PRAW

Release 4.0.0b21

November 04, 2016

Contents

PRAW, an acronym for "Python Reddit API Wrapper", is a python package that allows for simple access to Reddit's API. PRAW aims to be easy to use and internally follows all of Reddit's API rules. With PRAW there's no need to introduce `sleep` calls in your code. Give your client an appropriate user agent and you're set.

CHAPTER 1

Installation

PRAW is supported on python 2.7, 3.3, 3.4, and 3.5. The recommended way to install PRAW is via pip.

```
pip install --pre praw
```

Note: The `--pre` flag is needed to install PRAW4 as it is not yet the official version.

To install the latest development version of PRAW4 run the following instead:

```
pip install --upgrade https://github.com/praw-dev/praw/archive/praw4.zip
```

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

CHAPTER 2

PRAW Discussion and Support

For those new to python, or would otherwise consider themselves a python beginner, please consider asking questions on the r/learnpython subreddit. There are wonderful people there who can help with general python and simple PRAW related questions.

Otherwise, there are a few official places to ask questions about PRAW:

/r/redditdev is the best place on Reddit to ask PRAW related questions. This subreddit is for all Reddit API related discussion so please tag submissions with *[PRAW4]*. Please perform a search on the subreddit first to see if anyone has similar questions.

Real-time chat can be conducted via the praw-dev/praw channel on gitter.

Please do not directly message any of the contributors via Reddit, email, or gitter unless they have indicated otherwise. We strongly encourage everyone to help others with their questions.

Please file bugs and feature requests as issues on GitHub after first searching to ensure a similar issue was not already filed. If such an issue already exists please give it a thumbs up reaction. Comments to issues containing additional information are certainly welcome.

Note: This project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

CHAPTER 3

Documentation

PRAW's documentation is located at http://praw.readthedocs.io/en/praw4/.

CHAPTER 4

History

August 2010: Timothy Mellor created a github project called `reddit_api`.

March 2011: The python package `reddit` was registered and uploaded to pypi.

December 2011: Bryce Boe took over as maintainer of the `reddit` package.

June 2012: Bryce renamed the project `PRAW` and the repository was relocated to the newly created praw-dev organization on GitHub.

February 2016: Bryce began work on PRAW4, a complete rewrite of PRAW.

CHAPTER 5

License

PRAW4's source is provided under the Simplified BSD License.

- Copyright (c), 2016, Bryce Boe

Earlier versions of PRAW were released under GPLv3.

CHAPTER 6

Donations

Please consider donating to PRAW's maintainer via https://cash.me/$praw.

CHAPTER 7

Table of Contents

7.1 Getting Started

In this section, we are going over everything you need to know to start building bots using Python Reddit API Wrapper (PRAW). It's fun and easy. Let's get started.

7.1.1 Prerequisites

- **Python**: Obviously, 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 a great place to get help.
- **reddit**: A basic understanding of how reddit.com works is a must, although we can safely assume that a person who is reading the documentation of a reddit API wrapper must have that covered. Just in case you don't, here is the FAQ.

  You would also need a reddit account to register apps with reddit before you can use their API through PRAW.
- **user-agent**: A user-agent is a string that helps the reddit server identify the source of the requests. To use reddit API, you need a unique and descriptive user-agent string. 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-agent strings on reddit API's wiki page.
- **client ID & client secret**: These two are also mandatory for PRAW4. If you don't already have them, follow the "First Steps" section on this reddit API wiki page.

That's pretty much it! You are ready to learn how to do some of the most common tasks of reddit bot building!

7.1.2 Common Tasks

Get a `Reddit` instance

You need an instance of the `Reddit` class to do *anything* with PRAW. And there are "two kinds" of `Reddit` instance you can create, the read-only one and the regular one. They differ in the amount of settings information needed.

The read-only `Reddit` instance

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

1. user agent
2. client ID
3. client secret

You may choose to provide these by passing in three key-word arguments when calling the initializer of the `Reddit` class: `user_agent`, `client_id`, and `client_secret`. For example:

```
import praw

my_user_agent = "my user agent"
my_client_id = "my client ID"
my_client_secret = "my client secret"

reddit = praw.Reddit(user_agent=my_user_agent,
                     client_id=my_client_id,
                     client_secret=my_client_secret)
```

Just like that, now you've got a Reddit instance. Keep in mind though, this instance is in *read-only* mode.

