twitchAPI

Release 2.0

Nov 11, 2020

Contents:

1 Installation **3**

2 Usage **5**

3 Logging **7**

4 Indices and tables **9**
- 4.1 twitchAPI.twitch . . . 9
- 4.2 twitchAPI.webhook . . . 30
- 4.3 twitchAPI.pubsub . . . 35
- 4.4 twitchAPI.oauth . . . 39
- 4.5 twitchAPI.types . . . 41
- 4.6 twitchAPI.helper . . . 44

Python Module Index **47**

Index **49**

This is a full implementation of the Twitch Helix API and its Webhook in python 3.7.

On Github: https://github.com/Teekeks/pyTwitchAPI

On PyPi: https://pypi.org/project/twitchAPI/

Visit the changelog to see what has changed.

CHAPTER 1

Installation

Install using pip:

```
`pip install twitchAPI`
```

CHAPTER 2

Usage

For more detailed usage examples, see the links below

```
from twitchAPI.twitch import Twitch
from pprint import pprint
twitch = Twitch('my_app_key', 'my_app_secret')
# lets create a simple app authentication:
twitch.authenticate_app([])
pprint(twitch.get_users(logins=['your_twitch_username']))
```

CHAPTER 3

Logging

This module uses the *logging* module for creating Logs. Valid loggers are:

- *twitchAPI.twitch*
- *twitchAPI.pubsub*
- *twitchAPI.oauth*
- *twitchAPI.webhook*

CHAPTER 4

Indices and tables

- genindex
- modindex
- changelog

twitchAPI.twitch	The Twitch API client
twitchAPI.webhook	Full Implementation of the Twitch Webhook
twitchAPI.pubsub	PubSub client
twitchAPI.oauth	User OAuth Authenticator and helper functions
twitchAPI.types	Type Definitions
twitchAPI.helper	Helper functions

4.1 twitchAPI.twitch

4.1.1 The Twitch API client

This is the base of this library, it handles authentication renewal, error handling and permission management.

Look at the Twitch API reference for a more detailed documentation on what each endpoint does.

Example Usage:

```
from twitchAPI.twitch import Twitch
from pprint import pprint
twitch = Twitch('my_app_key', 'my_app_secret')
# lets create a simple app authentication:
twitch.authenticate_app([])
pprint(twitch.get_users(logins=['your_twitch_username']))
```

Authentication

The Twitch API knows 2 different authentications. App and User Authentication. Which one you need (or if one at all) depends on what calls you want to use.

Its always good to get at least App authentication even for calls where you dont need it since the rate limmits are way better for authenticated calls.

App Authentication

App authentication is super simple, just do the following:

```
from twitchAPI.twitch import Twitch
twitch = Twitch('my_app_id', 'my_app_secret')
# add App authentication
twitch.authenticate_app([])
```

User Authentication

To get a user auth token, the user has to explicitly click "Authorize" on the twitch website. You can use various online services to generate a token or use my build in authenticator.

See *twitchAPI.oauth* for more info.

Class Documentation:

class twitchAPI.twitch.**Twitch**(*app_id: str*, *app_secret: str*)

Twitch API client

Parameters

- **app_id** (*str*) – Your app id
- **app_secret** (*str*) – Your app secret

Variables **auto_refresh_auth** (*bool*) – If set to true, auto refresh the auth token once it expires. True

authenticate_app(*scope: List[twitchAPI.types.AuthScope]*) → None

Authenticate with a fresh generated app token

Parameters **scope** (*list[AuthScope]*) – List of Authorization scopes to use

Raises ***TwitchAuthorizationException*** – if the authentication fails

Returns None

check_automod_status(*broadcaster_id: str*, *msg_id: str*, *msg_text: str*, *user_id: str*) → dict

Determines whether a string message meets the channel's AutoMod requirements.

Requires User authentication with scope twitchAPI.types.AuthScope.MODERATION_READ

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#check-automod-status

Parameters

- **broadcaster_id** (*str*) – Provided broadcaster ID must match the user ID in the user auth token.

- **msg_id** (*str*) – Developer-generated identifier for mapping messages to results.
- **msg_text** (*str*) – Message text.
- **user_id** (*str*) – User ID of the sender.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

create_clip(*broadcaster_id: str*, *has_delay: bool = False*) → dict

Creates a clip programmatically. This returns both an ID and an edit URL for the new clip.

Requires User authentication with scope `twitchAPI.types.AuthScope.CLIPS_EDIT`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#create-clip

Parameters

- **broadcaster_id** (*str*) – Broadcaster ID of the stream from which the clip will be made.
- **has_delay** (*bool*) – If False, the clip is captured from the live stream when the API is called; otherwise, a delay is added before the clip is captured (to account for the brief delay between the broadcaster's stream and the viewer's experience of that stream). `False`

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

create_entitlement_grants_upload_url(*manifest_id: str*) → dict

Creates a URL where you can upload a manifest file and notify users that they have an entitlement. Entitlements are digital items that users are entitled to use. Twitch entitlements are granted to users gratis or as part of a purchase on Twitch.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#create-entitlement-grants-upload-url

Parameters **manifest_id** (*str*) – Unique identifier of the manifest file to be uploaded. Must be 1-64 characters.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the the authentication token became invalid and a re authentication failed

- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if length of manifest_id is not in range 1 to 64

Return type dict

create_stream_marker(*user_id: str*, *description: Optional[str] = None*) → dict

Creates a marker in the stream of a user specified by user ID.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_EDIT_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#create-stream-marker

Parameters

- **user_id** (*str*) – ID of the broadcaster in whose live stream the marker is created.
- **description** (*str*) – Description of or comments on the marker. Max length is 140 characters.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if description has more than 140 characters

Return type dict

create_user_follows(*from_id: str*, *to_id: str*, *allow_notifications: Optional[bool] = False*) → bool

Adds a specified user to the followers of a specified channel.

Requires User authentication with `twitchAPI.types.AuthScope.USER_EDIT_FOLLOWS`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#create-user-follows

Parameters

- **from_id** (*str*) – User ID of the follower
- **to_id** (*str*) – ID of the channel to be followed by the user
- **allow_notifications** (*bool*) – If true, the user gets email or push notifications (depending on the user's notification settings) when the channel goes live. `false`

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type bool

delete_user_follows(*from_id: str*, *to_id: str*) → bool
Deletes a specified user from the followers of a specified channel.

Requires User authentication with `twitchAPI.types.AuthScope.USER_EDIT_FOLLOWS`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#delete-user-follows

Parameters

- **from_id** (*str*) – User ID of the follower
- **to_id** (*str*) – Channel to be unfollowed by the user

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type bool

get_all_stream_tags(*after: Optional[str] = None*, *first: int = 20*, *tag_ids: Optional[List[str]] = None*) → dict
Gets the list of all stream tags defined by Twitch, optionally filtered by tag ID(s).

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-all-stream-tags

Parameters

- **after** (*str*) – Cursor for forward pagination
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. `20`
- **tag_ids** (*list[str]*) – IDs of tags. Maximum 100 entries

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100 or tag_ids has more than 100 entries

Return type dict

get_app_token() → Optional[str]
Returns the app token that the api uses or None when not authenticated.

