PRAW

Release 4.1.0

December 24, 2016

PRAW's documentation is organized into the following sections:

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

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 2.7, and Python 3.3 to 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.com works is a must. In the event you are not already familiar with Reddit start with their FAQ.

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 Redddit's First Steps Guide to create them.

User Agent A user agent is a unique indentifer 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 satisifed, 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

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:

```python
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.

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

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

```python
# continued from code above

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

# Output: 10 submission
```

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

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 user name, 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:

```python
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: 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. 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 2.7, 3.3, 3.4, and 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:

- *Script Application*
- *Web Application*
- *Installed Application*

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

1.3.1 Script Application

Script applications are the simplest type of application to work with because they don't involve any sort of callback process to obtain an `access_token`.

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

In order to use a **script** 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 **script** application.
>
> **username** The username of the Reddit account used to register the **script** application.

With this information authorizing as `username` using a **script** 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 script application is associated:

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

1.3.2 Web Application

A **web** application is useful for two primary purposes:

- You have a website and want to be able to access Reddit from your users' accounts.
- You want to limit the access one of your PRAW-based programs has to Reddit, or simply do not want to pass your username and password to PRAW.

When registering a **web** 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 **web** application 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 **web** application code flow. It also indicates that the `reddit` instance is now associated with that account.

1.3.3 Installed Application

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`.

The implicit flow is similar, 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 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.

1.3.5 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 **web** and **installed** applications start in the **read only** mode. With such applications **read only** mode is disabled when `authorize()`, `implicit()` are successfully called. **Script** 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
```

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 newewer 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 retreive 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:

(a) In the directory specified by the `XDG_CONFIG_HOME` environment varaible on operating systems that define such an environment variable (some modern Linux distributions).

(b) In the directory specified by `$HOME/.config` if the `HOME` environment variable is defined (Linux and Mac OS systems).

(c) 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

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

# 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.

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
```

1.5 The Reddit Instance

class praw.**Reddit**(*site_name=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:

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

   __init__(*site_name=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.

      Additional keyword arguments will be used to initialize the :class'.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

   auth = None
      An instance of *Auth*.

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

   comment(*id*)
      Return a lazy instance of *Comment* for id.

         Parameters **id** – The ID of the comment.

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

---

domain(*domain*)

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*, *params=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*)

Fetch information about each item in `fullnames`.

> **Parameters fullnames** – A list of fullnames for a comment, submission, or subreddit.
>
> **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`.

---

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')
```

post (*path*, *data=None*, *files=None*, *params=None*)
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 mappin (Default: None).
>
> - **params** – The query parameters to add to the request (Default: None).

random_subreddit (*nsfw=False*)
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*)
Return a lazy instance of [*Redditor*](#) for `name`.

> **Parameters** **name** – The name of the redditor.

request (*method*, *path*, *params=None*, *data=None*, *files=None*)
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 mappin (Default: None).

submission (*id=None*, *url=None*)
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())
```

1.5.1 reddit.front

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

__init__(*reddit*)
Initialize a Front instance.

comments
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='all'*, ***generator_kwargs*)
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*)
　　Return a ListingGenerator for gilded items.

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

hot(***generator_kwargs*)
　　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*)
　　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()
```

parse(*data*, *reddit*)
　　Return an instance of cls from data.

　　　　Parameters

　　　　　　• **data** – The structured data.

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

random_rising(***generator_kwargs*)
　　Return a ListingGenerator for random rising submissions.

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

rising(***generator_kwargs*)
　　Return a ListingGenerator for rising submissions.

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

top(*time_filter='all'*, ***generator_kwargs*)
　　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')
```

validate_time_filter(*time_filter*)
> Raise `ValueError` if time_filter is not valid.

> > **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

1.5.2 reddit.inbox

class praw.models.**Inbox**(*reddit*, *_data*)
> Inbox is a Listing class that represents the Inbox.

> **__init__**(*reddit*, *_data*)
> > Initialize a PRAWModel instance.

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

> **all**(***generator_kwargs*)
> > Return a ListingGenerator for all inbox comments and messages.

> > Additional keyword arguments are passed in the initialization of `ListingGenerator`.

> **comment_replies**(***generator_kwargs*)
> > Return a ListingGenerator for comment replies.

> > Additional keyword arguments are passed in the initialization of `ListingGenerator`.

> **mark_read**(*items*)
> > Mark Comments or Messages as read.

> > > **Parameters items** – A list containing Comments and Messages to be be marked as read relative to the authenticated user's inbox.

> > Requests are batched at 25 items (reddit limit).

> **mark_unread**(*items*)
> > Unmark Comments or Messages as read.

> > > **Parameters items** – A list containing Comments and Messages to be be marked as unread relative to the authenticated user's inbox.

> > Requests are batched at 25 items (reddit limit).

> **mentions**(***generator_kwargs*)
> > 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`.

> **message**(*message_id*)
> > Return a Message corresponding to message_id.

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

messages(***generator_kwargs*)
    Return a ListingGenerator for inbox messages.

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

parse(*data*, *reddit*)
    Return an instance of `cls` from `data`.

        Parameters

- **data** – The structured data.

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

sent(***generator_kwargs*)
    Return a ListingGenerator for sent messages.

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

submission_replies(***generator_kwargs*)
    Return a ListingGenerator for submission replies.

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

unread(*mark_read=False*, ***generator_kwargs*)
    Return a ListingGenerator for unread comments and messages.

        Parameters **mark_read** – Marks the messages as read when they're obtained (Default: False).

        Note: When marking messages as read, the entire batch (up to 100 at a time) is marked as read when fetched. Failure to consume the entire listing may result in missed messages if you only obtain unread messages.