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

You can do limited (read-only) things with this instance like "getting 10 'hot' submissions of the 'learnpython' subreddit":

```
# continue from code above

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

# Output: 10 submission titles
```

You can do some things with a read-only instance, but not a lot. In most cases you would need a regular instance.

The regular `Reddit` instance

In order to create a regular `Reddit` instance, two additional pieces of settings information is required:

4. your reddit user name, and
5. your reddit password

Again, you may choose to provide these by passing in key-word arguments, `username` and `password`, when you call the `Reddit` initializer, like this:

```
import praw

my_user_agent = "my user agent"
my_client_id = "my client ID"
my_client_secret = "my client secret"
my_username = "my username"
my_password = "my password"

reddit = praw.Reddit(user_agent=my_user_agent,
                     client_id=my_client_id,
                     client_secret=my_client_secret,
                     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:

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

Nonetheless, if you are uncomfortable of hard coding your credentials, we have two other options for you.

Providing Configuration Options

We have learned that five pieces of settings information are needed in order to create a regular `Reddit` instance:

1. user agent
2. client ID
3. client secret
4. your reddit user name
5. your reddit password

And we have been passing these as key-word arguments to the `Reddit` initializer. If you look at the source, however, you may notice that the `Reddit` initializer does not directly ask for any of these parameters. They are all passed along when creating a `Config` instance.

So what happens if you don't pass any arguments when calling `Reddit()`? Then the `Config` class will look for those settings in two locations in the following order of priority:

1. environment variables of the settings names prefixed with `praw_`. Specifically, these:
   - `praw_user_agent`
   - `praw_client_id`
   - `praw_client_secret`
   - `praw_username`
   - `praw_password`

   For example, you can invoke your scirpt like this:

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

2. in the `praw.ini` file you provide. The section name for these settings should be specified with an environment variable named `praw_site`; if no such environment variable is set, the default section name is `DEFAULT`. You can put your `praw.ini` file in one or both of the following places (both will be read if present):
   (a) the working directory when you invoke your script
   (b) your OS's config directory (for Linux, this is `$XDG_CONFIG_HOME` or `$HOME/.config`; for Windows, this is `${APPDATA}`)

   If you don't know how to write ini files, follow this example.

Get a subreddit

To get a `Subreddit` instance, all you need to know is the subreddit's display name. Pass that name when calling the `subreddit` method of your `Reddit` instance. For example:

```
# assuming you have a Reddit instance referenced by 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 ...
```

Get submissions from a subreddit

Now that you have a `Subreddit` instance, you can get some submissions (`Submission` instances) from it! There are several ways of sorting all the submissions of a subreddit: hot, new, top, etc. A `Subreddit` instance has a method for each of these sorting approaches, namely, these:

`controversial` `gilded` `hot` `new` `rising` `top`

Each of these methods will return `ListingGenerator`, something that you can iterate through. For example:

```
# assuming you have a Subreddit instance referenced by subreddit
for submission in subreddit.hot(limit=10):
    print(submission.title) # Output: the title of the submission
    print(submission.ups) # Output: upvote count
    print(submission.id) # Output: the ID of the submission
    print(submission.url) # Output: the URL the submission points to
                          # or the the submission URL if it's a self post
```

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

```
# assuming you have a Reddit instance referenced by 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/......")
```

Get redditors

There are several ways to get a redditor (a `Redditor` instance), two of the most common ones are:

- via the `author` attribute of a `Submission` instance
- call the `redditor` method on a `Reddit` instance

For example:

```
# assuming you have a Reddit instance referenced by reddit
# assuming you have a Submission instance referenced by submission
redditor1 = submission.author
print(redditor1.name) # Output: name of the redditor

redditor2 = reddit.redditor('bboe')
print(redditor2.link_karma) # Output: bboe's karma
```

Get comments

Submissions have a `comments` attribute that is a `CommentForest` instance. That instance is iterable and represents the top-level comments. If you instead want to iterate over *all* comments you can get a list of comments via the `list` method of a `CommentForest` instance. For example:

```
# assuming you have a Reddit instance referenced by reddit
# assuming you have a Submission instance referenced by submission
top_level_comments = list(submission.comments)
all_comments = submission.comments.list()
```

As you may be aware there will periodically be `MoreComments` instances scattered throughout the forest. Replace those at any time by calling the `replace_more` method on the `CommentForest` instances.

Get available attributes of an object

If you have a PRAW object, be it `Submission` or `Comment`, and you want to see what attributes are available and their values, use the built-in `vars` function of python. For example:

```
import pprint

# assuming you have a Reddit instance referenced by 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 to only make API calls when/if the information is needed. Here, before the print line, `submission` points to a lazy `Submission` object. When we try to print its title, information is needed, so it ceased to be lazy – PRAW makes the actual API call at this point. Now it is a good time to print out all the available attributes and their values!

