PyBattlerite Documentation

Release 0.5.17

PyBattlerite

Feb 02, 2018

Contents

Basic Usage:

```
import pybattlerite
import requests

brc = pybattlerite.Client('your-api-key')

# You can also provide an aiohttp.ClientSession to the BRClient constructor
session = requests.Session()
brc_a = pybattlerite.Client('your-api-key', session)

# Get 3 matches after specified time
# after and before can also be datetime.datetime objects
matches = brc.get_matches(limit=3, after="2017-11-22T20:34:58Z")

# Go to the next pages of matches
matches.next()

# Get telemetry data for one of the matches
telemetry = matches.matches[0].get_telemetry()
```

Async Usage:

```
import aiohttp
import asyncio
import pybattlerite

brc = pybattlerite.AsyncClient('your-api-key')

# You can also provide an aiohttp.ClientSession to the BRClient constructor
session = aiohttp.ClientSession()
brc_a = pybattlerite.AsyncClient('your-api-key', session)

# Get 3 matches after specified time
# after and before can also be datetime.datetime objects
matches = await brc.get_matches(limit=3, after="2017-11-22T20:34:58Z")

# Go to the next pages of matches
await matches.next()

# Get telemetry data for one of the matches
telemetry = await matches.matches[0].get_telemetry()
```

CHAPTER 1

pybattlerite.Client

class pybattlerite.client.**Client**(*key*, *session: requests.sessions.Session = None*, *lang: str = 'English'*)

Bases: pybattlerite.clientbase.ClientBase

Top level class for user to interact with the API.

Parameters **key** : str

The official Battlerite API key.

session : Optional[requests.Session]

lang : str, Default['English']

The language to localise game specific strings in.

Currently available languages are:

Brazilian, English, French, German, Italian, Japanese, Korean, Polish, Romanian, Russian, SChinese, Spanish, Turkish.

get_matches(*offset: int = None*, *limit: int = None*, *after=None*, *before=None*, *playerids: list = None*, *server_type: str = None*, *ranking_type: str = None*, *patch_version: list = None*)

Access the /matches endpoint and grab a list of matches

Parameters **offset** : Optional[int]

The nth number of match to start the page from.

limit : Optional[int]

Number of matches to return.

after : Optional[str or datetime.datetime]

Filter to return matches after provided time period, if an str is provided it should follow the **iso8601** format.

before : Optional[str or datetime.datetime]

Filter to return matches before provided time period, if an str is provided it should follow the **iso8601** format.

playerids : Optional[list]

Filter to only return matches with provided players in them by looking for their player IDs.

server_type : Optional[list(str)]

The match's server_type, can be either 'QUICK2V2', 'QUICK3V3' or 'PRIVATE'.

ranking_type : Optional[list(str)]

The match's rank type, either 'RANKED', 'UNRANKED' or 'NONE'.

patch_version : Optional[list(str)]

The Battlerite patch versions you want data for, this doesn't go through any tests so check your versions.

Returns *pybattlerite.models.MatchPaginator*

A MatchPaginator instance representing a get_matches request

get_players(*playerids: list = None*, *steamids: list = None*, *usernames: list = None*)
Get multiple players' info at once.

Parameters **playerids** : list

A list of playerids, either a *list* of strs or a *list* of ints. Max list length is 6.

steamids : list

A list of steamids, a *list* of ints, this accepts only *SteamID64* specification, check here for more details. Max list length is 6

usernames : list

A list of usernames, a *list* of strings, case insensitive. Max list length is 6

Returns list

A list of *pybattlerite.models.Player*

get_status()
Check if the API is up and running

Returns tuple(createdAt: str, version: str):

get_teams(*playerids: list*, *season: int*)
Get all teams for a player or group of players in a specified season.

Parameters **playerids** : list

A list of playerids to fetch teams for, this just fetches teams with these playerids in them, it is not an intersection of supplied playerids.

season : int

The season for which the teams of these playerids must be fetched

Returns list(`pybattlerite.models.Team`)

A list of Team objects representing each team from the request.

match_by_id(*match_id*)
Get a Match by its ID.

Parameters **match_id** : int or str

Returns *pybattlerite.models.Match*

A match object representing the requested match.

player_by_id(*player_id: int*)
Get a player's info by their ID.

Parameters **player_id** : int

Returns *pybattlerite.models.Player*

A Player object representing the requested player.

player_by_name(*username*)
Get a player's info by their ingame name.

Parameters **username** : str

Case insensitive username

Returns *pybattlerite.models.Player*

A Player object representing the requested player.

CHAPTER 2

pybattlerite.AsyncClient

class pybattlerite.asyncclient.**AsyncClient**(*key*, *session: aiohttp.client.ClientSession = None*, *lang: str = 'English'*)

Bases: pybattlerite.clientbase.ClientBase

Top level class for user to interact with the API.

Parameters **key** : str

The official Battlerite API key.

session : Optional[aiohttp.ClientSession]

lang : str, Default['English']

The language to localise game specific strings in.