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

1.5.3 reddit.live

class `praw.models.`**LiveHelper**(*reddit*, *_data*)
    Provide a set of functions to interact with LiveThreads.

    __call__(*id*)
        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*, *_data*)
        Initialize a PRAWModel instance.

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

    create(*title*, *description=None*, *nsfw=False*, *resources=None*)
        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.

parse(*data*, *reddit*)
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.

- **reddit** – An instance of [*Reddit*].

1.5.4 reddit.multireddit

class praw.models.**MultiredditHelper**(*reddit*, *_data*)
Provide a set of functions to interact with Multireddits.

__call__(*redditor*, *name*)
Return a lazy instance of [*Multireddit*].

Parameters

- **redditor** – A string or [*Redditor*] instance who owns the multireddit.

- **name** – The name of the multireddit.

__init__(*reddit*, *_data*)
Initialize a PRAWModel instance.

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

create(*display_name*, *subreddits*, *description_md=None*, *icon_name=None*, *key_color=None*, *visibility='private'*, *weighting_scheme='classic'*)
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.

parse(*data*, *reddit*)
> Return an instance of `cls` from `data`.

> > **Parameters**

> > > • **data** – The structured data.

> > > • **reddit** – An instance of [*Reddit*](#).

1.5.5 reddit.subreddit

class praw.models.**SubredditHelper**(*reddit*, *_data*)
> Provide a set of functions to interact with Subreddits.

> **__call__**(*display_name*)
> > Return a lazy instance of [*Subreddit*](#).

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

> **__init__**(*reddit*, *_data*)
> > Initialize a PRAWModel instance.

> > > **Parameters reddit** – An instance of [*Reddit*](#).

> **create**(*name*, *title=None*, *link_type='any'*, *subreddit_type='public'*, *wikimode='disabled'*, ***other_settings*)
> > 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.

> **parse**(*data*, *reddit*)
> > Return an instance of `cls` from `data`.

> > > **Parameters**

> > > > • **data** – The structured data.

> > > > • **reddit** – An instance of [*Reddit*](#).

1.5.6 reddit.subreddits

class praw.models.**Subreddits**(*reddit*, *_data*)
> Subreddits is a Listing class that provides various subreddit lists.

> **__init__**(*reddit*, *_data*)
> > Initialize a PRAWModel instance.

> > > **Parameters reddit** – An instance of [*Reddit*](#).

default(***generator_kwargs*)

> Return a [*ListingGenerator*](#) for default subreddits.

gold(***generator_kwargs*)

> Return a [*ListingGenerator*](#) for gold subreddits.

new(***generator_kwargs*)

> Return a [*ListingGenerator*](#) for new subreddits.

parse(*data*, *reddit*)

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

> > **Parameters**

> > > • **data** – The structured data.

> > > • **reddit** – An instance of [*Reddit*](#).

popular(***generator_kwargs*)

> Return a [*ListingGenerator*](#) for popular subreddits.

recommended(*subreddits*, *omit_subreddits=None*)

> 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*, ***generator_kwargs*)

> 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.

search_by_name(*query*, *include_nsfw=True*, *exact=False*)

> 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*)

> Return list of Subreddits whose topics match `query`.

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

stream()

> Yield new subreddits as they are created.

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

1.5.7 reddit.user

class `praw.models.`**User**(*reddit*)

> The user class provides methods for the currently authenticated user.

__init__(*reddit*)
Initialize a User instance.

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

blocked()
Return a RedditorList of blocked Redditors.

contributor_subreddits(***generator_kwargs*)
Return a ListingGenerator of subreddits user is a contributor of.

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

friends()
Return a RedditorList of friends.

karma()
Return a dictionary mapping subreddits to their karma.

me(*use_cache=True*)
Return a *Redditor* instance for the authenticated user.

> **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*)
Return a ListingGenerator of subreddits the user is a moderator of.

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

multireddits()
Return a list of multireddits belonging to the user.

parse(*data*, *reddit*)
Return an instance of `cls` from `data`.

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

subreddits(***generator_kwargs*)
Return a ListingGenerator of subreddits the user is subscribed to.

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

1.6 Working with PRAW's Models

1.6.1 Comment

class `praw.models.`**Comment**(*reddit*, *id=None*, *_data=None*)
A class that represents a reddit comments.

__init__(*reddit*, *id=None*, *_data=None*)
Construct an instance of the Comment object.

block()
>   Block the user who sent the item.

>>   **Returns** The json response from the server.

>   **Note:** Reddit does not permit blocking users unless you have a [*Comment*](#) or [*Message*](#) from them.

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]

delete()
>   Delete the object.

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]

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.

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.

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

mark_read()
>   Mark the item as read.

mark_unread()
>   Mark the item as unread.

mod
>   An instance of [*CommentModeration*](#).

parent()
>   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:

```python
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')]
```

parse(*data*, *reddit*)
> Return an instance of `cls` from `data`.

> > **Parameters**

> > > • **data** – The structured data.

> > > • **reddit** – An instance of *Reddit*.

permalink(*fast=False*)
> Return a permalink to the comment.

> > **Parameters** **fast** – Return the result as quickly as possible (Default: False).

> In order to determine the full permalink for a comment, the Submission may need to be fetched if it hasn't been already. Set `fast=True` if you want to bypass that possible load.

> A full permalink looks like: /r/redditdev/comments/2gmzqe/praw_https_enabled/cklhv0f

> A fast-loaded permalink for the same comment will look like: /comments/2gmzqe//cklhv0f

refresh()
> Refresh the comment's attributes.

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

replies
> An instance of *CommentForest*.

reply(*body*)
> Reply to the object.

> > **Parameters** **body** – The markdown formatted content for a comment.

