reddit_api_test` do:

```
title = 'My favorite picture'
image = '/path/to/image.png'
reddit.subreddit('reddit_api_test').submit_image(title, image)
```

submit_video(*title*, *video_path*, *videogif=False*, *thumbnail_path=None*, *flair_id=None*, *flair_text=None*, *resubmit=True*, *send_replies=True*, *nsfw=False*, *spoiler=False*, *timeout=10*, *collection_id=None*, *without_websockets=False*)

Add a video or videogif submission to the subreddit.

Parameters

- **title** – The title of the submission.
- **video_path** – The path to a video, to upload and post.
- **videogif** – A `bool` value. If `True`, the video is uploaded as a videogif, which is essentially a silent video (default: `False`).

- **thumbnail_path** – (Optional) The path to an image, to be uploaded and used as the thumbnail for this video. If not provided, the PRAW logo will be used as the thumbnail.
- **collection_id** – The UUID of a *Collection* to add the newly-submitted post to.
- **flair_id** – The flair template to select (default: `None`).
- **flair_text** – If the template's `flair_text_editable` value is True, this value will set a custom text (default: `None`).
- **resubmit** – When False, an error will occur if the URL has already been submitted (default: `True`).
- **send_replies** – When True, messages will be sent to the submission author when comments are made to the submission (default: `True`).
- **nsfw** – Whether or not the submission should be marked NSFW (default: False).
- **spoiler** – Whether or not the submission should be marked as a spoiler (default: False).
- **timeout** – Specifies a particular timeout, in seconds. Use to avoid "Websocket error" exceptions (default: 10).
- **without_websockets** – Set to `True` to disable use of WebSockets (see note below for an explanation). If `True`, this method doesn't return anything. (default: `False`).

Returns A *Submission* object for the newly created submission, unless `without_websockets` is `True`.

If `video_path` refers to a file that is not a video, PRAW will raise a *ClientException*.

Note: Reddit's API uses WebSockets to respond with the link of the newly created post. If this fails, the method will raise *WebSocketException*. Occasionally, the Reddit post will still be created. More often, there is an error with the image file. If you frequently get exceptions but successfully created posts, try setting the `timeout` parameter to a value above 10.

To disable the use of WebSockets, set `without_websockets=True`. This will make the method return `None`, though the post will still be created. You may wish to do this if you are running your program in a restricted network environment, or using a proxy that doesn't support WebSockets connections.

For example to submit a video to `r/reddit_api_test` do:

```
title = 'My favorite movie'
video = '/path/to/video.mp4'
reddit.subreddit('reddit_api_test').submit_video(title, video)
```

subscribe(*other_subreddits=None*)

Subscribe to the subreddit.

Parameters **other_subreddits** – When provided, also subscribe to the provided list of subreddits.

For example, to subscribe to `r/test`:

```
reddit.subreddit("test").subscribe()
```

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *`ListingGenerator`*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

`traffic`()

Return a dictionary of the subreddit's traffic statistics.

Raises `prawcore.NotFound` when the traffic stats aren't available to the authenticated user, that is, they are not public and the authenticated user is not a moderator of the subreddit.

The traffic method returns a dict with three keys. The keys are `day`, `hour` and `month`. Each key contains a list of lists with 3 or 4 values. The first value is a timestamp indicating the start of the category (start of the day for the `day` key, start of the hour for the `hour` key, etc.). The second, third, and fourth values indicate the unique pageviews, total pageviews, and subscribers, respectively.

Note: The `hour` key does not contain subscribers, and therefore each sub-list contains three values.

For example, to get the traffic stats for `r/test`:

```
stats=reddit.subreddit("test").traffic()
```

`unsubscribe`(*other_subreddits=None*)

Unsubscribe from the subreddit.

Parameters **`other_subreddits`** – When provided, also unsubscribe from the provided list of subreddits.

To unsubscribe from `r/test`:

```
reddit.subreddit('test').unsubscribe()
```

`widgets`

Provide an instance of *`SubredditWidgets`*.

Example usage

Get all sidebar widgets:

```
for widget in reddit.subreddit('redditdev').widgets.sidebar:
    print(widget)
```

Get ID card widget:

```
print(reddit.subreddit('redditdev').widgets.id_card)
```

`wiki`

Provide an instance of *`SubredditWiki`*.

This attribute can be used to discover all wikipages for a subreddit:

```
for wikipage in reddit.subreddit('iama').wiki:
    print(wikipage)
```

To fetch the content for a given wikipage try:

```
wikipage = reddit.subreddit('iama').wiki['proof']
print(wikipage.content_md)
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.8.11 WikiPage

class praw.models.**WikiPage**(*reddit: Reddit*, *subreddit: Subreddit*, *name: str*, *revision: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

An individual WikiPage object.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
`content_html`	The contents of the wiki page, as HTML.
`content_md`	The contents of the wiki page, as Markdown.
`may_revise`	A `bool` representing whether or not the authenticated user may edit the wiki page.
`name`	The name of the wiki page.
`revision_by`	The *`Redditor`* who authored this revision of the wiki page.
`revision_date`	The time of this revision, in Unix Time.
`subreddit`	The *`Subreddit`* this wiki page belongs to.

__init__(*reddit: Reddit*, *subreddit: Subreddit*, *name: str*, *revision: Optional[str] = None*, *_data: Optional[Dict[str, Any]] = None*)

Construct an instance of the WikiPage object.

Parameters **revision** – A specific revision ID to fetch. By default, fetches the most recent revision.

edit(*content: str*, *reason: Optional[str] = None*, ***other_settings*)

Edit this WikiPage's contents.

Parameters

- **content** – The updated Markdown content of the page.
- **reason** – (Optional) The reason for the revision.
- **other_settings** – Additional keyword arguments to pass.

For example, to replace the first wiki page of `r/test` with the phrase `test wiki page`:

```
page = next(iter(reddit.subreddit('test').wiki))
page.edit(content='test wiki page')
```

mod
Provide an instance of *WikiPageModeration*.

For example, to add `spez` as an editor on the wikipage `praw_test` try:

```
reddit.subreddit('test').wiki['praw_test'].mod.add('spez')
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

revision(*revision: str*)
Return a specific version of this page by revision ID.

To view revision `[ID]` of `'praw_test'` in `'/r/test'`:

```
page = reddit.subreddit('test').wiki['praw_test'].revision('[ID]')
```

revisions(***generator_kwargs*) → Generator[[_WikiPage, None], None]
Return a *ListingGenerator* for page revisions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To view the wiki revisions for `'praw_test'` in `'/r/test'` try:

```
for item in reddit.subreddit('test').wiki['praw_test'].revisions():
    print(item)
```

To get *WikiPage* objects for each revision:

```
for item in reddit.subreddit('test').wiki['praw_test'].revisions():
    print(item['page'])
```

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.9 Exceptions in PRAW

In addition to exceptions under the `praw.exceptions` namespace shown below, exceptions might be raised that inherit from `prawcore.PrawcoreException`. Please see the following resource for information on those exceptions: https://github.com/praw-dev/prawcore/blob/master/prawcore/exceptions.py

1.9.1 praw.exceptions

PRAW exception classes.

Includes two main exceptions: *APIException* for when something goes wrong on the server side, and *ClientException* when something goes wrong on the client side. Both of these classes extend *PRAWException*.

All other exceptions are subclassed from *ClientException*.

exception praw.exceptions.**APIException**(*error_type: str, message: str, field: Optional[str]*)
Indicate exception that involve responses from Reddit's API.

__init__(*error_type: str, message: str, field: Optional[str]*)
Initialize an instance of APIException.

Parameters

- **error_type** – The error type set on Reddit's end.
- **message** – The associated message for the error.
- **field** – The input field associated with the error if available.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**ClientException**
Indicate exceptions that don't involve interaction with Reddit's API.

__init__
Initialize self. See help(type(self)) for accurate signature.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**DuplicateReplaceException**
Indicate exceptions that involve the replacement of MoreComments.

__init__()
Initialize the class.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**InvalidFlairTemplateID**(*template_id: str*)
Indicate exceptions where an invalid flair template id is given.

__init__(*template_id: str*)
Initialize the class.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**InvalidImplicitAuth**
Indicate exceptions where an implicit auth type is used incorrectly.

__init__()
Instantize the class.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**InvalidURL**(*url: str*, *message: str = 'Invalid URL: {}'*)
Indicate exceptions where an invalid URL is entered.

__init__(*url: str*, *message: str = 'Invalid URL: {}'*)
Initialize the class.

Parameters

- **url** – The invalid URL.
- **message** – The message to display. Must contain a format identifier (`{}` or `{0}`). (default: `"Invalid URL: {}"`)

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**MissingRequiredAttributeException**
Indicate exceptions caused by not including a required attribute.

__init__
Initialize self. See help(type(self)) for accurate signature.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**PRAWException**
The base PRAW Exception that all other exception classes extend.

__init__
Initialize self. See help(type(self)) for accurate signature.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception praw.exceptions.**WebSocketException**(*message: str*, *exception: Exception*)
Indicate exceptions caused by use of WebSockets.

__init__(*message: str*, *exception: Exception*)
Initialize a WebSocketException.

Parameters

- **message** – The exception message.
- **exception** – The exception thrown by the websocket library.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

1.10 Other Classes

The following list of classes are provided here for complete documentation. You should not likely need to work with these classes directly, but rather through instances of them bound to an attribute of one of the PRAW models.

1.10.1 Collection

class praw.models.**Collection**(*reddit: Reddit*, *_data: Dict[str, Any] = None*, *collection_id: Optional[str] = None*, *permalink: Optional[str] = None*)
Class to represent a Collection.

Obtain an instance via:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
```

or

```
collection = reddit.subreddit('SUBREDDIT').collections(
    permalink='https://reddit.com/r/SUBREDDIT/collection/some_uuid')
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor that they will be the only attributes present.

Attribute	Description
`author`	The *`Redditor`* who created the collection.
`collection_id`	The UUID of the collection.
`created_at_utc`	Time the collection was created, represented in Unix Time.
`description`	The collection description.
`last_update_utc`	Time the collection was last updated, represented in Unix Time.
`link_ids`	A `list` of *`Submission`* fullnames.
`permalink`	The collection's permalink (to view on the web).
`sorted_links`	An iterable listing of the posts in this collection.
`title`	The title of the collection.

`__init__`(*reddit: Reddit*, *_data: Dict[str, Any] = None*, *collection_id: Optional[str] = None*, *permalink: Optional[str] = None*)

Initialize this collection.

Parameters

- **`reddit`** – An instance of *`Reddit`*.
- **`_data`** – Any data associated with the Collection (optional).
- **`collection_id`** – The ID of the Collection (optional).
- **`permalink`** – The permalink of the Collection (optional).

`__iter__`() → Generator[[Any, None], None]

Provide a way to iterate over the posts in this Collection.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
for submission in collection:
    print(submission.title, submission.permalink)
```

`__len__`() → int

Get the number of posts in this Collection.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
print(len(collection))
```

`follow`()

Follow this Collection.

Example usage:

```
reddit.subreddit('SUBREDDIT').collections('some_uuid').follow()
```

See also *unfollow()*.

mod

Get an instance of *CollectionModeration*.

Provides access to various methods, including *add_post()*, *delete()*, *reorder()*, and *update_title()*.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection.mod.update_title('My new title!')
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

subreddit

Get the subreddit that this collection belongs to.

For example:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
subreddit = collection.subreddit
```

unfollow()

Unfollow this Collection.

Example usage:

```
reddit.subreddit('SUBREDDIT').collections('some_uuid').unfollow()
```

See also *follow()*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.2 CollectionModeration

class praw.models.reddit.collections.**CollectionModeration**(*reddit: Reddit*, *collection_id: str*)

Class to support moderation actions on a *Collection*.

Obtain an instance via:

```
reddit.subreddit('SUBREDDIT').collections('some_uuid').mod
```

__init__(*reddit: Reddit*, *collection_id: str*)

Initialize an instance of CollectionModeration.

Parameters **collection_id** – The ID of a collection.

add_post(*submission: praw.models.reddit.submission.Submission*)
Add a post to the collection.

Parameters **submission** – The post to add, a *Submission*, its permalink as a `str`, its fullname as a `str`, or its ID as a `str`.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection.mod.add_post('bgibu9')
```

See also *remove_post()*.

delete()
Delete this collection.

Example usage:

```
reddit.subreddit('SUBREDDIT').collections('some_uuid').mod.delete()
```

See also *create()*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

remove_post(*submission: praw.models.reddit.submission.Submission*)
Remove a post from the collection.

Parameters **submission** – The post to remove, a *Submission*, its permalink as a `str`, its fullname as a `str`, or its ID as a `str`.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection.mod.remove_post('bgibu9')
```

See also *add_post()*.

reorder(*links: List[Union[str, praw.models.reddit.submission.Submission]]*)
Reorder posts in the collection.

Parameters **links** – A `list` of submissions, as *Submission*, permalink as a `str`, fullname as a `str`, or ID as a `str`.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
current_order = collection.link_ids
new_order = reversed(current_order)
collection.mod.reorder(new_order)
```

update_description(*description: str*)
Update the collection's description.

Parameters **description** – The new description.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection.mod.update_description('Please enjoy these links!')
```

See also *update_title()*.

update_title(*title: str*)
Update the collection's title.

Parameters **title** – The new title.

Example usage:

```
collection = reddit.subreddit('SUBREDDIT').collections('some_uuid')
collection.mod.update_title('Titley McTitleface')
```

See also *update_description()*.

1.10.3 SubredditCollections

class praw.models.reddit.collections.**SubredditCollections**(*reddit: Reddit*, *subreddit: praw.models.reddit.subreddit.Subreddit*, *_data: Optional[Dict[str, Any]] = None*)

Class to represent a Subreddit's *Collection*s.

Obtain an instance via:

```
reddit.subreddit('SUBREDDIT').collections
```

__call__(*collection_id: Optional[str] = None*, *permalink: Optional[str] = None*)
Return the *Collection* with the specified ID.

Parameters

- **collection_id** – The ID of a Collection (default: None).
- **permalink** – The permalink of a Collection (default: None).

Returns The specified Collection.

Exactly one of `collection_id` and `permalink` is required.

Example usage:

```
subreddit = reddit.subreddit('SUBREDDIT')

uuid = '847e4548-a3b5-4ad7-afb4-edbfc2ed0a6b'
collection = subreddit.collections(uuid)
print(collection.title)
print(collection.description)

permalink = 'https://www.reddit.com/r/SUBREDDIT/collection/' + uuid
collection = subreddit.collections(permalink=permalink)
print(collection.title)
print(collection.description)
```

__init__(*reddit: Reddit*, *subreddit: praw.models.reddit.subreddit.Subreddit*, *_data: Optional[Dict[str, Any]] = None*)
Initialize an instance of SubredditCollections.

__iter__()
Iterate over the Subreddit's *Collection*s.

Example usage:

```
for collection in reddit.subreddit('SUBREDDIT').collections:
    print(collection.permalink)
```

mod
Get an instance of *SubredditCollectionsModeration*.

Provides *create()*:

```
my_sub = reddit.subreddit('SUBREDDIT')
new_collection = my_sub.collections.mod.create('Title', 'desc')
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.4 SubredditCollectionsModeration

class praw.models.reddit.collections.**SubredditCollectionsModeration**(*reddit: Reddit*, *sub_fullname: str*, *_data: Optional[Dict[str, Any]] = None*)
Class to represent moderator actions on a Subreddit's Collections.

Obtain an instance via:

```
reddit.subreddit('SUBREDDIT').collections.mod
```

__init__(*reddit: Reddit*, *sub_fullname: str*, *_data: Optional[Dict[str, Any]] = None*)
Initialize the SubredditCollectionsModeration instance.

create(*title: str*, *description: str*)
Create a new *Collection*.

The authenticated account must have appropriate moderator permissions in the subreddit this collection belongs to.

Parameters

- **title** – The title of the collection, up to 300 characters.
- **description** – The description, up to 500 characters.

Returns The newly created *Collection*.

Example usage:

```
my_sub = reddit.subreddit('SUBREDDIT')
new_collection = my_sub.collections.mod.create('Title', 'desc')
new_collection.mod.add_post('bgibu9')
```

See also *delete()*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.5 SubmissionFlair

class praw.models.reddit.submission.**SubmissionFlair**(*submission: _Submission*)
Provide a set of functions pertaining to Submission flair.

__init__(*submission: _Submission*)
Create a SubmissionFlair instance.

Parameters **submission** – The submission associated with the flair functions.

choices() → List[Dict[str, Union[bool, list, str]]]
Return list of available flair choices.

Choices are required in order to use *select()*.

For example:

```
choices = submission.flair.choices()
```

select(*flair_template_id: str*, *text: Optional[str] = None*)
Select flair for submission.

Parameters

- **flair_template_id** – The flair template to select. The possible `flair_template_id` values can be discovered through *choices()*.
- **text** – If the template's `flair_text_editable` value is True, this value will set a custom text (default: None).

For example, to select an arbitrary editable flair text (assuming there is one) and set a custom value try:

```
choices = submission.flair.choices()
template_id = next(x for x in choices
                   if x['flair_text_editable'])['flair_template_id']
submission.flair.select(template_id, 'my custom value')
```

1.10.6 SubredditFlair

class praw.models.reddit.subreddit.**SubredditFlair**(*subreddit*)
Provide a set of functions to interact with a Subreddit's flair.

__call__(*redditor=None*, ***generator_kwargs*)
Return a *ListingGenerator* for Redditors and their flairs.

Parameters **redditor** – When provided, yield at most a single *Redditor* instance (default: None).

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

Usage:

```
for flair in reddit.subreddit('NAME').flair(limit=None):
    print(flair)
```

__init__(*subreddit*)
Create a SubredditFlair instance.

Parameters **subreddit** – The subreddit whose flair to work with.

configure(*position='right'*, *self_assign=False*, *link_position='left'*, *link_self_assign=False*, ***settings*)
Update the subreddit's flair configuration.

Parameters

- **position** – One of left, right, or False to disable (default: right).
- **self_assign** – (boolean) Permit self assignment of user flair (default: False).
- **link_position** – One of left, right, or False to disable (default: left).
- **link_self_assign** – (boolean) Permit self assignment of link flair (default: False).

Additional keyword arguments can be provided to handle new settings as Reddit introduces them.

delete(*redditor*)
Delete flair for a Redditor.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

Note: To delete the flair of many Redditors at once, please see *update()*.

---

delete_all()
Delete all Redditor flair in the Subreddit.

Returns List of dictionaries indicating the success or failure of each delete.

link_templates
Provide an instance of *SubredditLinkFlairTemplates*.

Use this attribute for interacting with a subreddit's link flair templates. For example to list all the link flair templates for a subreddit which you have the `flair` moderator permission on try:

```
for template in reddit.subreddit('NAME').flair.link_templates:
    print(template)
```

set(*redditor*, *text=''*, *css_class=''*, *flair_template_id=None*)
Set flair for a Redditor.

Parameters

- **redditor** – (Required) A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **text** – The flair text to associate with the Redditor or Submission (default: '').

- **css_class** – The css class to associate with the flair html (default: ‘’). Use either this or `flair_template_id`.
- **flair_template_id** – The ID of the flair template to be used (default: `None`). Use either this or `css_class`.

This method can only be used by an authenticated user who is a moderator of the associated Subreddit.

For example:

```
reddit.subreddit('redditdev').flair.set('bboe', 'PRAW author',
                                        css_class='mods')
template = '6bd28436-1aa7-11e9-9902-0e05ab0fad46'
reddit.subreddit('redditdev').flair.set('spez', 'Reddit CEO',
                                        flair_template_id=template)
```

templates

Provide an instance of *SubredditRedditorFlairTemplates*.

Use this attribute for interacting with a subreddit's flair templates. For example to list all the flair templates for a subreddit which you have the `flair` moderator permission on try:

```
for template in reddit.subreddit('NAME').flair.templates:
    print(template)
```

update(*flair_list*, *text=''*, *css_class=''*)

Set or clear the flair for many Redditors at once.

Parameters

- **flair_list** – Each item in this list should be either: the name of a Redditor, an instance of *Redditor*, or a dictionary mapping keys `user`, `flair_text`, and `flair_css_class` to their respective values. The `user` key should map to a Redditor, as described above. When a dictionary isn't provided, or the dictionary is missing one of `flair_text`, or `flair_css_class` attributes the default values will come from the the following arguments.
- **text** – The flair text to use when not explicitly provided in `flair_list` (default: ‘’).
- **css_class** – The css class to use when not explicitly provided in `flair_list` (default: ‘’).

Returns List of dictionaries indicating the success or failure of each update.

For example to clear the flair text, and set the `praw` flair css class on a few users try:

```
subreddit.flair.update(['bboe', 'spez', 'spladug'],
                       css_class='praw')
```

1.10.7 SubredditFlairTemplates

class praw.models.reddit.subreddit.**SubredditFlairTemplates**(*subreddit*)

Provide functions to interact with a Subreddit's flair templates.

__init__(*subreddit*)

Create a SubredditFlairTemplate instance.

Parameters **subreddit** – The subreddit whose flair templates to work with.

Note: This class should not be initialized directly. Instead obtain an instance via: `reddit.subreddit('subreddit_name').flair.templates` or `reddit.subreddit('subreddit_name').flair.link_templates`.

__iter__()
Abstract method to return flair templates.

delete(*template_id*)
Remove a flair template provided by `template_id`.

For example, to delete the first Redditor flair template listed, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.delete(template_info['id'])
```

static flair_type(*is_link*)
Return LINK_FLAIR or USER_FLAIR depending on `is_link` value.

update(*template_id*, *text=None*, *css_class=None*, *text_editable=None*, *background_color=None*, *text_color=None*, *mod_only=None*, *allowable_content=None*, *max_emojis=None*, *fetch=True*)
Update the flair template provided by `template_id`.

Parameters

- **template_id** – The flair template to update. If not valid then an exception will be thrown.
- **text** – The flair template's new text (required).
- **css_class** – The flair template's new css_class (default: '').
- **text_editable** – (boolean) Indicate if the flair text can be modified for each Redditor that sets it (default: False).
- **background_color** – The flair template's new background color, as a hex color.
- **text_color** – The flair template's new text color, either `'light'` or `'dark'`.
- **mod_only** – (boolean) Indicate if the flair can only be used by moderators.
- **allowable_content** – If specified, most be one of `'all'`, `'emoji'`, or `'text'` to restrict content to that type. If set to `'emoji'` then the `'text'` param must be a valid emoji string, for example `':snoo:'`.
- **max_emojis** – (int) Maximum emojis in the flair (Reddit defaults this value to 10).
- **fetch** – Whether or not PRAW will fetch existing information on the existing flair before updating (Default: True).

Warning: If parameter `fetch` is set to `False`, all parameters not provided will be reset to default (`None` or `False`) values.

For example to make a user flair template text_editable, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.update(
    template_info['id'],
```

(continues on next page)

(continued from previous page)

```
    template_info['flair_text'],
    text_editable=True)
```

1.10.8 SubredditLinkFlairTemplates

class praw.models.reddit.subreddit.**SubredditLinkFlairTemplates**(*subreddit*)

Provide functions to interact with link flair templates.

__init__(*subreddit*)

Create a SubredditFlairTemplate instance.

Parameters **subreddit** – The subreddit whose flair templates to work with.

Note: This class should not be initialized directly. Instead obtain an instance via: `reddit.subreddit('subreddit_name').flair.templates` or `reddit.subreddit('subreddit_name').flair.link_templates`.

__iter__()

Iterate through the link flair templates.

For example:

```
for template in reddit.subreddit('NAME').flair.link_templates:
    print(template)
```

add(*text*, *css_class=''*, *text_editable=False*, *background_color=None*, *text_color=None*, *mod_only=None*, *allowable_content=None*, *max_emojis=None*)

Add a link flair template to the associated subreddit.

Parameters

- **text** – The flair template's text (required).
- **css_class** – The flair template's css_class (default: '').
- **text_editable** – (boolean) Indicate if the flair text can be modified for each Redditor that sets it (default: False).
- **background_color** – The flair template's new background color, as a hex color.
- **text_color** – The flair template's new text color, either `'light'` or `'dark'`.
- **mod_only** – (boolean) Indicate if the flair can only be used by moderators.
- **allowable_content** – If specified, most be one of `'all'`, `'emoji'`, or `'text'` to restrict content to that type. If set to `'emoji'` then the `'text'` param must be a valid emoji string, for example `':snoo:'`.
- **max_emojis** – (int) Maximum emojis in the flair (Reddit defaults this value to 10).

For example, to add an editable link flair try:

```
reddit.subreddit('NAME').flair.link_templates.add(
    css_class='praw', text_editable=True)
```

clear()

Remove all link flair templates from the subreddit.