7.1.3 Next Steps

Now you know the basics of PRAW, the next steps are of course creating bots! Follow these complete tutorials to kickstart your bot-building journey!

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.

LMGTFY Bot

The goal of the LMGTFY Bot is to point users in the right direction when they ask a simple question that is unlikely to be upvoted or answered by other users.

Two examples of such questions are:

1. "What is the capital of Canada?"
2. "How many feet are in a yard?"

Once we identify these questions, the LMGTFY Bot will reply to the submission with an appropriate lmgtfy link. For the example questions those links are:

1. http://lmgtfy.com/?q=What+is+the+capital+of+Canada%3F
2. http://lmgtfy.com/?q=How+many+feet+are+in+a+yard%3F

Step 1: Getting Started Access to Reddit's API requires a set of OAuth2 credentials. Those credentials are obtained by registering an application with Reddit. To register an application and receive a set of OAuth2 credentials please follow only the "First Steps" section of Reddit's OAuth2 Quick Start Example wiki page.

Once the credentials are obtained we can begin writing the LMGTFY Bot. Start by creating an instance of *Reddit*:

```
import praw

reddit = praw.Reddit(user_agent='LMGTFY (by /u/USERNAME)',
                     client_id='CLIENT_ID', client_secret="CLIENT_SECRET",
                     username='USERNAME', password='PASSWORD')
```

In addition to the OAuth2 credentials, the username and password of the Reddit account that registered the application are required.

Note: This example demonstrates use of a *script* type application. For other application types please see Reddit's wiki page OAuth2 App Types.

Step 2: Monitoring New Submissions to /r/AskReddit PRAW provides a convenient way to obtain new submissions to a given subreddit. To indefinitely iterate over new submissions to a subreddit add:

```
subreddit = reddit.subreddit('AskReddit')
for submission in subreddit.stream.submissions():
    # do something with submission
```

Replace *AskReddit* with the name of another subreddit if you want to iterate through its new submissions. Additionally multiple subreddits can be specified by joining them with pluses, for example `AskReddit+NoStupidQuestions`. All subreddits can be specified using the special name `all`.

Step 3: Analyzing the Submission Titles Now that we have a stream of new submissions to /r/AskReddit, it is time to see if their titles contain a simple question. We naïvely define a simple question as:

1. It must contain no more than ten words.
2. It must contain one of the phrases "what is", "what are", or "who is".

Warning: These naïve criteria result in many false positives. It is strongly recommended that you develop more precise heuristics before launching a bot on any popular subreddits.

First we filter out titles that contain more than ten words:

```
if len(submission.title.split()) > 10:
        return
```

We then check to see if the submission's title contains any of the desired phrases:

```
questions = ['what is', 'who is', 'what are']
normalized_title = submission.title.lower()
for question_phrase in questions:
    if question_phrase in normalized_title:
        # do something with a matched submission
        break
```

String comparison in python is case sensitive. As a result, we only compare a normalized version of the title to our lower-case question phrases. In this case, "normalized" means only lower-case.

The `break` at the end prevents us from matching more than once on a single submission. For instance, what would happen without the `break` if a submission's title was "Who is or what are buffalo?"

Step 4: Automatically Replying to the Submission The LMGTFY Bot is nearly complete. We iterate through submissions, and find ones that appear to be simple questions. All that is remaining is to reply to those submissions with an appropriate lmgtfy link.

First we will need to construct a working lmgtfy link. In essence we want to pass the entire submission title to lmgtfy. However, there are certain characters that are not permitted in URLs or have other . For instance, the space character, ' ', is not permitted, and the question mark, '?', has a special meaning. Thus we will transform those into their URL-safe representation so that a question like "What is the capital of Canada?" is transformed into the link `http://lmgtfy.com/?q=What+is+the+capital+of+Canada%3F)`.