Currently available languages are:

Brazilian, English, French, German, Italian, Japanese, Korean, Polish, Romanian, Russian, SChinese, Spanish, Turkish.

get_matches(*offset: int = None*, *limit: int = None*, *after=None*, *before=None*, *playerids: list = None*, *server_type: list = None*, *ranking_type: list = None*, *patch_version: list = None*)

Access the /matches endpoint and grab a list of matches

Parameters **offset** : Optional[int]

The nth number of match to start the page from.

limit : Optional[int]

Number of matches to return.

after : Optional[str or datetime.datetime]

Filter to return matches after provided time period, if an str is provided it should follow the **iso8601** format.

before : Optional[str or datetime.datetime]

Filter to return matches before provided time period, if an str is provided it should follow the **iso8601** format.

playerids : Optional[list]

Filter to only return matches with provided players in them by looking for their player IDs.

server_type : Optional[list(str)]

The match's server_type, can be either 'QUICK2V2', 'QUICK3V3' or 'PRIVATE'.

ranking_type : Optional[list(str)]

The match's rank type, either 'RANKED', 'UNRANKED' or 'NONE'.

patch_version : Optional[list(str)]

The Battlerite patch versions you want data for, this doesn't go through any tests so check your versions.

Returns *pybattlerite.models.AsyncMatchPaginator*

A MatchPaginator instance representing a get_matches request

get_players(*playerids: list = None*, *steamids: list = None*, *usernames: list = None*)
Get multiple players' info at once.

Parameters **playerids** : list

A list of playerids, either a *list* of strs or a *list* of ints. Max list length is 6.

steamids : list

A list of steamids, a *list* of ints, this accepts only *SteamID64* specification, check here for more details. Max list length is 6

usernames : list

A list of usernames, a *list* of strings, case insensitive. Max list length is 6

Returns list

A list of *pybattlerite.models.Player*

get_status()
Check if the API is up and running

Returns tuple(createdAt: str, version: str):

get_teams(*playerids: list*, *season: int*)
Get all teams for a player or group of players in a specified season.

Parameters **playerids** : list

A list of playerids to fetch teams for, this just fetches teams with these playerids in them, it is not an intersection of supplied playerids.

season : int

The season for which the teams of these playerids must be fetched

Returns list(`pybattlerite.models.Team`)

A list of Team objects representing each team from the request.

match_by_id(*match_id*)
Get a Match by its ID.

Parameters **match_id** : int or str

Returns *pybattlerite.models.AsyncMatch*

A match object representing the requested match.

player_by_id(*player_id: int*)
Get a player's info by their ID.

Parameters **player_id** : int

Returns *pybattlerite.models.Player*

A Player object representing the requested player.

player_by_name(*username*)
Get a player's info by their ingame name.

Parameters **username** : str

Case insensitive username

Returns *pybattlerite.models.Player*

A Player object representing the requested player.

CHAPTER 3

pybattlerite.models

class pybattlerite.models.**AsyncMatch**(*data*, *session*, *included=None*)

Bases: *pybattlerite.models.MatchBase*

Extends *MatchBase* to add async *get_telemetry()*.

get_telemetry(*session=None*)

Get telemetry data for a match.

Parameters **session** : Optional[aiohttp.ClientSession]

Optional session to use to request telemetry data.

Returns *dict*

Match telemetry data

class pybattlerite.models.**AsyncMatchPaginator**(*matches*, *data*, *client*)

Bases: *pybattlerite.models.Paginator*

Returned only by the *get_matches* method of the async client.

first(*session=None*)

Move to the first page of matches.

Parameters **session** : Optional[aiohttp.ClientSession]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

next(*session=None*)

Move to the next page of matches.

Parameters **session** : Optional[aiohttp.ClientSession]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

Raises **BRPaginationError**

The current page is the last page of results

prev(*session=None*)
Move to the previous page of matches.

Parameters **session** : Optional[aiohttp.ClientSession]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

Raises **BRPaginationError**

The current page is the first page of results

class pybattlerite.models.**BaseBRObject**(*data*)
Bases: object

A base object for most data classes

Attributes

id	(int) A general unique ID for each type of data.

class pybattlerite.models.**Match**(*data*, *session*, *included=None*)
Bases: *pybattlerite.models.MatchBase*

Extends *MatchBase* to add *get_telemetry()*

get_telemetry(*session=None*)
Get telemetry data for a match.

Parameters **session** : Optional[requests.Session]

Optional session to use to request telemetry data.

Returns *dict*

Match telemetry data

class pybattlerite.models.**MatchBase**(*data*, *session*, *included=None*)
Bases: *pybattlerite.models.BaseBRObject*

A class that holds data for a match.

