BrawlStats Documentation

Release v4.1.0

SharpBit

Jan 04, 2021

Contents

- This library is a sync and async wrapper for the Brawl Stars API.
- Python 3.5.3 or later is required.

CHAPTER 1

Features

- Easy to use with an object oriented design.
- Use the same client for sync and async usage.
- Get a player profile and battlelog.
- Get a club and its members.
- Get the top 200 rankings for players, clubs, or a specific brawler.
- Get information about maps and more!
- Get information about current available brawlers.

CHAPTER 2

Installation

Install the latest stable build:

```
pip install brawlstats
```

2.1 API Reference

2.1.1 Client

class brawlstats.**Client**(*token*, *session=None*, *timeout=30*, *is_async=False*, ***options*)

A sync/async client class that lets you access the Brawl Stars API

Parameters

- **token** (*str*) – The API Key that you can get from https://developer.brawlstars.com
- **session** (*Union[requests.Session, aiohttp.ClientSession], optional*) – Use a current session or a make new one, by default None
- **timeout** (*int, optional*) – How long to wait in seconds before shutting down requests, by default 30
- **is_async** (*bool, optional*) – Setting this to `True` makes the client async, by default False
- **loop** (*asyncio.window_events._WindowsSelectorEventLoop, optional*) – The event loop to use for asynchronous operations, by default None
- **connector** (*aiohttp.TCPConnector, optional*) – Pass a TCPConnector into the client (aiohttp), by default None
- **debug** (*bool, optional*) – Whether or not to log info for debugging, by default False
- **prevent_ratelimit** (*bool, optional*) – Whether or not to wait between requests to prevent being ratelimited, by default False

- **base_url** (*str, optional*) – Sets a different base URL to make request to, by default None

__ainit__()
Task created to run *get_brawlers* asynchronously

get_battle_logs(*tag: brawlstats.utils.bstag*, *use_cache=True*) → brawlstats.models.BattleLog
Gets a player's battle logs.

Parameters

- **tag** (*str*) – A valid player tag. Valid characters: 0289PYLQGRJCUV
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A player battle object with all of its attributes.

Return type *BattleLog*

get_brawlers(*use_cache=True*) → brawlstats.models.Brawlers
Gets available brawlers and information about them.

Parameters **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A list of available brawlers and information about them.

Return type *Brawlers*

get_club(*tag: brawlstats.utils.bstag*, *use_cache=True*) → brawlstats.models.Club
Gets a club's stats.

Parameters

- **tag** (*str*) – A valid club tag. Valid characters: 0289PYLQGRJCUV
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A club object with all of its attributes.

Return type *Club*

get_club_members(*tag: brawlstats.utils.bstag*, *use_cache=True*) → brawlstats.models.Members
Gets the members of a club.

Parameters

- **tag** (*str*) – A valid club tag. Valid characters: 0289PYLQGRJCUV
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A list of the members in a club.

Return type *Members*

get_constants(*key: str = None*, *use_cache=True*) → brawlstats.models.Constants
Gets Brawl Stars constants extracted from the app.

Parameters

- **key** (*str, optional*) – Any key to get specific data, by default None
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns Data containing some Brawl Stars constants.

Return type *Constants*

get_player(*tag: brawlstats.utils.bstag*, *use_cache=True*) → brawlstats.models.Player
Gets a player's stats.

Parameters

- **tag** (*str*) – A valid player tag. Valid characters: 0289PYLQGRJCUV
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A player object with all of its attributes.

Return type *Player*

get_profile(*tag: brawlstats.utils.bstag*, *use_cache=True*) → brawlstats.models.Player
Gets a player's stats.

Parameters

- **tag** (*str*) – A valid player tag. Valid characters: 0289PYLQGRJCUV
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A player object with all of its attributes.

Return type *Player*

get_rankings(*, *ranking: str*, *region: str = None*, *limit: int = 200*, *brawler: Union[str*, *int] = None*, *use_cache=True*) → brawlstats.models.Ranking
Gets the top count players/clubs/brawlers.