> > **Returns** A *Comment* object for the newly created comment.

report(*reason*)
> Report this object to the moderators of its subreddit.

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

save(*category=None*)
> Save the object.

> > **Parameters** **category** – (Gold) The category to save to (Default: None). If your user does not have gold this value is ignored by Reddit.

submission
> Return the Submission object this comment belongs to.

unsave()
> Unsave the object.

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]

1.6.2 LiveThread

class praw.models.**LiveThread**(*reddit*, *id=None*, *_data=None*)
> An individual LiveThread object.

> **__getitem__**(*update_id*)
> > Return a lazy *LiveUpdate* instance.

> > ---
> > **Warning:** At this time, accesing lazy attributes, whose value have not loaded, raises `AttributeError`.
> > ---

> > **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      # raise ``AttributeError``
> > ```

> **__init__**(*reddit*, *id=None*, *_data=None*)
> > Initialize a lazy *LiveThread* instance.

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

> **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*.

> **parse**(*data*, *reddit*)
> > Return an instance of `cls` from `data`.

> > **Parameters**
> > > • **data** – The structured data.
> > > • **reddit** – An instance of *Reddit*.

> **updates**(***generator_kwargs*)
> > Return a *ListingGenerator* yields *LiveUpdate* s.

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

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

1.6.3 LiveUpdate

class praw.models.**LiveUpdate**(*reddit*, *thread_id=None*, *update_id=None*, *_data=None*)
    An individual *LiveUpdate* object.

    __init__(*reddit*, *thread_id=None*, *update_id=None*, *_data=None*)
        Initialize a lazy *LiveUpdate* instance.

        Either thread_id and update_id, or _data must be provided.

> **Warning:** At this time, accesing lazy attributes, whose value have not loaded, raises
> AttributeError.

    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       # raise ``AttributeError``
```

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.

parse(*data*, *reddit*)
    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.

1.6.4 Message

class praw.models.**Message**(*reddit*, *_data*)
    A class for private messages.

    __init__(*reddit*, *_data*)
        Construct an instance of the Message object.

    block()
        Block the user who sent the item.

        Returns The json response from the server.

Note: Reddit does not permit blocking users unless you have a *Comment* or *Message* from them.

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 the item as read.

mark_unread()
    Mark the item as unread.

classmethod parse(*data*, *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.

1.6.5 MoreComments

class `praw.models.`**MoreComments**(*reddit*, *_data*)
    A class indicating there are more comments.

    __init__(*reddit*, *_data*)
        Construct an instance of the MoreComments object.

    comments(*update=True*)
        Fetch and return the comments for a single MoreComments object.

    parse(*data*, *reddit*)
        Return an instance of `cls` from `data`.

        Parameters

- **data** – The structured data.

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

1.6.6 Multireddit

class `praw.models.`**Multireddit**(*reddit*, *_data*)
    A class for users' Multireddits.

    __init__(*reddit*, *_data*)
        Construct an instance of the Multireddit object.

    add(*subreddit*)
        Add a subreddit to this multireddit.

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

comments

> 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='all'*, ***generator_kwargs*)

> 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=None*)

> 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.

delete()

> Delete this multireddit.

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*)

> Return a ListingGenerator for gilded items.
>
> Additional keyword arguments are passed in the initialization of *ListingGenerator*.

hot(***generator_kwargs*)

> 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*)
> 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()
```

parse(*data*, *reddit*)
> Return an instance of `cls` from `data`.
>
> > **Parameters**
> >
> > - **data** – The structured data.
> >
> > - **reddit** – An instance of *Reddit*.

random_rising(***generator_kwargs*)
> Return a ListingGenerator for random rising submissions.
>
> Additional keyword arguments are passed in the initialization of *ListingGenerator*.

remove(*subreddit*)
> Remove a subreddit from this multireddit.
>
> > **Parameters subreddit** – The subreddit to remove from this multi.

rename(*display_name*)
> Rename this multireddit.
>
> > **Parameters display_name** – The new display name for this multireddit. Reddit will generate the `name` field from this display name.

rising(***generator_kwargs*)
> Return a ListingGenerator for rising submissions.
>
> Additional keyword arguments are passed in the initialization of *ListingGenerator*.

static sluggify(*title*)
> Return a slug version of the title.
>
> > **Parameters title** – The title to make a slug of.
>
> Adapted from reddit's utils.py.

top(*time_filter='all'*, ***generator_kwargs*)
> 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.

    - **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`.

validate_time_filter(*time_filter*)
    Raise `ValueError` if `time_filter` is not valid.

> **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

1.6.7 Redditor

class `praw.models.`**Redditor**(*reddit*, *name=None*, *_data=None*)
    A class representing the users of reddit.

    __init__(*reddit*, *name=None*, *_data=None*)
        Initialize a Redditor instance.

        Parameters

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

        - **name** – The name of the redditor.

    comments
        An instance of *SubListing* providing access to comments.

        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='all'*, ***generator_kwargs*)
        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*)
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*.

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

> **Parameters** **note** – A personal note about the user. Requires reddit Gold. (Default: None)

> **Returns** The json response from the server.

Calling this method subsequent times will update the note.

friend_info()
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 Gold.

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=1*)
Gild the Redditor.

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

gilded(***generator_kwargs*)
Return a ListingGenerator for gilded items.