Returns app token

Return type Union[str, None]

get_banned_events(*broadcaster_id: str*, *user_id: Optional[str] = None*, *after: Optional[str] = None*, *first: int = 20*) → dict
Returns all user bans and un-bans in a channel.

Requires User authentication with scope `twitchAPI.types.AuthScope.MODERATION_READ`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-banned-events

Parameters

- **broadcaster_id** (*str*) – Provided broadcaster ID must match the user ID in the user auth token.
- **user_id** (*str*) – Filters the results and only returns a status object for users who are banned in this channel and have a matching user_id
- **after** (*str*) – Cursor for forward pagination
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 ot 100

Return type dict

get_banned_users(*broadcaster_id: str*, *user_id: Optional[str] = None*, *after: Optional[str] = None*, *before: Optional[str] = None*) → dict

Returns all banned and timed-out users in a channel.

Requires User authentication with scope `twitchAPI.types.AuthScope.MODERATION_READ`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-banned-users

Parameters

- **broadcaster_id** (*str*) – Provided broadcaster ID must match the user ID in the user auth token.
- **user_id** (*str*) – Filters the results and only returns a status object for users who are banned in this channel and have a matching user_id.
- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_bits_leaderboard(*count: int = 10*, *period: twitchAPI.types.TimePeriod = <TimePeriod.ALL: 'all'>*, *started_at: Optional[datetime.datetime] = None*, *user_id: Optional[str] = None*) → dict

Gets a ranked list of Bits leaderboard information for an authorized broadcaster.

Requires User authentication with scope `twitchAPI.types.AuthScope.BITS_READ`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-bits-leaderboard

Parameters

- **count** (*int*) – Number of results to be returned. In range 1 to 100, `10`
- **period** (`TimePeriod`) – Time period over which data is aggregated, `twitchAPI.types.TimePeriod.ALL`
- **started_at** (*datetime*) – Timestamp for the period over which the returned data is aggregated.
- **user_id** (*str*) – ID of the user whose results are returned; i.e., the person who paid for the Bits.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

get_broadcaster_subscriptions(*broadcaster_id: str*, *user_ids: Optional[List[str]] = None*) → dict

Get all of a broadcaster's subscriptions.

Requires User authentication with scope `twitchAPI.types.AuthScope.CHANNEL_READ_SUBSCRIPTIONS`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-broadcaster-subscriptions

Parameters

- **broadcaster_id** (*str*) – User ID of the broadcaster. Must match the User ID in the Bearer token.
- **user_ids** (*list[str]*) – Unique identifier of account to get subscription status of. Maximum 100 entries

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if user_ids has more than 100 entries

Return type dict

get_channel_information(*broadcaster_id: str*) → dict

Gets channel information for users.

Requires App or user authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-channel-information

Parameters **broadcaster_id** (*str*) – ID of the channel to be updated

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_cheermotes(*broadcaster_id: str*) → dict

Retrieves the list of available Cheermotes, animated emotes to which viewers can assign Bits, to cheer in chat.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-cheermotes

Parameters **broadcaster_id** (*str*) – ID for the broadcaster who might own specialized Cheermotes.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_clips(*broadcaster_id: Optional[str] = None*, *game_id: Optional[str] = None*, *clip_id: Optional[List[str]] = None*, *after: Optional[str] = None*, *before: Optional[str] = None*, *ended_at: Optional[datetime.datetime] = None*, *started_at: Optional[datetime.datetime] = None*, *first: int = 20*) → dict

Gets clip information by clip ID (one or more), broadcaster ID (one only), or game ID (one only). Clips are returned sorted by view count, in descending order.

Requires App or User authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-clips

Parameters

- **broadcaster_id** (*str*) – ID of the broadcaster for whom clips are returned.
- **game_id** (*str*) – ID of the game for which clips are returned.
- **clip_id** (*list[str]*) – ID of the clip being queried. Limit: 100.
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20
- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination
- **ended_at** (*datetime*) – Ending date/time for returned clips
- **started_at** (*datetime*) – Starting date/time for returned clips

Raises

- ***UnauthorizedException*** – if user authentication is not set

- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if you try to query more than 100 clips in one call
- **ValueError** – if not exactly one of clip_id, broadcaster_id or game_id is given
- **ValueError** – if first is not in range 1 to 100

Return type dict

get_code_status(*code: List[str], user_id: int*) → dict

Gets the status of one or more provided Bits codes.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-code-status

Parameters

- **code** (*list[str]*) – The code to get the status of. Maximum of 20 entries
- **user_id** (*int*) – Represents the numeric Twitch user ID of the account which is going to receive the entitlement associated with the code.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if length of code is not in range 1 to 20

Return type dict

get_drops_entitlements(*id: Optional[str] = None, user_id: Optional[str] = None, game_id: Optional[str] = None, after: Optional[str] = None, first: Optional[int] = 20*) → dict

Gets a list of entitlements for a given organization that have been granted to a game, user, or both.

OAuth Token Client ID must have ownership of Game

Requires App authentication

See Twitch documentation for valid parameter combinations!

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-drops-entitlements

Parameters

- **id** (*str*) – Unique Identifier of the entitlement
- **user_id** (*str*) – A Twitch User ID
- **game_id** (*str*) – A Twitch Game ID
- **after** (*str*) – The cursor used to fetch the next page of data.
- **first** (*int*) – Maximum number of entitlements to return. Maximum: 100 20

Raises

- ***UnauthorizedException*** – if app authentication is not set

- **TwitchAuthorizationException** – if the used authentication token became invalid and a re authentication failed
- **TwitchBackendException** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

get_extension_analytics(*after: Optional[str] = None*, *extension_id: Optional[str] = None*, *first: int = 20*, *ended_at: Optional[datetime.datetime] = None*, *started_at: Optional[datetime.datetime] = None*, *report_type: Optional[twitchAPI.types.AnalyticsReportType] = None*) → dict

Gets a URL that extension developers can use to download analytics reports (CSV files) for their extensions. The URL is valid for 5 minutes.

Requires User authentication with scope `twitchAPI.types.AuthScope.ANALYTICS_READ_EXTENSION`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-extension-analytics

Parameters

- **after** (*str*) – cursor for forward pagination
- **extension_id** (*str*) – If this is specified, the returned URL points to an analytics report for just the specified extension.
- **first** (*int*) – Maximum number of objects returned, range 1 to 100, `20`
- **ended_at** (*datetime*) – Ending date/time for returned reports, if this is provided, *started_at* must also be specified.
- **started_at** (*datetime*) – Starting date/time for returned reports, if this is provided, *ended_at* must also be specified.
- **report_type** (`AnalyticsReportType`) – Type of analytics report that is returned

Return type dict

Raises

- **UnauthorizedException** – if user authentication is not set
- **MissingScopeException** – if the user authentication is missing the required scope
- **TwitchAuthorizationException** – if the used authentication token became invalid and a re authentication failed
- **TwitchBackendException** – if the Twitch API itself runs into problems
- **ValueError** – When you only supply *started_at* or *ended_at* without the other or when first is not in range 1 to 100

get_extension_transactions(*extension_id: str*, *transaction_id: Optional[str] = None*, *after: Optional[str] = None*, *first: int = 20*) → dict

Get Extension Transactions allows extension back end servers to fetch a list of transactions that have occurred for their extension across all of Twitch. A transaction is a record of a user exchanging Bits for an in-Extension digital good.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-extension-transactions

Parameters

- **extension_id** (*str*) – ID of the extension to list transactions for.

