r6sapi.py Documentation

Release 1.4.1

Billyoyo

Jan 08, 2021

CONTENTS

CHAPTER ONE

CONTENTS

1.1 Getting Started

1.1.1 Introduction

r6sapi.py is a module for easily getting information from the unofficial rainbow six siege api. It allows you to get things such as a players rank and specific stats for operators, gamemodes and queues The api requires authentication to process any api requests, so r6sapi requires your ubisoft login email and password.

1.1.2 Quick Example

```python
import asyncio
import r6sapi as api

@asyncio.coroutine
def run():
    auth = api.Auth("email", "password")

    player = yield from auth.get_player("billy_yoyo", api.Platforms.UPLAY)
    operator = yield from player.get_operator("sledge")

    print(operator.kills)

asyncio.get_event_loop().run_until_complete(run())
```

1.1.3 License

MIT

1.2 API Reference

1.2.1 Auth

class r6sapi.**Auth**(*email=None*, *password=None*, *token=None*, *appid=None*, *cachetime=120*, *max_connect_retries=1*, *session=None*, *refresh_session_period=180*)

Holds your authentication information. Used to retrieve Player objects Once you're done with the auth object, auth.close() should be called.

Parameters

- **email** (*Optional[str]*) – Your Ubisoft email
- **password** (*Optional[str]*) – Your Ubisoft password
- **token** (*Optional[str]*) – Your Ubisoft auth token, either supply this OR email/password
- **appid** (*Optional[str]*) – Your Ubisoft appid, not required
- **cachetime** (*Optional[float]*) – How long players are cached for (in seconds)
- **max_connect_retries** (*Optional[int]*) – How many times the auth client will automatically try to reconnect, high numbers can get you temporarily banned
- **refresh_session_period** (*Optional[int]*) – How frequently the http session should be refreshed, in seconds. Negative number for never. Defaults to 3 minutes.

session

aiohttp client session

token

your token

Type str

appid

your appid

Type str

sessionid

the current connections session id (will change upon attaining new key)

Type str

key

your current auth key (will change every time you connect)

Type str

spaceids

contains the spaceid for each platform

Type dict

profileid

your profileid (corresponds to your appid)

Type str

userid

your userid (corresponds to your appid)

Type str

cachetime
the time players are cached for

Type float

cache
the current player cache

Type dict

close()
This function is a *coroutine*.

Closes the session associated with the auth object

connect()
This function is a *coroutine*.

Connect to ubisoft, automatically called when needed

get_player(*name=None*, *platform=None*, *uid=None*)
This function is a *coroutine*.

Calls get_players and returns the first element, exactly one of uid and name must be given, platform must be given

Parameters

- **name** (`str`) – the name of the player you're searching for
- **platform** (`str`) – the name of the platform you're searching on (See `Platforms`)
- **uid** (`str`) – the uid of the player you're searching for

Returns player found

Return type `Player`

get_player_batch(*names=None*, *platform=None*, *uids=None*)
This function is a *coroutine*.

Calls get_player for each name and uid you give, and creates a player batch out of all the resulting player objects found. See `PlayerBatch` for how to use this.

Parameters

- **names** (`list[str]`) – a list of player names to add to the batch, can be none
- **uids** (`list[str]`) – a list of player uids to add to the batch, can be none
- **platform** (`str`) – the name of the platform you're searching for players on (See `Platforms`)

Returns the player batch

Return type `PlayerBatch`

get_players(*name=None*, *platform=None*, *uid=None*)
This function is a *coroutine*.

get a list of players matching the term on that platform, exactly one of uid and name must be given, platform must be given, this list almost always has only 1 element, so it's easier to use get_player

Parameters

- **name** (`str`) – the name of the player you're searching for
- **platform** (`str`) – the name of the platform you're searching on (See `Platforms`)

- **uid** (*str*) – the uid of the player you're searching for

Returns list of found players

Return type list[*Player*]

get_session()

This function is a *coroutine*.

Retrieves the current session, ensuring it's valid first

refresh_session()

This function is a *coroutine*.