For example:

```
reddit.subreddit('NAME').flair.link_templates.clear()
```

delete(*template_id*)
Remove a flair template provided by `template_id`.

For example, to delete the first Redditor flair template listed, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.delete(template_info['id'])
```

static flair_type(*is_link*)
Return LINK_FLAIR or USER_FLAIR depending on `is_link` value.

update(*template_id*, *text=None*, *css_class=None*, *text_editable=None*, *background_color=None*, *text_color=None*, *mod_only=None*, *allowable_content=None*, *max_emojis=None*, *fetch=True*)
Update the flair template provided by `template_id`.

Parameters

- **template_id** – The flair template to update. If not valid then an exception will be thrown.
- **text** – The flair template's new text (required).
- **css_class** – The flair template's new css_class (default: '').
- **text_editable** – (boolean) Indicate if the flair text can be modified for each Redditor that sets it (default: False).
- **background_color** – The flair template's new background color, as a hex color.
- **text_color** – The flair template's new text color, either `'light'` or `'dark'`.
- **mod_only** – (boolean) Indicate if the flair can only be used by moderators.
- **allowable_content** – If specified, most be one of `'all'`, `'emoji'`, or `'text'` to restrict content to that type. If set to `'emoji'` then the `'text'` param must be a valid emoji string, for example `':snoo:'`.
- **max_emojis** – (int) Maximum emojis in the flair (Reddit defaults this value to 10).
- **fetch** – Whether or not PRAW will fetch existing information on the existing flair before updating (Default: True).

Warning: If parameter `fetch` is set to `False`, all parameters not provided will be reset to default (`None` or `False`) values.

For example to make a user flair template text_editable, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.update(
    template_info['id'],
    template_info['flair_text'],
    text_editable=True)
```

1.10.9 SubredditRedditorFlairTemplates

class praw.models.reddit.subreddit.**SubredditRedditorFlairTemplates**(*subreddit*)
Provide functions to interact with Redditor flair templates.

__init__(*subreddit*)
Create a SubredditFlairTemplate instance.

Parameters **subreddit** – The subreddit whose flair templates to work with.

Note: This class should not be initialized directly. Instead obtain an instance via: `reddit.subreddit('subreddit_name').flair.templates` or `reddit.subreddit('subreddit_name').flair.link_templates`.

__iter__()
Iterate through the user flair templates.

For example:

```
for template in reddit.subreddit('NAME').flair.templates:
    print(template)
```

add(*text*, *css_class=''*, *text_editable=False*, *background_color=None*, *text_color=None*, *mod_only=None*, *allowable_content=None*, *max_emojis=None*)
Add a Redditor flair template to the associated subreddit.

Parameters

- **text** – The flair template's text (required).
- **css_class** – The flair template's css_class (default: '').
- **text_editable** – (boolean) Indicate if the flair text can be modified for each Redditor that sets it (default: False).
- **background_color** – The flair template's new background color, as a hex color.
- **text_color** – The flair template's new text color, either `'light'` or `'dark'`.
- **mod_only** – (boolean) Indicate if the flair can only be used by moderators.
- **allowable_content** – If specified, most be one of `'all'`, `'emoji'`, or `'text'` to restrict content to that type. If set to `'emoji'` then the `'text'` param must be a valid emoji string, for example `':snoo:'`.
- **max_emojis** – (int) Maximum emojis in the flair (Reddit defaults this value to 10).

For example, to add an editable Redditor flair try:

```
reddit.subreddit('NAME').flair.templates.add(
    css_class='praw', text_editable=True)
```

clear()
Remove all Redditor flair templates from the subreddit.

For example:

```
reddit.subreddit('NAME').flair.templates.clear()
```

delete(*template_id*)
Remove a flair template provided by `template_id`.

For example, to delete the first Redditor flair template listed, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.delete(template_info['id'])
```

static flair_type(*is_link*)
Return LINK_FLAIR or USER_FLAIR depending on `is_link` value.

update(*template_id*, *text=None*, *css_class=None*, *text_editable=None*, *background_color=None*, *text_color=None*, *mod_only=None*, *allowable_content=None*, *max_emojis=None*, *fetch=True*)
Update the flair template provided by `template_id`.

Parameters

- **template_id** – The flair template to update. If not valid then an exception will be thrown.
- **text** – The flair template's new text (required).
- **css_class** – The flair template's new css_class (default: '').
- **text_editable** – (boolean) Indicate if the flair text can be modified for each Redditor that sets it (default: False).
- **background_color** – The flair template's new background color, as a hex color.
- **text_color** – The flair template's new text color, either `'light'` or `'dark'`.
- **mod_only** – (boolean) Indicate if the flair can only be used by moderators.
- **allowable_content** – If specified, most be one of `'all'`, `'emoji'`, or `'text'` to restrict content to that type. If set to `'emoji'` then the `'text'` param must be a valid emoji string, for example `':snoo:'`.
- **max_emojis** – (int) Maximum emojis in the flair (Reddit defaults this value to 10).
- **fetch** – Whether or not PRAW will fetch existing information on the existing flair before updating (Default: True).

Warning: If parameter `fetch` is set to `False`, all parameters not provided will be reset to default (`None` or `False`) values.

For example to make a user flair template text_editable, try:

```
template_info = list(subreddit.flair.templates)[0]
subreddit.flair.templates.update(
    template_info['id'],
    template_info['flair_text'],
    text_editable=True)
```

1.10.10 LiveContributorRelationship

class praw.models.reddit.live.**LiveContributorRelationship**(*thread: _LiveThread*)
Provide methods to interact with live threads' contributors.

__call__() → List[praw.models.reddit.redditor.Redditor]
Return a *RedditorList* for live threads' contributors.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
for contributor in thread.contributor():
    print(contributor)
```

__init__(*thread: _LiveThread*)

Create a *LiveContributorRelationship* instance.

Parameters **thread** – An instance of *LiveThread*.

Note: This class should not be initialized directly. Instead obtain an instance via: `thread.contributor` where `thread` is a *LiveThread* instance.

---

accept_invite()

Accept an invite to contribute the live thread.

Usage:

```
thread = reddit.live('ydwwxneu7vsa')
thread.contributor.accept_invite()
```

invite(*redditor: Union[str, praw.models.reddit.redditor.Redditor], permissions: Optional[List[str]] = None*)

Invite a redditor to be a contributor of the live thread.

Raise *praw.exceptions.APIException* if the invitation already exists.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant. An empty list `[]` indicates no permissions, and when not provided (`None`), indicates full permissions.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
redditor = reddit.redditor('spez')

# 'manage' and 'settings' permissions
thread.contributor.invite(redditor, ['manage', 'settings'])
```

Seealso *LiveContributorRelationship.remove_invite()* to remove the invite for redditor.

leave()

Abdicate the live thread contributor position (use with care).

Usage:

```
thread = reddit.live('ydwwxneu7vsa')
thread.contributor.leave()
```

remove(*redditor: Union[str, praw.models.reddit.redditor.Redditor]*)

Remove the redditor from the live thread contributors.

Parameters **redditor** – A redditor fullname (e.g., `'t2_1w72'`) or *Redditor* instance.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
redditor = reddit.redditor('spez')
thread.contributor.remove(redditor)
thread.contributor.remove('t2_1w72')  # with fullname
```

remove_invite(*redditor: Union[str, praw.models.reddit.redditor.Redditor]*)

Remove the invite for redditor.

Parameters **redditor** – A redditor fullname (e.g., `'t2_1w72'`) or *Redditor* instance.

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
redditor = reddit.redditor('spez')
thread.contributor.remove_invite(redditor)
thread.contributor.remove_invite('t2_1w72')  # with fullname
```

Seealso *LiveContributorRelationship.invite()* to invite a redditor to be a contributor of the live thread.

update(*redditor: Union[str, praw.models.reddit.redditor.Redditor], permissions: Optional[List[str]] = None*)

Update the contributor permissions for `redditor`.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant (other permissions are removed). An empty list `[]` indicates no permissions, and when not provided (`None`), indicates full permissions.

For example, to grant all permissions to the contributor, try:

```
thread = reddit.live('ukaeu1ik4sw5')
thread.contributor.update('spez')
```

To grant 'access' and 'edit' permissions (and to remove other permissions), try:

```
thread.contributor.update('spez', ['access', 'edit'])
```

To remove all permissions from the contributor, try:

```
subreddit.moderator.update('spez', [])
```

update_invite(*redditor: Union[str, praw.models.reddit.redditor.Redditor], permissions: Optional[List[str]] = None*)

Update the contributor invite permissions for `redditor`.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant (other permissions are removed). An empty list `[]` indicates no permissions, and when not provided (`None`), indicates full permissions.

For example, to set all permissions to the invitation, try:

```
thread = reddit.live('ukaeu1ik4sw5')
thread.contributor.update_invite('spez')
```

To set 'access' and 'edit' permissions (and to remove other permissions) to the invitation, try:

```
thread.contributor.update_invite('spez', ['access', 'edit'])
```

To remove all permissions from the invitation, try:

```
thread.contributor.update_invite('spez', [])
```

1.10.11 LiveThreadContribution

class praw.models.reddit.live.**LiveThreadContribution**(*thread: praw.models.reddit.live.LiveThread*)

Provides a set of contribution functions to a LiveThread.

__init__(*thread: praw.models.reddit.live.LiveThread*)

Create an instance of *LiveThreadContribution*.

Parameters **thread** – An instance of *LiveThread*.

This instance can be retrieved through `thread.contrib` where thread is a *LiveThread* instance. E.g.,

```
thread = reddit.live('ukaeu1ik4sw5')
thread.contrib.add('### update')
```

add(*body: str*)

Add an update to the live thread.

Parameters **body** – The Markdown formatted content for the update.

Usage:

```
thread = reddit.live('ydwwxneu7vsa')
thread.contrib.add('test `LiveThreadContribution.add()`')
```

close()

Close the live thread permanently (cannot be undone).

Usage:

```
thread = reddit.live('ukaeu1ik4sw5')
thread.contrib.close()
```

update(*title: Optional[str] = None*, *description: Optional[str] = None*, *nsfw: Optional[bool] = None*, *resources: Optional[str] = None*, ***other_settings*)

Update settings of the live thread.

Parameters

- **title** – (Optional) The title of the live thread (default: None).
- **description** – (Optional) The live thread's description (default: None).
- **nsfw** – (Optional) Indicate whether this thread is not safe for work (default: None).
- **resources** – (Optional) Markdown formatted information that is useful for the live thread (default: None).

Does nothing if no arguments are provided.

Each setting will maintain its current value if `None` is specified.

Additional keyword arguments can be provided to handle new settings as Reddit introduces them.

Usage:

```
thread = reddit.live('xyu8kmjvfrww')

# update `title` and `nsfw`
updated_thread = thread.contrib.update(title=new_title, nsfw=True)
```

If Reddit introduces new settings, you must specify `None` for the setting you want to maintain:

```
# update `nsfw` and maintain new setting `foo`
thread.contrib.update(nsfw=True, foo=None)
```

1.10.12 LiveUpdateContribution

class praw.models.reddit.live.**LiveUpdateContribution**(*update: praw.models.reddit.live.LiveUpdate*)

Provides a set of contribution functions to LiveUpdate.

__init__(*update: praw.models.reddit.live.LiveUpdate*)

Create an instance of *LiveUpdateContribution*.

Parameters **update** – An instance of *LiveUpdate*.

This instance can be retrieved through `update.contrib` where update is a *LiveUpdate* instance. E.g.,

```
thread = reddit.live('ukaeu1ik4sw5')
update = thread['7827987a-c998-11e4-a0b9-22000b6a88d2']
update.contrib  # LiveUpdateContribution instance
update.contrib.remove()
```

remove()

Remove a live update.

Usage:

```
thread = reddit.live('ydwwxneu7vsa')
update = thread['6854605a-efec-11e6-b0c7-0eafac4ff094']
update.contrib.remove()
```

strike()

Strike a content of a live update.

```
thread = reddit.live('xyu8kmjvfrww')
update = thread['cb5fe532-dbee-11e6-9a91-0e6d74fabcc4']
update.contrib.strike()
```

To check whether the update is stricken or not, use `update.stricken` attribute. But note that accessing lazy attributes on updates (includes `update.stricken`) may raises `AttributeError`. See *LiveUpdate* for details.

1.10.13 CommentModeration

class praw.models.reddit.comment.**CommentModeration**(*comment: praw.models.reddit.comment.Comment*)

Provide a set of functions pertaining to Comment moderation.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.mod.approve()
```

__init__(*comment: praw.models.reddit.comment.Comment*)

Create a CommentModeration instance.

Parameters **comment** – The comment to moderate.

approve()

Approve a *Comment* or *Submission*.

Approving a comment or submission reverts a removal, resets the report counter, adds a green check mark indicator (only visible to other moderators) on the website view, and sets the `approved_by` attribute to the authenticated user.

Example usage:

```
# approve a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.approve()
# approve a submission:
submission = reddit.submission(id='5or86n')
submission.mod.approve()
```

distinguish(*how='yes'*, *sticky=False*)

Distinguish a *Comment* or *Submission*.

Parameters

- **how** – One of 'yes', 'no', 'admin', 'special'. 'yes' adds a moderator level distinguish. 'no' removes any distinction. 'admin' and 'special' require special user privileges to use.
- **sticky** – Comment is stickied if `True`, placing it at the top of the comment page regardless of score. If thing is not a top-level comment, this parameter is silently ignored.

Example usage:

```
# distinguish and sticky a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.distinguish(how='yes', sticky=True)
# undistinguish a submission:
submission = reddit.submission(id='5or86n')
submission.mod.distinguish(how='no')
```

See also *undistinguish()*

ignore_reports()

Ignore future reports on a *Comment* or *Submission*.

Calling this method will prevent future reports on this Comment or Submission from both triggering notifications and appearing in the various moderation listings. The report count will still increment on the Comment or Submission.

Example usage:

```
# ignore future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.ignore_reports()
# ignore future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.ignore_reports()
```

See also *unignore_reports()*

lock()

Lock a *Comment* or *Submission*.

Example usage:

```
# lock a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.lock()
# lock a submission:
submission = reddit.submission(id='5or86n')
submission.mod.lock()
```

See also *unlock()*

remove(*spam=False*, *mod_note=''*, *reason_id=None*)

Remove a *Comment* or *Submission*.

Parameters

- **mod_note** – A message for the other moderators.
- **spam** – When True, use the removal to help train the Subreddit's spam filter (default: False).
- **reason_id** – The removal reason ID.

If either `reason_id` or `mod_note` are provided, a second API call is made to add the removal reason.

Example usage:

```
# remove a comment and mark as spam:
comment = reddit.comment('dkk4qjd')
comment.mod.remove(spam=True)
# remove a submission
submission = reddit.submission(id='5or86n')
submission.mod.remove()
# remove a submission with a removal reason
reason = reddit.subreddit.mod.removal_reasons["110ni21zo23ql"]
submission = reddit.submission(id="5or86n")
submission.mod.remove(reason_id=reason.id)
```

send_removal_message(*message*, *title='ignored'*, *type='public'*)

Send a removal message for a *Comment* or *Submission*.

Warning: The object has to be removed before giving it a removal reason. Remove the object with *remove()*. Trying to add a removal reason without removing the object will result in `prawcore.exceptions.BadRequest` being thrown.

Reddit adds human-readable information about the object to the message.

Parameters

- **type** – One of 'public', 'private', 'private_exposed'. 'public' leaves a stickied comment on the post. 'private' sends a Modmail message with hidden username. 'private_exposed' sends a Modmail message without hidden username.
- **title** – The short reason given in the message. (Ignored if type is 'public'.)
- **message** – The body of the message.

If `type` is 'public', the new *Comment* is returned.

undistinguish()

Remove mod, admin, or special distinguishing from an object.

Also unstickies the object if applicable.

Example usage:

```
# undistinguish a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.undistinguish()
# undistinguish a submission:
submission = reddit.submission(id='5or86n')
submission.mod.undistinguish()
```

See also *distinguish()*

unignore_reports()

Resume receiving future reports on a Comment or Submission.

Future reports on this *Comment* or *Submission* will cause notifications, and appear in the various moderation listings.

Example usage:

```
# accept future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.unignore_reports()
# accept future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.unignore_reports()
```

See also *ignore_reports()*

unlock()

Unlock a *Comment* or *Submission*.

Example usage:

unlock a comment: comment = reddit.comment('dkk4qjd') comment.mod.unlock() # unlock a submission: submission = reddit.submission(id='5or86n') submission.mod.unlock()

See also *lock()*

1.10.14 SubmissionModeration

class praw.models.reddit.submission.**SubmissionModeration**(*submission: _Submission*)

Provide a set of functions pertaining to Submission moderation.

Example usage:

```
submission = reddit.submission(id="8dmv8z")
submission.mod.approve()
```

__init__(*submission: _Submission*)
Create a SubmissionModeration instance.

Parameters **submission** – The submission to moderate.

approve()
Approve a *Comment* or *Submission*.

Approving a comment or submission reverts a removal, resets the report counter, adds a green check mark indicator (only visible to other moderators) on the website view, and sets the `approved_by` attribute to the authenticated user.

Example usage:

```
# approve a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.approve()
# approve a submission:
submission = reddit.submission(id='5or86n')
submission.mod.approve()
```

contest_mode(*state: bool = True*)
Set contest mode for the comments of this submission.

Parameters **state** – (boolean) True enables contest mode, False, disables (default: True).

Contest mode have the following effects:

- The comment thread will default to being sorted randomly.
- Replies to top-level comments will be hidden behind "[show replies]" buttons.
- Scores will be hidden from non-moderators.
- Scores accessed through the API (mobile apps, bots) will be obscured to "1" for non-moderators.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.mod.contest_mode(state=True)
```

distinguish(*how='yes', sticky=False*)
Distinguish a *Comment* or *Submission*.

Parameters

- **how** – One of 'yes', 'no', 'admin', 'special'. 'yes' adds a moderator level distinguish. 'no' removes any distinction. 'admin' and 'special' require special user privileges to use.
- **sticky** – Comment is stickied if `True`, placing it at the top of the comment page regardless of score. If thing is not a top-level comment, this parameter is silently ignored.

Example usage:

```
# distinguish and sticky a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.distinguish(how='yes', sticky=True)
# undistinguish a submission:
```

(continues on next page)

(continued from previous page)

```
submission = reddit.submission(id='5or86n')
submission.mod.distinguish(how='no')
```

See also *undistinguish()*

flair(*text: str = ''*, *css_class: str = ''*, *flair_template_id: Optional[str] = None*)
Set flair for the submission.

Parameters

- **text** – The flair text to associate with the Submission (default: '').
- **css_class** – The css class to associate with the flair html (default: '').
- **flair_template_id** – The flair template id to use when flairing (Optional).

This method can only be used by an authenticated user who is a moderator of the Submission's Subreddit.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.mod.flair(text='PRAW', css_class='bot')
```

ignore_reports()
Ignore future reports on a *Comment* or *Submission*.

Calling this method will prevent future reports on this Comment or Submission from both triggering notifications and appearing in the various moderation listings. The report count will still increment on the Comment or Submission.

Example usage:

```
# ignore future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.ignore_reports()
# ignore future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.ignore_reports()
```

See also *unignore_reports()*

lock()
Lock a *Comment* or *Submission*.

Example usage:

```
# lock a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.lock()
# lock a submission:
submission = reddit.submission(id='5or86n')
submission.mod.lock()
```

See also *unlock()*

nsfw()
Mark as not safe for work.

This method can be used both by the submission author and moderators of the subreddit that the submission belongs to.

Example usage:

```
submission = reddit.subreddit('test').submit('nsfw test',
                                             selftext='nsfw')
submission.mod.nsfw()
```

See also *sfw()*

remove(*spam=False*, *mod_note=''*, *reason_id=None*)
Remove a *Comment* or *Submission*.

Parameters

- **mod_note** – A message for the other moderators.
- **spam** – When True, use the removal to help train the Subreddit's spam filter (default: False).
- **reason_id** – The removal reason ID.

If either `reason_id` or `mod_note` are provided, a second API call is made to add the removal reason.

Example usage:

```
# remove a comment and mark as spam:
comment = reddit.comment('dkk4qjd')
comment.mod.remove(spam=True)
# remove a submission
submission = reddit.submission(id='5or86n')
submission.mod.remove()
# remove a submission with a removal reason
reason = reddit.subreddit.mod.removal_reasons["110ni21zo23ql"]
submission = reddit.submission(id="5or86n")
submission.mod.remove(reason_id=reason.id)
```

send_removal_message(*message*, *title='ignored'*, *type='public'*)
Send a removal message for a *Comment* or *Submission*.

Warning: The object has to be removed before giving it a removal reason. Remove the object with *remove()*. Trying to add a removal reason without removing the object will result in `prawcore.exceptions.BadRequest` being thrown.

Reddit adds human-readable information about the object to the message.

Parameters

- **type** – One of 'public', 'private', 'private_exposed'. 'public' leaves a stickied comment on the post. 'private' sends a Modmail message with hidden username. 'private_exposed' sends a Modmail message without hidden username.
- **title** – The short reason given in the message. (Ignored if type is 'public'.)
- **message** – The body of the message.

If `type` is 'public', the new *Comment* is returned.

set_original_content()
Mark as original content.

This method can be used by moderators of the subreddit that the submission belongs to. If the subreddit has enabled the Original Content beta feature in settings, then the submission's author can use it as well.

Example usage:

```
submission = reddit.subreddit('test').submit('oc test',
                                             selftext='original')
submission.mod.set_original_content()
```

See also *unset_original_content()*

sfw()
Mark as safe for work.

This method can be used both by the submission author and moderators of the subreddit that the submission belongs to.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.mod.sfw()
```

See also *nsfw()*

spoiler()
Indicate that the submission contains spoilers.

This method can be used both by the submission author and moderators of the subreddit that the submission belongs to.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.mod.spoiler()
```

See also *unspoiler()*

sticky(*state: bool = True*, *bottom: bool = True*)
Set the submission's sticky state in its subreddit.

Parameters

- **state** – (boolean) True sets the sticky for the submission, false unsets (default: True).
- **bottom** – (boolean) When true, set the submission as the bottom sticky. If no top sticky exists, this submission will become the top sticky regardless (default: True).

This submission will replace an existing stickied submission if one exists.

For example:

```
submission = reddit.submission(id='5or86n')
submission.mod.sticky()
```

suggested_sort(*sort: str = 'blank'*)
Set the suggested sort for the comments of the submission.

Parameters **sort** – Can be one of: confidence, top, new, controversial, old, random, qa, blank (default: blank).

undistinguish()
Remove mod, admin, or special distinguishing from an object.

Also unstickies the object if applicable.

Example usage:

```
# undistinguish a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.undistinguish()
# undistinguish a submission:
submission = reddit.submission(id='5or86n')
submission.mod.undistinguish()
```

See also *distinguish()*

unignore_reports()
Resume receiving future reports on a Comment or Submission.

Future reports on this *Comment* or *Submission* will cause notifications, and appear in the various moderation listings.

Example usage:

```
# accept future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.unignore_reports()
# accept future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.unignore_reports()
```

See also *ignore_reports()*

unlock()
Unlock a *Comment* or *Submission*.

Example usage:

unlock a comment: comment = reddit.comment('dkk4qjd') comment.mod.unlock() # unlock a submission: submission = reddit.submission(id='5or86n') submission.mod.unlock()

See also *lock()*

unset_original_content()
Indicate that the submission is not original content.

This method can be used by moderators of the subreddit that the submission belongs to. If the subreddit has enabled the Original Content beta feature in settings, then the submission's author can use it as well.

Example usage:

```
submission = reddit.subreddit('test').submit('oc test',
                                             selftext='original')
submission.mod.unset_original_content()
```

See also *set_original_content()*

unspoiler()
Indicate that the submission does not contain spoilers.

This method can be used both by the submission author and moderators of the subreddit that the submission belongs to.

For example:

```
submission = reddit.subreddit('test').submit('not spoiler',
                                             selftext='spoiler')
submission.mod.unspoiler()
```

See also *spoiler()*

1.10.15 SubredditModeration

class praw.models.reddit.subreddit.**SubredditModeration**(*subreddit*)

Provides a set of moderation functions to a Subreddit.

For example, to accept a moderation invite from subreddit `r/test`:

```
reddit.subreddit('test').mod.accept_invite()
```

__init__(*subreddit*)

Create a SubredditModeration instance.