- **transaction_id** (*str*) – Transaction IDs to look up.
- **after** (*str*) – cursor for forward pagination
- **first** (*int*) – Maximum number of objects returned, range 1 to 100, 20

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

get_game_analytics(*after: Optional[str] = None*, *first: int = 20*, *game_id: Optional[str] = None*, *ended_at: Optional[datetime.datetime] = None*, *started_at: Optional[datetime.datetime] = None*, *report_type: Optional[twitchAPI.types.AnalyticsReportType] = None*) → dict

Gets a URL that game developers can use to download analytics reports (CSV files) for their games. The URL is valid for 5 minutes.

Requires User authentication with scope `twitchAPI.types.AuthScope.ANALYTICS_READ_GAMES`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-game-analytics

Parameters

- **after** (*str*) – cursor for forward pagination
- **first** (*int*) – Maximum number of objects returned, range 1 to 100, 20
- **game_id** (*str*) – Game ID
- **ended_at** (*datetime*) – Ending date/time for returned reports, if this is provided, *started_at* must also be specified.
- **started_at** (*datetime*) – Starting date/time for returned reports, if this is provided, *ended_at* must also be specified.
- **report_type** (AnalyticsReportType) – Type of analytics report that is returned.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – When you only supply *started_at* or *ended_at* without the other or when first is not in range 1 to 100

Return type dict

get_games(*game_ids: Optional[List[str]] = None*, *names: Optional[List[str]] = None*) → dict

Gets game information by game ID or name.

Requires User or App authentication. In total, only 100 game ids and names can be fetched at once.

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-games

Parameters

- **game_ids** (*list[str]*) – Game ID
- **names** (*list[str]*) – Game Name

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if neither game_ids nor names are given or if game_ids and names are more than 100 entries combined.

Return type dict

get_hype_train_events(*broadcaster_id: str*, *first: Optional[int] = 1*, *id: Optional[str] = None*, *cursor: Optional[str] = None*) → dict

Gets the information of the most recent Hype Train of the given channel ID. When there is currently an active Hype Train, it returns information about that Hype Train. When there is currently no active Hype Train, it returns information about the most recent Hype Train. After 5 days, if no Hype Train has been active, the endpoint will return an empty response.

Requires App or User authentication with `twitchAPI.types.AuthScope.CHANNEL_READ_HYPE_TRAIN`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-hype-train-events

Parameters

- **broadcaster_id** (*str*) – User ID of the broadcaster.
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. `1`
- **id** (*str*) – The id of the wanted event, if known
- **cursor** (*str*) – Cursor for forward pagination

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***MissingScopeException*** – if the user or app authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

get_moderator_events(*broadcaster_id: str*, *user_ids: Optional[List[str]] = None*) → dict

Returns a list of moderators or users added and removed as moderators from a channel.

Requires User authentication with scope `twitchAPI.types.AuthScope.MODERATION_READ`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-moderator-events

Parameters

- **broadcaster_id** (*str*) – Provided broadcaster ID must match the user ID in the user auth token.
- **user_ids** (*list[str]*) – Filters the results and only returns a status object for users who are moderator in this channel and have a matching user_id. Maximum 100

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if user_ids has more than 100 entries

Return type dict

get_moderators(*broadcaster_id: str*, *user_ids: Optional[List[str]] = None*, *after: Optional[str] = None*) → dict

Returns all moderators in a channel.

Requires User authentication with scope `twitchAPI.types.AuthScope.MODERATION_READ`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-moderators

Parameters

- **broadcaster_id** (*str*) – Provided broadcaster ID must match the user ID in the user auth token.
- **user_ids** (*list[str]*) – Filters the results and only returns a status object for users who are moderator in this channel and have a matching user_id. Maximum 100
- **after** (*str*) – Cursor for forward pagination

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if user_ids has more than 100 entries

Return type dict

get_stream_key(*broadcaster_id: str*) → dict

Gets the channel stream key for a user.

Requires User authentication with `twitchAPI.types.AuthScope.CHANNEL_READ_STREAM_KEY`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-stream-key

Parameters **broadcaster_id** (*str*) – User ID of the broadcaster

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope

- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_stream_markers(*user_id: str*, *video_id: str*, *after: Optional[str] = None*, *before: Optional[str] = None*, *first: int = 20*) → dict

Gets a list of markers for either a specified user's most recent stream or a specified VOD/video (stream), ordered by recency.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_READ_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-stream-markers

Only one of user_id and video_id must be specified.

Parameters

- **user_id** (*str*) – ID of the broadcaster from whose stream markers are returned.
- **video_id** (*str*) – ID of the VOD/video whose stream markers are returned.
- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination
- **first** (*int*) – Number of values to be returned when getting videos by user or game ID. Limit: 100. `20`

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100 or neither user_id nor video_id is provided

Return type dict

get_stream_tags(*broadcaster_id: str*) → dict

Gets the list of tags for a specified stream (channel).

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-stream-tags

Parameters **broadcaster_id** (*str*) – ID of the stream thats tags are going to be fetched

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_streams(*after: Optional[str] = None*, *before: Optional[str] = None*, *first: int = 20*, *game_id: Optional[List[str]] = None*, *language: Optional[List[str]] = None*, *user_id: Optional[List[str]] = None*, *user_login: Optional[List[str]] = None*) → dict

Gets information about active streams. Streams are returned sorted by number of current viewers, in descending order. Across multiple pages of results, there may be duplicate or missing streams, as viewers join and leave streams.

Requires App or User authentication.

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-streams

Parameters

- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20
- **game_id** (*list[str]*) – Returns streams broadcasting a specified game ID. You can specify up to 100 IDs.
- **language** (*list[str]*) – Stream language. You can specify up to 100 languages.
- **user_id** (*list[str]*) – Returns streams broadcast by one or more specified user IDs. You can specify up to 100 IDs.
- **user_login** (*list[str]*) – Returns streams broadcast by one or more specified user login names. You can specify up to 100 names.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100 or one of the following fields have more than 100 entries: *user_id, game_id, language, user_login*

Return type dict

get_top_games(*after: Optional[str] = None*, *before: Optional[str] = None*, *first: int = 20*) → dict

Gets games sorted by number of current viewers on Twitch, most popular first.

Requires App or User authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-top-games

Parameters

- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

- **ValueError** – if first is not in range 1 to 100

Return type dict

get_used_token() → Optional[str]
Returns the currently used token, can be either the app or user auth Token or None if no auth is set

Returns the currently used auth token or None if no Authentication is set

get_user_active_extensions(*user_id: Optional[str] = None*) → dict
Gets information about active extensions installed by a specified user, identified by a user ID or the authenticated user.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_READ_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-user-active-extensions

Parameters **user_id** (*str*) – ID of the user whose installed extensions will be returned.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_user_auth_scope() → List[twitchAPI.types.AuthScope]
Returns the set User auth Scope

get_user_auth_token() → Optional[str]
Returns the current user auth token, None if no user Authentication is set

Returns current user auth token

Return type str or None

get_user_extensions() → dict
Gets a list of all extensions (both active and inactive) for the authenticated user

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_READ_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-user-extensions

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

get_users(*user_ids: Optional[List[str]] = None*, *logins: Optional[List[str]] = None*) → dict
Gets information about one or more specified Twitch users. Users are identified by optional user IDs and/or login name. If neither a user ID nor a login name is specified, the user is the one authenticated.