There are a number of ways we could accomplish this task. For starters we could write a function to replace spaces with pluses, +, and question marks with `%3F`. However, there is even an easier way; using an existing built-in function to do so.

Add the following code where the "do something with a matched submission" comment is located:

```
from urllib.parse import quote_plus

reply_template = '[Let me google that for you](http://lmgtfy.com/?q={})'

url_title = quote_plus(submission.title)
reply_text = reply_template.format(url_title)
```

Note: This example assumes the use of Python 3. For Python 2 replace `from urllib.parse import quote_plus` with `from urllib import quote_plus`.

Now that we have the reply text, replying to the submission is easy:

```
submission.reply(reply_text)
```

If all went well, your comment should have been made. If your bot account is brand new, you will likely run into rate limit issues. These rate limits will persist until that account acquires sufficient karma.

Step 5: Cleaning Up The Code While we have a working bot, we have added little segments here and there. If we were to continue to do so in this fashion our code would be quite unreadable. Let's clean it up some.

The first thing we should do is put all of our import statements at the top of the file. It is common to list built-in packages before third party ones:

```
from urllib.parse import quote_plus

import praw
```

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

```
QUESTIONS = ['what is', 'who is', 'what are']
REPLY_TEMPLATE = '[Let me google that for you](http://lmgtfy.com/?q={})'
```

We then extract the segment of code pertaining to processing a single submission into its own function:

```
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:

```
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:

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

The Complete LMGTFY Bot The following is the complete LMGTFY Bot:

```
from urllib.parse import quote_plus

import praw

QUESTIONS = ['what is', 'who is', 'what are']
REPLY_TEMPLATE = '[Let me google that for you](http://lmgtfy.com/?q={})'
```

```
def main():
    reddit = praw.Reddit(user_agent='LMGTFY (by /u/USERNAME)',
                         client_id='CLIENT_ID', client_secret="CLIENT_SECRET",
                         username='USERNAME', password='PASSWORD')

    subreddit = reddit.subreddit('AskReddit')
    for submission in subreddit.stream.submissions():
        process_submission(submission)


def process_submission(submission):
    # 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()
```

Comment Extraction and Parsing

A common use for Reddit's API is to extract comments from submissions and use them to perform keyword or phrase analysis.

As always, you need to begin by creating an instance of *Reddit*:

```
import praw

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()*.

Extracting comments with PRAW

Assume we want to process the comments for this submission: https://www.reddit.com/r/funny/comments/3g1jfi/buttons/

We first need to obtain a submission object. We can do that either with the entire URL:

```
submission = reddit.submission(url='https://www.reddit.com/r/funny/comments/3g1jfi/buttons/')
```

or with the submission's ID which comes after *comments/* in the URL:

```
submission = reddit.submission(id='3g1jfi')
```

With a submission object we can then interact with its *CommentForest* through the submission's *comments* attribute. A *CommentForest* is a list of top-level comments each of which contains a *CommentForest* of replies.

If we wanted to output only the `body` of the top level comments in the thread we could do:

```
for top_level_comment in submission.comments:
    print(top_level_comment.body)
```

While running this you will most likely encounter the exception `AttributeError: 'MoreComments' object has no attribute 'body'`. This submission's comment forest contains a number of *MoreComments* objects. These objects represent the "load more comments", and "continue this thread" links encountered on the website. While we could ignore *MoreComments* in our code, like so:

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

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 minimum 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:

```
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:

```
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:

```
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:

```
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 comments extracted via PRAW. This discrepancy is normal as that count includes deleted, removed, and spam comments.

7.2 Change Log

7.2.1 4.0.0bX

PRAW 4 introduces significant breaking changes. The numerous changes are not listed here, only the feature removals. Please read through Getting Started 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.
- 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.
- Remove non-oauth functions `Message.collapse` and `Message.uncollapse`.

For changes prior to version 4.0 please see: 3.4.0 changelog

7.3 Code Overview

Here you will find an overview of PRAW's objects and methods, but not the objects attributes which are generated dynamically from reddit's responses and are thus impossible to accurately describe statically.

7.3.1 Top Level Classes

class `praw.`**Reddit**(*site_name=None*, ***config_settings*)

Provide convenient access to reddit's API.

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 `Config` object. This can be used to specify configuration settings during instantiation of the `Reddit` instance. For more details please see: https://praw.readthedocs.org/en/stable/pages/configuration_files.html

Required settings are:

•client_id

•client_secret (for installed applications set this value to `None`)

•user_agent

auth = **None**

An instance of *Auth*.

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.

front = None
An instance of *Front*.

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*.

live = None
An instance of *LiveHelper*.

multireddit = None
An instance of *MultiredditHelper*.

post (*path*, *data=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.
- **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*)
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.

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*.

subreddits = None
An instance of *Subreddits*.

user = None
An instance of *User*.

7.3.2 Models

Provide the PRAW models.

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

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 `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*.

url(*scopes*, *state*, *duration='permanent'*)
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** – (web app only) 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 for installed apps as only temporary tokens can be generated for them.

class `praw.models.`**Comment**(*reddit*, *id=None*, *_data=None*)

A class that represents a reddit comments.

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 they you have an Inboxable item 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.

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.

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** – The category to save to (Default: None).

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]

---

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

Initialize a Front instance.

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Parameters

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

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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

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

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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).

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

class `praw.models.`**Listing**(*reddit*, *_data*)
A listing is a collection of RedditBase instances.

Initialize a PRAWModel instance.

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

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

Parameters

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

class praw.models.**ListingGenerator**(*reddit*, *url*, *limit=100*, *params=None*)
Instances of this class generate RedditBase instances.

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.

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*.

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

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*.

class praw.models.**LiveThread**(*reddit*, *_data*)
An individual LiveThread object.

Initialize a RedditBase instance.

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*.

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

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 they you have an Inboxable item 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*.

class praw.models.**ModAction**(*reddit*, *_data*)
Represent a moderator action.

Initialize a RedditBase instance.

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*.

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

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*.

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

Construct an instance of the Multireddit object.

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

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

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Parameters

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

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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.

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

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*.

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

Initialize a Redditor instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **name** – The name of the redditor.

comments
An attribute representing the comments made by the Redditor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

submissions
An attribute representing the submissions made by the Redditor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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

class `praw.models.`**RedditorList**(*reddit*, *_data*)
A list of Redditors. Works just like a regular list.

Initialize a BaseList instance.

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

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

Parameters

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

class `praw.models.`**Submission**(*reddit*, *id=None*, *url=None*, *_data=None*)
A class for submissions to reddit.

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]

---

comments
An instance of *CommentForest*.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

hide()
Hide Submission.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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*.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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** – The category to save to (Default: None).

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

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 to the `ListingGenerator` constructor.

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]

---

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

class praw.models.**Subreddit**(*reddit*, *display_name=None*, *_data=None*)
A class for Subreddits.

Initialize a Subreddit instance.

Parameters

- **reddit** – An instance of *Reddit*.
- **display_name** – The name of the subreddit.

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 to the `ListingGenerator` constructor.

flair
An instance of *SubredditFlair*.

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 to the `ListingGenerator` constructor.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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*.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

Parameters

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

random()
Return a random Submission.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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

stream
An instance of *SubredditStream*.

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.

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.

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 to the `ListingGenerator` constructor.

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

wiki
An instance of *SubredditWiki*.

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

Initialize a PRAWModel instance.

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

