Rocket Snake Documentation

Release 0.1.5

Hugo Berg

Aug 10, 2017

Contents

Contents:

CHAPTER 1

Rocket Snake

A python async API client for the https://rocketleaguestats.com/ api. Get an api key here.

Status

Stable and supports all the RLS features.

Documentation?

Check the docs directory or read the online documentation on readthedocs.

CHAPTER 2

Installation

At the command line:

```
$ pip install rocket_snake
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv rocket_snake
$ pip install rocket_snake
```

CHAPTER 3

Usage

To use Rocket Snake in a project:

```
import asyncio
import rocket_snake as rs

from pprint import pprint


async def example_function():

    client = rs.RLS_Client("API KEY GOES HERE")

    print("\nPlaylists:")
    pprint(await client.get_playlists())
    print("\nSeasons:")
    pprint(await client.get_seasons())
    print("\nPlatforms:")
    pprint(await client.get_platforms())
    print("\nTiers:")
    pprint(await client.get_tiers())


print("Creating and starting an asyncio event loop...")

my_loop = asyncio.get_event_loop()
my_loop.run_until_complete(example_function())

print("The event loop has now exited after executing the example.")
```

CHAPTER 4

API Reference

The following section outlines the API of Rocket Snake.

The Client

class `rocket_snake.`**`RLS_Client`**(*api_key: str = None*, *auto_rate_limit: bool = True*, *event_loop: asyncio.events.AbstractEventLoop = None*, *_api_version: int = 1*)

Represents the client, does everything. Initialize with api key and some other settings if you want to.

Parameters

- **`api_key`** (`str`) – The key for the https://rocketleaguestats.com api. If not supplied, an InvalidArgumentException will be thrown.
- **`auto_rate_limit`** (`bool`, default is `True`.) – If the api should automatically delay execution of request to satisfy the default ratelimiting. When this is True automatic ratelimiting is enabled.
- **`event_loop`** (`asyncio.AbstractEventLoop`) – The asyncio event loop that should be used. If not supplied, the default one returned by `asyncio.get_event_loop()` is used.
- **`_api_version`** – What version endpoint to use. Do not change if you don't know what you're doing.
- **`_api_version`** – `int`, default is `1`.

`get_platforms`()

Gets the supported platforms for the api.

:return The platforms. :rtype `list` of `str`.

`get_player`(*unique_id: str*, *platform: str*)

Gets a single player from the api for a single player.

Parameters

- **unique_id** (`str`.) – The string to search for. Depending on the platform parameter, this can represent Xbox Gamertag, Xbox user ID, steam 64 ID, or PSN username.
- **platform** (One of the platform constants in *`rocket_snake.constants`*, they are all `str`.) – The platform to search on. This should be one of the platforms defined in rocket_snake/constants.py.

:return A `data_classes.Player` object. :rtype A `data_classes.Player`, which is the player that was requested. :raise: *`exceptions.APINotFoundError`* if the player could be found.

get_players (*unique_id_platform_pairs: list*)

Does what *`RLS_Client.get_player()`* does but for up to 10 players at once.

> **Warning:** This function can take really long to execute, sometimes up to 15 seconds or more. This is because the API automatically updates the users' data from Rocket League itself, and sometimes doesn't. There is currently no way of finding out how long using this function will take, as the API sometimes doesn't update the users' data, and therefore returns the data quickly.

Parameters **unique_id_platform_pairs** (A `list` of **:tuples:‘tuple‘s of unique ids and platform, where both the unique ids and platforms are strings. The platform strings can be found in :mod:‘rocket_snake.constants‘**, and the unique ids are of the same type as what *`RLS_Client.get_player()`* uses. Example: `[("ExampleUniqueID1", constants.STEAM), ("ExampleUniqueID1OnXBOX", constants.XBOX1)]`) – The users you want to search for. These are specified by unique id and platform.

:return The players that could be found. :rtype A `list` of `data_classes.Player` objects.

If a player could not be found, the corresponding index (the index in the `unique_id_platform_pairs` `list`) in the returned `list` will be None.

get_playlists ()

Gets the supported playlists for the api.

:return The supported playlists (basically gamemodes, separate per platform) for the api. :rtype A `list` of `data_classes.Playlists`.

get_ranked_leaderboard (*playlist*)

Gets the leaderboard for ranked playlists from RLS.

Parameters **playlist** (A `data_classes.Playlist` or `int` if you pass a playlist id.) – The playlist you want to get a leaderboard for.

:return The leaderboard, that is, the top players in the requested ranked playlist. The list is usually around 100 players long.

:rtype A `list` of `data_classes.Player` objects, where the first one is the one with the highest rank in the requested playlist and current season, and the list is descending.

get_seasons ()

Gets the supported seasons for the api.

:return The supported seasons for the api. One of them has `Season.time_ended == None`, and `Season.is_c` which means it's the current season.

:rtype A `list` of `data_classes.Seasons`.

get_stats_leaderboard(*stat_type: str*)
Gets a list of the top 100 rocket league players according to a specified stat.

Parameters **stat_type** (One of the LEADERBOARD_* constants in *rocket_snake.constants*.) – What statistic you want to get a leaderboard for.