Attributes

id	(int) A general unique ID for each type of data.
created_at	(datetime.datetime) Time when the match was created.
duration	(int) The match's duration in seconds.
game_mode	(str) The gamemode this match was of.
patch	(str) The Battlerite patch version this match was played on.
shard_id	(str) The shard on which this match data resides, only *global* is available at the moment.
map_id	(int) The match's map's ID.
type	(str) The match type, ex: 'QUICK2V2 '.
rosters	(list) A list of *Roster* objects representing rosters taking part in the match.
rounds	(list) A list of *Round* objects representing each round in the match.
spectators	(list) A list of *Participant* objects representing spectators, this seems unimplemented in the game as of now.
telemetry_url	(str) URL for the telemetry file for this match
session	(Optional[aiohttp.ClientSession]) Optional session to use to request match data.

class pybattlerite.models.**MatchPaginator**(*matches*, *data*, *client*)

Bases: *pybattlerite.models.Paginator*

Returned only by the *get_matches* method of the client.

first(*session=None*)

Move to the first page of matches.

Parameters **session** : Optional[requests.Session]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

next(*session=None*)

Move to the next page of matches.

Parameters **session** : Optional[requests.Session]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

Raises **BRPaginationError**

The current page is the last page of results

prev(*session=None*)

Move to the previous page of matches.

Parameters **session** : Optional[requests.Session]

Optional session to use to make this request.

Returns *list*

A list of *Match*.

Raises **BRPaginationError**

The current page is the first page of results

class pybattlerite.models.**Paginator**(*matches*, *data*, *client*)
Bases: object

Returned only by BRClient.get_matches

class pybattlerite.models.**Participant**(*participant*, *included*)
Bases: *pybattlerite.models.BaseBRObject*

A class that holds data about a participant in a match.

Attributes

id	(int) A general unique ID for each type of data.
actor	(int)
shard_id	(str)
attachment	(int)
emote	(int)
mount	(int)
outfit	(int)
side	(int)
ability_uses	(Optional[int])
damage_done	(Optional[int])
damage_received	(Optional[int])
deaths	(Optional[int])
disables_done	(Optional[int])
disables_received	(Optional[int])
energy_gained	(Optional[int])
energy_used	(Optional[int])
healing_done	(Optional[int])
healing_received	(Optional[int])
kills	(Optional[int])
score	(Optional[int])
time_alive	(Optional[int])
user_id	(Optional[int])
player	(Optional[*Player*])

class pybattlerite.models.**Player**(*data*, *lang: str = 'English'*)
Bases: *pybattlerite.models.BaseBRObject*

A class that holds general user data, if this is through a Match, this will not have name, picture and title, only an ID

Attributes

id	(int) A general unique ID for each type of data.
name	(str) The player's name
picture	(int) The picture ID for this player
title	(int) This player's ingame title

class pybattlerite.models.**Roster**(*roster*, *included*)
Bases: *pybattlerite.models.BaseBRObject*

A class that holds data about one of the two teams in a match.

Attributes

id	(int) A general unique ID for each type of data.
shard_id	(str) The shard on which this roster data resides, only *global* is available at the moment.
score	(int) The roster's overall score for a match.
won	(bool) Signifies if the roster won a match.
participants	(list) A list of *Participant* representing players that are part of the roster.

class pybattlerite.models.**Round**(*_round*, *included*)
Bases: *pybattlerite.models.BaseBRObject*

A class that holds general data about a round.

Attributes

id	(int) A general unique ID for each type of data.
duration	(int) Duration in seconds of this round.
ordinal	(int) The round number this round was in a match.
winning_team	(int) Signifies which of the teams/rosters won the round.

CHAPTER 4

pybattlerite.utils

class pybattlerite.utils.**Localizer**(*lang*)

Bases: object

Use this to manually localize any data with an available *loc_id*.

Parameters **lang** : str

The language to localise game specific strings in.

Currently available languages are:

Brazilian, English, French, German, Italian, Japanese, Korean, Polish, Romanian, Russian, SChinese, Spanish, Turkish.

localize(*_id*)

Call to return a localized string for your id.

Parameters **_id** : str

An id that refers to the name of an object in multiple languages.

Example: '035ad4c27697469e8163040ae0a4f796'

Returns str

A localized string in the *Localizer*'s set language.

CHAPTER 5

pybattlerite.errors

exception pybattlerite.errors.**BRFilterException**(*error*)
Bases: Exception

Raised when an invalid filter value is supplied.

exception pybattlerite.errors.**BRPaginationError**(*error*)
Bases: Exception

Raised when asyncbattlerite.models.MatchPaginator.next() or asyncbattlerite.models.MatchPaginator.prev() are called when the paginator is on the last or first page respectively.

exception pybattlerite.errors.**BRRequestException**(*response*, *data*)
Bases: Exception

General purpose exception, base class for other request related exceptions.

Parameters **response** : aiohttp.ClientResponse

data : dict

The json response from the API

exception pybattlerite.errors.**BRServerException**(*response*, *data*)
Bases: *pybattlerite.errors.BRRequestException*

Exception that signifies that the server failed to respond with valid data.

exception pybattlerite.errors.**EmptyResponseException**(*error*)
Bases: Exception

Raised when any request is 200 OK, but the data is empty.

exception pybattlerite.errors.**NotFoundException**(*response*, *data*)
Bases: *pybattlerite.errors.BRRequestException*

For the 404s

Python Module Index

p

Index