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

gildings(***generator_kwargs*)
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*.

hidden(***generator_kwargs*)
>   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*.

hot(***generator_kwargs*)
>   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 mail permissions.
>
>   **Returns** The json response from the server.

multireddits()
>   Return a list of the redditor's public multireddits.

new(***generator_kwargs*)
>   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()
```

parse(*data*, *reddit*)
>   Return an instance of `cls` from `data`.
>
>   **Parameters**
>
>   - **data** – The structured data.
>
>   - **reddit** – An instance of *Reddit*.

saved(***generator_kwargs*)

> 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`*.

submissions

> An instance of *`SubListing`* providing access to submissions.
>
> For example, to output the title's of top 100 of all time submissions for /u/spez try:

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

top(*time_filter='all'*, ***generator_kwargs*)

> 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:

```python
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')
```

unblock()

> Unblock the Redditor.
>
> > **Returns** The json response from the server.
>
> Blocking must be done from a Message, Comment Reply or Submission Reply.

unfriend()

> Unfriend the Redditor.

upvoted(***generator_kwargs*)

> 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`*.

validate_time_filter(*time_filter*)

> Raise *`ValueError`* if `time_filter` is not valid.
>
> > **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

1.6.8 Submission

class praw.models.**Submission**(*reddit*, *id=None*, *url=None*, *_data=None*)

A class for submissions to reddit.

__init__(*reddit*, *id=None*, *url=None*, *_data=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]

---

comment_sort = None

Specify the sort order for comments

comments

An instance of *CommentForest*.

delete()

Delete the object.

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]

---

duplicates(***generator_kwargs*)

Return a ListingGenerator for the submission's duplicates.

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

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.

flair

An instance of *SubmissionFlair*.

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.

gilded(***generator_kwargs*)
    Return a ListingGenerator for gilded items.

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

hide()
    Hide Submission.

static id_from_url(*url*)
    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 *ClientException* if URL is not a valid submission URL.

mod
    An instance of *SubmissionModeration*.

parse(*data*, *reddit*)
    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.

report(*reason*)
    Report this object to the moderators of its subreddit.

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

save(*category=None*)
    Save the object.

>   **Parameters category** – (Gold) The category to save to (Default: None). If your user does not have gold this value is ignored by Reddit.

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

unhide()
    Unhide Submission.

unsave()
    Unsave the object.

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]

1.6.9 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 like so:

```
for submission in reddit.subreddit('redditdev+learnpython').top('all'):
    print(submission)
```

> Subreddits can be filtered from combined listings as follows:

```
for submission in reddit.subreddit('all-redditdev').new():
    print(submission)
```

> **__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
> An instance of *SubredditRelationship*.

> For example to ban a user try:

```
reddit.subreddit('SUBREDDIT').banned.add('NAME', reason='a reason')
```

comments
> 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

An instance of *ContributorRelationship*.

controversial(*time_filter='all'*, ***generator_kwargs*)

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')
```

filters

An instance of *SubredditFilters*.

flair

An instance of *SubredditFlair*.

Provides the interface 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)
```

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*)

Return a ListingGenerator for gilded items.

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

hot(***generator_kwargs*)

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 mail permissions.

Returns The json response from the server.

mod
An instance of *SubredditModeration*.

moderator
An instance of *ModeratorRelationship*.

muted
An instance of *SubredditRelationship*.

new(***generator_kwargs*)
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()
```

parse(*data*, *reddit*)
Return an instance of `cls` from `data`.

Parameters

- **data** – The structured data.

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

quaran
An instance of *SubredditQuarantine*.

This property is named `quaran` because `quarantine` is an Subreddit attribute returned by Reddit to indicate whether or not a Subreddit is quarantined.

random()
Return a random Submission.

random_rising(***generator_kwargs*)
Return a ListingGenerator for random rising submissions.

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

rising(***generator_kwargs*)
Return a ListingGenerator for rising submissions.

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

search(*query*, *sort='relevance'*, *syntax='cloudsearch'*, *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: cloudsearch).
> - **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:

```python
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.

stream
    An instance of *SubredditStream*.

stylesheet
    An instance of *SubredditStylesheet*.

submissions(*start=None*, *end=None*, *extra_query=None*)
    Yield submissions created between timestamps `start` and `end`.

> **Parameters**
>
> - **start** – A UNIX timestamp indicating the earliest creation time of submission yielded during the call. A value of `None` will consider all submissions older than `end` (Default: None).
> - **end** – A UNIX timestamp indicating the latest creation time of a submission yielded during the call. A value of `None` will consider all submissions newer than `start` (Default: None).
> - **extra_query** – A cloudsearch query that will be combined via (and `timestamp:start..end EXTRA_QUERY`) to futher filter results (Default: None).

Submissions are yielded newest first.

Example: If you want to obtain all submissions to `/r/politics` on November 8, 2016 PST. First you need to determine the start and end dates as UNIX timestamps. Using http://www.epochconverter.com/ those timestamps are 1478592000, and 1478678400 respectively. The following outputs all such submissions' titles.

```python
subreddit = reddit.subreddit('politics')
for submission in subreddit.submissions(1478592000, 1478678400):
    print(submission.title)
```

As of this writing there are 809 results.

submit(*title*, *selftext=None*, *url=None*, *resubmit=True*, *send_replies=True*)
    Add a submission to the subreddit.

>   **Parameters**
>
>   - **title** – The title of the submission.
>
>   - **selftext** – The markdown formatted content for a text submission.
>
>   - **url** – The URL for a link submission.
>
>   - **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).
>
>   **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:

```
url = 'https://praw.readthedocs.io'
reddit.subreddit('reddit_api_test').submit(url=url)
```

subscribe(*other_subreddits=None*)
    Subscribe to the subreddit.

>   **Parameters other_subreddits** – When provided, also subscribe to the provided list of subreddits.

top(*time_filter='all'*, ***generator_kwargs*)
    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.

unsubscribe(*other_subreddits=None*)
    Unsubscribe from the subreddit.

>   **Parameters other_subreddits** – When provided, also unsubscribe to the provided list of subreddits.

validate_time_filter(*time_filter*)
    Raise [ValueError](#) if time_filter is not valid.

> **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

wiki
>   An instance of *SubredditWiki*.

1.6.10 WikiPage

class `praw.models.`**`WikiPage`**(*reddit*, *subreddit*, *name*, *revision=None*, *_data=None*)
>   An individual WikiPage object.
>
>   **`__init__`**(*reddit*, *subreddit*, *name*, *revision=None*, *_data=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*, *reason=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.
>
>   **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*.
>
>   **mod**
>   >   An instance of *WikiPageModeration*.
>
>   **parse**(*data*, *reddit*)
>   >   Return an instance of `cls` from `data`.
>   >
>   >   > **Parameters**
>   >   >   - **data** – The structured data.
>   >   >   - **reddit** – An instance of *Reddit*.
>
>   **revisions**(***generator_kwargs*)
>   >   Return a generator for page revisions.
>   >
>   >   Additional keyword arguments are passed in the initialization of *ListingGenerator*.

1.7 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

1.7.1 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*.

exception praw.exceptions.**APIException**(*error_type*, *message*, *field*)
Indicate exception that involve responses from Reddit's API.

> **__getitem__**
> x.__getitem__(y) <==> x[y]

> **__init__**(*error_type*, *message*, *field*)
> 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.

exception praw.exceptions.**ClientException**
Indicate exceptions that don't involve interaction with Reddit's API.

> **__getitem__**
> x.__getitem__(y) <==> x[y]

> **__init__**
> x.__init__(...) initializes x; see help(type(x)) for signature

exception praw.exceptions.**PRAWException**
The base PRAW Exception that all other exception classes extend.

> **__getitem__**
> x.__getitem__(y) <==> x[y]

> **__init__**
> x.__init__(...) initializes x; see help(type(x)) for signature

1.8 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.

1.8.1 SubmissionFlair

class praw.models.reddit.submission.**SubmissionFlair**(*submission*)
Provide a set of functions pertaining to Submission flair.

> **__init__**(*submission*)
> Create a SubmissionFlair instance.
>
> > **Parameters submission** – The submission associated with the flair functions.

> **choices**()
> Return list of available flair choices.

---

select (*flair_template_id*, *text=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).

1.8.2 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 generator for Redditors and their associated flair.

            Parameters **redditor** – Yield at most a single *Redditor* instance.

        This method is intended to be used like:

```
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.

    __iter__ ()
        Iterate through the Redditors and their associated flair.

> **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer iterating over `subreddit.flair` instead of `subreddit.flair`.

    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** – An instance of *Redditor*, or the name of a Redditor.

> **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.

set(*redditor=None*, *text=''*, *css_class=''*, *thing=None*)
Set flair for a Redditor.

> **Parameters**
>
> - **redditor** – (Required) An instance of Redditor or a string that is the name of a Redditor.
>
> - **text** – The flair text to associate with the Redditor or Submission (Default: '').
>
> - **css_class** – The css class to associate with the flair html (Default: '').

This method can only be used by an authenticated user who is a moderator of the associated Subreddit.

> **Warning:** The use of this method to set the flair of a *Submission* is deprecated and will be removed in PRAW 5. Use *flair()* instead.

> **Warning:** The use of the keyword argument `thing` is deprecated and will be removed in PRAW 5. Use `redditor` instead.

Example:

```
reddit.subreddit('redditdev').flair.set('bboe', 'PRAW author')
```

update(*flair_list*, *text=''*, *css_class=''*)
Set or clear the flair for many Redditors at once.

> **Parameters**
>
> - **redditor_flair_list** – Each item in this list should be either: a Redditor name string, a Redditor, or a dictionary containing the keys `user`, `flair_text`, and `flair_css_class`. The `user` key should map to a Redditor name string, or a Redditor. 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.

1.8.3 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.

__iter__()
Iterate through the user flair templates.

add(*text*, *css_class=''*, *text_editable=False*, *is_link=False*)
Add a 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).
>
> - **is_link** – (boolean) When True, add a link flair template rather than a Redditor flair template (default: False).

clear(*is_link=False*)
Remove all flair templates from the subreddit.

> **Parameters is_link** – (boolean) When True, clear all link flair templates rather than a Redditor flair templates (default: False).

delete(*template_id*)
Remove a flair template provided by `template_id`.

static flair_type(*is_link*)
Return LINK_FLAIR or USER_FLAIR depending on `is_link` value.

update(*template_id*, *text*, *css_class=''*, *text_editable=False*)
Update the flair templated provided by `template_id`.

> **Parameters**
>
> - **template_id** – The flair template to update.
>
> - **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).

1.8.4 CommentModeration

class praw.models.reddit.comment.**CommentModeration**(*comment*)
Provide a set of functions pertaining to Comment moderation.

__init__(*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.

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 priviliges 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.

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.

remove(*spam=False*)
Remove a [*Comment*](#) or [*Submission*](#).

> **Parameters** **spam** – When True, use the removal to help train the Subreddit's spam filter (Default: False)

undistinguish()
Remove mod, admin, or special distinguishing on object.

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.

1.8.5 SubmissionModeration

class praw.models.reddit.submission.**SubmissionModeration**(*submission*)
Provide a set of functions pertaining to Submission moderation.

__init__(*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.

contest_mode(*state=True*)
Set contest mode for the comments of this submission.

> **Parameters** **state** – (boolean) True enables contest mode, False, disables.

> **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.

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 priviliges 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.

flair(*text='', css_class=''*)
  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: '').

  This method can only be used by an authenticated user who is a moderator of the Submission's Subreddit.