Requires App authentication if either user_ids or logins is provided, otherwise requires a User authentication. If you have user Authentication and want to get your email info, you also need the authentication scope `twitchAPI.types.AuthScope.USER_READ_EMAIL`

If you provide user_ids and/or logins, the maximum combined entries should not exceed 100.

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-users

Parameters

- **user_ids** (*list[str]*) – User ID. Multiple user IDs can be specified. Limit: 100.
- **logins** (*list[str]*) – User login name. Multiple login names can be specified. Limit: 100.

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if more than 100 combined user_ids and logins where provided

Return type dict

get_users_follows(*after: Optional[str] = None*, *first: int = 20*, *from_id: Optional[str] = None*, *to_id: Optional[str] = None*) → dict

Gets information on follow relationships between two Twitch users. Information returned is sorted in order, most recent follow first.

Requires App authentication.

You have to use at least one of the following fields: from_id, to_id For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-users-follows

Parameters

- **after** (*str*) – Cursor for forward pagination
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. `20`
- **from_id** (*str*) – User ID. The request returns information about users who are being followed by the from_id user.
- **to_id** (*str*) – User ID. The request returns information about users who are following the to_id user.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100 or neither from_id nor to_id is provided

Return type dict

get_videos(*ids: Optional[List[str]] = None*, *user_id: Optional[str] = None*, *game_id: Optional[str] = None*, *after: Optional[str] = None*, *before: Optional[str] = None*, *first: int = 20*, *language: Optional[str] = None*, *period: twitchAPI.types.TimePeriod = <TimePeriod.ALL: 'all'>*, *sort: twitchAPI.types.SortMethod = <SortMethod.TIME: 'time'>*, *video_type: twitchAPI.types.VideoType = <VideoType.ALL: 'all'>*) → dict

Gets video information by video ID (one or more), user ID (one only), or game ID (one only).

Requires App authentication.

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-videos

Parameters

- **ids** (*list[str]*) – ID of the video being queried. Limit: 100.
- **user_id** (*str*) – ID of the user who owns the video.
- **game_id** (*str*) – ID of the game the video is of.
- **after** (*str*) – Cursor for forward pagination
- **before** (*str*) – Cursor for backward pagination
- **first** (*int*) – Number of values to be returned when getting videos by user or game ID. Limit: 100. 20
- **language** (*str*) – Language of the video being queried.
- **period** (TimePeriod) – Period during which the video was created.
- **sort** (SortMethod) – Sort order of the videos.
- **video_type** (VideoType) – Type of video.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100, ids has more than 100 entries or none of ids, user_id nor game_id is provided.

Return type dict

get_webhook_subscriptions(*first: Optional[int] = 20*, *after: Optional[str] = None*) → dict

Gets the Webhook subscriptions of the authenticated user, in order of expiration.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#get-webhook-subscriptions

Parameters

- **first** (*int*) – Number of values to be returned per page. Limit: 100. 20
- **after** (*str*) – Cursor for forward pagination

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

- **ValueError** – if first is not in range 1 to 100

Return type dict

modify_channel_information(*broadcaster_id: str*, *game_id: Optional[str] = None*, *broadcaster_language: Optional[str] = None*, *title: Optional[str] = None*) → bool

Modifies channel information for users.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_EDIT_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#modify-channel-information

Parameters

- **broadcaster_id** (*str*) – ID of the channel to be updated
- **game_id** (*str*) – The current game ID being played on the channel
- **broadcaster_language** (*str*) – The language of the channel
- **title** (*str*) – The title of the stream

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if none of the following fiends are specified: *game_id, broadcaster_language, title*

Return type bool

redeem_code(*code: List[str], user_id: int*) → dict

Redeems one or more provided Bits codes to the authenticated Twitch user.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#redeem-code

Parameters

- **code** (*list[str]*) – The code to redeem to the authenticated user's account. Maximum of 20 entries
- **user_id** (*int*) – Represents the numeric Twitch user ID of the account which is going to receive the entitlement associated with the code.

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if length of code is not in range 1 to 20

Return type dict

refresh_used_token()
Refreshes the currently used token

replace_stream_tags(*broadcaster_id: str, tag_ids: List[str]*) → dict
Applies specified tags to a specified stream, overwriting any existing tags applied to that stream. If no tags are specified, all tags previously applied to the stream are removed. Automated tags are not affected by this operation.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_EDIT_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#replace-stream-tags

Parameters

- **broadcaster_id** (*str*) – ID of the stream for which tags are to be replaced.
- **tag_ids** (*list[str]*) – IDs of tags to be applied to the stream. Maximum of 100 supported.

Returns {}

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if more than 100 tag_ids where provided

Return type dict

search_categories(*query: str*, *first: Optional[int] = 20*, *after: Optional[str] = None*) → dict
Returns a list of games or categories that match the query via name either entirely or partially.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#search-categories

Parameters

- **query** (*str*) – search query
- **first** (*int*) – Maximum number of objects to return. Maximum: 100 `20`
- **after** (*str*) – Cursor for forward pagination

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

search_channels(*query: str*, *first: Optional[int] = 20*, *after: Optional[str] = None*, *live_only: Optional[bool] = False*) → dict

Returns a list of channels (users who have streamed within the past 6 months) that match the query via channel name or description either entirely or partially.

Requires App authentication

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#search-channels

Parameters

- **query** (*str*) – search query
- **first** (*int*) – Maximum number of objects to return. Maximum: 100 `20`
- **after** (*str*) – Cursor for forward pagination
- **live_only** (*bool*) – Filter results for live streams only. `False`

Raises

- ***UnauthorizedException*** – if app authentication is not set
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if first is not in range 1 to 100

Return type dict

set_user_authentication(*token: str*, *scope: List[twitchAPI.types.AuthScope]*, *refresh_token: Optional[str] = None*) → None

Set a user token to be used.

Parameters

- **token** (*str*) – the generated user token
- **scope** (*list[AuthScope]*) – List of Authorization Scopes that the given user token has
- **refresh_token** (*str*) – The generated refresh token, has to be provided if `auto_refresh_auth` is True

Returns None

Raises **ValueError** – if `auto_refresh_auth` is True but refresh_token is not set

start_commercial(*broadcaster_id: str*, *length: int*) → dict

Starts a commercial on a specified channel.

Requires User authentication with `twitchAPI.types.AuthScope.CHANNEL_EDIT_COMMERCIAL`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#start-commercial

Parameters

- **broadcaster_id** (*str*) – ID of the channel requesting a commercial
- **length** (*int*) – Desired length of the commercial in seconds. , one of these: [30, 60, 90, 120, 150, 180]

Raises

- ***UnauthorizedException*** – if user authentication is not set

- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems
- **ValueError** – if length is not one of these: *30, 60, 90, 120, 150, 180*

Return type dict

update_user(*description: str*) → dict
Updates the description of the Authenticated user.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_EDIT`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#update-user

Parameters **description** (*str*) – User's account description

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

update_user_extensions(*data: dict*) → dict
"Updates the activation state, extension ID, and/or version number of installed extensions for the authenticated user.