Parameters **subreddit** – The subreddit to moderate.

accept_invite()

Accept an invitation as a moderator of the community.

edited(*only=None*, ***generator_kwargs*)

Return a *ListingGenerator* for edited comments and submissions.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print all items in the edited queue try:

```
for item in reddit.subreddit('mod').mod.edited(limit=None):
    print(item)
```

inbox(***generator_kwargs*)

Return a *ListingGenerator* for moderator messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

See `unread` for unread moderator messages.

To print the last 5 moderator mail messages and their replies, try:

```
for message in reddit.subreddit('mod').mod.inbox(limit=5):
    print("From: {}, Body: {}".format(message.author, message.body))
    for reply in message.replies:
        print("From: {}, Body: {}".format(reply.author, reply.body))
```

log(*action=None*, *mod=None*, ***generator_kwargs*)

Return a *ListingGenerator* for moderator log entries.

Parameters

- **action** – If given, only return log entries for the specified action.
- **mod** – If given, only return log entries for actions made by the passed in Redditor.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print the moderator and subreddit of the last 5 modlog entries try:

```
for log in reddit.subreddit('mod').mod.log(limit=5):
    print("Mod: {}, Subreddit: {}".format(log.mod, log.subreddit))
```

modqueue(*only=None*, ***generator_kwargs*)
Return a *ListingGenerator* for modqueue items.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print all modqueue items try:

```
for item in reddit.subreddit('mod').mod.modqueue(limit=None):
    print(item)
```

removal_reasons
Provide an instance of *SubredditRemovalReasons*.

Use this attribute for interacting with a subreddit's removal reasons. For example to list all the removal reasons for a subreddit which you have the `posts` moderator permission on, try:

```
for removal_reason in reddit.subreddit('NAME').mod.removal_reasons:
    print(removal_reason)
```

A single removal reason can be lazily retrieved via:

```
reddit.subreddit('NAME').mod.removal_reasons['reason_id']
```

Note: Attempting to access attributes of an nonexistent removal reason will result in a *ClientException*.

reports(*only=None*, ***generator_kwargs*)
Return a *ListingGenerator* for reported comments and submissions.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print the user and mod report reasons in the report queue try:

```
for reported_item in reddit.subreddit('mod').mod.reports():
    print("User Reports: {}".format(reported_item.user_reports))
    print("Mod Reports: {}".format(reported_item.mod_reports))
```

settings()
Return a dictionary of the subreddit's current settings.

spam(*only=None*, ***generator_kwargs*)
Return a *ListingGenerator* for spam comments and submissions.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print the items in the spam queue try:

```
for item in reddit.subreddit('mod').mod.spam():
    print(item)
```

stream
Provide an instance of *SubredditModerationStream*.

Streams can be used to indefinitely retrieve Moderator only items from *SubredditModeration* made to moderated subreddits, like:

```
for log in reddit.subreddit('mod').mod.stream.log():
    print("Mod: {}, Subreddit: {}".format(log.mod, log.subreddit))
```

unmoderated(***generator_kwargs*)
Return a *ListingGenerator* for unmoderated submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To print the items in the unmoderated queue try:

```
for item in reddit.subreddit('mod').mod.unmoderated():
    print(item)
```

unread(***generator_kwargs*)
Return a *ListingGenerator* for unread moderator messages.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

See `inbox` for all messages.

To print the mail in the unread modmail queue try:

```
for message in reddit.subreddit('mod').mod.unread():
    print("From: {}, To: {}".format(message.author, message.dest))
```

update(***settings*)
Update the subreddit's settings.

Parameters

- **allow_images** – Allow users to upload images using the native image hosting. Only applies to link-only subreddits.
- **allow_post_crossposts** – Allow users to crosspost submissions from other subreddits.
- **allow_top** – Allow the subreddit to appear on `r/all` as well as the default and trending lists.
- **collapse_deleted_comments** – Collapse deleted and removed comments on comments pages by default.
- **comment_score_hide_mins** – The number of minutes to hide comment scores.
- **description** – Shown in the sidebar of your subreddit.
- **disable_contributor_requests** (*bool*) – Specifies whether redditors may send automated modmail messages requesting approval as a submitter.
- **domain** – Domain name with a cname that points to {subreddit}.reddit.com.
- **exclude_banned_modqueue** – Exclude posts by site-wide banned users from modqueue/unmoderated.
- **header_hover_text** – The text seen when hovering over the snoo.
- **hide_ads** – Don't show ads within this subreddit. Only applies to Premium-user only subreddits.

- **key_color** – A 6-digit rgb hex color (e.g. `'#AABBCC'`), used as a thematic color for your subreddit on mobile.
- **lang** – A valid IETF language tag (underscore separated).
- **link_type** – The types of submissions users can make. One of `any`, `link`, `self`.
- **over_18** – Viewers must be over 18 years old (i.e. NSFW).
- **public_description** – Public description blurb. Appears in search results and on the landing page for private subreddits.
- **public_traffic** – Make the traffic stats page public.
- **restrict_commenting** (*bool*) – Specifies whether approved users have the ability to comment.
- **restrict_posting** (*bool*) – Specifies whether approved users have the ability to submit posts.
- **show_media** – Show thumbnails on submissions.
- **show_media_preview** – Expand media previews on comments pages.
- **spam_comments** – Spam filter strength for comments. One of `all`, `low`, `high`.
- **spam_links** – Spam filter strength for links. One of `all`, `low`, `high`.
- **spam_selfposts** – Spam filter strength for selfposts. One of `all`, `low`, `high`.
- **spoilers_enabled** – Enable marking posts as containing spoilers.
- **sr** – The fullname of the subreddit whose settings will be updated.
- **submit_link_label** – Custom label for submit link button (None for default).
- **submit_text** – Text to show on submission page.
- **submit_text_label** – Custom label for submit text post button (None for default).
- **subreddit_type** – One of `archived`, `employees_only`, `gold_only`, `gold_restricted`, `private`, `public`, `restricted`.
- **suggested_comment_sort** – All comment threads will use this sorting method by default. Leave None, or choose one of `confidence`, `controversial`, `new`, `old`, `qa`, `random`, `top`.
- **title** – The title of the subreddit.
- **wiki_edit_age** – Account age, in days, required to edit and create wiki pages.
- **wiki_edit_karma** – Subreddit karma required to edit and create wiki pages.
- **wikimode** – One of `anyone`, `disabled`, `modonly`.

Additional keyword arguments can be provided to handle new settings as Reddit introduces them.

Settings that are documented here and aren't explicitly set by you in a call to *SubredditModeration.update()* should retain their current value. If they do not please file a bug.

Warning: Undocumented settings, or settings that were very recently documented, may not retain their current value when updating. This often occurs when Reddit adds a new setting but forgets to add that setting to the API endpoint that is used to fetch the current settings.

1.10.16 SubredditWidgetsModeration

class praw.models.**SubredditWidgetsModeration**(*subreddit*, *reddit*)

Class for moderating a subreddit's widgets.

Get an instance of this class from *SubredditWidgets.mod*.

Example usage:

```
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
reddit.subreddit('learnpython').widgets.mod.add_text_area(
    'My title', '**bold text**', styles)
```

Note: To use this class's methods, the authenticated user must be a moderator with appropriate permissions.

__init__(*subreddit*, *reddit*)

Initialize the class.

add_button_widget(*short_name*, *description*, *buttons*, *styles*, ***other_settings*)

Add and return a *ButtonWidget*.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **description** – Markdown text to describe the widget.
- **buttons** – A `list` of `dicts` describing buttons, as specified in Reddit docs. As of this writing, the format is:

  Each button is either a text button or an image button. A text button looks like this:

```
{
  "kind": "text",
  "text": a string no longer than 30 characters,
  "url": a valid URL,
  "color": a 6-digit rgb hex color, e.g. `#AABBCC`,
  "textColor": a 6-digit rgb hex color, e.g. `#AABBCC`,
  "fillColor": a 6-digit rgb hex color, e.g. `#AABBCC`,
  "hoverState": {...}
}
```

  An image button looks like this:

```
{
  "kind": "image",
  "text": a string no longer than 30 characters,
  "linkUrl": a valid URL,
  "url": a valid URL of a reddit-hosted image,
  "height": an integer,
  "width": an integer,
  "hoverState": {...}
}
```

  Both types of buttons have the field `hoverState`. The field does not have to be included (it is optional). If it is included, it can be one of two types: text or image. A text `hoverState` looks like this:

```
{
  "kind": "text",
  "text": a string no longer than 30 characters,
  "color": a 6-digit rgb hex color, e.g. `#AABBCC`,
  "textColor": a 6-digit rgb hex color, e.g. `#AABBCC`,
  "fillColor": a 6-digit rgb hex color, e.g. `#AABBCC`
}
```

An image `hoverState` looks like this:

```
{
  "kind": "image",
  "url": a valid URL of a reddit-hosted image,
  "height": an integer,
  "width": an integer
}
```

Note: The method *`upload_image()`* can be used to upload images to Reddit for a `url` field that holds a Reddit-hosted image.

Note: An image `hoverState` may be paired with a text widget, and a text `hoverState` may be paired with an image widget.

- **`styles`** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
my_image = widget_moderation.upload_image('/path/to/pic.jpg')
buttons = [
    {
        'kind': 'text',
        'text': 'View source',
        'url': 'https://github.com/praw-dev/praw',
        'color': '#FF0000',
        'textColor': '#00FF00',
        'fillColor': '#0000FF',
        'hoverState': {
            'kind': 'text',
            'text': 'ecruos weiV',
            'color': '#FFFFFF',
            'textColor': '#000000',
            'fillColor': '#0000FF'
        }
    },
    {
        'kind': 'image',
        'text': 'View documentation',
        'linkUrl': 'https://praw.readthedocs.io',
        'url': my_image,
        'height': 200,
        'width': 200,
```

(continues on next page)

(continued from previous page)

```
        'hoverState': {
            'kind': 'image',
            'url': my_image,
            'height': 200,
            'width': 200
        }
    }
]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
new_widget = widget_moderation.add_button_widget(
    'Things to click', 'Click some of these *cool* links!',
    buttons, styles)
```

add_calendar(*short_name*, *google_calendar_id*, *requires_sync*, *configuration*, *styles*, ***other_settings*)

Add and return a *Calendar* widget.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **google_calendar_id** – An email-style calendar ID. To share a Google Calendar, make it public, then find the "Calendar ID."
- **requires_sync** – A `bool`.
- **configuration** – A `dict` as specified in Reddit docs.

  For example:

  ```
  {'numEvents': 10,
   'showDate': True,
   'showDescription': False,
   'showLocation': False,
   'showTime': True,
   'showTitle': True}
  ```

- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
config = {'numEvents': 10,
          'showDate': True,
          'showDescription': False,
          'showLocation': False,
          'showTime': True,
          'showTitle': True}
cal_id = 'y6nm89jy427drk8l71w75w9wjn@group.calendar.google.com'
new_widget = widget_moderation.add_calendar('Upcoming Events',
                                            cal_id, True,
                                            config, styles)
```

add_community_list(*short_name*, *data*, *styles*, *description=''*, ***other_settings*)

Add and return a *CommunityList* widget.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **data** – A `list` of subreddits. Subreddits can be represented as `str` (e.g. the string `'redditdev'`) or as *Subreddit* (e.g. `reddit.subreddit('redditdev')`). These types may be mixed within the list.
- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.
- **description** – A `str` containing Markdown (default: `''`).

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
subreddits = ['learnpython', reddit.subreddit('redditdev')]
new_widget = widget_moderation.add_community_list('My fav subs',
                                                  subreddits,
                                                  styles,
                                                  'description')
```

add_custom_widget(*short_name*, *text*, *css*, *height*, *image_data*, *styles*, ***other_settings*)
Add and return a *CustomWidget*.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **text** – The Markdown text displayed in the widget.
- **css** – The CSS for the widget, no longer than 100000 characters.

Note: As of this writing, Reddit will not accept empty CSS. If you wish to create a custom widget without CSS, consider using `'/**/'` (an empty comment) as your CSS.

- **height** – The height of the widget, between 50 and 500.
- **image_data** – A `list` of `dicts` as specified in Reddit docs. Each `dict` represents an image and has the key `'url'` which maps to the URL of an image hosted on Reddit's servers. Images should be uploaded using *upload_image()*.

  For example:

```
[{'url': 'https://some.link',  # from upload_image()
  'width': 600, 'height': 450,
  'name': 'logo'},
 {'url': 'https://other.link',  # from upload_image()
  'width': 450, 'height': 600,
  'name': 'icon'}]
```

- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
image_paths = ['/path/to/image1.jpg', '/path/to/image2.png']
```

(continues on next page)

(continued from previous page)

```
image_urls = [widget_moderation.upload_image(img_path)
              for img_path in image_paths]
image_dicts = [{'width': 600, 'height': 450, 'name': 'logo',
                'url': image_urls[0]},
               {'width': 450, 'height': 600, 'name': 'icon',
                'url': image_urls[1]}]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
new_widget = widget_moderation.add_custom_widget('My widget',
                                                 '# Hello world!',
                                                 '/**/', 200,
                                                 image_dicts, styles)
```

add_image_widget(*short_name*, *data*, *styles*, ***other_settings*)

Add and return an *ImageWidget*.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **data** – A `list` of `dicts` as specified in Reddit docs. Each `dict` has the key `'url'` which maps to the URL of an image hosted on Reddit's servers. Images should be uploaded using *upload_image()*.

  For example:

```
[{'url': 'https://some.link',  # from upload_image()
  'width': 600, 'height': 450,
  'linkUrl': 'https://github.com/praw-dev/praw'},
 {'url': 'https://other.link',  # from upload_image()
  'width': 450, 'height': 600,
  'linkUrl': 'https://praw.readthedocs.io'}]
```

- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
image_paths = ['/path/to/image1.jpg', '/path/to/image2.png']
image_dicts = [{'width': 600, 'height': 450, 'linkUrl': '',
                'url': widget_moderation.upload_image(img_path)}
               for img_path in image_paths]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
new_widget = widget_moderation.add_image_widget('My cool pictures',
                                                image_dicts, styles)
```

add_menu(*data*, ***other_settings*)

Add and return a *Menu* widget.

Parameters **data** – A `list` of `dicts` describing menu contents, as specified in Reddit docs. As of this writing, the format is:

```
[
  {
    "text": a string no longer than 20 characters,
    "url": a valid URL
  },
```

(continues on next page)

(continued from previous page)

```
  OR

  {
    "children": [
      {
         "text": a string no longer than 20 characters,
         "url": a valid URL,
      },
      ...
     ],
    "text": a string no longer than 20 characters,
  },
  ...
]
```

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
menu_contents = [
    {'text': 'My homepage', 'url': 'https://example.com'},
    {'text': 'Python packages',
     'children': [
         {'text': 'PRAW', 'url': 'https://praw.readthedocs.io/'},
         {'text': 'requests', 'url': 'http://python-requests.org'}
     ]},
    {'text': 'Reddit homepage', 'url': 'https://reddit.com'}
]
new_widget = widget_moderation.add_menu(menu_contents)
```

add_post_flair_widget(*short_name*, *display*, *order*, *styles*, ***other_settings*)

Add and return a *PostFlairWidget*.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **display** – Display style. Either `'cloud'` or `'list'`.
- **order** – A `list` of flair template IDs. You can get all flair template IDs in a subreddit with:

  ```
  flairs = [f['id'] for f in subreddit.flair.link_templates]
  ```

- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
subreddit = reddit.subreddit('mysub')
widget_moderation = subreddit.widgets.mod
flairs = [f['id'] for f in subreddit.flair.link_templates]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
new_widget = widget_moderation.add_post_flair_widget('Some flairs',
                                                     'list',
                                                     flairs, styles)
```

add_text_area(*short_name*, *text*, *styles*, ***other_settings*)
Add and return a *TextArea* widget.

Parameters

- **short_name** – A name for the widget, no longer than 30 characters.
- **text** – The Markdown text displayed in the widget.
- **styles** – A `dict` with keys `backgroundColor` and `headerColor`, and values of hex colors. For example, `{'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}`.

Example usage:

```
widget_moderation = reddit.subreddit('mysub').widgets.mod
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
new_widget = widget_moderation.add_text_area('My cool title',
                                             '*Hello* **world**!',
                                             styles)
```

reorder(*new_order*, *section='sidebar'*)
Reorder the widgets.

Parameters

- **new_order** – A list of widgets. Represented as a `list` that contains `Widget` objects, or widget IDs as strings. These types may be mixed.
- **section** – The section to reorder. (default: `'sidebar'`)

Example usage:

```
widgets = reddit.subreddit('mysub').widgets
order = list(widgets.sidebar)
order.reverse()
widgets.mod.reorder(order)
```

upload_image(*file_path*)
Upload an image to Reddit and get the URL.

Parameters **file_path** – The path to the local file.

Returns The URL of the uploaded image as a `str`.

This method is used to upload images for widgets. For example, it can be used in conjunction with *add_image_widget()*, *add_custom_widget()*, and *add_button_widget()*.

Example usage:

```
my_sub = reddit.subreddit('my_sub')
image_url = my_sub.widgets.mod.upload_image('/path/to/image.jpg')
images = [{'width': 300, 'height': 300,
           'url': image_url, 'linkUrl': ''}]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
my_sub.widgets.mod.add_image_widget('My cool pictures', images,
                                    styles)
```

1.10.17 ThingModerationMixin

class praw.models.reddit.mixins.**ThingModerationMixin**

Provides moderation methods for Comments and Submissions.

__init__

Initialize self. See help(type(self)) for accurate signature.

approve()

Approve a *Comment* or *Submission*.

Approving a comment or submission reverts a removal, resets the report counter, adds a green check mark indicator (only visible to other moderators) on the website view, and sets the `approved_by` attribute to the authenticated user.

Example usage:

```
# approve a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.approve()
# approve a submission:
submission = reddit.submission(id='5or86n')
submission.mod.approve()
```

distinguish(*how='yes'*, *sticky=False*)

Distinguish a *Comment* or *Submission*.

Parameters

- **how** – One of 'yes', 'no', 'admin', 'special'. 'yes' adds a moderator level distinguish. 'no' removes any distinction. 'admin' and 'special' require special user privileges to use.
- **sticky** – Comment is stickied if `True`, placing it at the top of the comment page regardless of score. If thing is not a top-level comment, this parameter is silently ignored.

Example usage:

```
# distinguish and sticky a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.distinguish(how='yes', sticky=True)
# undistinguish a submission:
submission = reddit.submission(id='5or86n')
submission.mod.distinguish(how='no')
```

See also *undistinguish()*

ignore_reports()

Ignore future reports on a *Comment* or *Submission*.

Calling this method will prevent future reports on this Comment or Submission from both triggering notifications and appearing in the various moderation listings. The report count will still increment on the Comment or Submission.

Example usage:

```
# ignore future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.ignore_reports()
# ignore future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.ignore_reports()
```

See also *unignore_reports()*

lock()
Lock a *Comment* or *Submission*.

Example usage:

```
# lock a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.lock()
# lock a submission:
submission = reddit.submission(id='5or86n')
submission.mod.lock()
```

See also *unlock()*

remove(*spam=False*, *mod_note=''*, *reason_id=None*)
Remove a *Comment* or *Submission*.

Parameters

- **mod_note** – A message for the other moderators.
- **spam** – When True, use the removal to help train the Subreddit's spam filter (default: False).
- **reason_id** – The removal reason ID.

If either `reason_id` or `mod_note` are provided, a second API call is made to add the removal reason.

Example usage:

```
# remove a comment and mark as spam:
comment = reddit.comment('dkk4qjd')
comment.mod.remove(spam=True)
# remove a submission
submission = reddit.submission(id='5or86n')
submission.mod.remove()
# remove a submission with a removal reason
reason = reddit.subreddit.mod.removal_reasons["110ni21zo23ql"]
submission = reddit.submission(id="5or86n")
submission.mod.remove(reason_id=reason.id)
```

send_removal_message(*message*, *title='ignored'*, *type='public'*)
Send a removal message for a *Comment* or *Submission*.

Warning: The object has to be removed before giving it a removal reason. Remove the object with *remove()*. Trying to add a removal reason without removing the object will result in `prawcore.exceptions.BadRequest` being thrown.

Reddit adds human-readable information about the object to the message.

Parameters

- **type** – One of 'public', 'private', 'private_exposed'. 'public' leaves a stickied comment on the post. 'private' sends a Modmail message with hidden username. 'private_exposed' sends a Modmail message without hidden username.
- **title** – The short reason given in the message. (Ignored if type is 'public'.)
- **message** – The body of the message.

If `type` is 'public', the new *Comment* is returned.

undistinguish()
Remove mod, admin, or special distinguishing from an object.

Also unstickies the object if applicable.

Example usage:

```
# undistinguish a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.undistinguish()
# undistinguish a submission:
submission = reddit.submission(id='5or86n')
submission.mod.undistinguish()
```

See also *distinguish()*

unignore_reports()
Resume receiving future reports on a Comment or Submission.

Future reports on this *Comment* or *Submission* will cause notifications, and appear in the various moderation listings.

Example usage:

```
# accept future reports on a comment:
comment = reddit.comment('dkk4qjd')
comment.mod.unignore_reports()
# accept future reports on a submission:
submission = reddit.submission(id='5or86n')
submission.mod.unignore_reports()
```

See also *ignore_reports()*

unlock()
Unlock a *Comment* or *Submission*.

Example usage:

unlock a comment: comment = reddit.comment('dkk4qjd') comment.mod.unlock() # unlock a submission: submission = reddit.submission(id='5or86n') submission.mod.unlock()

See also *lock()*

1.10.18 WidgetModeration

class praw.models.**WidgetModeration**(*widget*, *subreddit*, *reddit*)
Class for moderating a particular widget.

Example usage:

```
widget = reddit.subreddit('my_sub').widgets.sidebar[0]
widget.mod.update(shortName='My new title')
widget.mod.delete()
```

__init__(*widget*, *subreddit*, *reddit*)
Initialize the widget moderation object.

delete()
Delete the widget.

Example usage:

```
widget.mod.delete()
```

update(***kwargs*)
Update the widget. Returns the updated widget.

Parameters differ based on the type of widget. See Reddit documentation or the document of the particular type of widget. For example, update a text widget like so:

```
text_widget.mod.update(shortName='New text area', text='Hello!')
```

Note: Most parameters follow the `lowerCamelCase` convention. When in doubt, check the Reddit documentation linked above.

1.10.19 WikiPageModeration

class praw.models.reddit.wikipage.**WikiPageModeration**(*wikipage: _WikiPage*)
Provides a set of moderation functions for a WikiPage.

For example, to add `spez` as an editor on the wikipage `praw_test` try:

```
reddit.subreddit('test').wiki['praw_test'].mod.add('spez')
```

__init__(*wikipage: _WikiPage*)
Create a WikiPageModeration instance.

Parameters **wikipage** – The wikipage to moderate.

add(*redditor: praw.models.reddit.redditor.Redditor*)
Add an editor to this WikiPage.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

To add `'spez'` as an editor on the wikipage `'praw_test'` try:

```
reddit.subreddit('test').wiki['praw_test'].mod.add('spez')
```

remove(*redditor: praw.models.reddit.redditor.Redditor*)
Remove an editor from this WikiPage.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

To remove `'spez'` as an editor on the wikipage `'praw_test'` try:

```
reddit.subreddit('test').wiki['praw_test'].mod.remove('spez')
```

settings() → Dict[str, Any]
Return the settings for this WikiPage.

update(*listed: bool*, *permlevel: int*, ***other_settings*) → Dict[str, Any]
Update the settings for this WikiPage.

Parameters

- **listed** – (boolean) Show this page on page list.

- **permlevel** – (int) Who can edit this page? (0) use subreddit wiki permissions, (1) only approved wiki contributors for this page may edit (see *WikiPageModeration.add()*), (2) only mods may edit and view
- **other_settings** – Additional keyword arguments to pass.

Returns The updated WikiPage settings.

To set the wikipage 'praw_test' in '/r/test' to mod only and disable it from showing in the page list, try:

```
reddit.subreddit('test').wiki['praw_test'].mod.update(listed=False,
                                                      permlevel=2)
```

1.10.20 ContributorRelationship

class praw.models.reddit.subreddit.**ContributorRelationship**(*subreddit*, *relationship*)

Provides methods to interact with a Subreddit's contributors.

Contributors are also known as approved submitters.

Contributors of a subreddit can be iterated through like so:

```
for contributor in reddit.subreddit('redditdev').contributor():
    print(contributor)
```

__call__(*redditor=None*, ***generator_kwargs*)

Return a *ListingGenerator* for Redditors in the relationship.

Parameters **redditor** – When provided, yield at most a single *Redditor* instance. This is useful to confirm if a relationship exists, or to fetch the metadata associated with a particular relationship (default: None).

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

__init__(*subreddit*, *relationship*)

Create a SubredditRelationship instance.

Parameters

- **subreddit** – The subreddit for the relationship.
- **relationship** – The name of the relationship.

add(*redditor*, ***other_settings*)

Add `redditor` to this relationship.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

leave()

Abdicate the contributor position.

remove(*redditor*)

Remove `redditor` from this relationship.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

1.10.21 ModeratorRelationship

class praw.models.reddit.subreddit.**ModeratorRelationship**(*subreddit*, *relationship*)
Provides methods to interact with a Subreddit's moderators.

Moderators of a subreddit can be iterated through like so:

```
for moderator in reddit.subreddit('redditdev').moderator():
    print(moderator)
```

__call__(*redditor=None*)
Return a list of Redditors who are moderators.

Parameters **redditor** – When provided, return a list containing at most one *Redditor* instance. This is useful to confirm if a relationship exists, or to fetch the metadata associated with a particular relationship (default: None).

Note: Unlike other relationship callables, this relationship is not paginated. Thus it simply returns the full list, rather than an iterator for the results.

To be used like:

```
moderators = reddit.subreddit('nameofsub').moderator()
```

For example, to list the moderators along with their permissions try:

```
for moderator in reddit.subreddit('SUBREDDIT').moderator():
    print('{}: {}'.format(moderator, moderator.mod_permissions))
```

__init__(*subreddit*, *relationship*)
Create a SubredditRelationship instance.

Parameters

- **subreddit** – The subreddit for the relationship.
- **relationship** – The name of the relationship.

add(*redditor*, *permissions=None*, ***other_settings*)
Add or invite `redditor` to be a moderator of the subreddit.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant. An empty list `[]` indicates no permissions, and when not provided `None`, indicates full permissions.

An invite will be sent unless the user making this call is an admin user.

For example, to invite `'spez'` with `'posts'` and `'mail'` permissions to `r/test`, try:

```
reddit.subreddit('test').moderator.add('spez', ['posts', 'mail'])
```

invite(*redditor*, *permissions=None*, ***other_settings*)
Invite `redditor` to be a moderator of the subreddit.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant. An empty list `[]` indicates no permissions, and when not provided `None`, indicates full permissions.

For example, to invite `'spez'` with `posts` and `mail` permissions to `r/test`, try:

```
reddit.subreddit('test').moderator.invite('spez', ['posts', 'mail'])
```

leave()

Abdicate the moderator position (use with care).

For example:

```
reddit.subreddit('subredditname').moderator.leave()
```

remove(*redditor*)

Remove `redditor` from this relationship.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

remove_invite(*redditor*)

Remove the moderator invite for `redditor`.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

For example:

```
reddit.subreddit('subredditname').moderator.remove_invite('spez')
```

update(*redditor*, *permissions=None*)

Update the moderator permissions for `redditor`.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant. An empty list `[]` indicates no permissions, and when not provided, `None`, indicates full permissions.

For example, to add all permissions to the moderator, try:

```
subreddit.moderator.update('spez')
```

To remove all permissions from the moderator, try:

```
subreddit.moderator.update('spez', [])
```

update_invite(*redditor*, *permissions=None*)

Update the moderator invite permissions for `redditor`.

Parameters

- **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.
- **permissions** – When provided (not `None`), permissions should be a list of strings specifying which subset of permissions to grant. An empty list `[]` indicates no permissions, and when not provided, `None`, indicates full permissions.

For example, to grant the `flair` and `mail` permissions to the moderator invite, try:

```
subreddit.moderator.update_invite('spez', ['flair', 'mail'])
```

1.10.22 SubredditRelationship

class praw.models.reddit.subreddit.**SubredditRelationship**(*subreddit*, *relationship*)

Represents a relationship between a redditor and subreddit.

Instances of this class can be iterated through in order to discover the Redditors that make up the relationship.

For example, banned users of a subreddit can be iterated through like so:

```
for ban in reddit.subreddit('redditdev').banned():
    print('{}: {}'.format(ban, ban.note))
```

__call__(*redditor=None*, ***generator_kwargs*)

Return a *ListingGenerator* for Redditors in the relationship.

Parameters **redditor** – When provided, yield at most a single *Redditor* instance. This is useful to confirm if a relationship exists, or to fetch the metadata associated with a particular relationship (default: None).

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

__init__(*subreddit*, *relationship*)

Create a SubredditRelationship instance.

Parameters

- **subreddit** – The subreddit for the relationship.
- **relationship** – The name of the relationship.

add(*redditor*, ***other_settings*)

Add `redditor` to this relationship.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

remove(*redditor*)

Remove `redditor` from this relationship.

Parameters **redditor** – A redditor name (e.g., `'spez'`) or *Redditor* instance.

1.10.23 SubredditFilters

class praw.models.reddit.subreddit.**SubredditFilters**(*subreddit*)

Provide functions to interact with the special Subreddit's filters.

Members of this class should be utilized via `Subreddit.filters`. For example, to add a filter, run:

```
reddit.subreddit('all').filters.add('subreddit_name')
```

__init__(*subreddit*)

Create a SubredditFilters instance.

Parameters **subreddit** – The special subreddit whose filters to work with.

As of this writing filters can only be used with the special subreddits `all` and `mod`.

__iter__()
Iterate through the special subreddit's filters.

This method should be invoked as:

```
for subreddit in reddit.subreddit('NAME').filters:
    ...
```

add(*subreddit*)
Add `subreddit` to the list of filtered subreddits.

Parameters **subreddit** – The subreddit to add to the filter list.

Items from subreddits added to the filtered list will no longer be included when obtaining listings for `r/all`.

Alternatively, you can filter a subreddit temporarily from a special listing in a manner like so:

```
reddit.subreddit('all-redditdev-learnpython')
```

Raises `prawcore.NotFound` when calling on a non-special subreddit.

remove(*subreddit*)
Remove `subreddit` from the list of filtered subreddits.

Parameters **subreddit** – The subreddit to remove from the filter list.

Raises `prawcore.NotFound` when calling on a non-special subreddit.

1.10.24 SubredditQuarantine

class praw.models.reddit.subreddit.**SubredditQuarantine**(*subreddit*)
Provides subreddit quarantine related methods.

To opt-in into a quarantined subreddit:

```
reddit.subreddit('test').quaran.opt_in()
```

__init__(*subreddit*)
Create a SubredditQuarantine instance.

Parameters **subreddit** – The subreddit associated with the quarantine.

opt_in()
Permit your user access to the quarantined subreddit.

Usage:

```
subreddit = reddit.subreddit('QUESTIONABLE')
next(subreddit.hot())  # Raises prawcore.Forbidden

subreddit.quaran.opt_in()
next(subreddit.hot())  # Returns Submission
```

opt_out()
Remove access to the quarantined subreddit.

Usage:

```
subreddit = reddit.subreddit('QUESTIONABLE')
next(subreddit.hot())  # Returns Submission

subreddit.quaran.opt_out()
next(subreddit.hot())  # Raises prawcore.Forbidden
```

1.10.25 SubredditStream

class praw.models.reddit.subreddit.**SubredditStream**(*subreddit*)

Provides submission and comment streams.

__init__(*subreddit*)

Create a SubredditStream instance.

Parameters **subreddit** – The subreddit associated with the streams.

comments(***stream_options*)

Yield new comments as they become available.

Comments are yielded oldest first. Up to 100 historical comments will initially be returned.

Keyword arguments are passed to *stream_generator()*.

Note: While PRAW tries to catch all new comments, some high-volume streams, especially the r/all stream, may drop some comments.

For example, to retrieve all new comments made to the `iama` subreddit, try:

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

To only retreive new submissions starting when the stream is created, pass *skip_existing=True*:

```
subreddit = reddit.subreddit('iama')
for comment in subreddit.stream.comments(skip_existing=True):
    print(comment)
```

submissions(***stream_options*)

Yield new submissions as they become available.

Submissions are yielded oldest first. Up to 100 historical submissions will initially be returned.

Keyword arguments are passed to *stream_generator()*.

Note: While PRAW tries to catch all new submissions, some high-volume streams, especially the r/all stream, may drop some submissions.

For example to retrieve all new submissions made to all of Reddit, try:

```
for submission in reddit.subreddit('all').stream.submissions():
    print(submission)
```

1.10.26 SubredditModerationStream

class praw.models.reddit.subreddit.**SubredditModerationStream**(*subreddit*)
Provides moderator streams.

__init__(*subreddit*)
Create a SubredditModerationStream instance.

Parameters **subreddit** – The moderated subreddit associated with the streams.

edited(*only=None*, ***stream_options*)
Yield edited comments and submissions as they become available.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Keyword arguments are passed to *stream_generator()*.

For example, to retrieve all new edited submissions/comments made to all moderated subreddits, try:

```
for item in reddit.subreddit('mod').mod.stream.edited():
    print(item)
```

log(*action=None*, *mod=None*, ***stream_options*)
Yield moderator log entries as they become available.

Parameters

- **action** – If given, only return log entries for the specified action.
- **mod** – If given, only return log entries for actions made by the passed in Redditor.

For example, to retrieve all new mod actions made to all moderated subreddits, try:

```
for log in reddit.subreddit('mod').mod.stream.log():
    print("Mod: {}, Subreddit: {}".format(log.mod, log.subreddit))
```

modmail_conversations(*other_subreddits=None*, *sort=None*, *state=None*, ***stream_options*)
Yield unread new modmail messages as they become available.

Parameters

- **other_subreddits** – A list of *Subreddit* instances for which to fetch conversations (default: None).
- **sort** – Can be one of: mod, recent, unread, user (default: recent).
- **state** – Can be one of: all, archived, highlighted, inprogress, mod, new, notifications, (default: all). "all" does not include internal or archived conversations.

Keyword arguments are passed to *stream_generator()*.

To print new mail in the unread modmail queue try:

```
for message in reddit.subreddit('all').mod.stream.modmail_conversations():
    print("From: {}, To: {}".format(message.owner, message.participant))
```

modqueue(*only=None*, ***stream_options*)
Yield comments/submissions in the modqueue as they become available.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Keyword arguments are passed to *stream_generator()*.

To print all new modqueue items try:

```
for item in reddit.subreddit('mod').mod.stream.modqueue():
    print(item)
```

reports(*only=None*, ***stream_options*)

Yield reported comments and submissions as they become available.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Keyword arguments are passed to *stream_generator()*.

To print new user and mod report reasons in the report queue try:

```
for item in reddit.subreddit('mod').mod.stream.reports():
    print(item)
```

spam(*only=None*, ***stream_options*)

Yield spam comments and submissions as they become available.

Parameters **only** – If specified, one of `'comments'`, or `'submissions'` to yield only results of that type.

Keyword arguments are passed to *stream_generator()*.

To print new items in the spam queue try:

```
for item in reddit.subreddit('mod').mod.stream.spam():
    print(item)
```

unmoderated(***stream_options*)

Yield unmoderated submissions as they become available.

Keyword arguments are passed to *stream_generator()*.

To print new items in the unmoderated queue try:

```
for item in reddit.subreddit('mod').mod.stream.unmoderated():
    print(item)
```

unread(***stream_options*)

Yield unread old modmail messages as they become available.

Keyword arguments are passed to *stream_generator()*.

See `inbox` for all messages.

To print new mail in the unread modmail queue try:

```
for message in reddit.subreddit('mod').mod.stream.unread():
    print("From: {}, To: {}".format(message.author, message.dest))
```

1.10.27 SubredditStylesheet

class praw.models.reddit.subreddit.**SubredditStylesheet**(*subreddit*)

Provides a set of stylesheet functions to a Subreddit.

For example, to add the css data `.test{color:blue}` to the existing stylesheet:

```
subreddit = reddit.subreddit('SUBREDDIT')
stylesheet = subreddit.stylesheet()
stylesheet += ".test{color:blue}"
subreddit.stylesheet.update(stylesheet)
```

__call__()
Return the subreddit's stylesheet.

To be used as:

```
stylesheet = reddit.subreddit('SUBREDDIT').stylesheet()
```

__init__(*subreddit*)
Create a SubredditStylesheet instance.

Parameters **subreddit** – The subreddit associated with the stylesheet.

An instance of this class is provided as:

```
reddit.subreddit('SUBREDDIT').stylesheet
```

delete_banner()
Remove the current subreddit (redesign) banner image.

Succeeds even if there is no banner image.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_banner()
```

delete_banner_additional_image()
Remove the current subreddit (redesign) banner additional image.

Succeeds even if there is no additional image. Will also delete any configured hover image.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_banner_additional_image()
```

delete_banner_hover_image()
Remove the current subreddit (redesign) banner hover image.

Succeeds even if there is no hover image.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_banner_hover_image()
```

delete_header()
Remove the current subreddit header image.

Succeeds even if there is no header image.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_header()
```

delete_image(*name*)
Remove the named image from the subreddit.

Succeeds even if the named image does not exist.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_image('smile')
```

delete_mobile_header()
Remove the current subreddit mobile header.

Succeeds even if there is no mobile header.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_mobile_header()
```

delete_mobile_icon()
Remove the current subreddit mobile icon.

Succeeds even if there is no mobile icon.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.delete_mobile_icon()
```

update(*stylesheet*, *reason=None*)
Update the subreddit's stylesheet.

Parameters **stylesheet** – The CSS for the new stylesheet.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.update(
    'p { color: green; }', 'color text green')
```

upload(*name*, *image_path*)
Upload an image to the Subreddit.

Parameters

- **name** – The name to use for the image. If an image already exists with the same name, it will be replaced.
- **image_path** – A path to a jpeg or png image.

Returns A dictionary containing a link to the uploaded image under the key `img_src`.

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload('smile', 'img.png')
```

upload_banner(*image_path*)
Upload an image for the subreddit's (redesign) banner image.

Parameters **image_path** – A path to a jpeg or png image.

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_banner('banner.png')
```

upload_banner_additional_image(*image_path*, *align=None*)
Upload an image for the subreddit's (redesign) additional image.

Parameters

- **image_path** – A path to a jpeg or png image.
- **align** – Either `left`, `centered`, or `right`. (default: `left`).

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_banner_additional_image(
↪'banner.png')
```

upload_banner_hover_image(*image_path*)
Upload an image for the subreddit's (redesign) additional image.

Parameters **image_path** – A path to a jpeg or png image.

Fails if the Subreddit does not have an additional image defined

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_banner_hover_image('banner.png
↪')
```

upload_header(*image_path*)
Upload an image to be used as the Subreddit's header image.

Parameters **image_path** – A path to a jpeg or png image.

Returns A dictionary containing a link to the uploaded image under the key `img_src`.

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_header('header.png')
```

upload_mobile_header(*image_path*)
Upload an image to be used as the Subreddit's mobile header.

Parameters **image_path** – A path to a jpeg or png image.

Returns A dictionary containing a link to the uploaded image under the key `img_src`.

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_mobile_header(
    'header.png')
```

upload_mobile_icon(*image_path*)

Upload an image to be used as the Subreddit's mobile icon.

Parameters **image_path** – A path to a jpeg or png image.

Returns A dictionary containing a link to the uploaded image under the key `img_src`.

Raises `prawcore.TooLarge` if the overall request body is too large.

Raises *APIException* if there are other issues with the uploaded image. Unfortunately the exception info might not be very specific, so try through the website with the same image to see what the problem actually might be.

For example:

```
reddit.subreddit('SUBREDDIT').stylesheet.upload_mobile_icon(
    'icon.png')
```

1.10.28 SubredditWidgets

class praw.models.**SubredditWidgets**(*subreddit*)

Class to represent a subreddit's widgets.

Create an instance like so:

```
widgets = reddit.subreddit('redditdev').widgets
```

Data will be lazy-loaded. By default, PRAW will not request progressively loading images from Reddit. To enable this, instantiate a SubredditWidgets object, then set the attribute `progressive_images` to `True` before performing any action that would result in a network request.

```
widgets = reddit.subreddit('redditdev').widgets
widgets.progressive_images = True
for widget in widgets.sidebar:
    # do something
```

Access a subreddit's widgets with the following attributes:

```
print(widgets.id_card)
print(widgets.moderators_widget)
print(widgets.sidebar)
print(widgets.topbar)
```

The attribute *id_card* contains the subreddit's ID card, which displays information like the number of subscribers.

The attribute *moderators_widget* contains the subreddit's moderators widget, which lists the moderators of the subreddit.

The attribute *sidebar* contains a list of widgets which make up the sidebar of the subreddit.

The attribute *topbar* contains a list of widgets which make up the top bar of the subreddit.

To edit a subreddit's widgets, use *mod*. For example:

```
widgets.mod.add_text_area('My title', '**bold text**',
                          {'backgroundColor': '#FFFF66',
                           'headerColor': '#3333EE'})
```

For more information, see *SubredditWidgetsModeration*.

To edit a particular widget, use `.mod` on the widget. For example:

```
for widget in widgets.sidebar:
    widget.mod.update(shortName='Exciting new name')
```

For more information, see *WidgetModeration*.

Currently available Widgets:

- *ButtonWidget*
- *Calendar*
- *CommunityList*
- *CustomWidget*
- *IDCard*
- *ImageWidget*
- *Menu*
- *ModeratorsWidget*
- *PostFlairWidget*
- *RulesWidget*
- *TextArea*

__init__(*subreddit*)
Initialize the class.

Parameters **subreddit** – The *Subreddit* the widgets belong to.

id_card
Get this subreddit's *IDCard* widget.

items
Get this subreddit's widgets as a dict from ID to widget.

mod
Get an instance of *SubredditWidgetsModeration*.

Note: Using any of the methods of *SubredditWidgetsModeration* will likely result in the data of this *SubredditWidgets* being outdated. To re-sync, call *refresh()*.

---

moderators_widget
Get this subreddit's *ModeratorsWidget*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

refresh()
Refresh the subreddit's widgets.

By default, PRAW will not request progressively loading images from Reddit. To enable this, set the attribute `progressive_images` to `True` prior to calling `refresh()`.

```
widgets = reddit.subreddit('redditdev').widgets
widgets.progressive_images = True
widgets.refresh()
```

sidebar
Get a list of Widgets that make up the sidebar.

topbar
Get a list of Widgets that make up the top bar.

1.10.29 SubredditWiki

class praw.models.reddit.subreddit.**SubredditWiki**(*subreddit*)
Provides a set of wiki functions to a Subreddit.

__getitem__(*page_name*)
Lazily return the WikiPage for the subreddit named `page_name`.

This method is to be used to fetch a specific wikipage, like so:

```
wikipage = reddit.subreddit('iama').wiki['proof']
print(wikipage.content_md)
```

__init__(*subreddit*)
Create a SubredditWiki instance.

Parameters **subreddit** – The subreddit whose wiki to work with.

__iter__()
Iterate through the pages of the wiki.

This method is to be used to discover all wikipages for a subreddit:

```
for wikipage in reddit.subreddit('iama').wiki:
    print(wikipage)
```

create(*name*, *content*, *reason=None*, ***other_settings*)
Create a new wiki page.

Parameters

- **name** – The name of the new WikiPage. This name will be normalized.
- **content** – The content of the new WikiPage.

- **reason** – (Optional) The reason for the creation.
- **other_settings** – Additional keyword arguments to pass.

To create the wiki page `praw_test` in `r/test` try:

```
reddit.subreddit('test').wiki.create(
    'praw_test', 'wiki body text', reason='PRAW Test Creation')
```

revisions(***generator_kwargs*)

Return a *ListingGenerator* for recent wiki revisions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

To view the wiki revisions for `'praw_test'` in `r/test` try:

```
for item in reddit.subreddit('test').wiki['praw_test'].revisions():
    print(item)
```

1.10.30 ButtonWidget

class praw.models.**ButtonWidget**(*reddit*, *_data*)

Class to represent a widget containing one or more buttons.

Find an existing one:

```
button_widget = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.ButtonWidget):
        button_widget = widget
        break

for button in button_widget:
    print(button.text, button.url)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
buttons = [
    {
        'kind': 'text',
        'text': 'View source',
        'url': 'https://github.com/praw-dev/praw',
        'color': '#FF0000',
        'textColor': '#00FF00',
        'fillColor': '#0000FF',
        'hoverState': {
            'kind': 'text',
            'text': 'ecruos weiV',
            'color': '#000000',
            'textColor': '#FFFFFF',
            'fillColor': '#0000FF'
        }
    },
    {
        'kind': 'text',
        'text': 'View documentation',
```

(continues on next page)

(continued from previous page)

```
        'url': 'https://praw.readthedocs.io',
        'color': '#FFFFFF',
        'textColor': '#FFFF00',
        'fillColor': '#0000FF'
    },
]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
button_widget = widgets.mod.add_button_widget(
    'Things to click', 'Click some of these *cool* links!',
    buttons, styles)
```

For more information on creation, see *add_button_widget()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
button_widget = button_widget.mod.update(shortName='My fav buttons',
                                         styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
button_widget.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`buttons`	A `list` of *Button*s. These can also be accessed just by iterating over the *ButtonWidget* (e.g. `for button in button_widget`).
`description`	The description, in Markdown.
`description_html`	The description, in HTML.
`id`	The widget ID.
`kind`	The widget kind (always `'button'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *Subreddit* the button widget belongs to.

__contains__(*item: Any*) → bool
Test if item exists in the list.

__getitem__(*index: int*) → Any
Return the item at position index in the list.

__init__(*reddit*, *_data*)
Initialize an instance of the class.

__iter__() → Iterator[Any]
Return an iterator to the list.

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

mod
Get an instance of *WidgetModeration* for this widget.

Note: Using any of the methods of *WidgetModeration* will likely make outdated the data in the *SubredditWidgets* that this widget belongs to. To remedy this, call *refresh()*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.31 Calendar

class praw.models.**Calendar**(*reddit*, *_data*)
Class to represent a calendar widget.

Find an existing one:

```
calendar = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.Calendar):
        calendar = widget
        break

print(calendar.googleCalendarId)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
config = {'numEvents': 10,
          'showDate': True,
          'showDescription': False,
          'showLocation': False,
          'showTime': True,
          'showTitle': True}
cal_id = 'y6nm89jy427drk8l71w75w9wjn@group.calendar.google.com'
calendar = widgets.mod.add_calendar(
    'Upcoming Events', cal_id, True, config, styles)
```

For more information on creation, see *add_calendar()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
calendar = calendar.mod.update(shortName='My fav events',
                               styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
calendar.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`configuration`	A `dict` describing the calendar configuration.
`data`	A `list` of `dicts` that represent events.
`id`	The widget ID.
`kind`	The widget kind (always `'calendar'`).
`requiresSync`	A `bool`.
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *`Subreddit`* the button widget belongs to.

`__init__`(*reddit*, *_data*)
Initialize an instance of the class.

`mod`
Get an instance of *`WidgetModeration`* for this widget.

Note: Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.
- **`reddit`** – An instance of *`Reddit`*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.32 CommunityList

`class` `praw.models.`**`CommunityList`**(*reddit*, *_data*)
Class to represent a Related Communities widget.

Find an existing one:

```
community_list = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
```

(continues on next page)

(continued from previous page)

```
    if isinstance(widget, praw.models.CommunityList):
        community_list = widget
        break

print(community_list)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
subreddits = ['learnpython', reddit.subreddit('announcements')]
community_list = widgets.mod.add_community_list('Related subreddits',
                                                subreddits, styles,
                                                'description')
```

For more information on creation, see *add_community_list()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
community_list = community_list.mod.update(shortName='My fav subs',
                                           styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
community_list.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`data`	A `list` of *Subreddit*s. These can also be iterated over by iterating over the *CommunityList* (e.g. `for sub in community_list`).
`id`	The widget ID.
`kind`	The widget kind (always `'community-list'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *Subreddit* the button widget belongs to.

__contains__(*item: Any*) → bool
Test if item exists in the list.

__getitem__(*index: int*) → Any
Return the item at position index in the list.

__init__(*reddit*, *_data*)
Initialize an instance of the class.

__iter__() → Iterator[Any]
Return an iterator to the list.

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

mod
Get an instance of *WidgetModeration* for this widget.

Note: Using any of the methods of *WidgetModeration* will likely make outdated the data in the *SubredditWidgets* that this widget belongs to. To remedy this, call *refresh()*.

---

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.33 CustomWidget

class praw.models.**CustomWidget**(*reddit*, *_data*)
Class to represent a custom widget.

Find an existing one:

```
custom = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.CustomWidget):
        custom = widget
        break

print(custom.text)
print(custom.css)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
custom = widgets.mod.add_custom_widget(
    'My custom widget', '# Hello world!', '/**/', 200, [], styles)
```

For more information on creation, see *add_custom_widget()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
custom = custom.mod.update(shortName='My fav customization',
                           styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
custom.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`css`	The CSS of the widget, as a `str`.
`height`	The height of the widget, as an `int`.
`id`	The widget ID.
`imageData`	A `list` of *`ImageData`* that belong to the widget.
`kind`	The widget kind (always `'custom'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`stylesheetUrl`	A link to the widget's stylesheet.
`subreddit`	The *`Subreddit`* the button widget belongs to.
`text`	The text contents, as Markdown.
`textHtml`	The text contents, as HTML.

`__init__`(*reddit*, *_data*)
Initialize the class.

`mod`
Get an instance of *`WidgetModeration`* for this widget.

Note: Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.
- **`reddit`** – An instance of *`Reddit`*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.34 IDCard

`class` praw.models.**`IDCard`**(*reddit*, *_data*)
Class to represent an ID card widget.