Closes the current session and opens a new one

1.2.2 Player

class r6sapi.**Player**(*auth*, *data*)

Contains information about a specific player

auth

the auth object used to find this player

Type *Auth*

id

the players profile id

Type str

userid

the players user id

Type str

platform

the platform this player is on

Type str

platform_url

the URL name for this platform (used internally)

Type str

id_on_platform

the players ID on the platform

Type str

name

the players name on the platform

Type str

url

a link to the players profile

Type str

icon_url

a link to the players avatar

Type str

xp
the amount of xp the player has, must call check_level or load_level first

Type int

level
the level of the player, must call check_level or load_level first

Type int

ranks
dict containing already found ranks ("region_name:season": *Rank*)

Type dict

operators
dict containing already found operators (operator_name: *Operator*)

Type dict

gamemodes
dict containing already found gamemodes (gamemode_id: *Gamemode*)

Type dict

weapons
dict containing already found weapons (weapon_id: *Weapon*)

Type dict

casual
stats for the casual queue, must call load_queues or check_queues first

Type *GameQueue*

ranked
stats for the ranked queue, must call load_queues or check_queues first

Type *GameQueue*

deaths
the number of deaths the player has (must call load_general or check_general first)

Type int

kills
the number of kills the player has (must call load_general or check_general first)

Type int

kill_assists
the number of kill assists the player has (must call load_general or check_general first)

Type int

penetration_kills
the number of penetration kills the player has (must call load_general or check_general first)

Type int

melee_kills
the number of melee kills the player has (must call load_general or check_general first)

Type int

revives
the number of revives the player has (must call load_general or check_general first)

Type int

matches_won
the number of matches the player has won (must call load_general or check_general first)

Type int

matches_lost
the number of matches the player has lost (must call load_general or check_general first)

Type int

matches_played
the number of matches the player has played (must call load_general or check_general first)

Type int

time_played
the amount of time in seconds the player has played for (must call load_general or check_general first)

Type int

bullets_fired
the amount of bullets the player has fired (must call load_general or check_general first)

Type int

bullets_hit
the amount of bullets the player has hit (must call load_general or check_general first)

Type int

headshots
the amount of headshots the player has hit (must call load_general or check_general first)

Type int

terrorist_hunt
contains all of the above state (from deaths to headshots) inside a gamequeue object.

Type *GameQueue*

check_gamemodes(*data=None*)
This function is a *coroutine*.

Checks the players gamemode stats, only loading them if they haven't already been found

Returns dict of all the gamemodes found (gamemode_name: *Gamemode*)

Return type dict

check_general(*data=None*)
This function is a *coroutine*.

Checks the players general stats, only loading them if they haven't already been found

check_level()
This function is a *coroutine*.

Check the players XP and level, only loading it if it hasn't been loaded yet

check_queues(*data=None*)
This function is a *coroutine*.

Checks the players game queues, only loading them if they haven't already been found

check_terrohunt(*data=None*)

This function is a *coroutine*.

Checks the players general stats for terrorist hunt, only loading them if they haven't been loaded already

check_weapons(*data=None*)

This function is a *coroutine*.

Check the players weapon stats, only loading them if they haven't already been found

Returns list of all the weapon objects found

Return type list[`Weapon`]

get_all_operators(*data=None*)

This function is a *coroutine*.

Checks the player stats for all operators, loading them all again if any aren't found This is significantly more efficient than calling get_operator for every operator name.

Returns the dictionary of all operators found

Return type dict[`Operator`]

get_operator(*operator*, *data=None*)

This function is a *coroutine*.

Checks the players stats for this operator, only loading them if they haven't already been found

Parameters **operator** (`str`) – the name of the operator

Returns the operator object found

Return type `Operator`

get_rank(*region*, *season=- 1*, *data=None*)

This function is a *coroutine*.

Checks the players rank for this region, only loading it if it hasn't already been found

Parameters

- **region** (`str`) – the name of the region you want to get the rank for
- **season** (`Optional[int]`) – the season you want to get the rank for (defaults to -1, latest season)

Returns the players rank for this region and season

Return type `Rank`

load_all_operators(*data=None*)

This function is a *coroutine*.

Loads the player stats for all operators

Returns the dictionary of all operators found

Return type dict[`Operator`]

load_gamemodes(*data=None*)

This function is a *coroutine*.

Loads the players gamemode stats

Returns dict of all the gamemodes found (gamemode_name: `Gamemode`)

Return type dict

load_general (*data=None*)

This function is a *coroutine*.

Loads the players general stats

load_level (*data=None*)

This function is a *coroutine*.

Load the players XP and level

load_operator (*operator*, *data=None*)

This function is a *coroutine*.