Requires User authentication with scope `twitchAPI.types.AuthScope.USER_EDIT_BROADCAST`

For detailed documentation, see here: https://dev.twitch.tv/docs/api/reference#update-user-extensions

Parameters **data** (*dict*) – The user extension data to be written

Raises

- ***UnauthorizedException*** – if user authentication is not set
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchBackendException*** – if the Twitch API itself runs into problems

Return type dict

4.2 twitchAPI.webhook

4.2.1 Full Implementation of the Twitch Webhook

The Webhook runs in its own thread, calling the given callback function whenever an webhook event happens.

Look at the Twitch Webhook reference to find the topics you are interested in.

Requirements

You need to have a public IP with a port open. That port will be 80 by default. Authentication is off by default but you can choose to authenticate to use some Webhook Topics or to get more information.

Note: Please note that Your Endpoint URL has to be HTTPS if you need authentication which means that you probably need a reverse proxy like nginx. This lib currently does not provide any way to add https on its own.

Short code example:

```
from twitchAPI.twitch import Twitch
from twitchAPI.webhook import TwitchWebHook
from pprint import pprint

def callback_stream_changed(uuid, data):
    print('Callback for UUID ' + str(uuid))
    pprint(data)

twitch = Twitch(td['app_id'], td['secret'])
twitch.authenticate_app([])

user_info = twitch.get_users(logins=['my_twitch_user'])
user_id = user_info['data'][0]['id']
# basic setup
hook = TwitchWebHook("https://my.cool.domain.net:8080", 'my_app_id', 8080)
hook.authenticate(twitch)
hook.start()
print('subscribing to hook:')
success, uuid = hook.subscribe_stream_changed(user_id, callback_stream_changed)
pprint(success)
pprint(twitch.get_webhook_subscriptions())
# the webhook is now running and you are subscribed to the topic you want to listen␣
↪to. lets idle a bit...
input('press Enter to shut down...\n')
hook.stop()
print('done')
```

Subscription handling

You can subscribe to webhook topics using the `subscribe_` prefixed methods.

If `wait_for_subscription_confirm` is True (default), this will wait for the full handshake and confirmation to happen, otherwise the returned success value might be inaccurate in case the subscription itself succeeded but the final handshake failed.

You can unsubscribe from a webhook subscription at any time by using `unsubscribe()`

If `unsubscribe_on_stop` is True (default), you dont need to manually unsubscribe from topics.

By deafult, subscriptions will be automatically renewed one minute before they run out for as long as the webhook is running.

You can also use *`unsubscribe_all()`* to unsubscribe from all topic subscriptions at once. This will also unsubscribe from topics that where left over from a previous run.

Class Documentation:

class twitchAPI.webhook.**TwitchWebHook**(*callback_url: str*, *api_client_id: str*, *port: int*)
Webhook integration for the Twitch Helix API.

Parameters

- **callback_url** (*str*) – The full URL of the webhook.
- **api_client_id** (*str*) – The id of your API client
- **port** (*int*) – the port on which this webhook should run

Variables

- **secret** (*str*) – A random secret string. Set this for added security.
- **callback_url** (*str*) – The full URL of the webhook.
- **subscribe_least_seconds** (*int*) – The duration in seconds for how long you want to subscribe to webhhoks. Min 300 Seconds, Max 864000 Seconds. 600
- **auto_renew_subscription** (*bool*) – If True, automatically renew all webhooks once they get close to running out. **Only disable this if you know what you are doing.** True
- **wait_for_subscription_confirm** (*bool*) – Set this to false if you dont want to wait for a subscription confirm. True
- **wait_for_subscription_confirm_timeout** (*int*) – Max time in seconds to wait for a subscription confirmation. Only used if wait_for_subscription_confirm is set to True. 30
- **unsubscribe_on_stop** (*bool*) – Unsubscribe all currently active Webhooks on calling *stop()* True

authenticate(*twitch: twitchAPI.twitch.Twitch*) → None
Set authentication for the Webhook. Can be either a app or user token.

Parameters **twitch** (Twitch) – a authenticated instance of *Twitch*

Return type None

Raises **RuntimeError** – if the callback URL does not use HTTPS

renew_subscription(*uuid: uuid.UUID*) → bool
Renew existing topic subscription

Parameters **uuid** – UUID of the subscription to renew

Return type bool

Returns True if renewal worked. Note that you still need to wait for the handshake to make sure its renewed.

start()
Starts the Webhook

Return type None

Raises

- **ValueError** – if subscribe_least_seconds is not in range 300 to 864000
- **RuntimeError** – if webhook is already running

stop()
Stops the Webhook

Please make sure to unsubscribe from all subscriptions!

Return type None

subscribe_channel_ban_change_events(*broadcaster_id: str, user_id: Optional[str], callback_func: Callable[[uuid.UUID, dict], None]*) → Tuple[bool, uuid.UUID]
Subscribe to Channel Ban Change Events

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-channel-ban-change-events for documentation

Parameters

- **broadcaster_id** – str
- **user_id** – str or None
- **callback_func** – function for callback

Return type bool, UUID

subscribe_extension_transaction_created(*extension_id: str, callback_func: Optional[Callable[[uuid.UUID, dict], None]]*) → Tuple[bool, uuid.UUID]
Subscribe to Extension transaction topic

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-extension-transaction-created for documentation

Parameters

- **extension_id** – str
- **callback_func** – function for callback

Return type bool, UUID

subscribe_hype_train_events(*broadcaster_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → Tuple[bool, uuid.UUID]
Subscribe to Hype Train Events

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-hype-train-event for documentation

Parameters

- **broadcaster_id** – str
- **callback_func** – function for callback

Return type bool, UUID

subscribe_moderator_change_events(*broadcaster_id: str, user_id: Optional[str], callback_func: Callable[[uuid.UUID, dict], None]*) → Tuple[bool, uuid.UUID]
Subscribe to Moderator Change Events topic

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-moderator-change-events for documentation

Parameters

- **broadcaster_id** – str
- **user_id** – str or None

- **callback_func** – function for callback

Return type bool, UUID

subscribe_stream_changed(*user_id: str, callback_func: Optional[Callable[[uuid.UUID, dict], None]]*) → Tuple[bool, uuid.UUID]

Subscribe to stream changed topic

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-stream-changed for documentation

Parameters

- **user_id** – str
- **callback_func** – function for callback

Return type bool, UUID

subscribe_subscription_events(*broadcaster_id: str, callback_func: Callable[[uuid.UUID, dict], None], user_id: Optional[str] = None, gifter_id: Optional[str] = None, gifter_name: Optional[str] = None*) → Tuple[bool, uuid.UUID]

Subscribe to Subscription Events Topic

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-subscription-events for documentation

Parameters

- **broadcaster_id** – str
- **callback_func** – function for callback
- **user_id** – optional str
- **gifter_id** – optional str
- **gifter_name** – optional str

Return type bool, UUID

subscribe_user_changed(*user_id: str, callback_func: Optional[Callable[[uuid.UUID, dict], None]]*) → Tuple[bool, uuid.UUID]

Subscribe to subscription event topic

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-user-changed for documentation

Parameters

- **user_id** – str
- **callback_func** – function for callback

Return type bool, UUID

subscribe_user_follow(*from_id: Optional[str], to_id: Optional[str], callback_func: Optional[Callable[[uuid.UUID, dict], None]]*) → Tuple[bool, uuid.UUID]

Subscribe to user follow topic.