```
widgets = reddit.subreddit('redditdev').widgets
id_card = widgets.id_card
print(id_card.subscribersText)
```

Update one (requires proper moderator permissions):

```
widgets.id_card.mod.update(currentlyViewingText='Bots')
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`currentlyViewingCount`	The number of Redditors viewing the subreddit.
`currentlyViewingText`	The text displayed next to the view count. For example, "users online".
`description`	The subreddit description.
`id`	The widget ID.
`kind`	The widget kind (always `'id-card'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *`Subreddit`* the button widget belongs to.
`subscribersCount`	The number of subscribers to the subreddit.
`subscribersText`	The text displayed next to the subscriber count. For example, "users subscribed".

`__init__`(*reddit*, *_data*)
Initialize an instance of the class.

`mod`
Get an instance of *`WidgetModeration`* for this widget.

> **Note:** Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.
- **`reddit`** – An instance of *`Reddit`*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.35 ImageWidget

`class` `praw.models.`**`ImageWidget`**(*reddit*, *_data*)
Class to represent an image widget.

Find an existing one:

```
image_widget = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.ImageWidget):
        image_widget = widget
        break

for image in image_widget:
    print(image.url)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
image_paths = ['/path/to/image1.jpg', '/path/to/image2.png']
image_dicts = [{'width': 600, 'height': 450, 'linkUrl': '',
                'url': widgets.mod.upload_image(img_path)}
               for img_path in image_paths]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
image_widget = widgets.mod.add_image_widget('My cool pictures',
                                            image_dicts, styles)
```

For more information on creation, see *add_image_widget()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
image_widget = image_widget.mod.update(shortName='My fav images',
                                       styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
image_widget.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`data`	A list of the *Image*s in this widget. Can be iterated over by iterating over the *ImageWidget* (e.g. `for img in image_widget`).
`id`	The widget ID.
`kind`	The widget kind (always `'image'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *Subreddit* the button widget belongs to.

__contains__(*item: Any*) → bool
Test if item exists in the list.

__getitem__(*index: int*) → Any
Return the item at position index in the list.

__init__(*reddit*, *_data*)
Initialize an instance of the class.

__iter__() → Iterator[Any]
Return an iterator to the list.

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

mod
Get an instance of *WidgetModeration* for this widget.

Note: Using any of the methods of *WidgetModeration* will likely make outdated the data in the *SubredditWidgets* that this widget belongs to. To remedy this, call *refresh()*.

---

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.36 Menu

class praw.models.**Menu**(*reddit*, *_data*)
Class to represent the top menu widget of a subreddit.

Menus can generally be found as the first item in a subreddit's top bar.

```
topbar = reddit.subreddit('redditdev').widgets.topbar
if len(topbar) > 0:
    probably_menu = topbar[0]
    assert isinstance(probably_menu, praw.models.Menu)
    for item in probably_menu:
        if isinstance(item, praw.models.Submenu):
            print(item.text)
            for child in item:
                print('\t', child.text, child.url)
        else:  # MenuLink
            print(item.text, item.url)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
menu_contents = [
    {'text': 'My homepage', 'url': 'https://example.com'},
    {'text': 'Python packages',
     'children': [
         {'text': 'PRAW', 'url': 'https://praw.readthedocs.io/'},
         {'text': 'requests', 'url': 'http://python-requests.org'}
     ]},
```

(continues on next page)

(continued from previous page)

```
    {'text': 'Reddit homepage', 'url': 'https://reddit.com'}
]
menu = widgets.mod.add_menu(menu_contents)
```

For more information on creation, see *add_menu()*.

Update one (requires proper moderator permissions):

```
menu_items = list(menu)
menu_items.reverse()
menu = menu.mod.update(data=menu_items)
```

Delete one (requires proper moderator permissions):

```
menu.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`data`	A list of the *MenuLink*s and *Submenu*s in this widget. Can be iterated over by iterating over the *Menu* (e.g. `for item in menu`).
`id`	The widget ID.
`kind`	The widget kind (always `'menu'`).
`subreddit`	The *Subreddit* the button widget belongs to.

__contains__(*item: Any*) → bool
Test if item exists in the list.

__getitem__(*index: int*) → Any
Return the item at position index in the list.

__init__(*reddit*, *_data*)
Initialize an instance of the class.

__iter__() → Iterator[Any]
Return an iterator to the list.

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

mod
Get an instance of *WidgetModeration* for this widget.

> **Note:** Using any of the methods of *WidgetModeration* will likely make outdated the data in the *SubredditWidgets* that this widget belongs to. To remedy this, call *refresh()*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.

- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.37 ModeratorsWidget

class praw.models.**ModeratorsWidget**(*reddit*, *_data*)

Class to represent a moderators widget.

```
widgets = reddit.subreddit('redditdev').widgets
print(widgets.moderators_widget)
```

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
widgets.moderators_widget.mod.update(styles=new_styles)
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`id`	The widget ID.
`kind`	The widget kind (always `'moderators'`).
`mods`	A list of the *Redditor*s that moderate the subreddit. Can be iterated over by iterating over the *ModeratorsWidget* (e.g. `for mod in widgets.moderators_widget`).
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *Subreddit* the button widget belongs to.
`totalMods`	The total number of moderators in the subreddit.

__contains__(*item: Any*) → bool

Test if item exists in the list.

__getitem__(*index: int*) → Any

Return the item at position index in the list.

__init__(*reddit*, *_data*)

Initialize the moderators widget.

__iter__() → Iterator[Any]

Return an iterator to the list.

__len__() → int

Return the number of items in the list.

mod

Get an instance of *WidgetModeration* for this widget.

> **Note:** Using any of the methods of *WidgetModeration* will likely make outdated the data in the *SubredditWidgets* that this widget belongs to. To remedy this, call *refresh()*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.38 PostFlairWidget

class praw.models.**PostFlairWidget**(*reddit*, *_data*)
Class to represent a post flair widget.

Find an existing one:

```
post_flair_widget = None
widgets = reddit.subreddit('redditdev').widgets
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.PostFlairWidget):
        post_flair_widget = widget
        break

for flair in post_flair_widget:
    print(flair)
    print(post_flair_widget.templates[flair])
```

Create one (requires proper moderator permissions):

```
subreddit = reddit.subreddit('redditdev')
widgets = subreddit.widgets
flairs = [f['id'] for f in subreddit.flair.link_templates]
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
post_flair = widgets.mod.add_post_flair_widget('Some flairs', 'list',
                                               flairs, styles)
```

For more information on creation, see *add_post_flair_widget()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
post_flair = post_flair.mod.update(shortName='My fav flairs',
                                   styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
post_flair.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`display`	The display style of the widget, either `'cloud'` or `'list'`.
`id`	The widget ID.
`kind`	The widget kind (always `'post-flair'`).
`order`	A list of the flair IDs in this widget. Can be iterated over by iterating over the *`PostFlairWidget`* (e.g. `for flair_id in post_flair`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *`Subreddit`* the button widget belongs to.
`templates`	A `dict` that maps flair IDs to `dicts` that describe flairs.

`__contains__`(*item: Any*) → bool
Test if item exists in the list.

`__getitem__`(*index: int*) → Any
Return the item at position index in the list.

`__init__`(*reddit*, *_data*)
Initialize an instance of the class.

`__iter__`() → Iterator[Any]
Return an iterator to the list.

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

`mod`
Get an instance of *`WidgetModeration`* for this widget.

Note: Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.
- **`reddit`** – An instance of *`Reddit`*.

1.10.39 RulesWidget

`class` `praw.models.`**`RulesWidget`**(*reddit*, *_data*)
Class to represent a rules widget.

```
widgets = reddit.subreddit('redditdev').widgets
rules_widget = None
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.RulesWidget):
        rules_widget = widget
        break
from pprint import pprint; pprint(rules_widget.data)
```

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
rules_widget.mod.update(display='compact', shortName='The LAWS',
                        styles=new_styles)
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`data`	A list of the subreddit rules. Can be iterated over by iterating over the *`RulesWidget`* (e.g. `for rule in rules_widget`).
`display`	The display style of the widget, either `'full'` or `'compact'`.
`id`	The widget ID.
`kind`	The widget kind (always `'subreddit-rules'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *`Subreddit`* the button widget belongs to.

`__contains__`(*item: Any*) → bool
Test if item exists in the list.

`__getitem__`(*index: int*) → Any
Return the item at position index in the list.

`__init__`(*reddit*, *_data*)
Initialize the rules widget.

`__iter__`() → Iterator[Any]
Return an iterator to the list.

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

`mod`
Get an instance of *`WidgetModeration`* for this widget.

> **Note:** Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.

- **reddit** – An instance of *Reddit*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.40 TextArea

class praw.models.**TextArea**(*reddit*, *_data*)

Class to represent a text area widget.

Find a text area in a subreddit:

```
widgets = reddit.subreddit('redditdev').widgets
text_area = None
for widget in widgets.sidebar:
    if isinstance(widget, praw.models.TextArea):
        text_area = widget
        break
print(text_area.text)
```

Create one (requires proper moderator permissions):

```
widgets = reddit.subreddit('redditdev').widgets
styles = {'backgroundColor': '#FFFF66', 'headerColor': '#3333EE'}
text_area = widgets.mod.add_text_area('My cool title',
                                      '*Hello* **world**!',
                                      styles)
```

For more information on creation, see *add_text_area()*.

Update one (requires proper moderator permissions):

```
new_styles = {'backgroundColor': '#FFFFFF', 'headerColor': '#FF9900'}
text_area = text_area.mod.update(shortName='My fav text',
                                 styles=new_styles)
```

Delete one (requires proper moderator permissions):

```
text_area.mod.delete()
```

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`id`	The widget ID.
`kind`	The widget kind (always `'textarea'`).
`shortName`	The short name of the widget.
`styles`	A `dict` with the keys `'backgroundColor'` and `'headerColor'`.
`subreddit`	The *`Subreddit`* the button widget belongs to.
`text`	The widget's text, as Markdown.
`textHtml`	The widget's text, as HTML.

`__init__`(*reddit*, *_data*)
Initialize an instance of the class.

`mod`
Get an instance of *`WidgetModeration`* for this widget.

Note: Using any of the methods of *`WidgetModeration`* will likely make outdated the data in the *`SubredditWidgets`* that this widget belongs to. To remedy this, call *`refresh()`*.

`classmethod parse`(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **`data`** – The structured data.
- **`reddit`** – An instance of *`Reddit`*.

Note: This list of attributes is not complete. PRAW dynamically provides the attributes that Reddit returns via the API. Because those attributes are subject to change on Reddit's end, PRAW makes no effort to document them, other than to instruct you on how to discover what is available. See *Determine Available Attributes of an Object* for detailed information.

1.10.41 Auth

`class` `praw.models.`**`Auth`**(*reddit: Reddit*, *_data: Optional[Dict[str, Any]]*)
Auth provides an interface to Reddit's authorization.

`__init__`(*reddit: Reddit*, *_data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **`reddit`** – An instance of *`Reddit`*.

`authorize`(*code: str*) → Optional[str]
Complete the web authorization flow and return the refresh token.

Parameters **`code`** – The code obtained through the request to the redirect uri.

Returns The obtained refresh token, if available, otherwise `None`.

The session's active authorization will be updated upon success.

`implicit`(*access_token: str*, *expires_in: int*, *scope: str*) → None
Set the active authorization to be an implicit authorization.

Parameters

- **access_token** – The access_token obtained from Reddit's callback.
- **expires_in** – The number of seconds the `access_token` is valid for. The origin of this value was returned from Reddit's callback. You may need to subtract an offset before passing in this number to account for a delay between when Reddit prepared the response, and when you make this function call.
- **scope** – A space-delimited string of Reddit OAuth2 scope names as returned from Reddit's callback.

Raise *InvalidImplicitAuth* if *Reddit* was initialized for a non-installed application type.

limits

Return a dictionary containing the rate limit info.

The keys are:

Remaining The number of requests remaining to be made in the current rate limit window.

Reset_timestamp A unix timestamp providing an upper bound on when the rate limit counters will reset.

Used The number of requests made in the current rate limit window.

All values are initially `None` as these values are set in response to issued requests.

The `reset_timestamp` value is an upper bound as the real timestamp is computed on Reddit's end in preparation for sending the response. This value may change slightly within a given window due to slight changes in response times and rounding.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

scopes() → Set[str]

Return a set of scopes included in the current authorization.

For read-only authorizations this should return `{'*'}`.

url(*scopes: List[str]*, *state: str*, *duration: str = 'permanent'*, *implicit: bool = False*) → str

Return the URL used out-of-band to grant access to your application.

Parameters

- **scopes** – A list of OAuth scopes to request authorization for.
- **state** – A string that will be reflected in the callback to `redirect_uri`. This value should be temporarily unique to the client for whom the URL was generated for.
- **duration** – Either `permanent` or `temporary` (default: permanent). `temporary` authorizations generate access tokens that last only 1 hour. `permanent` authorizations additionally generate a refresh token that can be indefinitely used to generate new hour-long access tokens. This value is ignored when `implicit=True`.
- **implicit** – For **installed** applications, this value can be set to use the implicit, rather than the code flow. When True, the `duration` argument has no effect as only temporary tokens can be retrieved.

1.10.42 Button

class praw.models.**Button**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Class to represent a single button inside a *ButtonWidget*.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`color`	The hex color used to outline the button.
`height`	Image height. Only present on image buttons.
`hoverState`	A `dict` describing the state of the button when hovered over. Optional.
`kind`	Either `'text'` or `'image'`.
`linkUrl`	A link that can be visited by clicking the button. Only present on image buttons.
`text`	The text displayed on the button.
`url`	If the button is a text button, a link that can be visited by clicking the button. If the button is an image button, the URL of a Reddit-hosted image.
`width`	Image width. Only present on image buttons.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.43 CommentForest

class praw.models.comment_forest.**CommentForest**(*submission: Submission, comments: Optional[List[Comment]] = None*)

A forest of comments starts with multiple top-level comments.

Each of these comments can be a tree of replies.

__getitem__(*index: int*)

Return the comment at position `index` in the list.

This method is to be used like an array access, such as:

```
first_comment = submission.comments[0]
```

Alternatively, the presence of this method enables one to iterate over all top_level comments, like so:

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

__init__(*submission: Submission, comments: Optional[List[Comment]] = None*)

Initialize a CommentForest instance.

Parameters

- **submission** – An instance of *Subreddit* that is the parent of the comments.
- **comments** – Initialize the Forest with a list of comments (default: None).

__len__() → int
Return the number of top-level comments in the forest.

list() → Union[Comment, praw.models.reddit.more.MoreComments]
Return a flattened list of all Comments.

This list may contain *MoreComments* instances if *replace_more()* was not called first.

replace_more(*limit: int = 32*, *threshold: int = 0*) → List[praw.models.reddit.more.MoreComments]
Update the comment forest by resolving instances of MoreComments.

Parameters

- **limit** – The maximum number of *MoreComments* instances to replace. Each replacement requires 1 API request. Set to `None` to have no limit, or to `0` to remove all *MoreComments* instances without additional requests (default: 32).
- **threshold** – The minimum number of children comments a *MoreComments* instance must have in order to be replaced. *MoreComments* instances that represent "continue this thread" links unfortunately appear to have 0 children. (default: 0).

Returns A list of *MoreComments* instances that were not replaced.

For example, to replace up to 32 *MoreComments* instances of a submission try:

```
submission = reddit.submission('3hahrw')
submission.comments.replace_more()
```

Alternatively, to replace *MoreComments* instances within the replies of a single comment try:

```
comment = reddit.comment('d8r4im1')
comment.refresh()
comment.replies.replace_more()
```

Note: This method can take a long time as each replacement will discover at most 20 new *Comment* or *MoreComments* instances. As a result, consider looping and handling exceptions until the method returns successfully. For example:

```
while True:
    try:
        submission.comments.replace_more()
        break
    except PossibleExceptions:
        print('Handling replace_more exception')
        sleep(1)
```

1.10.44 CommentHelper

class praw.models.listing.mixins.subreddit.**CommentHelper**(*subreddit: Subreddit*)
Provide a set of functions to interact with a subreddit's comments.

__call__(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for the Subreddit's comments.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method should be used in a way similar to the example below:

```
for comment in reddit.subreddit('redditdev').comments(limit=25):
    print(comment.author)
```

__init__(*subreddit: Subreddit*)
Initialize a CommentHelper instance.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.45 Config

class praw.config.**Config**(*site_name: str*, *config_interpolation: Optional[str] = None*, ***settings*)
A class containing the configuration for a reddit site.

__init__(*site_name: str*, *config_interpolation: Optional[str] = None*, ***settings*)
Initialize a Config instance.

short_url
Return the short url or raise a ClientException when not set.

1.10.46 DomainListing

class praw.models.**DomainListing**(*reddit: Reddit*, *domain: str*)
Provide a set of functions to interact with domain listings.

__init__(*reddit: Reddit*, *domain: str*)
Initialize a DomainListing instance.

Parameters

- **reddit** – An instance of Reddit.
- **domain** – The domain for which to obtain listings.

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

hot(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

new(***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

random_rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for random rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get random rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').random_rising():
    print(submission.title)
```

rising(***generator_kwargs*) → Generator[[Submission, None], None]
Return a *ListingGenerator* for rising submissions.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

For example, to get rising submissions for subreddit `r/test`:

```
for submission in reddit.subreddit('test').rising():
    print(submission.title)
```

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise ValueError if time_filter is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

1.10.47 Emoji

class praw.models.reddit.emoji.**Emoji**(*reddit: Reddit*, *subreddit: Subreddit*, *name: str*, *_data: Optional[Dict[str, Any]] = None*)
An individual Emoji object.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
mod_flair_only	Whether the emoji is restricted for mod use only.
name	The name of the emoji.
post_flair_allowed	Whether the emoji may appear in post flair.
url	The URL of the emoji image.
user_flair_allowed	Whether the emoji may appear in user flair.

__init__(*reddit: Reddit*, *subreddit: Subreddit*, *name: str*, *_data: Optional[Dict[str, Any]] = None*)
Construct an instance of the Emoji object.

delete()
Delete an emoji from this subreddit by Emoji.

To delete 'test' as an emoji on the subreddit 'praw_test' try:

```
reddit.subreddit('praw_test').emoji['test'].delete()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of cls from data.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

update(*mod_flair_only: Optional[bool] = None*, *post_flair_allowed: Optional[bool] = None*, *user_flair_allowed: Optional[bool] = None*)
Update the permissions of an emoji in this subreddit.

Parameters

- **mod_flair_only** – (boolean) Indicate whether the emoji is restricted to mod use only. Respects pre-existing settings if not provided.
- **post_flair_allowed** – (boolean) Indicate whether the emoji may appear in post flair. Respects pre-existing settings if not provided.
- **user_flair_allowed** – (boolean) Indicate whether the emoji may appear in user flair. Respects pre-existing settings if not provided.

Note: In order to retain pre-existing values for those that are not explicitly passed, a network request is issued. To avoid that network request, explicitly provide all values.

To restrict the emoji `test` in subreddit `wowemoji` to mod use only, try:

```
reddit.subreddit("wowemoji").emoji["test"].update(mod_flair_only=True)
```

1.10.48 ListingGenerator

class praw.models.**ListingGenerator**(*reddit: Reddit*, *url: str*, *limit: int = 100*, *params: Optional[Dict[str, str]] = None*)
Instances of this class generate *RedditBase* instances.

Warning: This class should not be directly utilized. Instead you will find a number of methods that return instances of the class:

http://praw.readthedocs.io/en/latest/search.html?q=ListingGenerator

__init__(*reddit: Reddit*, *url: str*, *limit: int = 100*, *params: Optional[Dict[str, str]] = None*)
Initialize a ListingGenerator instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **url** – A URL returning a reddit listing.
- **limit** – The number of content entries to fetch. If `limit` is None, then fetch as many entries as possible. Most of reddit's listings contain a maximum of 1000 items, and are returned 100 at a time. This class will automatically issue all necessary requests (default: 100).
- **params** – A dictionary containing additional query string parameters to send with the request.

__iter__() → Iterator[Any]
Permit ListingGenerator to operate as an iterator.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.49 Image

class praw.models.**Image**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Class to represent an image that's part of a *ImageWidget*.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
height	Image height.
linkUrl	A link that can be visited by clicking the image.
url	The URL of the (Reddit-hosted) image.
width	Image width.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any

Return an instance of cls from data.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.50 ImageData

class praw.models.**ImageData**(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Class for image data that's part of a *CustomWidget*.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
height	The image height.
name	The image name.
url	The URL of the image on Reddit's servers.
width	The image width.

__init__(*reddit: Reddit, _data: Optional[Dict[str, Any]]*)

Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.51 MenuLink

class praw.models.**MenuLink**(*reddit: Reddit*, *_data: Optional[Dict[str, Any]]*)
Class to represent a single link inside a menu or submenu.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
`text`	The text of the menu link.
`url`	The URL that the menu item links to.

__init__(*reddit: Reddit*, *_data: Optional[Dict[str, Any]]*)
Initialize a PRAWModel instance.

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.52 Modmail

class praw.models.reddit.subreddit.**Modmail**(*subreddit*)
Provides modmail functions for a subreddit.

For example, to send a new modmail from the subreddit `r/test` to user `u/spez` with the subject `test` along with a message body of `hello`:

```
reddit.subreddit('test').modmail.create('test', 'hello', 'spez')
```

__call__(*id=None*, *mark_read=False*)
Return an individual conversation.

Parameters

- **id** – A reddit base36 conversation ID, e.g., `2gmz`.
- **mark_read** – If True, conversation is marked as read (default: False).

For example:

```
reddit.subreddit('redditdev').modmail('2gmz', mark_read=True)
```

To print all messages from a conversation as Markdown source:

```
conversation = reddit.subreddit('redditdev').modmail('2gmz', mark_read=True)
for message in conversation.messages:
    print(message.body_markdown)
```

`ModmailConversation.user` is a special instance of *`Redditor`* with extra attributes describing the non-moderator user's recent posts, comments, and modmail messages within the subreddit, as well as information on active bans and mutes. This attribute does not exist on internal moderator discussions.

For example, to print the user's ban status:

```
conversation = reddit.subreddit('redditdev').modmail('2gmz', mark_read=True)
print(conversation.user.ban_status)
```

To print a list of recent submissions by the user:

```
conversation = reddit.subreddit('redditdev').modmail('2gmz', mark_read=True)
print(conversation.user.recent_posts)
```

`__init__`(*subreddit*)
Construct an instance of the Modmail object.

`bulk_read`(*other_subreddits=None*, *state=None*)
Mark conversations for subreddit(s) as read.

Due to server-side restrictions, 'all' is not a valid subreddit for this method. Instead, use *`subreddits()`* to get a list of subreddits using the new modmail.

Parameters

- **`other_subreddits`** – A list of *`Subreddit`* instances for which to mark conversations (default: None).
- **`state`** – Can be one of: all, archived, highlighted, inprogress, mod, new, notifications, (default: all). "all" does not include internal or archived conversations.

Returns A list of *`ModmailConversation`* instances that were marked read.

For example, to mark all notifications for a subreddit as read:

```
subreddit = reddit.subreddit('redditdev')
subreddit.modmail.bulk_read(state='notifications')
```

`conversations`(*after=None*, *limit=None*, *other_subreddits=None*, *sort=None*, *state=None*)
Generate *`ModmailConversation`* objects for subreddit(s).

Parameters

- **`after`** – A base36 modmail conversation id. When provided, the listing begins after this conversation (default: None).
- **`limit`** – The maximum number of conversations to fetch. If None, the server-side default is 25 at the time of writing (default: None).
- **`other_subreddits`** – A list of *`Subreddit`* instances for which to fetch conversations (default: None).
- **`sort`** – Can be one of: mod, recent, unread, user (default: recent).

- **state** – Can be one of: all, archived, highlighted, inprogress, mod, new, notifications, (default: all). "all" does not include internal or archived conversations.

For example:

```
conversations = reddit.subreddit('all').modmail.conversations(state='mod')
```

create(*subject*, *body*, *recipient*, *author_hidden=False*)
Create a new modmail conversation.

Parameters

- **subject** – The message subject. Cannot be empty.
- **body** – The message body. Cannot be empty.
- **recipient** – The recipient; a username or an instance of *Redditor*.
- **author_hidden** – When True, author is hidden from non-moderators (default: False).

Returns A *ModmailConversation* object for the newly created conversation.