Parameters

- **ranking** (*str*) – The type of ranking. Must be "players", "clubs", "brawlers".
- **region** (*str, optional*) – The region to retrieve from. Must be a 2 letter country code, by default None
- **limit** (*int, optional*) – The number of top players or clubs to fetch, by default 200
- **brawler** (*Union[str, int], optional*) – The brawler name or ID, by default None
- **use_cache** (*bool, optional*) – Whether to use the internal 3 minutes cache, by default True

Returns A player or club ranking that contains a list of players or clubs.

Return type *Ranking*

Raises

- `ValueError` – The brawler name or ID is invalid.
- `ValueError` – *rankings* is not "players", "clubs", or "brawlers"
- `ValueError` – *limit* is not between 1 and 200, inclusive.

2.1.2 Data Models

class brawlstats.models.**Player**(**args*, ***kwargs*)
A player object with all of its attributes.

get_club() → brawlstats.models.Club
Gets the player's club.

Returns A list of the members in a club, or None if the player is not in a club.

Return type *Club* or None

class brawlstats.models.**Club**(*client*, *data*)
A club object with all of its attributes.

get_members() → brawlstats.models.Members
Gets the members of a club.

Returns A list of the members in a club.

Return type *Members*

class brawlstats.models.**Ranking**(*client*, *data*)
A player or club ranking that contains a list of players or clubs.

class brawlstats.models.**BattleLog**(*client*, *data*)
A player battle object with all of its attributes.

class brawlstats.models.**Members**(*client*, *data*)
A list of the members in a club.

class brawlstats.models.**Constants**(*client*, *data*)
Data containing some Brawl Stars constants.

class brawlstats.models.**Brawlers**(*client*, *data*)
A list of available brawlers and information about them.

2.1.3 Attributes of Data Models

Note: These are subject to change at any time. Visit https://developer.brawlstars.com/#/documentation to view up-to-date information on the API.

2.1.4 Player

A full player object (all its statistics)

Attributes:

Name	Type
tag	str
name	str
name_color	str
trophies	int
highest_trophies	int
power_play_points	int
highest_power_play_points	int
exp_level	int
exp_points	int
is_qualified_from_championship_challenge	bool
x3vs3_victories	int
team_victories	int
solo_victories	int
duo_victories	int
best_robo_rumble_time	int
best_time_as_big_brawler	int
club.tag	str
club.name	str
brawlers	List[Brawler]

2.1.5 Club

A full club object to get a club's statistics. In order to get this, you must get it from the client or a player object.

Attributes:

Name	Type
tag	str
name	str
description	str
type	str
trophies	int
required_trophies	int
members	List[Member]

2.1.6 Members

Returns a list of club members. Get this by accessing Club.members or Club.get_members()

```
members = club.members
print(members[0].name, members[0].role) # prints best player's name and role (sorted␣
↪by trophies)
```

Attributes:

Name	Type
tag	str
name	str
name_color	str
role	str
trophies	int

2.1.7 Ranking

Returns a list of top players, clubs, or brawlers. To access this, do `ranking[index]`

Player/Brawler attributes:

Name	Type
tag	str
name	str
name_color	str
trophies	int
rank	int
club.name	str

Club attributes:

Name	Type
tag	str
name	str
trophies	int
rank	int
member_count	int

2.1.8 Brawler

Returns a brawler object with the following attributes. You can retrieve a profile's brawler info by getting `Profile.brawlers`

```
brawlers = profile.brawlers
top_brawler = brawlers[0] # first index in list = highest trophies
print(top_brawler.name, top_brawler.trophies) # prints best brawler's name and␣
↪trophies
```

Attributes:

Name	Type
id	int
name	str
power	int
rank	int
trophies	int
highest_trophies	int
star_powers	List[SP]

2.1.9 Star Power

Attributes:

Name	Type
id	int
name	str

2.1.10 Battle Logs

Returns a list of objects with this structure:

Attributes:


```
{
    "battleTime":"20190706T151526.000Z",
    "event":{
        "id":15000126,
        "mode":"duoShowdown",
        "map":"Royal Runway"
    },
    "battle":{
        "mode":"duoShowdown",
        "type":"ranked",
        "rank":1,
        "trophyChange":9,
        "teams":[
            [
                {
                    "tag":"#Y2QPGG",
                    "name":"Lex_YouTube",
                    "brawler":{
                        "id":16000005,
                        "name":"SPIKE",
                        "power":10,
                        "trophies":495
                    }
                },
                {
                    "tag":"#8Q229LJY",
                    "name":"Brandon",
                    "brawler":{
                        "id":16000003,
                        "name":"BROCK",
                        "power":10,
                        "trophies":495
                    }
                },
                    {
                    "tag":"#29RGL0QJ0",
                    "name":"smallwhitepeen1",
                    "brawler":{
                        "id":16000007,
                        "name":"JESSIE",
                        "power":7,
                        "trophies":486
```


(continues on next page)

(continued from previous page)


```
                    }
                }
            ],
            [
                {
                    "tag":"#CYLVL8LY",
                    "name":"TST|ROYER™",
                    "brawler":{
                        "id":16000019,
                        "name":"PENNY",
                        "power":8,
                        "trophies":541
                    }
                },
                {
                    "tag":"#8P2URCR0",
                    "name":"ANOTHER",
                    "brawler":{
                        "id":16000023,
                        "name":"LEON",
                        "power":8,
                        "trophies":559
                    }
                },
                {
                    "tag":"#8LRY92QP",
                    "name":"Marshmello",
                    "brawler":{
                        "id":16000021,
                        "name":"GENE",
                        "power":7,
                        "trophies":448
                    }
                }
            ]
        ]
    }
}
```


2.1.11 Brawlers

Returns list of available brawlers and information about them with this structure:

Attributes:

```
[
    Brawler
]
```

2.2 Exceptions

The possible exceptions thrown by the library.

exception brawlstats.errors.**RequestError**(*code*, *message*)
The base class for all errors.

exception brawlstats.errors.**Forbidden**(*code*, *url*, *message*)
Raised if your API Key is invalid.

exception brawlstats.errors.**NotFoundError**(*code*, ***kwargs*)
Raised if an invalid player tag or club tag has been passed.

exception brawlstats.errors.**RateLimitError**(*code*, *url*)
Raised when the rate limit is reached.

exception brawlstats.errors.**UnexpectedError**(*url*, *code*, *text*)
Raised if an unknown error has occured.

exception brawlstats.errors.**ServerError**(*code*, *url*)
Raised if the API is down.

2.3 Setting Up Logging

brawlstats logs errors and debug information via the `logging` python module. It is strongly recommended that the logging module is configured, as no errors or warnings will be output if it is not set up. Configuration of the `logging` module can be as simple as

```
import logging

logging.basicConfig(level=logging.DEBUG)
```

Placed at the start of the application. This will output the logs from *brawlstats* as well as other libraries that uses the `logging` module directly to the console.

The optional `level` argument specifies what level of events to log out and can any of `CRITICAL`, `ERROR`, `WARNING`, `INFO`, and `DEBUG` and if not specified defaults to `WARNING`.

More advanced setups are possible with the `logging` module. For example, to write the logs to a file called `brawlstars.log` instead of outputting them to to the console, the following snippet can be used:

```
import brawlstats
import logging

logger = logging.getLogger('brawlstats')
logger.setLevel(logging.DEBUG)
handler = logging.FileHandler(filename='brawlstars.log', encoding='utf-8', mode='w')
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(name)s:
↪%(message)s'))
logger.addHandler(handler)
```

This is recommended, especially at verbose levels such as `INFO`, and `DEBUG` as there are a lot of events logged and it would clog the stdout of your program.

Currently, the following things are logged:

- `DEBUG`: API Requests

For more information, check the documentation and tutorial of the `logging` module.

CHAPTER 3

Indices

- genindex
- search

Index

Symbols

B

C

F

G

M

N

P

R

S

U