Set only from_id if you want to know if User with that id follows someone.

Set only to_id if you want to know if someone follows User with that id.

Set both if you only want to know if from_id follows to_id.

See https://dev.twitch.tv/docs/api/webhooks-reference#topic-user-follows for documentation

Parameters

- **from_id** – str or None

- **to_id** – str or None
- **callback_func** – function for callback

Return type bool, UUID

unsubscribe_all(*twitch: twitchAPI.twitch.Twitch*) → bool
Unsubscribe from all Webhooks that use the callback URL set in *callback_url*

If 'wait_for_subscription_confirm' is False, the response might be True even tho the unsubscribe action failed.

Parameters **twitch** (Twitch) – App authorized instance of *Twitch*

Return type bool

Returns True if all webhooks could be unsubscribed, otherwise False.

4.3 twitchAPI.pubsub

4.3.1 PubSub client

This is a full implementation of the PubSub API of twitch. PubSub enables you to subscribe to a topic, for updates (e.g., when a user cheers in a channel).

Read more about it on the Twitch API Documentation.

Note: You **always** need User Authentication while using this!

Short code example:

```
from twitchAPI.pubsub import PubSub
from twitchAPI.twitch import Twitch
from pprint import pprint
from uuid import UUID

def callback_whisper(uuid: UUID, data: dict) -> None:
    print('got callback for UUID ' + str(uuid))
    pprint(data)

twitch = Twitch('my_app_id', 'my_app_secret')
twitch.authenticate_app([])
twitch.set_user_authentication('my_user_auth_token', [AuthScope.WHISPERS_READ], 'my_
↪user_auth_refresh_token')
user_id = twitch.get_users(logins=['my_username'])['data'][0]['id']

pubsub = PubSub(twitch)
pubsub.start()
# you can either start listening before or after you started pubsub.
uuid = pubsub.listen_whispers(user_id, callback_whisper)
input('press ENTER to close...')
# you do not need to unlisten to topics before stopping but you can listen and␣
↪unlisten at any moment you want
pubsub.unlisten(uuid)
pubsub.stop()
```

Class Documentation:

class twitchAPI.pubsub.**PubSub**(*twitch: twitchAPI.twitch.Twitch*)
The PubSub client

Variables

- **ping_frequency** (*int*) – with which frequency in seconds a ping command is send. You probably don't want to change this. This should never be shorter than 12 + *ping_jitter* seconds to avoid problems with the pong timeout. 120
- **ping_jitter** (*int*) – time in seconds added or subtracted from *ping_frequency*. You probably don't want to change this. 4
- **listen_confirm_timeout** (*int*) – maximum time in seconds waited for a listen confirm. 30

listen_bits(*channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID
You are notified when anyone cheers in the specified channel.

Requires the twitchAPI.types.AuthScope.BITS_READ AuthScope.

Parameters

- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_bits_badge_notification(*channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID
You are notified when a user earns a new Bits badge in the given channel, and chooses to share the notification with chat.

Requires the twitchAPI.types.AuthScope.BITS_READ AuthScope.

Parameters

- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem

- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_bits_v1(*channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID

You are notified when anyone cheers in the specified channel.

Requires the `twitchAPI.types.AuthScope.BITS_READ` AuthScope.

Parameters

- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_channel_points(*channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID

You are notified when a custom reward is redeemed in the channel.

Requires the `twitchAPI.types.AuthScope.CHANNEL_READ_REDEMPTIONS` AuthScope.

Parameters

- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_channel_subscriptions(*channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID

You are notified when anyone subscribes (first month), resubscribes (subsequent months), or gifts a subscription to a channel. Subgift subscription messages contain recipient information.

Requires the `twitchAPI.types.AuthScope.CHANNEL_SUBSCRIPTIONS` AuthScope.

Parameters

- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_chat_moderator_actions(*user_id: str, channel_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID

Supports moderators listening to the topic, as well as users listening to the topic to receive their own events. Examples of moderator actions are bans, unbans, timeouts, deleting messages, changing chat mode (followers-only, subs-only), changing AutoMod levels, and adding a mod.

Requires the `twitchAPI.types.AuthScope.CHANNEL_MODERATE` AuthScope.

Parameters

- **user_id** (*str*) – ID of the User
- **channel_id** (*str*) – ID of the Channel
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

listen_whispers(*user_id: str, callback_func: Callable[[uuid.UUID, dict], None]*) → uuid.UUID

You are notified when anyone whispers the specified user or the specified user whispers to anyone.

Requires the `twitchAPI.types.AuthScope.WHISPERS_READ` AuthScope.

Parameters

- **user_id** (*str*) – ID of the User
- **callback_func** (*Callable[[UUID, dict], None]*) – Function called on event

Returns UUID of this subscription

Return type UUID

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the subscription response is something else than suspected
- ***PubSubListenTimeoutException*** – if the subscription is not confirmed in the time set by *listen_confirm_timeout*
- ***MissingScopeException*** – if required AuthScope is missing from Token

start() → None
Start the PubSub Client

Raises **RuntimeError** – if already started

stop() → None
Stop the PubSub Client

Raises **RuntimeError** – if the client is not running

unlisten(*uuid: uuid.UUID*) → None
Stop listening to a specific Topic subscription.

Parameters **uuid** (*UUID*) – The UUID of the subscription you want to stop listening to

Raises

- ***TwitchAuthorizationException*** – if Token is not valid
- ***TwitchBackendException*** – if the Twitch Server has a problem
- ***TwitchAPIException*** – if the server response is something else than suspected
- ***PubSubListenTimeoutException*** – if the unsubscription is not confirmed in the time set by *listen_confirm_timeout*

4.4 twitchAPI.oauth

4.4.1 User OAuth Authenticator and helper functions

This tool is an alternative to various online services that give you a user auth token.

Requirements

Since this tool opens a browser tab for the Twitch authentication, you can only use this tool on enviroments that can open a browser window and render the twitch.tv website.

For my authenticator you have to add the following URL as a "OAuth Redirect URL": `http://localhost:17563` You can set that here in your twitch dev dashboard.

Code example

```
from twitchAPI.twitch import Twitch
from twitchAPI.oauth import UserAuthenticator
from twitchAPI.types import AuthScope

twitch = Twitch('my_app_id', 'my_app_secret')

target_scope = [AuthScope.BITS_READ]
auth = UserAuthenticator(twitch, target_scope, force_verify=False)
# this will open your default browser and prompt you with the twitch verification␣
↪website
token, refresh_token = auth.authenticate()
# add User authentication
twitch.set_user_authentication(token, target_scope, refresh_token)
```

Class Documentation:

class twitchAPI.oauth.**UserAuthenticator**(*twitch: twitchAPI.twitch.Twitch, scopes: List[twitchAPI.types.AuthScope], force_verify: bool = False*)

Simple to use client for the Twitch User authentication flow.

Parameters

- **twitch** (Twitch) – A twitch instance
- **scopes** (*list[AuthScope]*) – List of the desired Auth scopes
- **force_verify** (*bool*) –

  If this is true, the user will always be prompted for authorization by twitch, False

  var str url The reachable URL that will be opened in the browser. http://localhost:17563

  var int port The port that will be used. 17653

  var str host the host the webserver will bind to. 0.0.0.0

authenticate(*callback_func=None*)

Start the user authentication flow

If callback_func is not set, authenticate will wait till the authentication process finnished and then return the access_token and the refresh_token

Parameters **callback_func** – Function to call once the authentication finnished.