```
subreddit = reddit.subreddit('redditdev')
redditor = reddit.redditor('bboe')
subreddit.modmail.create('Subject', 'Body', redditor)
```

subreddits()
Yield subreddits using the new modmail that the user moderates.

For example:

```
subreddits = reddit.subreddit('all').modmail.subreddits()
```

unread_count()
Return unread conversation count by conversation state.

At time of writing, possible states are: archived, highlighted, inprogress, mod, new, notifications.

Returns A dict mapping conversation states to unread counts.

For example, to print the count of unread moderator discussions:

```
subreddit = reddit.subreddit('redditdev')
unread_counts = subreddit.modmail.unread_count()
print(unread_counts['mod'])
```

1.10.53 ModmailMessage

class praw.models.**ModmailMessage**(*reddit: Reddit, _data: Dict[str, Any]*)
A class for modmail messages.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)
Initialize a RedditBase instance (or a subclass).

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.

- **reddit** – An instance of *Reddit*.

1.10.54 Preferences

class praw.models.**Preferences**(*reddit: Reddit*)

A class for Reddit preferences.

The Preferences class provides access to the Reddit preferences of the currently authenticated user.

__call__() → Dict[str, Union[int, str]]

Return the preference settings of the authenticated user as a dict.

This method is intended to be accessed as `reddit.user.preferences()` like so:

```
preferences = reddit.user.preferences()
print(preferences['show_link_flair'])
```

See https://www.reddit.com/dev/api#GET_api_v1_me_prefs for the list of possible values.

__init__(*reddit: Reddit*)

Create a Preferences instance.

Parameters **reddit** – The Reddit instance.

update(***preferences*)

Modify the specified settings.

Parameters

- **3rd_party_data_personalized_ads** – Allow Reddit to use data provided by third-parties to show you more relevant advertisements on Reddit (boolean).
- **3rd_party_site_data_personalized_ads** – Allow personalization of advertisements using third-party website data (boolean).
- **3rd_party_site_data_personalized_content** – Allow personalization of content using third-party website data (boolean).
- **activity_relevant_ads** – Allow Reddit to use your activity on Reddit to show you more relevant advertisements (boolean).
- **allow_clicktracking** – Allow Reddit to log my outbound clicks for personalization (boolean).
- **beta** – I would like to beta test features for Reddit (boolean).
- **clickgadget** – Show me links I've recently viewed (boolean).
- **collapse_read_messages** – Collapse messages after I've read them (boolean).
- **compress** – Compress the link display (boolean).
- **creddit_autorenew** – Use a creddit to automatically renew my gold if it expires (boolean).
- **default_comment_sort** – Default comment sort (one of `'confidence'`, `'top'`, `'new'`, `'controversial'`, `'old'`, `'random'`, `'qa'`, `'live'`).
- **domain_details** – Show additional details in the domain text when available, such as the source subreddit or the content author's url/name (boolean).
- **email_digests** – Send email digests (boolean).
- **email_messages** – Send messages as emails (boolean).

- **email_unsubscribe_all** – Unsubscribe from all emails (boolean).
- **enable_default_themes** – Use reddit theme (boolean).
- **g** – Location (one of `'GLOBAL'`, `'AR'`, `'AU'`, `'BG'`, `'CA'`, `'CL'`, `'CO'`, `'CZ'`, `'FI'`, `'GB'`, `'GR'`, `'HR'`, `'HU'`, `'IE'`, `'IN'`, `'IS'`, `'JP'`, `'MX'`, `'MY'`, `'NZ'`, `'PH'`, `'PL'`, `'PR'`, `'PT'`, `'RO'`, `'RS'`, `'SE'`, `'SG'`, `'TH'`, `'TR'`, `'TW'`, `'US'`, `'US_AK'`, `'US_AL'`, `'US_AR'`, `'US_AZ'`, `'US_CA'`, `'US_CO'`, `'US_CT'`, `'US_DC'`, `'US_DE'`, `'US_FL'`, `'US_GA'`, `'US_HI'`, `'US_IA'`, `'US_ID'`, `'US_IL'`, `'US_IN'`, `'US_KS'`, `'US_KY'`, `'US_LA'`, `'US_MA'`, `'US_MD'`, `'US_ME'`, `'US_MI'`, `'US_MN'`, `'US_MO'`, `'US_MS'`, `'US_MT'`, `'US_NC'`, `'US_ND'`, `'US_NE'`, `'US_NH'`, `'US_NJ'`, `'US_NM'`, `'US_NV'`, `'US_NY'`, `'US_OH'`, `'US_OK'`, `'US_OR'`, `'US_PA'`, `'US_RI'`, `'US_SC'`, `'US_SD'`, `'US_TN'`, `'US_TX'`, `'US_UT'`, `'US_VA'`, `'US_VT'`, `'US_WA'`, `'US_WI'`, `'US_WV'`, `'US_WY'`).
- **hide_ads** – Hide ads (boolean).
- **hide_downs** – Don't show me submissions after I've downvoted them, except my own (boolean).
- **hide_from_robots** – Don't allow search engines to index my user profile (boolean).
- **hide_locationbar** – Hide location bar (boolean).
- **hide_ups** – Don't show me submissions after I've upvoted them, except my own (boolean).
- **highlight_controversial** – Show a dagger on comments voted controversial (boolean).
- **highlight_new_comments** – Highlight new comments (boolean).
- **ignore_suggested_sort** – Ignore suggested sorts (boolean).
- **in_redesign_beta** – In redesign beta (boolean).
- **label_nsfw** – Label posts that are not safe for work (boolean).
- **lang** – Interface language (IETF language tag, underscore separated).
- **legacy_search** – Show legacy search page (boolean).
- **live_orangereds** – Send message notifications in my browser (boolean).
- **mark_messages_read** – Mark messages as read when I open my inbox (boolean).
- **media** – Thumbnail preference (one of `'on'`, `'off'`, `'subreddit'`).
- **media_preview** – Media preview preference (one of `'on'`, `'off'`, `'subreddit'`).
- **min_comment_score** – Don't show me comments with a score less than this number (int between `-100` and `100`).
- **min_link_score** – Don't show me submissions with a score less than this number (int between `-100` and `100`).
- **monitor_mentions** – Notify me when people say my username (boolean).
- **newwindow** – Open links in a new window (boolean).
- **no_profanity** – Don't show thumbnails or media previews for anything labeled NSFW (boolean).
- **no_video_autoplay** – Don't autoplay Reddit videos on the desktop comments page (boolean).

- **num_comments** – Display this many comments by default (int between 1 and 500).
- **numsites** – Number of links to display at once (int between 1 and 100).
- **organic** – Show the spotlight box on the home feed (boolean).
- **other_theme** – Subreddit theme to use (subreddit name).
- **over_18** – I am over eighteen years old and willing to view adult content (boolean).
- **private_feeds** – Enable private RSS feeds (boolean).
- **profile_opt_out** – View user profiles on desktop using legacy mode (boolean).
- **public_votes** – Make my votes public (boolean).
- **research** – Allow my data to be used for research purposes (boolean).
- **search_include_over_18** – Include not safe for work (NSFW) search results in searches (boolean).
- **show_flair** – Show user flair (boolean).
- **show_gold_expiration** – Show how much gold you have remaining on your user-page (boolean).
- **show_link_flair** – Show link flair (boolean).
- **show_promote** – Show promote (boolean).
- **show_stylesheets** – Allow subreddits to show me custom themes (boolean).
- **show_trending** – Show trending subreddits on the home feed (boolean).
- **store_visits** – Store visits (boolean)
- **theme_selector** – Theme selector (subreddit name).
- **threaded_messages** – Show message conversations in the inbox (boolean).
- **threaded_modmail** – Enable threaded modmail display (boolean).
- **top_karma_subreddits** – Top karma subreddits (boolean).
- **use_global_defaults** – Use global defaults (boolean).

Additional keyword arguments can be provided to handle new settings as Reddit introduces them.

See https://www.reddit.com/dev/api#PATCH_api_v1_me_prefs for the most up-to-date list of possible parameters.

This is intended to be used like so:

```
reddit.user.preferences.update(show_link_flair=True)
```

This method returns the new state of the preferences as a `dict`, which can be used to check whether a change went through.

```
original_preferences = reddit.user.preferences()
new_preferences = reddit.user.preferences.update(invalid_param=123)
print(original_preferences == new_preferences)  # True, no change
```

Warning: Passing an unknown parameter name or an illegal value (such as an int when a boolean is expected) does not result in an error from the Reddit API. As a consequence, any invalid input will fail silently. To verify that changes have been made, use the return value of this method, which is a dict of the preferences after the update action has been performed.

Some preferences have names that are not valid keyword arguments in Python. To update these, construct a `dict` and use `**` to unpack it as keyword arguments:

```
reddit.user.preferences.update(
    **{'3rd_party_data_personalized_ads': False})
```

1.10.55 RedditBase

class praw.models.reddit.base.**RedditBase**(*reddit: Reddit, _data: Dict[str, Any]*)

Base class that represents actual Reddit objects.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)

Initialize a RedditBase instance (or a subclass).

Parameters **reddit** – An instance of *Reddit*.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.56 RedditorList

class praw.models.**RedditorList**(*reddit: Reddit, _data: Dict[str, Any]*)

A list of Redditors. Works just like a regular list.

__contains__(*item: Any*) → bool

Test if item exists in the list.

__getitem__(*index: int*) → Any

Return the item at position index in the list.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)

Initialize a BaseList instance.

Parameters **reddit** – An instance of *Reddit*.

__iter__() → Iterator[Any]

Return an iterator to the list.

__len__() → int

Return the number of items in the list.

classmethod parse(*data: Dict[str, Any], reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.

- **reddit** – An instance of *Reddit*.

1.10.57 RemovalReason

class praw.models.reddit.removal_reasons.**RemovalReason**(*reddit: Reddit*, *subreddit: Subreddit*, *reason_id: str*, *_data: Optional[Dict[str, Any]] = None*)

An individual Removal Reason object.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
`id`	The id of the removal reason.
`message`	The message of the removal reason.
`title`	The title of the removal reason.

__init__(*reddit: Reddit*, *subreddit: Subreddit*, *reason_id: str*, *_data: Optional[Dict[str, Any]] = None*)

Construct an instance of the Removal Reason object.

delete()

Delete a removal reason from this subreddit.

To delete `'141vv5c16py7d'` from the subreddit `'NAME'` try:

```
reddit.subreddit('NAME').removal_reasons['141vv5c16py7d'].mod.delete()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any

Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

update(*message: str*, *title: str*)

Update the removal reason from this subreddit.

Parameters

- **message** – The removal reason's new message (required).
- **title** – The removal reason's new title (required).

To update `'141vv5c16py7d'` from the subreddit `'NAME'` try:

```
reddit.subreddit('NAME').removal_reasons['141vv5c16py7d'].mod.update(
    message='New message',
    title='New title')
```

1.10.58 SubListing

class praw.models.listing.mixins.redditor.**SubListing**(*reddit: Reddit*, *base_path: str*, *subpath: str*)

Helper class for generating *ListingGenerator* objects.

__init__(*reddit: Reddit*, *base_path: str*, *subpath: str*)

Initialize a SubListing instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **base_path** – The path to the object up to this point.
- **subpath** – The additional path to this sublisting.

controversial(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for controversial submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise ValueError if time_filter is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').controversial('week')
reddit.multireddit('samuraisam', 'programming').controversial('day')
reddit.redditor('spez').controversial('month')
reddit.redditor('spez').comments.controversial('year')
reddit.redditor('spez').submissions.controversial('all')
reddit.subreddit('all').controversial('hour')
```

hot(***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for hot items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').hot()
reddit.multireddit('samuraisam', 'programming').hot()
reddit.redditor('spez').hot()
reddit.redditor('spez').comments.hot()
reddit.redditor('spez').submissions.hot()
reddit.subreddit('all').hot()
```

new(***generator_kwargs*) → Generator[[Any, None], None]

Return a *ListingGenerator* for new items.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').new()
reddit.multireddit('samuraisam', 'programming').new()
reddit.redditor('spez').new()
reddit.redditor('spez').comments.new()
reddit.redditor('spez').submissions.new()
reddit.subreddit('all').new()
```

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

top(*time_filter: str = 'all'*, ***generator_kwargs*) → Generator[[Any, None], None]
Return a *ListingGenerator* for top submissions.

Parameters **time_filter** – Can be one of: all, day, hour, month, week, year (default: all).

Raise `ValueError` if `time_filter` is invalid.

Additional keyword arguments are passed in the initialization of *ListingGenerator*.

This method can be used like:

```
reddit.domain('imgur.com').top('week')
reddit.multireddit('samuraisam', 'programming').top('day')
reddit.redditor('spez').top('month')
reddit.redditor('spez').comments.top('year')
reddit.redditor('spez').submissions.top('all')
reddit.subreddit('all').top('hour')
```

1.10.59 Submenu

class praw.models.**Submenu**(*reddit: Reddit*, *_data: Dict[str, Any]*)
Class to represent a submenu of links inside a menu.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
children	A list of the *MenuLink*s in this submenu. Can be iterated over by iterating over the *Submenu* (e.g. `for menu_link in submenu`).
text	The name of the submenu.

__contains__(*item: Any*) → bool
Test if item exists in the list.

__getitem__(*index: int*) → Any
Return the item at position index in the list.

__init__(*reddit: Reddit*, *_data: Dict[str, Any]*)
Initialize a BaseList instance.

Parameters **reddit** – An instance of *Reddit*.

__iter__() → Iterator[Any]
Return an iterator to the list.

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

classmethod parse(*data: Dict[str, Any]*, *reddit: Reddit*) → Any
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

1.10.60 SubredditEmoji

class praw.models.reddit.emoji.**SubredditEmoji**(*subreddit: Subreddit*)
Provides a set of functions to a Subreddit for emoji.

__getitem__(*name: str*) → praw.models.reddit.emoji.Emoji
Lazily return the Emoji for the subreddit named `name`.

Parameters **name** – The name of the emoji

This method is to be used to fetch a specific emoji url, like so:

```
emoji = reddit.subreddit('praw_test').emoji['test']
print(emoji)
```

__init__(*subreddit: Subreddit*)
Create a SubredditEmoji instance.

Parameters **subreddit** – The subreddit whose emoji are affected.

__iter__() → List[praw.models.reddit.emoji.Emoji]
Return a list of Emoji for the subreddit.

This method is to be used to discover all emoji for a subreddit:

```
for emoji in reddit.subreddit('praw_test').emoji:
    print(emoji)
```

add(*name: str*, *image_path: str*, *mod_flair_only: Optional[bool] = None*, *post_flair_allowed: Optional[bool] = None*, *user_flair_allowed: Optional[bool] = None*) → praw.models.reddit.emoji.Emoji
Add an emoji to this subreddit.

Parameters

- **name** – The name of the emoji
- **image_path** – A path to a jpeg or png image.
- **mod_flair_only** – (boolean) When provided, indicate whether the emoji is restricted to mod use only. (Default: `None`)
- **post_flair_allowed** – (boolean) When provided, indicate whether the emoji may appear in post flair. (Default: `None`)
- **user_flair_allowed** – (boolean) When provided, indicate whether the emoji may appear in user flair. (Default: `None`)

Returns The Emoji added.

To add `test` to the subreddit `praw_test` try:

```
reddit.subreddit("praw_test").emoji.add("test", "test.png")
```

1.10.61 SubredditMessage

class praw.models.**SubredditMessage**(*reddit: Reddit, _data: Dict[str, Any]*)

A class for messages to a subreddit.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

Attribute	Description
author	Provides an instance of *Redditor*.
body	The body of the message.
created_utc	Time the message was created, represented in Unix Time.
dest	Provides an instance of *Redditor*. The recipient of the message.
id	The ID of the message.
name	The full ID of the message, prefixed with 't4'.
subject	The subject of the message.
subreddit	If the message was sent from a subreddit, provides an instance of *Subreddit*.
was_comment	Whether or not the message was a comment reply.

__init__(*reddit: Reddit, _data: Dict[str, Any]*)

Construct an instance of the Message object.

block()

Block the user who sent the item.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
comment = reddit.comment('dkk4qjd')
comment.block()

# or, identically:

comment.author.block()
```

collapse()

Mark the item as collapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and collapse it
message = next(inbox)
message.collapse()
```

See also *uncollapse()*

delete()
Delete the message.

Note: Reddit does not return an indication of whether or not the message was successfully deleted.

For example, to delete the most recent message in your inbox:

```
next(reddit.inbox.all()).delete()
```

fullname
Return the object's fullname.

A fullname is an object's kind mapping like `t3` followed by an underscore and the object's base36 ID, e.g., `t1_c5s96e0`.

mark_read()
Mark a single inbox item as read.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox.unread()

for message in inbox:
    # process unread messages
```

See also *mark_unread()*

To mark the whole inbox as read with a single network request, use *praw.models.Inbox.mark_read()*

mark_unread()
Mark the item as unread.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox(limit=10)

for message in inbox:
    # process messages
```

See also *mark_read()*

mute()
Mute the sender of this SubredditMessage.

For example, to mute the sender of the first SubredditMessage in the authenticated users' inbox:

```
from praw.models import SubredditMessage
msg = next(message for message in reddit.inbox.all()
           if isinstance(message, SubredditMessage))
msg.mute()
```

classmethod parse(*data: Dict[str, Any], reddit: Reddit*)
Return an instance of Message or SubredditMessage from `data`.

Parameters

- **data** – The structured data.
- **reddit** – An instance of *Reddit*.

reply(*body*)
Reply to the object.

Parameters **body** – The Markdown formatted content for a comment.

Returns A *Comment* object for the newly created comment or `None` if Reddit doesn't provide one.

A `None` value can be returned if the target is a comment or submission in a quarantined subreddit and the authenticated user has not opt-ed in to viewing the content. When this happens the comment will be sucessfully created on Reddit and can be retried by drawing the comment from the user's comment history.

Note: Some items, such as locked submissions/comments or non-replyable messages will throw `prawcore.exceptions.Forbidden` when attempting to reply to them.

Example usage:

```
submission = reddit.submission(id='5or86n')
submission.reply('reply')

comment = reddit.comment(id='dxolpyc')
comment.reply('reply')
```

uncollapse()
Mark the item as uncollapsed.

Note: This method pertains only to objects which were retrieved via the inbox.

Example usage:

```
inbox = reddit.inbox()

# select first inbox item and uncollapse it
message = next(inbox)
message.uncollapse()
```

See also *collapse()*

unmute()
Unmute the sender of this SubredditMessage.

For example, to unmute the sender of the first SubredditMessage in the authenticated users' inbox:

```
from praw.models import SubredditMessage
msg = next(message for message in reddit.inbox.all()
           if isinstance(message, SubredditMessage))
msg.unmute()
```

1.10.62 SubredditRemovalReasons

class praw.models.reddit.removal_reasons.**SubredditRemovalReasons**(*subreddit: Subreddit*)

Provide a set of functions to a Subreddit's removal reasons.

__getitem__(*reason_id: str*) → praw.models.reddit.removal_reasons.RemovalReason
Lazily return the Removal Reason for the subreddit with id `reason_id`.

Parameters **reason_id** – The id of the removal reason

This method is to be used to fetch a specific removal reason, like so:

```
reason_id = '141vv5c16py7d'
reason = reddit.subreddit('NAME').mod.removal_reasons[reason_id]
print(reason)
```

__init__(*subreddit: Subreddit*)
Create a SubredditRemovalReasons instance.

Parameters **subreddit** – The subreddit whose removal reasons to work with.

__iter__() → Generator[[praw.models.reddit.removal_reasons.RemovalReason, None], None]
Return a list of Removal Reasons for the subreddit.

This method is used to discover all removal reasons for a subreddit:

```
for removal_reason in reddit.subreddit('NAME').mod.removal_reasons:
    print(removal_reason)
```

add(*message: str*, *title: str*) → praw.models.reddit.removal_reasons.RemovalReason
Add a removal reason to this subreddit.

Parameters

- **message** – The message associated with the removal reason.
- **title** – The title of the removal reason

Returns The RemovalReason added.

The message will be prepended with *Hi u/username,* automatically.

To add `'Test'` to the subreddit `'NAME'` try:

```
reddit.subreddit('NAME').removal_reasons.mod.add(
    message='Foobar',
    title='Test')
```

1.10.63 RedditorStream

class praw.models.reddit.redditor.**RedditorStream**(*redditor: praw.models.reddit.redditor.Redditor*)

Provides submission and comment streams.

__init__(*redditor: praw.models.reddit.redditor.Redditor*)
Create a RedditorStream instance.

Parameters **redditor** – The redditor associated with the streams.

comments(***stream_options*) → Generator[[Comment, None], None]
Yield new comments as they become available.

Comments are yielded oldest first. Up to 100 historical comments will initially be returned.

Keyword arguments are passed to *stream_generator()*.

For example, to retrieve all new comments made by redditor spez, try:

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

submissions(***stream_options*) → Generator[[Submission, None], None]
Yield new submissions as they become available.

Submissions are yielded oldest first. Up to 100 historical submissions will initially be returned.

Keyword arguments are passed to *stream_generator()*.

For example to retrieve all new submissions made by redditor spez, try:

```
for submission in reddit.redditor('spez').stream.submissions():
    print(submission)
```

1.10.64 Trophy

class praw.models.**Trophy**(*reddit: Reddit, _data: Dict[str, Any]*)
Represent a trophy.

End users should not instantiate this class directly. *Redditor.trophies()* can be used to get a list of the redditor's trophies.

Typical Attributes

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see *Determine Available Attributes of an Object*), there is not a guarantee that these attributes will always be present, nor is this list necessarily comprehensive.

Attribute	Description
award_id	The ID of the trophy (sometimes None).
description	The description of the trophy (sometimes None).
icon_40	The URL of a 41x41 px icon for the trophy.
icon_70	The URL of a 71x71 px icon for the trophy.
name	The name of the trophy.
url	A relevant URL (sometimes None).

__str__() → str
Return a name of the trophy.

1.10.65 Util

class praw.models.util.**BoundedSet**(*max_items: int*)
A set with a maximum size that evicts the oldest items when necessary.

This class does not implement the complete set interface.

__contains__(*item: Any*) → bool
Test if the BoundedSet contains item.

__init__(*max_items: int*)
Construct an instance of the BoundedSet.

add(*item: Any*)
Add an item to the set discarding the oldest item if necessary.

class praw.models.util.**ExponentialCounter**(*max_counter: int*)
A class to provide an exponential counter with jitter.

__init__(*max_counter: int*)
Initialize an instance of ExponentialCounter.

Parameters **max_counter** – The maximum base value. Note that the computed value may be 3.125% higher due to jitter.

counter() → int
Increment the counter and return the current value with jitter.

reset()
Reset the counter to 1.

praw.models.util.**permissions_string**(*permissions: Optional[List[str]], known_permissions: Set[str]*) → str
Return a comma separated string of permission changes.

Parameters

- **permissions** – A list of strings, or None. These strings can exclusively contain + or – prefixes, or contain no prefixes at all. When prefixed, the resulting string will simply be the joining of these inputs. When not prefixed, all permissions are considered to be additions, and all permissions in the known_permissions set that aren't provided are considered to be removals. When None, the result is +all.
- **known_permissions** – A set of strings representing the available permissions.

praw.models.util.**stream_generator**(*function: Callable[Any, Any], pause_after: Optional[int] = None, skip_existing: bool = False, attribute_name: str = 'fullname', exclude_before: bool = False, **function_kwargs*) → Generator[[Any, None], None]
Yield new items from ListingGenerators and None when paused.

Parameters

- **function** – A callable that returns a ListingGenerator, e.g. subreddit.comments or subreddit.new.
- **pause_after** – An integer representing the number of requests that result in no new items before this function yields None, effectively introducing a pause into the stream. A negative value yields None after items from a single response have been yielded, regardless of number of new items obtained in that response. A value of 0 yields None after every response resulting in no new items, and a value of None never introduces a pause (default: None).
- **skip_existing** – When True does not yield any results from the first request thereby skipping any items that existed in the stream prior to starting the stream (default: False).
- **attribute_name** – The field to use as an id (default: "fullname").
- **exclude_before** – When True does not pass params to functions (default: False).

Additional keyword arguments will be passed to `function`.

Note: This function internally uses an exponential delay with jitter between subsequent responses that contain no new results, up to a maximum delay of just over a 16 seconds. In practice that means that the time before pause for `pause_after=N+1` is approximately twice the time before pause for `pause_after=N`.

For example, to create a stream of comment replies, try:

```
reply_function = reddit.inbox.comment_replies
for reply in praw.models.util.stream_generator(reply_function):
    print(reply)
```

To pause a comment stream after six responses with no new comments, try:

```
subreddit = reddit.subreddit('redditdev')
for comment in subreddit.stream.comments(pause_after=6):
    if comment is None:
        break
    print(comment)
```

To resume fetching comments after a pause, try:

```
subreddit = reddit.subreddit('help')
comment_stream = subreddit.stream.comments(pause_after=5)

for comment in comment_stream:
    if comment is None:
        break
    print(comment)
# Do any other processing, then try to fetch more data
for comment in comment_stream:
    if comment is None:
        break
    print(comment)
```

To bypass the internal exponential backoff, try the following. This approach is useful if you are monitoring a subreddit with infrequent activity, and you want the to consistently learn about new items from the stream as soon as possible, rather than up to a delay of just over sixteen seconds.