  Example:

  ```
  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.

lock()
  Lock the submission.

nsfw()
  Mark as not safe for work.

remove(*spam=False*)
  Remove a *Comment* or *Submission*.

  Parameters spam – When True, use the removal to help train the Subreddit's spam filter (Default: False)

sfw()
  Mark as safe for work.

sticky(*state=True*, *bottom=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.

suggested_sort(*sort='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 on object.

---

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.

unlock()
> Unlock the submission.

1.8.6 SubredditModeration

class praw.models.reddit.subreddit.**SubredditModeration**(*subreddit*)
> Provides a set of moderation functions to a Subreddit.

> **__init__**(*subreddit*)
> > Create a SubredditModeration instance.

> > **Parameters subreddit** – The subreddit to moderate.

> **accept_invite**()
> > Accept an invitation as a moderator of the community.

> static **approve**(*thing*)
> > DEPRECATED.

> > > **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling comment.mod.approve(), submission.mod.approve().

> static **distinguish**(*thing*, *how='yes'*, *sticky=False*)
> > DEPRECATED.

> > > **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling comment.mod.distinguish(), submission.mod.distinguish().

> **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*.

> static **ignore_reports**(*thing*)
> > DEPRECATED.

> > > **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling comment.mod.ignore_reports(), submission.mod.ignore_reports().

> **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.

> **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.

modqueue(*only=None*, ***generator_kwargs*)
    Return a ListingGenerator for comments/submissions in the modqueue.

        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*.

static remove(*thing*, *spam=False*)
    DEPRECATED.

> **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling `comment.mod.remove()`, `submission.mod.remove()`.

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*.

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*.

static undistinguish(*thing*)
    DEPRECATED.

> **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling `comment.mod.undistinguish()`, `submission.mod.undistinguish()`.

static unignore_reports(*thing*)
    DEPRECATED.

> **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling `comment.mod.unignore_reports()`, `submission.mod.unignore_reports()`.

unmoderated(***generator_kwargs*)
    Return a ListingGenerator for unmoderated submissions.

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

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.

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_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.

- **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 gold-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.

- **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`.

- **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.

Unspecified settings will maintain their current value.

1.8.7 WikiPageModeration

class praw.models.reddit.wikipage.**WikiPageModeration**(*wikipage*)
:   Provides a set of moderation functions for a WikiPage.

    __init__(*wikipage*)
    :   Create a WikiPageModeration instance.

        Parameters wikipage – The wikipage to moderate.

    add(*redditor*)
    :   Add an editor to this WikiPage.

        Parameters redditor – A string or [*Redditor*](#) instance.

    remove(*redditor*)
    :   Remove an editor from this WikiPage.

        Parameters redditor – A string or [*Redditor*](#) instance.

    settings()
    :   Return the settings for this WikiPage.

    update(*listed*, *permlevel*, ***other_settings*)
    :   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 *add*), (2) only mods may edit and view

        - **other_settings** – Additional keyword arguments to pass.

        Returns The updated WikiPage settings.

1.8.8 ContributorRelationship

class praw.models.reddit.subreddit.**ContributorRelationship**(*subreddit*, *relationship*)
:   Provides methods to interact with a Subreddit's contributors.

    Contributors of a subreddit can be iterated through like so:

    ```python
    for contributor in reddit.subreddit('redditdev').contributors():
        print(contributor)
    ```

    __call__(*redditor=None*, ***generator_kwargs*)
    :   Return a generator for Redditors belonging to this relationship.

Parameters **redditor** – 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.

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.

__iter__()
Iterate through the Redditors belonging to this relationship.

> **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling `subreddit.banned(limit=None)` instead of `subreddit.banned` and similar for other relationships.

add(*redditor*, ***other_settings*)
Add `redditor` to this relationship.

Parameters **redditor** – A string or *Redditor* instance.

leave()
Abdicate the contributor position.

remove(*redditor*)
Remove `redditor` from this relationship.

Parameters **redditor** – A string or *Redditor* instance.

1.8.9 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').moderators():
    print(moderator)
```

__call__(*redditor=None*, ***generator_kwargs*)
Return a generator for Redditors belonging to this relationship.

Parameters **redditor** – 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.

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.

__iter__()
> Iterate through the Redditors belonging to this relationship.

> Warning: (Deprecated) This method will be removed in PRAW 5. Prefer calling `subreddit.banned(limit=None)` instead of `subreddit.banned` and similar for other relationships.

add(*redditor*, *permissions=None*, ***other_settings*)
> Add or invite `redditor` to be a moderator of the subreddit.

> > **Parameters**
> > - **redditor** – A string 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.

invite(*redditor*, *permissions=None*, ***other_settings*)
> Invite `redditor` to be a moderator of the subreddit.

> > **Parameters**
> > - **redditor** – A string 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.

leave()
> Abdicate the moderator position (use with care).

remove(*redditor*)
> Remove `redditor` from this relationship.

> > **Parameters redditor** – A string or *Redditor* instance.

update(*redditor*, *permissions=None*)
> Update the moderator permissions for `redditor`.

> > **Parameters**
> > - **redditor** – A string 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.

update_invite(*redditor*, *permissions=None*)
> Update the moderator invite permissions for `redditor`.

> > **Parameters**
> > - **redditor** – A string 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.

1.8.10 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:

```python
for ban in reddit.subreddit('redditdev').banned():
    print('{}: {}'.format(ban, ban.note))
```

    __call__(*redditor=None*, ***generator_kwargs*)
    : Return a generator for Redditors belonging to this relationship.

        Parameters redditor – 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.

        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.