Returns None if callback_func is set, otherwise access_token and refresh_token

Return type None or (str, str)

stop()

Manually stop the flow

Return type None

twitchAPI.oauth.**refresh_access_token**(*refresh_token: str, app_id: str, app_secret: str*)

Simple helper function for refreshing a user access token.

Parameters

- **refresh_token** (*str*) – the current refresh_token

- **app_id** (*str*) – the id of your app
- **app_secret** (*str*) – the secret key of your app

Returns access_token, refresh_token

Return type (str, str)

4.5 twitchAPI.types

Type Definitions

class twitchAPI.types.**AnalyticsReportType**
Enum of all Analytics report types

Variables

- **V1** –
- **V2** –

class twitchAPI.types.**AuthScope**
Enum of Authentication scopes

Variables

- **ANALYTICS_READ_EXTENSION** –
- **ANALYTICS_READ_GAMES** –
- **BITS_READ** –
- **CHANNEL_READ_SUBSCRIPTIONS** –
- **CHANNEL_READ_STREAM_KEY** –
- **CHANNEL_EDIT_COMMERCIAL** –
- **CHANNEL_READ_HYPE_TRAIN** –
- **CHANNEL_MANAGE_BROADCAST** –
- **CHANNEL_READ_REDEMPTIONS** –
- **CLIPS_EDIT** –
- **USER_EDIT** –
- **USER_EDIT_BROADCAST** –
- **USER_READ_BROADCAST** –
- **USER_READ_EMAIL** –
- **USER_EDIT_FOLLOWS** –
- **CHANNEL_MODERATE** –
- **CHAT_EDIT** –
- **CHAT_READ** –
- **WHISPERS_READ** –
- **WHISPERS_EDIT** –
- **MODERATION_READ** –

- **CHANNEL_SUBSCRIPTIONS** –

class twitchAPI.types.**AuthType**
Type of authentication required. Only internal use

Variables

- **NONE** – No auth required
- **USER** – User auth required
- **APP** – app auth required

class twitchAPI.types.**CodeStatus**
Enum Code Status, see https://dev.twitch.tv/docs/api/reference#get-code-status for more documentation

Variables

- **SUCCESSFULLY_REDEEMED** –
- **ALREADY_CLAIMED** –
- **EXPIRED** –
- **USER_NOT_ELIGIBLE** –
- **NOT_FOUND** –
- **INACTIVE** –
- **UNUSED** –
- **INCORRECT_FORMAT** –
- **INTERNAL_ERROR** –
- **UNKNOWN_VALUE** –

class twitchAPI.types.**HypeTrainContributionMethod**
Enum of valid Hype Train contribution types

Variables

- **BITS** –
- **SUBS** –
- **UNKNOWN** –

exception twitchAPI.types.**MissingScopeException**
authorization is missing scope

class twitchAPI.types.**ModerationEventType**
Enum of moderation event types

Variables

- **BAN** –
- **UNBAN** –
- **UNKNOWN** –

exception twitchAPI.types.**PubSubListenTimeoutException**
when a a PubSub listen command times out

class twitchAPI.types.**PubSubResponseError**

Variables

- **BAD_MESSAGE** – message is malformed
- **BAD_AUTH** – user auth token is invalid
- **SERVER** – server error
- **BAD_TOPIC** – topic is invalid
- **NONE** – no Error
- **UNKNOWN** – a unknown error

class twitchAPI.types.**SortMethod**
Enum of valid sort methods

Variables

- **TIME** –
- **TRENDING** –
- **VIEWS** –

class twitchAPI.types.**TimePeriod**
Enum of valid Time periods

Variables

- **ALL** –
- **DAY** –
- **WEEK** –
- **MONTH** –
- **YEAR** –

exception twitchAPI.types.**TwitchAPIException**
Base Twitch API Exception

exception twitchAPI.types.**TwitchAuthorizationException**
Exception in the Twitch Authorization

exception twitchAPI.types.**TwitchBackendException**
when the Twitch API itself is down

exception twitchAPI.types.**UnauthorizedException**
Not authorized to use this

class twitchAPI.types.**VideoType**
Enum of valid video types

Variables

- **ALL** –
- **UPLOAD** –
- **ARCHIVE** –
- **HIGHLIGHT** –
- **UNKNOWN** –

4.6 twitchAPI.helper

Helper functions

twitchAPI.helper.**build_scope**(*scopes: List[twitchAPI.types.AuthScope]*) → str
Builds a valid scope string from list

Parameters **scopes** (*list[AuthScope]*) – list of *AuthScope*

Return type str

Returns the valid auth scope string

twitchAPI.helper.**build_url**(*url: str*, *params: dict*, *remove_none=False*, *split_lists=False*) → str
Build a valid url string

Parameters

- **url** – base URL
- **params** – dictionary of URL parameter
- **remove_none** – optional bool, if set all params that have a None value get removed
- **split_lists** – optional bool, if set all params that are a list will be split over multiple url parameter with the same name

Returns URL

Return type str

twitchAPI.helper.**extract_uuid_str_from_url**(*url: str*) → Optional[str]
Extracts a UUID string from a URL

Parameters **url** (*str*) – The URL to parse

Returns UUID string extracted from given URL or None if no UUID found

Return type Union[str, None]

twitchAPI.helper.**fields_to_enum**(*data: Union[dict, list]*, *fields: List[str]*, *_enum: Type[enum.Enum]*, *default: Optional[enum.Enum]*) → Union[dict, list]
Itterates a dict or list and tries to replace every dict entry with key in fields with the correct Enum value

Parameters

- **list] data** (*Union[dict,*) – dict or list
- **fields** (*list[str]*) – list of keys to be replaced
- **_enum** – Type of Enum to be replaced
- **default** – The default value if _enum does not contain the field value

Return type dict or list

twitchAPI.helper.**get_json**(*request: aiohttp.web_request.Request*) → Union[list, dict, None]
Tries to retrieve the json object from the body

Parameters **request** – the request

Returns the object in the body or None

twitchAPI.helper.**get_uuid**()
Returns a random UUID

Return type UUID

twitchAPI.helper.**make_fields_datetime**(*data: Union[dict, list], fields: List[str]*)
Itterates over dict or list recursivly to replace string fields with datetime

Parameters

- **list] data** (*union[dict,*) – dict or list
- **fields** (*list[str]*) – list of keys to be replaced

Return type union[dict, list]

Python Module Index

t

twitchAPI.helper, 44
twitchAPI.oauth, 39
twitchAPI.pubsub, 35
twitchAPI.twitch, 9
twitchAPI.types, 41
twitchAPI.webhook, 30

Index

A

AnalyticsReportType (*class in twitchAPI.types*), 41
authenticate() (*twitchAPI.oauth.UserAuthenticator method*), 40
authenticate() (*twitchAPI.webhook.TwitchWebHook method*), 32
authenticate_app() (*twitchAPI.twitch.Twitch method*), 10
AuthScope (*class in twitchAPI.types*), 41
AuthType (*class in twitchAPI.types*), 42