```
subreddit = reddit.subreddit('help')
for comment in subreddit.stream.comments(pause_after=0):
    if comment is None:
        continue
    print(comment)
```

1.11 Comment Extraction and Parsing

A common use for Reddit's API is to extract comments from submissions and use them to perform keyword or phrase analysis.

As always, you need to begin by creating an instance of *Reddit*:

```
import praw
```

(continues on next page)

(continued from previous page)

```
reddit = praw.Reddit(user_agent='Comment Extraction (by /u/USERNAME)',
                     client_id='CLIENT_ID', client_secret="CLIENT_SECRET",
                     username='USERNAME', password='PASSWORD')
```

Note: If you are only analyzing public comments, entering a username and password is optional.

In this document we will detail the process of finding all the comments for a given submission. If you instead want process all comments on Reddit, or comments belonging to one or more specific subreddits, please see *praw.models.reddit.subreddit.SubredditStream.comments()*.

1.11.1 Extracting comments with PRAW

Assume we want to process the comments for this submission: https://www.reddit.com/r/funny/comments/3g1jfi/buttons/

We first need to obtain a submission object. We can do that either with the entire URL:

```
submission = reddit.submission(url='https://www.reddit.com/r/funny/comments/3g1jfi/
↪buttons/')
```

or with the submission's ID which comes after `comments/` in the URL:

```
submission = reddit.submission(id='3g1jfi')
```

With a submission object we can then interact with its *CommentForest* through the submission's *comments* attribute. A *CommentForest* is a list of top-level comments each of which contains a *CommentForest* of replies.

If we wanted to output only the `body` of the top level comments in the thread we could do:

```
for top_level_comment in submission.comments:
    print(top_level_comment.body)
```

While running this you will most likely encounter the exception `AttributeError: 'MoreComments' object has no attribute 'body'`. This submission's comment forest contains a number of *MoreComments* objects. These objects represent the "load more comments", and "continue this thread" links encountered on the website. While we could ignore *MoreComments* in our code, like so:

```
from praw.models import MoreComments
for top_level_comment in submission.comments:
    if isinstance(top_level_comment, MoreComments):
        continue
    print(top_level_comment.body)
```

1.11.2 The `replace_more` method

In the previous snippet, we used `isinstance` to check whether the item in the comment list was a *MoreComments* so that we could ignore it. But there is a better way: the *CommentForest* object has a method called *replace_more()*, which replaces or removes *MoreComments* objects from the forest.

Each replacement requires one network request, and its response may yield additional *MoreComments* instances. As a result, by default, *replace_more()* only replaces at most thirty-two *MoreComments* instances – all other

instances are simply removed. The maximum number of instances to replace can be configured via the `limit` parameter. Additionally a `threshold` parameter can be set to only perform replacement of *`MoreComments`* instances that represent a minimum number of comments; it defaults to 0, meaning all *`MoreComments`* instances will be replaced up to `limit`.

A `limit` of 0 simply removes all *`MoreComments`* from the forest. The previous snippet can thus be simplified:

```
submission.comments.replace_more(limit=0)
for top_level_comment in submission.comments:
    print(top_level_comment.body)
```

Note: Calling *`replace_more()`* is destructive. Calling it again on the same submission instance has no effect.

Meanwhile, a `limit` of `None` means that all *`MoreComments`* objects will be replaced until there are none left, as long as they satisfy the `threshold`.

```
submission.comments.replace_more(limit=None)
for top_level_comment in submission.comments:
    print(top_level_comment.body)
```

Now we are able to successfully iterate over all the top-level comments. What about their replies? We could output all second-level comments like so:

```
submission.comments.replace_more(limit=None)
for top_level_comment in submission.comments:
    for second_level_comment in top_level_comment.replies:
        print(second_level_comment.body)
```

However, the comment forest can be arbitrarily deep, so we'll want a more robust solution. One way to iterate over a tree, or forest, is via a breadth-first traversal using a queue:

```
submission.comments.replace_more(limit=None)
comment_queue = submission.comments[:]  # Seed with top-level
while comment_queue:
    comment = comment_queue.pop(0)
    print(comment.body)
    comment_queue.extend(comment.replies)
```

The above code will output all the top-level comments, followed by second-level, third-level, etc. While it is awesome to be able to do your own breadth-first traversals, *`CommentForest`* provides a convenience method, *`list()`*, which returns a list of comments traversed in the same order as the code above. Thus the above can be rewritten as:

```
submission.comments.replace_more(limit=None)
for comment in submission.comments.list():
    print(comment.body)
```

You can now properly extract and parse all (or most) of the comments belonging to a single submission. Combine this with *submission iteration* and you can build some really cool stuff.

Finally, note that the value of `submission.num_comments` may not match up 100% with the number of comments extracted via PRAW. This discrepancy is normal as that count includes deleted, removed, and spam comments.

1.12 Obtaining a Refresh Token

The following program can be used to obtain a refresh token with the desired scopes. Such a token can be used in conjunction with the `refresh_token` keyword argument using in initializing an instance of *Reddit*. A list of all possible scopes can be found in the reddit API docs

```
#!/usr/bin/env python

"""This example demonstrates the flow for retrieving a refresh token.

In order for this example to work your application's redirect URI must be set
to http://localhost:8080.

This tool can be used to conveniently create refresh tokens for later use with
your web application OAuth2 credentials.

"""
import praw
import random
import socket
import sys


def receive_connection():
    """Wait for and then return a connected socket..

    Opens a TCP connection on port 8080, and waits for a single client.

    """
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind(("localhost", 8080))
    server.listen(1)
    client = server.accept()[0]
    server.close()
    return client


def send_message(client, message):
    """Send message to client and close the connection."""
    print(message)
    client.send("HTTP/1.1 200 OK\r\n\r\n{}".format(message).encode("utf-8"))
    client.close()


def main():
    """Provide the program's entry point when directly executed."""
    print(
        "Go here while logged into the account you want to create a "
        "token for: https://www.reddit.com/prefs/apps/"
    )
    print(
        "Click the create an app button. Put something in the name "
        "field and select the script radio button."
    )
    print(
        "Put http://localhost:8080 in the redirect uri field and "
```

(continues on next page)

(continued from previous page)

```
        "click create app"
    )
    client_id = input(
        "Enter the client ID, it's the line just under "
        "Personal use script at the top: "
    )
    client_secret = input(
        "Enter the client secret, it's the line next " "to secret: "
    )
    commaScopes = input(
        "Now enter a comma separated list of scopes, or "
        "all for all tokens: "
    )

    if commaScopes.lower() == "all":
        scopes = ["*"]
    else:
        scopes = commaScopes.strip().split(",")

    reddit = praw.Reddit(
        client_id=client_id.strip(),
        client_secret=client_secret.strip(),
        redirect_uri="http://localhost:8080",
        user_agent="praw_refresh_token_example",
    )
    state = str(random.randint(0, 65000))
    url = reddit.auth.url(scopes, state, "permanent")
    print("Now open this url in your browser: " + url)
    sys.stdout.flush()

    client = receive_connection()
    data = client.recv(1024).decode("utf-8")
    param_tokens = data.split(" ", 2)[1].split("?", 1)[1].split("&")
    params = {
        key: value
        for (key, value) in [token.split("=") for token in param_tokens]
    }

    if state != params["state"]:
        send_message(
            client,
            "State mismatch. Expected: {} Received: {}".format(
                state, params["state"]
            ),
        )
        return 1
    elif "error" in params:
        send_message(client, params["error"])
        return 1

    refresh_token = reddit.auth.authorize(params["code"])
    send_message(client, "Refresh token: {}".format(refresh_token))
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

1.13 Submission Stream Reply Bot

Most redditors have seen bots in action on the site. Reddit bots can perform a number of tasks including providing useful information, e.g., an Imperial to Metric units bot; convenience, e.g., a link corrector bot; or analytical information, e.g., redditor analyzer bot for writing complexity.

PRAW provides a simple way to build your own bot using the python programming language. As a result, it is little surprise that a majority of bots on Reddit are powered by PRAW.

This tutorial will show you how to build a bot that monitors a particular subreddit, /r/AskReddit, for new submissions containing simple questions and replies with an appropriate link to lmgtfy (Let Me Google That For You).

There are three key components we will address to perform this task:

1. Monitor new submissions.
2. Analyze the title of each submission to see if it contains a simple question.
3. Reply with an appropriate lmgtfy link.

1.13.1 LMGTFY Bot

The goal of the LMGTFY Bot is to point users in the right direction when they ask a simple question that is unlikely to be upvoted or answered by other users.

Two examples of such questions are:

1. "What is the capital of Canada?"
2. "How many feet are in a yard?"

Once we identify these questions, the LMGTFY Bot will reply to the submission with an appropriate lmgtfy link. For the example questions those links are:

1. http://lmgtfy.com/?q=What+is+the+capital+of+Canada%3F
2. http://lmgtfy.com/?q=How+many+feet+are+in+a+yard%3F

Step 1: Getting Started

Access to Reddit's API requires a set of OAuth2 credentials. Those credentials are obtained by registering an application with Reddit. To register an application and receive a set of OAuth2 credentials please follow only the "First Steps" section of Reddit's OAuth2 Quick Start Example wiki page.

Once the credentials are obtained we can begin writing the LMGTFY Bot. Start by creating an instance of *Reddit*:

```
import praw

reddit = praw.Reddit(user_agent='LMGTFY (by /u/USERNAME)',
                     client_id='CLIENT_ID', client_secret="CLIENT_SECRET",
                     username='USERNAME', password='PASSWORD')
```

In addition to the OAuth2 credentials, the username and password of the Reddit account that registered the application are required.

Note: This example demonstrates use of a *script* type application. For other application types please see Reddit's wiki page OAuth2 App Types.

Step 2: Monitoring New Submissions to /r/AskReddit

PRAW provides a convenient way to obtain new submissions to a given subreddit. To indefinitely iterate over new submissions to a subreddit add:

```
subreddit = reddit.subreddit('AskReddit')
for submission in subreddit.stream.submissions():
    # do something with submission
```

Replace `AskReddit` with the name of another subreddit if you want to iterate through its new submissions. Additionally multiple subreddits can be specified by joining them with pluses, for example `AskReddit+NoStupidQuestions`. All subreddits can be specified using the special name `all`.

Step 3: Analyzing the Submission Titles

Now that we have a stream of new submissions to /r/AskReddit, it is time to see if their titles contain a simple question. We naïvely define a simple question as:

1. It must contain no more than ten words.
2. It must contain one of the phrases "what is", "what are", or "who is".

Warning: These naïve criteria result in many false positives. It is strongly recommended that you develop more precise heuristics before launching a bot on any popular subreddits.

First we filter out titles that contain more than ten words:

```
if len(submission.title.split()) > 10:
        return
```

We then check to see if the submission's title contains any of the desired phrases:

```
questions = ['what is', 'who is', 'what are']
normalized_title = submission.title.lower()
for question_phrase in questions:
    if question_phrase in normalized_title:
        # do something with a matched submission
        break
```

String comparison in python is case sensitive. As a result, we only compare a normalized version of the title to our lower-case question phrases. In this case, "normalized" means only lower-case.

The `break` at the end prevents us from matching more than once on a single submission. For instance, what would happen without the `break` if a submission's title was "Who is or what are buffalo?"

Step 4: Automatically Replying to the Submission

The LMGTFY Bot is nearly complete. We iterate through submissions, and find ones that appear to be simple questions. All that is remaining is to reply to those submissions with an appropriate lmgtfy link.

First we will need to construct a working lmgtfy link. In essence we want to pass the entire submission title to lmgtfy. However, there are certain characters that are not permitted in URLs or have other . For instance, the space character, ' ', is not permitted, and the question mark, '?', has a special meaning. Thus we will transform those into their URL-safe representation so that a question like "What is the capital of Canada?" is transformed into the link `http://lmgtfy.com/?q=What+is+the+capital+of+Canada%3F)`.

There are a number of ways we could accomplish this task. For starters we could write a function to replace spaces with pluses, +, and question marks with `%3F`. However, there is even an easier way; using an existing built-in function to do so.

Add the following code where the "do something with a matched submission" comment is located:

```
from urllib.parse import quote_plus

reply_template = '[Let me google that for you](http://lmgtfy.com/?q={})'

url_title = quote_plus(submission.title)
reply_text = reply_template.format(url_title)
```

Note: This example assumes the use of Python 3. For Python 2 replace `from urllib.parse import quote_plus` with `from urllib import quote_plus`.

Now that we have the reply text, replying to the submission is easy:

```
submission.reply(reply_text)
```

If all went well, your comment should have been made. If your bot account is brand new, you will likely run into rate limit issues. These rate limits will persist until that account acquires sufficient karma.

Step 5: Cleaning Up The Code

While we have a working bot, we have added little segments here and there. If we were to continue to do so in this fashion our code would be quite unreadable. Let's clean it up some.

The first thing we should do is put all of our import statements at the top of the file. It is common to list built-in packages before third party ones:

```
from urllib.parse import quote_plus

import praw
```

Next we extract a few constants that are used in our script:

```
QUESTIONS = ["what is", "who is", "what are"]
REPLY_TEMPLATE = "[Let me google that for you](http://lmgtfy.com/?q={})"
```

We then extract the segment of code pertaining to processing a single submission into its own function:

```
    for submission in subreddit.stream.submissions():
        process_submission(submission)


def process_submission(submission):
    # Ignore titles with more than 10 words as they probably are not simple
    # questions.
    if len(submission.title.split()) > 10:
        return

    normalized_title = submission.title.lower()
    for question_phrase in QUESTIONS:
        if question_phrase in normalized_title:
```

(continues on next page)

(continued from previous page)

```
            url_title = quote_plus(submission.title)
            reply_text = REPLY_TEMPLATE.format(url_title)
```

Observe that we added some comments and a `print` call. The `print` addition informs us every time we are about to reply to a submission, which is useful to ensure the script is running.

Next, it is a good practice to not have any top-level executable code in case you want to turn your Python script into a Python module, i.e., import it from another Python script or module. A common way to do that is to move the top-level code to a `main` function:

```
def main():
    reddit = praw.Reddit(
        user_agent="LMGTFY (by /u/USERNAME)",
        client_id="CLIENT_ID",
        client_secret="CLIENT_SECRET",
        username="USERNAME",
        password="PASSWORD",
    )
```

Finally we need to call `main` only in the cases that this script is the one being executed:

```
            # A reply has been made so do not attempt to match other phrases.
            break


if __name__ == "__main__":
    main()
```

The Complete LMGTFY Bot

The following is the complete LMGTFY Bot:

```
from urllib.parse import quote_plus

import praw

QUESTIONS = ["what is", "who is", "what are"]
REPLY_TEMPLATE = "[Let me google that for you](http://lmgtfy.com/?q={})"


def main():
    reddit = praw.Reddit(
        user_agent="LMGTFY (by /u/USERNAME)",
        client_id="CLIENT_ID",
        client_secret="CLIENT_SECRET",
        username="USERNAME",
        password="PASSWORD",
    )

    subreddit = reddit.subreddit("AskReddit")
    for submission in subreddit.stream.submissions():
        process_submission(submission)


def process_submission(submission):
```

(continues on next page)

(continued from previous page)

```
    # Ignore titles with more than 10 words as they probably are not simple
    # questions.
    if len(submission.title.split()) > 10:
        return

    normalized_title = submission.title.lower()
    for question_phrase in QUESTIONS:
        if question_phrase in normalized_title:
            url_title = quote_plus(submission.title)
            reply_text = REPLY_TEMPLATE.format(url_title)
            print("Replying to: {}".format(submission.title))
            submission.reply(reply_text)
            # A reply has been made so do not attempt to match other phrases.
            break


if __name__ == "__main__":
    main()
```

1.14 Change Log

1.14.1 Unreleased

Added

- `config_interpolation` parameter for *Reddit* supporting basic and extended modes.
- Add *Redditors.partial_redditors()* that returns lightweight redditor objects that contain only a few fields. This is useful for resolving Redditor IDs to their usernames in bulk.
- *User.friends()* has a new parameter `user` that takes either an instance of *Redditor* or a string containing a redditor name and returns an instance of *Redditor* if the authenticated user is friends with the user, otherwise throws an exception.
- *SubmissionModeration.flair()* has the parameter `flair_template_id` for applying flairs with template IDs.
- *update()* supports modifying an emoji's permissions.
- *add()* now supports optionally passing booleans to set an emoji's permissions upon upload.
- Methods *SubredditLinkFlairTemplates.update()* and *SubredditRedditorFlairTemplates.update()* contain a new parameter, `fetch`, that toggles the automatic fetching of existing data from Reddit. It is set to True by default.
- Values in methods *SubredditLinkFlairTemplates.update()* and *SubredditRedditorFlairTemplates.update()* that are left as the defaults will no longer be over-written if the `fetch` parameter is set to `True`, but will fill in existing values for the flair template.
- The parameter `text` for methods *SubredditLinkFlairTemplates.update()* and *SubredditRedditorFlairTemplates.update()* is no longer required.

Removed

- Converting *APIException* to string will no longer escape unicode characters.
- Module `praw.models.modaction` no longer exists. Please use the module `praw.models.mod_action`, or directly import `ModAction` from `praw.models`.

- Methods *SubredditLinkFlairTemplates.update()* and *SubredditRedditorFlairTemplates.update()* will no longer create flairs that are using an invalid template id, but instead throw a *InvalidFlairTemplateID*.

1.14.2 6.5.1 (2020/01/07)

Fixed

- Removed usages of `NoReturn` that caused PRAW to fail due to `ImportError` in Python `<3.5.4` and `<3.6.2`.

1.14.3 6.5.0 (2020/01/05)

Added

- *set_original_content()* supports marking a submission as original content.
- *unset_original_content()* supports unmarking a submission as original content.
- *Redditor.moderated()* to get a list of a Redditor's moderated subreddits.
- Parameter `without_websockets` to *submit_image()* and *submit_video()* to submit without using WebSockets.
- *Reddit.redditor()* supports `fullname` param to fetch a Redditor by the fullname instead of name. *Redditor* constructor now also has `fullname` param.
- Add *RemovalReason* and *SubredditRemovalReasons* to work with removal reasons
- Attribute `removal_reasons` to *SubredditModeration* to interact with new removal reason classes
- Parameters `mod_note` and `reason_id` to *ThingModerationMixin.remove()* to optionally apply a removal reason on removal
- Add *SubredditModerationStream* to enable moderation streams
- Attribute `stream` to *SubredditModeration* to interact with new moderation streams
- Add *SubredditModerationStream.edited()* to allow streaming of *SubredditModeration.edited()*
- Add *SubredditModerationStream.log()* to allow streaming of *SubredditModeration.log()*
- Add *SubredditModerationStream.modmail_conversations()* to allow streaming of *Modmail.conversations()*
- Add *SubredditModerationStream.modqueue()* to allow streaming of *SubredditModeration.modqueue()*
- Add *SubredditModerationStream.reports()* to allow streaming of *SubredditModeration.reports()*
- Add *SubredditModerationStream.spam()* to allow streaming of *SubredditModeration.spam()*
- Add *SubredditModerationStream.unmoderated()* to allow streaming of *SubredditModeration.unmoderated()*
- Add *SubredditModerationStream.unread()* to allow streaming of *SubredditModeration.unread()*

- Parameter `exclude_before` to *`stream_generator()`* to allow *`SubredditModerationStream.modmail_conversations()`* to work
- Parameters `allowable_content` and `max_emojis` to *`add()`*, *`add()`*, and *`update()`*, as well as its child classes.

Deprecated

- *`moderator_subreddits()`* as *`Redditor.moderated()`* provides more functionality.
- The file for ModActions (praw/models/modaction.py) has been moved to praw/models/mod_action.py and the previous has been Deprecated.

Expected Changes

- The behavior of *`APIException`* will no longer unicode-escape strings in the next minor release

1.14.4 6.4.0 (2019/09/21)

Added

- *`crosspost()`* support parameter `flair_id` to flair the submission immediately upon crossposting.
- *`crosspost()`* support parameter `flair_text` to set a custom text to the flair immediately upon crossposting.
- *`crosspost()`* support parameter `nsfw` to mark the submission NSFW immediately upon crossposting.
- *`crosspost()`* support parameter `spoiler` to mark the submission as a spoiler immediately upon crossposting.

Fixed

- *`add_community_list()`* has parameter `description` to support unannounced upstream Reddit API changes.
- *`update()`* supports passing a list of *`Subreddit`* objects.

Changed

- Removed `css_class` parameter cannot be used with `background_color`, `text_color`, or `mod_only` constraint on methods:
  - `SubredditFlairTemplates.update()`
  - `SubredditRedditorFlairTemplates.add()`
  - `SubredditLinkFlairTemplates.add()`

Removed

- Drop official support for Python 2.7.
- `Multireddit.rename()` no longer works due to a change in the Reddit API.

1.14.5 6.3.1 (2019/06/10)

Removed

- `SubredditListingMixin.gilded()`, as this was supposed to be removed in 6.0.0 after deprecation in 5.2.0.

1.14.6 6.3.0 (2019/06/09)

Added

- Collections (*Collection* and helper classes).
- *submit()*, *submit_image()*, and *submit_video()* can be used to submit a post directly to a collection.
- `praw.util.camel_to_snake` and `praw.util.snake_case_keys`.
- Comments can now be locked and unlocked via `comment.mod.lock()` and `comment.mod.unlock()`. See: (*ThingModerationMixin.lock()* and *ThingModerationMixin.unlock()*).
- `align` parameter to *SubredditStylesheet.upload_banner_additional_image()*

Changed

- *Reddit.info()* now accepts any non-str iterable for fullnames (not just `list`).
- *Reddit.info()* now returns a generator instead of a list when using the `url` parameter.

1.14.7 6.2.0 (2019/05/05)

Added

- *SubredditStylesheet.upload_banner()*
- *SubredditStylesheet.upload_banner_additional_image()*
- *SubredditStylesheet.upload_banner_hover_image()*
- *SubredditStylesheet.delete_banner()*
- *SubredditStylesheet.delete_banner_additional_image()*
- *SubredditStylesheet.delete_banner_hover_image()*
- *submit()*, *submit_image()*, and *submit_video()* support parameter `nsfw` to mark the submission NSFW immediately upon posting.
- *submit()*, *submit_image()*, and *submit_video()* support parameter `spoiler` to mark the submission as a spoiler immediately upon posting.
- *submit_image()* and *submit_video()* support parameter `timeout`. Default timeout has been raised from 2 seconds to 10 seconds.
- Added parameter `function_kwargs` to *stream_generator()* to pass additional kwargs to `function`.

Fixed

- *Subreddit.random()* returns `None` instead of raising *ClientException* when the subreddit does not support generating random submissions.

Other

- Bumped minimum prawcore version to 1.0.1.

1.14.8 6.1.1 (2019/01/29)

Added

- *set()* supports parameter `flair_template_id` for giving a user redesign flair.

1.14.9 6.1.0 (2019/01/19)

Added

- Add method *Redditor.trophies()* to get a list of the Redditor's trophies.
- Add class *PostFlairWidget*.
- Add attributes `reply_limit` and `reply_sort` to class *Comment*
- Add class *SubredditWidgetsModeration* (accessible through *SubredditWidgets.mod*) and method *add_text_area()*.
- Add class *WidgetModeration* (accessible through the `.mod` attribute on any widget) with methods *update()* and *delete()*.
- Add method *Reddit.put()* for HTTP PUT requests.
- Add methods *add_calendar()* and *add_community_list()*.
- Add methods *add_image_widget()* and *upload_image()*.
- Add method *add_custom_widget()*.
- Add method *add_post_flair_widget()*.
- Add method *add_menu()*.
- Add method *add_button_widget()*.
- Add method *reorder()* to reorder a subreddit's widgets.
- Add *Redditors* (`reddit.redditors`) to provide Redditor listings.
- Add *submit_image()* for submitting native images to Reddit.
- Add *submit_video()* for submitting native videos and videogifs to Reddit.