    __iter__()
    : Iterate through the Redditors belonging to this relationship.

        > **Warning:** (Deprecated) This method will be removed in PRAW 5. Prefer calling
        > subreddit.banned(limit=None) instead of subreddit.banned and similar for other re-
        > lationships.

    add(*redditor*, ***other_settings*)
    : Add redditor to this relationship.

        Parameters redditor – A string or *Redditor* instance.

    remove(*redditor*)
    : Remove redditor from this relationship.

        Parameters redditor – A string or *Redditor* instance.

1.8.11 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.fiters. For example to add a filter run:

```python
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.

1.8.12 SubredditQuarantine

class `praw.models.reddit.subreddit.`**SubredditQuarantine**(*subreddit*)
> Provides submission and comment streams.
>
> **__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
```

1.8.13 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`()

        Yield new comments as they become available.

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

    `submissions`()

        Yield new submissions as they become available.

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

1.8.14 SubredditStylesheet

class `praw.models.reddit.subreddit.`**`SubredditStylesheet`**(*subreddit*)

    Provides a set of stylesheet functions to a Subreddit.

    `__call__`()

        Return the subreddit's stylesheet.

    `__init__`(*subreddit*)

        Create a SubredditStylesheet instance.

            Parameters **`subreddit`** – The subreddit associated with the stylesheet.

    `delete_header`()

        Remove the current subreddit header image.

        Succeeds even if there is no header image.

    `delete_image`(*name*)

        Remove the named image from the subreddit.

        Succeeds even if the named image does not exist.

    `update`(*stylesheet*, *reason=None*)

        Update the subreddit's stylesheet.

            Parameters **`stylesheet`** – The CSS for the new stylesheet.

    `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`.

    `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.

1.8.15 SubredditWiki

class praw.models.reddit.subreddit.**SubredditWiki**(*subreddit*)
Provides a set of moderation functions to a Subreddit.

__getitem__(*page_name*)
Lazily return the WikiPage for the subreddit named page_name.

__init__(*subreddit*)
Create a SubredditModeration instance.

Parameters **subreddit** – The subreddit to moderate.

__iter__()
Iterate through the pages of the wiki.

create(*name*, *content*, *reason=None*, ***other_settings*)
Create a new wiki page.

Parameters

- **name** – The name of the new WikiPage. This name will be normalied.

- **content** – The content of the new WikiPage.

- **reason** – (Optional) The reason for the creation.

- **other_settings** – Additional keyword arguments to pass.

revisions(***generator_kwargs*)
Return a generator for recent wiki revisions.

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

1.8.16 Auth

class praw.models.**Auth**(*reddit*, *_data*)
Auth provides an interface to Reddit's authorization.

__init__(*reddit*, *_data*)
Initialize a PRAWModel instance.

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

authorize(*code*)
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*, *expires_in*, *scope*)
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 class:.*ClientException* if *Reddit* was initialized for a non-installed application type.

parse(*data*, *reddit*)
    Return an instance of `cls` from `data`.

    Parameters

    - **data** – The structured data.

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

scopes()
    Return a set of scopes included in the current authorization.

    For read-only authorizations this should return `{'*'}`.

url(*scopes*, *state*, *duration='permanent'*, *implicit=False*)
    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.

1.8.17 CommentForest

class praw.models.comment_forest.**CommentForest**(*submission*, *comments=None*)
    A forest of comments starts with multiple top-level comments.

    Each of these comments can be a tree of replies.

__getitem__(*index*)
    Return the comment at position `index` in the list.

__init__(*submission*, *comments=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__()
    Return the number of top-level comments in the forest.

list()
> Return a flattened list of all Comments.
>
> This list may contain *MoreComments* instances if *replace_more()* was not called first.

replace_more(*limit=32*, *threshold=0*)
> 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.replace_more()
```

1.8.18 CommentHelper

class praw.models.listing.mixins.subreddit.**CommentHelper**(*subreddit*)
> Provide a set of functions to interact with a subreddit's comments.
>
> **__call__**(***generator_kwargs*)
> > 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*)
> > Initialize a CommentHelper instance.
>
> **gilded**(***generator_kwargs*)
> > Return a ListingGenerator for gilded items.
> >
> > Additional keyword arguments are passed in the initialization of *ListingGenerator*.
>
> **parse**(*data*, *reddit*)
> > Return an instance of `cls` from `data`.
> >
> > > **Parameters**
> > >
> > > • **data** – The structured data.
> > >
> > > • **reddit** – An instance of *Reddit*.

1.8.19 DomainListing

class praw.models.**DomainListing**(*reddit*, *domain*)

    Provide a set of functions to interact with domain listings.

    __init__(*reddit*, *domain*)

        Initialize a DomainListing instance.

            Parameters

- **reddit** – An instance of Reddit.

- **domain** – The domain for which to obtain listings.

    controversial(*time_filter='all'*, ***generator_kwargs*)

        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*)

        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*)

        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()
```

    parse(*data*, *reddit*)

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

            Parameters

- **data** – The structured data.

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

random_rising(***generator_kwargs*)
   Return a ListingGenerator for random rising submissions.

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

rising(***generator_kwargs*)
   Return a ListingGenerator for rising submissions.