B

build_scope() (*in module twitchAPI.helper*), 44
build_url() (*in module twitchAPI.helper*), 44

C

check_automod_status() (*twitchAPI.twitch.Twitch method*), 10
CodeStatus (*class in twitchAPI.types*), 42
create_clip() (*twitchAPI.twitch.Twitch method*), 11
create_entitlement_grants_upload_url() (*twitchAPI.twitch.Twitch method*), 11
create_stream_marker() (*twitchAPI.twitch.Twitch method*), 12
create_user_follows() (*twitchAPI.twitch.Twitch method*), 12

D

delete_user_follows() (*twitchAPI.twitch.Twitch method*), 12

E

extract_uuid_str_from_url() (*in module twitchAPI.helper*), 44

F

fields_to_enum() (*in module twitchAPI.helper*), 44

G

get_all_stream_tags() (*twitchAPI.twitch.Twitch method*), 13
get_app_token() (*twitchAPI.twitch.Twitch method*), 13
get_banned_events() (*twitchAPI.twitch.Twitch method*), 13
get_banned_users() (*twitchAPI.twitch.Twitch method*), 14
get_bits_leaderboard() (*twitchAPI.twitch.Twitch method*), 14
get_broadcaster_subscriptions() (*twitchAPI.twitch.Twitch method*), 15
get_channel_information() (*twitchAPI.twitch.Twitch method*), 15
get_cheermotes() (*twitchAPI.twitch.Twitch method*), 16
get_clips() (*twitchAPI.twitch.Twitch method*), 16
get_code_status() (*twitchAPI.twitch.Twitch method*), 17
get_drops_entitlements() (*twitchAPI.twitch.Twitch method*), 17
get_extension_analytics() (*twitchAPI.twitch.Twitch method*), 18
get_extension_transactions() (*twitchAPI.twitch.Twitch method*), 18
get_game_analytics() (*twitchAPI.twitch.Twitch method*), 19
get_games() (*twitchAPI.twitch.Twitch method*), 19
get_hype_train_events() (*twitchAPI.twitch.Twitch method*), 20
get_json() (*in module twitchAPI.helper*), 44
get_moderator_events() (*twitchAPI.twitch.Twitch method*), 20
get_moderators() (*twitchAPI.twitch.Twitch method*), 21
get_stream_key() (*twitchAPI.twitch.Twitch method*), 21
get_stream_markers() (*twitchAPI.twitch.Twitch*

method), 22
get_stream_tags() (*twitchAPI.twitch.Twitch method*), 22
get_streams() (*twitchAPI.twitch.Twitch method*), 22
get_top_games() (*twitchAPI.twitch.Twitch method*), 23
get_used_token() (*twitchAPI.twitch.Twitch method*), 24
get_user_active_extensions() (*twitchAPI.twitch.Twitch method*), 24
get_user_auth_scope() (*twitchAPI.twitch.Twitch method*), 24
get_user_auth_token() (*twitchAPI.twitch.Twitch method*), 24
get_user_extensions() (*twitchAPI.twitch.Twitch method*), 24
get_users() (*twitchAPI.twitch.Twitch method*), 24
get_users_follows() (*twitchAPI.twitch.Twitch method*), 25
get_uuid() (*in module twitchAPI.helper*), 44
get_videos() (*twitchAPI.twitch.Twitch method*), 25
get_webhook_subscriptions() (*twitchAPI.twitch.Twitch method*), 26

H

HypeTrainContributionMethod (*class in twitchAPI.types*), 42

L

listen_bits() (*twitchAPI.pubsub.PubSub method*), 36
listen_bits_badge_notification() (*twitchAPI.pubsub.PubSub method*), 36
listen_bits_v1() (*twitchAPI.pubsub.PubSub method*), 37
listen_channel_points() (*twitchAPI.pubsub.PubSub method*), 37
listen_channel_subscriptions() (*twitchAPI.pubsub.PubSub method*), 37
listen_chat_moderator_actions() (*twitchAPI.pubsub.PubSub method*), 38
listen_whispers() (*twitchAPI.pubsub.PubSub method*), 38

M

make_fields_datetime() (*in module twitchAPI.helper*), 45
MissingScopeException, 42
ModerationEventType (*class in twitchAPI.types*), 42
modify_channel_information() (*twitchAPI.twitch.Twitch method*), 27

P

PubSub (*class in twitchAPI.pubsub*), 36
PubSubListenTimeoutException, 42
PubSubResponseError (*class in twitchAPI.types*), 42

R

redeem_code() (*twitchAPI.twitch.Twitch method*), 27
refresh_access_token() (*in module twitchAPI.oauth*), 40
refresh_used_token() (*twitchAPI.twitch.Twitch method*), 27
renew_subscription() (*twitchAPI.webhook.TwitchWebHook method*), 32
replace_stream_tags() (*twitchAPI.twitch.Twitch method*), 28

S

search_categories() (*twitchAPI.twitch.Twitch method*), 28
search_channels() (*twitchAPI.twitch.Twitch method*), 28
set_user_authentication() (*twitchAPI.twitch.Twitch method*), 29
SortMethod (*class in twitchAPI.types*), 43
start() (*twitchAPI.pubsub.PubSub method*), 39
start() (*twitchAPI.webhook.TwitchWebHook method*), 32
start_commercial() (*twitchAPI.twitch.Twitch method*), 29
stop() (*twitchAPI.oauth.UserAuthenticator method*), 40
stop() (*twitchAPI.pubsub.PubSub method*), 39
stop() (*twitchAPI.webhook.TwitchWebHook method*), 32
subscribe_channel_ban_change_events() (*twitchAPI.webhook.TwitchWebHook method*), 33
subscribe_extension_transaction_created() (*twitchAPI.webhook.TwitchWebHook method*), 33
subscribe_hype_train_events() (*twitchAPI.webhook.TwitchWebHook method*), 33
subscribe_moderator_change_events() (*twitchAPI.webhook.TwitchWebHook method*), 33
subscribe_stream_changed() (*twitchAPI.webhook.TwitchWebHook method*), 34
subscribe_subscription_events() (*twitchAPI.webhook.TwitchWebHook method*), 34

subscribe_user_changed()
(*twitchAPI.webhook.TwitchWebHook method*), 34
subscribe_user_follow()
(*twitchAPI.webhook.TwitchWebHook method*), 34

T

TimePeriod (*class in twitchAPI.types*), 43
Twitch (*class in twitchAPI.twitch*), 10
twitchAPI.helper (*module*), 44
twitchAPI.oauth (*module*), 39
twitchAPI.pubsub (*module*), 35
twitchAPI.twitch (*module*), 9
twitchAPI.types (*module*), 41
twitchAPI.webhook (*module*), 30
TwitchAPIException, 43
TwitchAuthorizationException, 43
TwitchBackendException, 43
TwitchWebHook (*class in twitchAPI.webhook*), 32

U

UnauthorizedException, 43
unlisten() (*twitchAPI.pubsub.PubSub method*), 39
unsubscribe_all()
(*twitchAPI.webhook.TwitchWebHook method*), 35
update_user() (*twitchAPI.twitch.Twitch method*), 30
update_user_extensions()
(*twitchAPI.twitch.Twitch method*), 30
UserAuthenticator (*class in twitchAPI.oauth*), 40

V

VideoType (*class in twitchAPI.types*), 43