Changed

- *User.me()* returns `None` in *read_only* mode.
- *SubredditLinkFlairTemplates.__iter__()* uses the v2 flair API endpoint. This change will result in additional fields being returned. All fields that were previously returned will still be returned.
- *SubredditRedditorFlairTemplates.__iter__()* uses the v2 flair API endpoint. The method will still return the exact same items.
- Methods *add()*, *add()*, *update()*, and *update()* can add and update redesign-style flairs with the v2 flair API endpoint. They can still update pre-redesign-style flairs with the older endpoint.

Fixed

- Widgets of unknown types are parsed as `Widget` s rather than raising an exception

1.14.10 6.0.0 (2018/07/24)

Added

- Add method *WikiPage.revision()* to get a specific wiki page revision.
- Added parameter `skip_existing` to *stream_generator()* to skip existing items when starting a stream.
- Add method *Front.best()* to get the front page "best" listing.

- Add *Subreddit.widgets*, *SubredditWidgets*, and widget subclasses like *TextArea* to support fetching Reddit widgets.
- Add method *Submission.mark_visited()* to mark a submission as visited on the Reddit backend.

Fixed

- Fix `RecursionError` on *SubredditEmoji*'s `repr` and `str`.
- *SubredditFilters.add()* and *SubredditFilters.remove()* also accept a *Subreddit* for the `subreddit` parameter.
- Remove restriction which prevents installed (non-confidential) apps from using OAuth2 authorization code grant flow.

Removed

- `Subreddit.submissions` as the API endpoint backing the method is no more. See https://www.reddit.com/r/changelog/comments/7tus5f/update_to_search_api/.

1.14.11 5.4.0 (2018/03/27)

Added

- Add method *patch()* to *Reddit* class to support HTTP PATCH requests.
- Add class *Preferences* to access and update Reddit preferences.
- Add attribute *User.preferences* to access an instance of *Preferences*.
- Add method *Message.delete()*.
- Add class *Emoji* to work with custom subreddit emoji.

Deprecated

- `Subreddit.submissions` as the API endpoint backing the method is going away. See https://www.reddit.com/r/changelog/comments/7tus5f/update_to_search_api/.

Fixed

- Fix bug with positive `pause_after` values in streams provided by *stream_generator()* where the wait time was not reset after a yielded `None`.
- Parse URLs with trailing slashes and no `'comments'` element when creating *Submission* objects.
- Fix bug where `Subreddit.submissions` returns a same submission more than once
- Fix bug where `ListingGenerator` fetches the same batch of submissions in an infinite loop when `'before'` parameter is provided.

Removed

- Removed support for Python 3.3 as it is no longer supported by requests.

1.14.12 5.3.0 (2017/12/16)

Added

- *Multireddit.stream*, to stream submissions and comments from a Multireddit.
- *Redditor.block()*

Fixed

- Now raises `prawcore.UnavailableForLegalReasons` instead of an `AssertionError` when encountering a HTTP 451 response.

1.14.13 5.2.0 (2017/10/24)

Changed

- An attribute on *LiveUpdate* now works as lazy attribute (i.e. populate an attribute when the attribute is first accessed).

Deprecated

- `subreddit.comments.gilded` because there isn't actually an endpoint that returns only gilded comments. Use `subreddit.gilded` instead.

Fixed

- Removed `comment.permalink()` because `comment.permalink` is now an attribute returned by Reddit.

1.14.14 5.1.0 (2017/08/31)

Added

- *Redditor.stream*, with methods *RedditorStream.submissions()* and *RedditorStream.comments()* to stream a Redditor's comments or submissions
- *RedditorStream* has been added to facilitate *Redditor.stream*
- *Inbox.collapse()* to mark messages as collapsed.
- *Inbox.uncollapse()* to mark messages as uncollapsed.
- Raise *ClientException* when calling *refresh()* when the comment does not appear in the resulting comment tree.
- *Submission.crosspost()* to crosspost to a subreddit.

Fixed

- Calling *refresh()* on a directly fetched, deeply nested *Comment* will additionally pull in as many parent comments as possible (currently 8) enabling significantly quicker traversal to the top-most *Comment* via successive *parent()* calls.
- Calling *refresh()* previously could have resulted in a `AttributeError: 'MoreComments' object has no attribute '_replies'` exception. This situation will now result in a *ClientException*.
- Properly handle `BAD_CSS_NAME` errors when uploading stylesheet images with invalid filenames. Previously an `AssertionError` was raised.
- *Submission*'s `gilded` attribute properly returns the expected value from reddit.

1.14.15 5.0.1 (2017/07/11)

Fixed

- Calls to *hide()* and *unhide()* properly batch into requests of 50 submissions at a time.
- Lowered the average maximum delay between inactive stream checks by 4x to 16 seconds. It was previously 64 seconds, which was too long.

1.14.16 5.0.0 (2017/07/04)

Added

- *Comment.disable_inbox_replies()*, *Comment.enable_inbox_replies()* *Submission.disable_inbox_replies()*, and *Submission.enable_inbox_replies()* to toggle inbox replies on comments and submissions.

Changed

- `cloudsearch` is no longer the default syntax for *Subreddit.search()*. `lucene` is now the default syntax so that PRAW's default is aligned with Reddit's default.
- *Reddit.info()* will now take either a list of fullnames or a single URL string.
- *Subreddit.submit()* accepts a flair template ID and text.

Fixed

- Fix accessing *LiveUpdate.contrib* raises `AttributeError`.

Removed

- Iterating directly over *SubredditRelationship* (e.g., `subreddit.banned`, `subreddit.contributor`, `subreddit.moderator`, etc) and *SubredditFlair* is no longer possible. Iterate instead over their callables, e.g. `subreddit.banned()` and `subreddit.flair()`.
- The following methods are removed: `Subreddit.mod.approve`, `Subreddit.mod.distinguish`, `Subreddit.mod.ignore_reports`, `Subreddit.mod.remove`, `Subreddit.mod.undistinguish`, `Subreddit.mod.unignore_reports`.
- Support for passing a *Submission* to *SubredditFlair.set()* is removed.
- The `thing` argument to *SubredditFlair.set()* is removed.
- Return values from *Comment.block()*, *Message.block()*, *SubredditMessage.block()*, *SubredditFlair.delete()*, *friend()*, *Redditor.message()*, *Subreddit.message()*, *select()*, and *unfriend()* are removed as they do not provide any useful information.
- `praw.ini` no longer reads in `http_proxy` and `https_proxy` settings.
- `is_link` parameter of *SubredditRedditorFlairTemplates.add()* and *SubredditRedditorFlairTemplates.clear()*. Use *SubredditLinkFlairTemplates* instead.

1.14.17 4.6.0 (2017/07/04)

The release's sole purpose is to announce the deprecation of the `is_link` parameter as described below:

Added

- *SubredditFlair.link_templates* to manage link flair templates.

Deprecated

- `is_link` parameter of *SubredditRedditorFlairTemplates.add()* and *SubredditRedditorFlairTemplates.clear()*. Use *SubredditLinkFlairTemplates* instead.

1.14.18 4.5.1 (2017/05/07)

Fixed

- Calling *parent()* works on *Comment* instances obtained via *comment_replies()*.

1.14.19 4.5.0 (2017/04/29)

Added

- *unread_count()* to get unread count by conversation state.
- *bulk_read()* to mark conversations as read by conversation state.
- *subreddits()* to fetch subreddits using new modmail.
- *create()* to create a new modmail conversation.
- *read()* to mark modmail conversations as read.
- *unread()* to mark modmail conversations as unread.
- *conversations()* to get new modmail conversations.
- *highlight()* to highlight modmail conversations.
- *unhighlight()* to unhighlight modmail conversations.
- *mute()* to mute modmail conversations.
- *unmute()* to unmute modmail conversations.
- *archive()* to archive modmail conversations.
- *unarchive()* to unarchive modmail conversations.
- *reply()* to reply to modmail conversations.
- *__call__()* to get a new modmail conversation.
- *Inbox.stream()* to stream new items in the inbox.
- Exponential request delay to all streams when no new items are returned in a request. The maximum delay between requests is 66 seconds.

Changed

- *submit()* accepts `selftext=''` to create a title-only submission.
- *Reddit* accepts `requestor_class=cls` for a customized requestor class and `requestor_kwargs={'param': value}` for passing arguments to requestor initialization.
- *comments()*, *submissions()*, and *stream()* accept a `pause_after` argument to allow pausing of the stream. The default value of `None` retains the preexisting behavior.

Deprecated

- `cloudsearch` will no longer be the default syntax for *Subreddit.search()* in PRAW 5. Instead `lucene` will be the default syntax so that PRAW's default is aligned with Reddit's default.

Fixed

- Fix bug where *WikiPage* revisions with deleted authors caused `TypeError`.
- *Submission* attributes `comment_limit` and `comment_sort` maintain their values after making instances non-lazy.

1.14.20 4.4.0 (2017/02/21)

Added

- *LiveThreadContribution.update()* to update settings of a live thread.
- `reset_timestamp` to `limits()` to provide insight into when the current rate limit window will expire.
- *upload_mobile_header()* to upload subreddit mobile header.
- *upload_mobile_icon()* to upload subreddit mobile icon.
- *delete_mobile_header()* to remove subreddit mobile header.
- *delete_mobile_icon()* to remove subreddit mobile icon.
- *LiveUpdateContribution.strike()* to strike a content of a live thread.
- *LiveContributorRelationship.update()* to update contributor permissions for a redditor.
- *LiveContributorRelationship.update_invite()* to update contributor invite permissions for a redditor.
- *LiveThread.discussions()* to get submissions linking to the thread.
- *LiveThread.report()* to report the thread violating the Reddit rules.
- *LiveHelper.now()* to get the currently featured live thread.
- *LiveHelper.info()* to fetch information about each live thread in live thread IDs.

Fixed

- Uploading an image resulting in too large of a request (>500 KB) now raises `prawcore.TooLarge` instead of an `AssertionError`.
- Uploading an invalid image raises *APIException*.
- *Redditor* instances obtained via *moderator* (e.g., `reddit.subreddit('subreddit').moderator()`) will contain attributes with the relationship metadata (e.g., `mod_permissions`).
- *Message* instances retrieved from the inbox now have attributes `author`, `dest` `replies` and `subreddit` properly converted to their appropriate PRAW model.

1.14.21 4.3.0 (2017/01/19)

Added

- *LiveContributorRelationship.leave()* to abdicate the live thread contributor position.
- *LiveContributorRelationship.remove()* to remove the redditor from the live thread contributors.
- `limits()` to provide insight into number of requests made and remaining in the current rate limit window.
- *LiveThread.contrib* to obtain an instance of *LiveThreadContribution*.
- *LiveThreadContribution.add()* to add an update to the live thread.
- *LiveThreadContribution.close()* to close the live thread permanently.
- *LiveUpdate.contrib* to obtain an instance of *LiveUpdateContribution*.
- *LiveUpdateContribution.remove()* to remove a live update.
- *LiveContributorRelationship.accept_invite()* to accept an invite to contribute the live thread.

- *SubredditHelper.create()* and *SubredditModeration.update()* have documented support for `spoilers_enabled`. Note, however, that *SubredditModeration.update()* will currently unset the `spoilers_enabled` value until such a time that Reddit returns the value along with the other settings.
- *spoiler()* and *unspoiler()* to change a submission's spoiler status.

Fixed

- *LiveContributorRelationship.invite()* and *LiveContributorRelationship.remove_invite()* now hit endpoints, which starts with 'api/', for consistency.
- *ModeratorRelationship.update()*, and *ModeratorRelationship.update_invite()* now always remove known unlisted permissions.

1.14.22 4.2.0 (2017/01/07)

Added

- *Subreddit.rules()* to get the rules of a subreddit.
- *LiveContributorRelationship*, which can be obtained through *LiveThread.contributor*, to interact with live threads' contributors.
- *remove_invite()* to remove a moderator invite.
- *LiveContributorRelationship.invite()* to send a contributor invitation.
- *LiveContributorRelationship.remove_invite()* to remove the contributor invitation.

Deprecated

- Return values from *Comment.block()*, *Message.block()*, *SubredditMessage.block()*, *SubredditFlair.delete()*, *friend()*, *Redditor.message()*, *Subreddit.message()*, *select()*, and *unfriend()* will be removed in PRAW 5 as they do not provide any useful information.

Fixed

- *hide()* and *unhide()* now accept a list of additional submissions.
- *replace_more()* is now recoverable. Previously, when an exception was raised during the work done by *replace_more()*, all unreplaced *MoreComments* instances were lost. Now *MoreComments* instances are only removed once their children have been added to the *CommentForest* enabling callers of *replace_more()* to call the method as many times as required to complete the replacement.
- Working with contributors on *SubredditWiki* is done consistently through `contributor` not `contributors`.
- `Subreddit.moderator()` works.
- `live_thread.contributor()` now returns *RedditorList* correctly.

Removed

- `validate_time_filter` is no longer part of the public interface.

1.14.23 4.1.0 (2016/12/24)

Added

- *praw.models.Subreddits.search_by_topic()* to search subreddits by topic. (see: https://www.reddit.com/dev/api/#GET_api_subreddits_by_topic).

- *praw.models.LiveHelper.__call__()* to provide interface to praw.models.LiveThread.__init__.
- *SubredditFilters* to work with filters for special subreddits, like /r/all.
- Added callables for *SubredditRelationship* and *SubredditFlair* so that limit and other parameters can be passed.
- Add *reply()* to *Message* which was accidentally missed previously.
- Add sticky parameter to *CommentModeration.distinguish()* to sticky comments.
- *flair()* to add a submission's flair from an instance of *Submission*.
- *Comment.parent()* to obtain the parent of a *Comment*.
- *opt_in()* and *opt_out()* to *Subreddit* to permit working with quarantined subreddits.
- *LiveUpdate* to represent an individual update in a *LiveThread*.
- Ability to access an individual *LiveUpdate* via reddit.live('THREAD_ID')['UPDATE_ID'].
- *LiveThread.updates()* to iterate the updates of the thread.

Changed

- *me()* now caches its result in order to reduce redundant requests for methods that depend on it. Set use_cache=False when calling to bypass the cache.
- *replace_more()* can be called on *Comment* replies.

Deprecated

- validate_time_filter will be removed from the public interface in PRAW 4.2 as it was never intended to be part of it to begin with.
- Iterating directly over *SubredditRelationship* (e.g., subreddit.banned, subreddit.contributor, subreddit.moderator, etc) and *SubredditFlair* will be removed in PRAW 5. Iterate instead over their callables, e.g. subreddit.banned() and subreddit.flair().
- The following methods are deprecated to be removed in PRAW 5 and are replaced with similar Comment.mod... and Submission.mod... alternatives: Subreddit.mod.approve, Subreddit.mod.distinguish, Subreddit.mod.ignore_reports, Subreddit.mod.remove, Subreddit.mod.undistinguish, Subreddit.mod.unignore_reports.
- Support for passing a *Submission* to *SubredditFlair.set()* will be removed in PRAW 5. Use *flair()* instead.
- The thing argument to *SubredditFlair.set()* is replaced with redditor and will be removed in PRAW 5.

Fixed

- *SubredditModeration.update()* accurately updates exclude_banned_modqueue, header_hover_text, show_media and show_media_preview values.
- Instances of *Comment* obtained through the inbox (including mentions) are now refreshable.
- Searching /r/all should now work as intended for all users.
- Accessing an invalid attribute on an instance of *Message* will raise AttributeError instead of *PRAWException*.

1.14.24 4.0.0 (2016/11/29)

Fixed

- Fix bug where ipython tries to access attribute `_ipython_canary_method_should_not_exist_` resulting in a useless fetch.
- Fix bug where Comment replies becomes `[]` after attempting to access an invalid attribute on the Comment.
- Reddit.wiki[...] converts the passed in page name to lower case as pages are only saved in lower case and non-lower case page names results in a Redirect exception (thanks pcjonathan).

1.14.25 4.0.0rc3 (2016/11/26)

Added

- `implicit` parameter to *`url()`* to support the implicit flow for **installed** applications (see: https://github.com/reddit/reddit/wiki/OAuth2#authorization-implicit-grant-flow)
- *`scopes()`* to discover which scopes are available to the current authentication
- Lots of documentation: http://praw.readthedocs.io/

1.14.26 4.0.0rc2 (2016/11/20)

Fixed

- *`authorize()`* properly sets the session's Authentication (thanks @williammck).

1.14.27 4.0.0rc1 (2016/11/20)

PRAW 4 introduces significant breaking changes. The numerous changes are not listed here, only the feature removals. Please read through *Quick Start* to help with updating your code to PRAW 4. If you require additional help please ask on /r/redditdev or via Slack.

Added

- *`praw.models.Comment.block()`*, *`praw.models.Message.block()`*, and *`praw.models.SubredditMessage.block()`* to permit blocking unwanted user contact.
- *`praw.models.LiveHelper.create()`* to create new live threads.
- *`praw.models.Redditor.unblock()`* to undo a block.
- *`praw.models.Subreddits.gold()`* to iterate through gold subreddits.
- *`praw.models.Subreddits.search()`* to search for subreddits by name and description.
- *`praw.models.Subreddits.stream()`* to obtain newly created subreddits in near-realtime.
- *`praw.models.User.karma()`* to retrieve the current user's subreddit karma.
- `praw.models.reddit.submission.SubmissionModeration.lock` and `praw.models.reddit.submission.SubmissionModeration.unlock` to change a Submission's lock state.
- *`praw.models.reddit.subreddit.SubredditFlairTemplates.delete()`* to delete a single flair template.
- *`praw.models.reddit.subreddit.SubredditModeration.unread()`* to iterate over unread moderation messages.

- *praw.models.reddit.subreddit.ModeratorRelationship.invite()* to invite a moderator to a subreddit.
- *praw.models.reddit.subreddit.ModeratorRelationship.update()* to update a moderator's permissions.
- *praw.models.reddit.subreddit.ModeratorRelationship.update_invite()* to update an invited moderator's permissions.
- *praw.models.Front.random_rising()*, *praw.models.Subreddit.random_rising()* and *praw.models.Multireddit.random_rising()*.
- *WikiPage* supports a revision argument.
- *revisions()* to obtain a list of recent revisions to a subreddit.
- *revisions()* to obtain a list of revisions for a wiki page.
- Support installed-type OAuth apps.
- Support read-only OAuth for all application types.
- Support script-type OAuth apps.

Changed

Note: Only prominent changes are listed here.

- `helpers.comments_stream` is now *praw.models.reddit.subreddit.SubredditStream.comments()*
- `helpers.submissions_between` is now `Subreddit.submissions`. This new method now only iterates through newest submissions first and as a result makes approximately 33% fewer requests.
- `helpers.submission_stream` is now *praw.models.reddit.subreddit.SubredditStream.submissions()*

Removed

- Removed *Reddit*'s `login` method. Authentication must be done through OAuth.
- Removed `praw-multiprocess` as this functionality is no longer needed with PRAW 4.
- Removed non-oauth functions `Message.collapse` and `Message.uncollapse` `is_username_available`.
- Removed captcha related functions.

For changes prior to version 4.0 please see: 3.4.0 changelog

1.15 Contributing to PRAW

PRAW gladly welcomes new contributions. As with most larger projects, we have an established consistent way of doing things. A consistent style increases readability, decreases bug-potential and makes it faster to understand how everything works together.

PRAW follows **PEP 8** and **PEP 257**. The `pre_push.py` script can be used to test for compliance with these PEPs in addition to providing a few other checks. The following are PRAW-specific guidelines in addition to those PEP's.

1.15.1 Code

- Within a single file classes are sorted alphabetically where inheritance permits.
- Within a class, methods are sorted alphabetically within their respective groups with the following as the grouping order:
  - Static methods
  - Class methods
  - Properties
  - Instance Methods
- Use descriptive names for the catch-all keyword argument. E.g., `**other_options` rather than `**kwargs`.

1.15.2 Testing

Contributions to PRAW requires 100% test coverage as reported by Coveralls. If you know how to add a feature, but aren't sure how to write the necessary tests, please open a PR anyway so we can work with you to write the necessary tests.

Running the Test Suite

Github Actions automatically runs all updates to known branches and pull requests. However, it's useful to be able to run the tests locally. The simplest way is via:

```
python setup.py test
```

Without any configuration or modification, all the tests should pass.

Adding and Updating Integration Tests

PRAW's integration tests utilize Betamax to record an interaction with Reddit. The recorded interaction is then replayed for subsequent test runs.

To safely record a cassette without leaking your account credentials, PRAW utilizes a number of environment variables which are replaced with placeholders in the cassettes. The environment variables are (listed in bash export format):

```
export prawtest_client_id=myclientid
export prawtest_client_secret=myclientsecret
export prawtest_password=mypassword
export prawtest_test_subreddit=reddit_api_test
export prawtest_username=myusername
export prawtest_user_agent=praw_pytest
```

By setting these environment variables prior to running `python setup.py test`, when adding or updating cassettes, instances of `mypassword` will be replaced by the placeholder text `<PASSWORD>` and similar for the other environment variables.

To use tokens instead of username/password set `prawtest_refresh_token` instead of `prawtest_password` and `prawtest_username`.

When adding or updating a cassette, you will likely want to force requests to occur again rather than using an existing cassette. The simplest way to rebuild a cassette is to first delete it, and then rerun the test suite.

Please always verify that only the requests you expect to be made are contained within your cassette.

1.15.3 Documentation

- All publicly available functions, classes and modules should have a docstring.
- Use correct terminology. A subreddit's fullname is something like `t5_xyfc7`. The correct term for a subreddit's "name" like python is its display name.

Static Checker

PRAW's test suite comes with a checker tool that can warn you of using incorrect documentation styles (using `.. code::` instead of `.. code-block::`, using `/r/` instead of `r/`, etc.).

class tools.static_word_checks.**StaticChecker**(*replace: bool*)
Run simple checks on the entire document or specific lines.

__init__(*replace: bool*)
Instantiates the class.

Parameters **replace** – Whether or not to make replacements.

check_for_code_statement(*filename: str*, *content: str*) → bool
Checks the code for `.. code::` statements.

Parameters

- **filename** – The name of the file to check & replace.
- **content** – The content of the file

Returns A boolean with the status of the check

check_for_double_syntax(*filename: str*, *content: str*) → bool
Checks a file for double-slash statements (`/r/` and `/u/`).

Parameters

- **filename** – The name of the file to check & replace.
- **content** – The content of the file

Returns A boolean with the status of the check

check_for_noreturn(*filename: str*, *line_number: int*, *content: str*) → bool
Checks a line for `NoReturn` statements.

Parameters

- **filename** – The name of the file to check & replace.
- **line_number** – The line number
- **content** – The content of the line

Returns A boolean with the status of the check

run_checks() → bool
Scan a directory and run the checks.

The directory is assumed to be the praw directory located in the parent directory of the file, so if this file exists in `~/praw/tools/static_word_checks.py`, it will check `~/praw/praw`.

It runs the checks located in the `self.full_file_checks` and `self.line_checks` lists, with full file checks being run first.

Full-file checks are checks that can also fix the errors they find, while the line checks can just warn about found errors.

- Full file checks:
  - *check_for_code_statement()*
  - *check_for_double_syntax()*
- Line checks
  - *check_for_noreturn()*

1.15.4 Files to Update

AUTHORS.rst

For your first contribution, please add yourself to the end of the respective list in the `AUTHORS.rst` file.

CHANGES.rst

For feature additions, bugfixes, or code removal please add an appropriate entry to `CHANGES.rst`. If the `Unreleased` section does not exist at the top of `CHANGES.rst` please add it. See commit 280525c16ba28cdd69cdbb272a0e2764b1c7e6a0 for an example.

1.15.5 See Also

Please also read through: https://github.com/praw-dev/praw/blob/master/.github/CONTRIBUTING.md

1.16 References

- PRAW Source Code.
- Reddit Source Code. This repository has been archived and is no longer updated.
- Reddit API Wiki Page.
- Reddit API Documentation.
- Reddit Help. Frequently asked questions and a knowledge base for the site.
- Reddit Markdown Primer. A guide to the Markdown formatting used on the site.
- Reddit Status. Indicates when Reddit is up or down.
- r/changelog. Significant changes to Reddit's codebase will be announced here in non-developer speak.
- r/redditdev. Ask questions about Reddit's codebase, PRAW, and other API clients here.

1.17 Index

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W