Loads the players stats for the operator

Parameters **operator** (`str`) – the name of the operator

Returns the operator object found

Return type `Operator`

load_queues (*data=None*)

This function is a *coroutine*.

Loads the players game queues

load_rank (*region*, *season=- 1*, *data=None*)

This function is a *coroutine*. Loads the players rank for this region and season

Parameters

- **region** (`str`) – the name of the region you want to get the rank for
- **season** (`Optional[int]`) – the season you want to get the rank for (defaults to -1, latest season)

Returns the players rank for this region and season

Return type `Rank`

load_terrohunt (*data=None*)

This function is a *coroutine*.

Loads the player's general stats for terrorist hunt

load_weapons (*data=None*)

This function is a *coroutine*.

Load the players weapon stats

Returns list of all the weapon objects found

Return type list[`Weapon`]

1.2.3 PlayerBatch

class r6sapi.**PlayerBatch** (*players*)

Accumulates requests for multiple players' stats in to a single request, saving time.

Acts as a proxy for any asynchronous method in `Player`. The response of the method will be a dictionary of the responses from each player, with the player ids as keys.

This class is also an iterable, and iterates over the `Player` objects contained in the batch. Individual players in the batch can be accessed via their ID using an item accessor (player_batch[player.id])

Parameters **players** (list[`Player`]) – the list of players in the batch

1.2.4 Rank

class r6sapi.**Rank**(*data*, *rank_definitions*)

Contains information about your rank

RANKS

Names of the ranks

Type list[str]

RANK_CHARMS

URLs for the rank charms

Type list[str]

UNRANKED

the unranked bracket id

Type int

COPPER

the copper bracket id

Type int

BRONZE

the bronze bracket id

Type int

SILVER

the silver bracket id

Type int

GOLD

the gold bracket id

Type int

PLATINUM

the platinum bracket id

Type int

DIAMOND

the diamond bracket id

Type int

max_mmr

the maximum MMR the player has achieved

Type int

mmr

the MMR the player currently has

Type int

wins

the number of wins this player has this season

Type int

`losses`
the number of losses this player has this season

Type int

`abandons`
the number of abandons this player has this season

Type int

`rank_id`
the id of the players current rank

Type int

`rank`
the name of the players current rank

Type str

`max_rank`
the id of the players max rank

Type int

`next_rank_mmr`
the mmr required for the player to achieve their next rank

Type int

`season`
the season this rank is for

Type int

`region`
the region this rank is for

Type str

`skill_mean`
the mean for this persons skill

Type float

`skill_stdev`
the standard deviation for this persons skill

Type float

`get_bracket`(*rank_id=None*)
Get rank bracket

Returns the id for the rank bracket this rank is in

Return type `int`

`get_bracket_name`(*rank_id=None*)
Get rank bracket name

Returns the name for the rank bracket this rank is in

Return type `str`

`get_charm_url`()
Get charm URL for the bracket this rank is in

Returns the URL for the charm

Return type `str`

`get_icon_url``()`

Get URL for this rank's icon

Returns the URL for the rank icon

Return type `str`

`get_max_rank_name``()`

Get rank name of max rank

Returns the name for this rank

Return type `str`

`get_rank_name``(`*rank_id=None*`)`

Get rank name

Returns the name for this rank

Return type `str`

1.2.5 Operator

`class` `r6sapi.`**`Operator`**`(`*name*, *stats=None*, *unique_stats=None*`)`

Contains information about an operator

`name`

the name of the operator

Type str

`wins`

the number of wins the player has on this operator

Type int

`losses`

the number of losses the player has on this operator

Type int

`kills`

the number of kills the player has on this operator

Type int

`deaths`

the number of deaths the player has on this operator

Type int

`headshots`

the number of headshots the player has on this operator

Type int

`melees`

the number of melee kills the player has on this operator

Type int

dbnos
the number of DBNO (down-but-not-out)'s the player has on this operator

Type int

xp
the total amount of xp the player has on this operator

Type int

time_played
the amount of time the player has played this operator for in seconds

Type int

statistic
the value for this operators unique statistic (depreciated in favour of unique_stats)

Type int

statistic_name
the human-friendly name for this operators statistic (depreciated in favour of unique_stats)

Type str

unique_stats
mapping of an operator's unique stat to number of times that stat has been achieved (e.g. kills with a gadget)

Type dict[`UniqueOperatorStat`, int]