:return A ordered list of Player objects, where the first one is the one with the highest stat (descending). :rtype A `list` of `data_classes.Player` objects, where the first one is the one with the highest amount of the requested stat, and the list is descending.

get_tiers()
Gets the supported tiers for the api.

:return The supported tiers for the api. :rtype A `list` of `data_classes.Tiers`.

search_player(*display_name: str*, *get_all: bool = False*)
Searches for a displayname and returns the results, this does not search all of Rocket League, but only the https://rocketleaguestats.com database.

Parameters

- **display_name** (`str`) – The displayname you want to search for.
- **get_all** – Whether to get all search results or not. If this is True, the function may take many seconds to return, since it will get all the search results from the API one page at a time. If this is False, the function will only return with the first (called "page" in the http api) 20 results or less.

:type `bool`, default is `False`. :return The search results. :rtype A `list` of `data_classes.Player` objects, where the first one is the top result.

If the search didn't return any players, this `list` is empty (`[]`).

The Exceptions

This module (`rocket_snake.exceptions`) defines the exceptions that are specific to Rocket Snake. All of them are subclasses of some `builtins` exception, but not all of them are direct subclasses.

class `exceptions.`**NoAPIKeyError**

A subclass of `ValueError`, and is raised when an API key isn't provided to the *RLS_Client*.

class `exceptions.`**APIServerError**

A subclass of `ConnectionError`, and is raised when the client gets an error when trying to request something from the API server.

class `exceptions.`**APINotFoundError**

A subclass of `ConnectionError`, and is raised when the API server can't find what was requested (e.g. if a player with the requested displayname doesn't exists)

class `exceptions.`**APIBadResponseCodeError**

A subclass of `ConnectionError`, and is raised when the API returns a response code that isn't successful, but can't be identified as a more specific error.

class `exceptions.`**RatelimitError**

A subclass of `APIBadResponseCodeError`, and is raised when the *RLS_Client* gets ratelimited by the API server but didn't handle ratelimiting at all or not properly.

class `exceptions.`**InvalidAPIKeyError**

A subclass of `APIBadResponseCodeError`, and is raised when the *`RLS_Client`* has been initialised with an invalid API key and tries to execute a request to the API server.

The Constants

This module (`rocket_snake.constants`) defines constants that are used when requesting information from the API.

Note: The value of these should not be hardcoded in your code, since these might change at any time.

These constants are all uppercase. Here is the rundown:

Platform related constants	
Name	Description
`STEAM`	This is the string that represents Steam as a platform.
`PS4`	^ But for Playstation 4.
`XBOX1`	^ But for Xbox One.
`ALL_PLATFORMS`	A `set` of the previous platform strings.
`STEAM_ID`	This is the ID of the steam platform string.
`PS4_ID`	^ But for Playstation 4.
`XBOX1_ID`	^ But for Xbox One.
`ALL_IDS`	A `set` of the previous platform IDs.
`ID_PLATFORM_LUT`	A `dict` with the members of `ALL_IDS` as keys and the members of `ALL_PLATFORMS` as values.
`PLATFORM_ID_LUT`	The inverse of `ID_PLATFORM_LUT`, that is, platforms as keys and IDs as values.

Leaderboard related constants	
Name	Description
`LEADERBOARD_WINS`	This is the string that represents a leaderboard based/filtered on number of wins.
`LEADERBOARD_GOALS`	^ But for number of goals.
`LEADERBOARD_MVPS`	^ But for number of MVPs.
`LEADERBOARD_SAVES`	^ But for number of saves.
`LEADERBOARD_SHOTS`	^ But for number of shots.
`LEADERBOARD_ASSISTS`	^ But for number of assists
`LEADERBOARD_TYPES`	A `set` of all the previous leaderboard filter types.

Platform related constants	
Name	Description
`RANKED_DUEL_ID`	This is the ID of the ranked duels playlist.
`RANKED_DOUBLES_ID`	^ But for the ranked doubles playlist.
`RANKED_SOLO_STANDARD_ID`	^ But for the ranked solo standard playlist.
`RANKED_STANDARD_ID`	^ But for the ranked standard playlist.
`RANKED_PLAYLISTS_IDS`	A `set` of all the previous playlist IDs.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/drummersbrother/rocket_snake/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Rocket Snake could always use more documentation, whether as part of the official Rocket Snake docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/drummersbrother/rocket_snake/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *rocket_snake* for local development.

1. Fork the *rocket_snake* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/rocket_snake.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv rocket_snake
   $ cd rocket_snake/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 3.5 and 3.6. Check https://travis-ci.org/drummersbrother/rocket_snake/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 6

Credits

Development Lead

- Hugo Berg <hb11002@icloud.com>

Contributors

None yet. Why not be the first?

CHAPTER 7

History

CHAPTER 8

0.1.5 (2017-08-10)

- Fix possible authorisation key leak in error logs.
- Fix indexing on `RankedSeason` objects.

CHAPTER 9

0.1.4 (2017-07-24)

- Use the correct loop for async timeouts.

CHAPTER 10

0.1.3 (2017-07-24)

- Add proper CI.
- Improve the documentation.
- Stop checking SSL certs to allow machines with broken cert validation chains.

CHAPTER 11

0.1.2 (2017-07-21)

- First release on PyPI.

CHAPTER 12

0.1.0 (2017-07-21)

- First release on TestPyPI.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index