create(*name*, *link_type*, *subreddit_type*, *title*, *wikimode*, ***other_settings*)
Create a new subreddit.

Parameters

- **link_type** – The types of submissions users can make. One of `any`, `link`, `self`.
- **name** – The name for the new subreddit.
- **subreddit_type** – One of `archived`, `employees_only`, `gold_only`, `gold_restricted`, `private`, `public`, `restricted`.
- **title** – The title of the subreddit.
- **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*.

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

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 they you have an Inboxable item 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*.

unmute()
Unmute the sender of this SubredditMessage.

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

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 default subreddits.

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).

stream()
Yield new subreddits as they are created.

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

class praw.models.**User**(*reddit*, *_data*)
The user class provides methods for the currently authenticated user.

Initialize a PRAWModel instance.

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

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 to the `ListingGenerator` constructor.

friends()
Return a RedditorList of friends.

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

me()
Return a Redditor instance for the authenticated user.

moderator_subreddits(***generator_kwargs*)
Return a ListingGenerator of subreddits the user is a moderator of.

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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 to the `ListingGenerator` constructor.

class praw.models.**WikiPage**(*reddit*, *subreddit*, *name*, *_data=None*)
An individual WikiPage object.

Construct an instance of the WikiPage object.

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*.

7.3.3 Submission Utility Classes

class praw.models.comment_forest.**CommentForest**(*submission*)
A forest of comments starts with multiple top-level comments.

Each of these comments can be a tree of replies.

Initialize a CommentForest instance.

Parameters **submission** – An instance of *Subreddit* that is the parent of the comments.

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.

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

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).

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

Create a SubmissionModeration instance.

Parameters **submission** – The submission to moderate.

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.

lock()
Lock the submission.

nsfw()
Mark as not safe for work.

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).

unlock()
Lock the submission.

7.3.4 Subreddit Utility Classes

class praw.models.reddit.subreddit.**SubredditFlair**(*subreddit*)
Provide a set of functions to interact with a Subreddit's flair.

Create a SubredditFlair instance.

Parameters **subreddit** – The subreddit whose flair to work with.

delete_all()
Delete all Redditor flair in the Subreddit.

Returns List of dictionaries indicating the success or failure of each delete.

set(*thing*, *text=''*, *css_class=''*)
Set flair for a Redditor or Submission.

Parameters

- **thing** – An instance of Redditor or Submission, or a string. When a string is provided it will be treated as 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.

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.

class praw.models.reddit.subreddit.**SubredditFlairTemplates**(*subreddit*)
Provide functions to interact with a Subreddit's flair templates.

Create a SubredditFlairTemplate instance.

Parameters **subreddit** – The subreddit whose flair templates to work with.

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).

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

Create a SubredditModeration instance.

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

approve(*thing*)
Approve a Comment or Submission.

Parameters **thing** – An instance of 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(*thing*, *how='yes'*)
Distinguish a Comment or Submission.

Parameters

- **thing** – An instance of Comment or Submission.
- **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.

ignore_reports(*thing*)
Ignore future reports on a Comment or Submission.

Parameters **thing** – An instance of 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.

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

Additional keyword arguments are passed to the `ListingGenerator` constructor.

See `unread` for unread moderator messages.

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

Parameters

- **thing** – An instance of Comment or Submission.
- **spam** – When True, use the removal to help train the Subreddit's spam filter (Default: False)

settings()
Return a dictionary of the subreddit's current settings.

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

Returns The json response from the server.

unignore_reports(*thing*)
Resume receiving future reports on a Comment or Submission.

Parameters **thing** – An instance of Comment or Submission.

Future reports on this Comment or Submission will cause notifications, and appear in the various moderation listings.

unread(***generator_kwargs*)
Return a ListingGenerator for unread moderator messages.

Additional keyword arguments are passed to the `ListingGenerator` constructor.

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_modqueue_banned** – 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** – Expand media previews on comments pages.
- **show_thumbnails** – Show thumbnails on submissions.
- **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.

class `praw.models.reddit.subreddit.`**SubredditRelationship**(*subreddit*, *relationship*)

Represents a relationship between a redditor and subreddit.

Create a SubredditRelationship instance.

Parameters

- **subreddit** – The subreddit for the relationship.
- **relationship** – The name of the relationship.

add(*redditor*)

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.

class praw.models.reddit.subreddit.**SubredditStream**(*subreddit*)
Provides submission and comment streams.

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 historial comments will initially be returned.

submissions()
Yield new submissions as they become available.

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

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

Create a SubredditModeration instance.

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

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.

7.3.5 Other Classes

class praw.config.**Config**(*site_name*, ***settings*)
A class containing the configuration for a reddit site.

Initialize a Config instance.

short_url
Return the short url or raise a ClientException when not set.

class praw.objector.**Objector**(*reddit*)
The objector builds RedditBase objects.

Initialize an Objector instance.

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

kind(*instance*)
Return the kind from the instance class.

Parameters **instance** – An instance of a subclass of RedditBase.

objectify(*data*)
Create RedditBase objects from data.

Parameters **data** – The structured data.

register(*kind*, *cls*)
Register a class for a given kind.

Parameters

- **kind** – The kind in the parsed data to map to `cls`.
- **cls** – A RedditBase class.

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

7.3.6 Exceptions

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

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

Construct an 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.

7.4 Contributor Guidelines

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. The following are PRAW-specific guidelines in to those PEP's.

7.4.1 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.

7.4.2 Testing

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

7.4.3 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 subreddits "name" like python is its display name.

CHAPTER 8

Other Relevant Pages

- 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

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W