1.2.6 Weapon

class r6sapi.**Weapon**(*weaponType*, *stats=None*)
Contains information about a weapon

type
the weapon type

Type int

name
the human-friendly name for this weapon type

Type str

kills
the number of kills the player has for this weapon

Type int

headshots
the number of headshots the player has for this weapon

Type int

hits
the number of bullet this player has hit with this weapon

Type int

shots
the number of bullets this player has shot with this weapon

Type int

1.2.7 Gamemode

class `r6sapi.`**`Gamemode`**(*gamemodeType*, *stats=None*)

Contains information about a gamemode

`type`

the gamemode id

Type str

`name`

the human-readable name for this gamemode

Type str

`won`

the number of wins the player has on this gamemode

Type int

`lost`

the number of losses the player has on this gamemode

Type int

`played`

the number of games this player has played on this gamemode

Type int

`best_score`

the best score this player has achieved on this gamemode

Type int

1.2.8 GameQueue

class `r6sapi.`**`GameQueue`**(*name*, *stats=None*)

Contains information about a specific game queue

`name`

the name for this gamemode (always either "ranked" or "casual"

Type str

`won`

the number of wins the player has on this gamemode

Type int

`lost`

the number of losses the player has on this gamemode

Type int

`time_played`

the amount of time in seconds the player has spent playing on this gamemode

Type int

`played`

the number of games the player has played on this gamemode

Type int

kills
the number of kills the player has on this gamemode

Type int

deaths
the number of deaths the player has on this gamemode

Type int

1.2.9 Platforms

class r6sapi.**Platforms**
Platforms supported

UPLAY
name of the uplay platform

Type str

XBOX
name of the xbox platform

Type str

PLAYSTATION
name of the playstation platform

Type str

1.2.10 RankedRegions

class r6sapi.**RankedRegions**
Ranked regions supported

EU
name of the european data centre

Type str

NA
name of the north american data centre

Type str

ASIA
name of the asian data centre

Type str

1.2.11 WeaponTypes

class r6sapi.**WeaponTypes**

Weapon Types

ASSAULT_RIFLE

the assault rifle weapon id

Type int

SUBMACHINE_GUN

the submachine gun weapon id

Type int

MARKSMAN_RIFLE

the marksman rifle weapon id

Type int

SHOTGUN

the shotgun weapon id

Type int

HANDGUN

the handgun weapon id

Type int

LIGHT_MACHINE_GUN

the light machine gun weapon id

Type int

MACHINE_PISTOL

the machine pistol weapon id

Type int

1.3 How It Works

1.3.1 Introduction

Most of the API endpoints can be fairly easily retrieved by going on to the network tab and monitoring the requests sent. Your browser, as usual, will send a load of unnecessary headers with the request, and a quick bit of testing will show that the only two required are the "Authorization" header and the "Ubi-AppId" header. (Also the request must have content-type set to application/json)

1.3.2 Experimenting

When you're logged in to your account on the website, your "Authorization" header looks like `Ubi_v1 t=[token]` where `[token]` is a load of random characters. Your Ubi-AppId is a string of characters split by "-", so if we attempt to simply copy/paste these two and use them in our code, it will work but this type of token is called a "ticket" and is only temporary. Eventually you'll get a response telling you your token is invalid, meaning you need to resend the information you used to recieve your ticket in the first place.

1.3.3 Logging In

So clearly some sort of auth login logic is required, where you recieve a new ticket every time your current one runs out. So if you monitor the requests sent when you log in, you'll see the very first request sent has the authorization header set to `Basic [token]`. This time `[token]` appears to be constant, and the response you get from this endpoint gives you a valid ticket, along with some other things. Great, now there's two things left to do: figure out how to generate this token from username/id and figure out how to get you appid

1.3.4 Generating The Token

To do this I read through the javascript on the login page until I found the bit that converts your username and password in to a base64 number. This is actually, very simply, `base64.encode(email + ":" + password)`. Nice and simple, this solves our first problem.

1.3.5 Getting the AppId

Turns out the AppId doesn't seem to matter at all, after reading through the code I couldn't figure out where the AppId gets decided. I believe it's generated server-side by ubisoft based on your IP, but either way I did manage to find a default AppId in the code, so unless one is specified, just using that one seems to work.

1.3.6 Conclusion

That's basically the end of it, I convert the username and password in to a basic token, then every time a request gets an unauthorized I try and fetch a new one. Then using the default appid, I can access any of the endpoints easily.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- search

INDEX