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

top(*time_filter='all'*, ***generator_kwargs*)
   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')
```

validate_time_filter(*time_filter*)
   Raise `ValueError` if `time_filter` is not valid.

   > **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

1.8.20 ListingGenerator

class praw.models.**ListingGenerator**(*reddit*, *url*, *limit=100*, *params=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*, *url*, *limit=100*, *params=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__()
> Permit ListingGenerator to operate as an iterator.

next()
> Permit ListingGenerator to operate as a generator in py2.

parse(*data*, *reddit*)
> Return an instance of `cls` from `data`.
>
> > **Parameters**
> >
> > • **data** – The structured data.
> >
> > • **reddit** – An instance of [*Reddit*](#).

1.8.21 RedditBase

class praw.models.reddit.base.**RedditBase**(*reddit*, *_data*)
> Base class that represents actual Reddit objects.

__init__(*reddit*, *_data*)
> Initialize a RedditBase instance (or a subclass).
>
> > **Parameters reddit** – An instance of [*Reddit*](#).

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*.

parse(*data*, *reddit*)
> Return an instance of `cls` from `data`.
>
> > **Parameters**
> >
> > • **data** – The structured data.
> >
> > • **reddit** – An instance of [*Reddit*](#).

1.8.22 SubListing

class praw.models.listing.mixins.redditor.**SubListing**(*reddit*, *base_path*, *subpath*)
> Helper class for generating SubListing objects.

__init__(*reddit*, *base_path*, *subpath*)
> 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='all'*, ***generator_kwargs*)
> 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*)
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*)
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()
```

parse(*data*, *reddit*)
Return an instance of `cls` from `data`.

> **Parameters**
>
> > * **data** – The structured data.
> >
> > * **reddit** – An instance of *Reddit*.

top(*time_filter='all'*, ***generator_kwargs*)
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')
```

validate_time_filter(*time_filter*)
>   Raise [ValueError](#) if time_filter is not valid.

>   **Warning:** (Deprecated) This method will not be part of the public interface in PRAW 4.1+.

1.8.23 SubredditMessage

class praw.models.**SubredditMessage**(*reddit*, *_data*)
>   A class for messages to a subreddit.

>   **__init__**(*reddit*, *_data*)
>   >   Construct an instance of the Message object.

>   **block**()
>   >   Block the user who sent the item.

>   >   >   **Returns** The json response from the server.

>   >   **Note:** Reddit does not permit blocking users unless you have a [Comment](#) or [Message](#) from them.

>   **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 the item as read.

>   **mark_unread**()
>   >   Mark the item as unread.

>   **mute**(*_unmute=False*)
>   >   Mute the sender of this SubredditMessage.

>   **parse**(*data*, *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.

>   **unmute**()
>   >   Unmute the sender of this SubredditMessage.

1.9 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*:

```python
import praw

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()*.

1.9.1 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:

```python
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:

```python
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:

```python
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:

```python
from praw.models import MoreComments
for top_level_comment in submission.comments:
    if isinstance(top_level_comment, MoreComments):
        continue
    print(top_level_comment.body)
```

The preferred way is to use the *replace_more()* method of the *CommentForest*. Calling *replace_more()* will replace or remove all the *MoreComments* objects in the comment 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` paramter. Additionally a

---

threshold parameter can be set to only perform replacement of *MoreComments* instances that represent a mini-mum number of comments; it defaults to 0, meaning all *MoreComments* instances will be replaced up to limit.

We can rewrite the snippet above as the following, which simply removes all *MoreComments* instances from the comment forest:

```python
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.

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:

```python
submission.comments.replace_more(limit=0)
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 breath-first traversal using a queue:

```python
submission.comments.replace_more(limit=0)
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:

```python
submission.comments.replace_more(limit=0)
for comment in submission.comments.list():
    print(comment.body)
```

Now 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 com-ments extracted via PRAW. This discrepancy is normal as that count includes deleted, removed, and spam comments.

1.10 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*.

```python
#!/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."""
    if len(sys.argv) < 2:
        print('Usage: {} SCOPE...'.format(sys.argv[0]))
        return 1

    reddit = praw.Reddit(client_id='YOUR_CLIENT_ID',
                         client_secret='YOUR_CLIENT_SECRET',
                         redirect_uri='http://localhost:8080',
                         user_agent='praw_refresh_token_example')
    state = str(random.randint(0, 65000))
    url = reddit.auth.url(sys.argv[1:], state, 'permanent')
    print(url)

    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())
```

1.11 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.

1.11.1 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:

```python
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:

```python
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:

```python
from urllib.parse import quote_plus

import praw
```

Next we extract a few constants that are used in our script:

```python
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:

```python
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:
            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
```

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:

```python
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)
```

Finally we need to call `main` only in the cases that this script is the one being executed:

```python
if __name__ == '__main__':
    main()
```

The Complete LMGTFY Bot

The following is the complete LMGTFY Bot:

```python
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):
    # 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()
```

1.12 Change Log

1.12.1 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 4.1+ 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*.

1.12.2 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).

1.12.3 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/

1.12.4 4.0.0rc2 (2016/11/20)

Fixed

- *authorize()* properly sets the session's Authentication (thanks @williammck).

1.12.5 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 in the praw-dev/praw channel on gitter.

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 *`praw.models.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

1.13 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**. You can use `pre_push.py` to test for compliance with these PEP's and a few other checks. The following are PRAW-specific guidelines in addition to those PEP's.

1.13.1 Code

- Objects are sorted alphabetically.

- Things should maintain the same name throughout the code.

- Things should be stored in the same data structure throughout the code.

- `**kwargs` should be given descriptive names.

1.13.2 Testing

- All additions to the code require 100% test coverage. If you're not sure where to begin with testing, ask.

1.13.3 Documentation

- All publicly available functions, classes and modules should have a docstring.

- Use correct terminology. A subreddits name is something like ' t5_xyfc7'. The correct term for a subreddit's "name" like python is its display name.

1.14 References

- PRAW's Source Code
- Reddit's Source Code

- Reddit's API Wiki Page

- Reddit's API Documentation

- Reddit Markdown Primer

- reddit.com's FAQ

- reddit.com's Status Twitterbot. Tweets when Reddit goes 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

p

Symbols