

---

# **twitchAPI**

*Release 2.0*

**Feb 24, 2021**



---

## Contents:

---

|          |                             |           |
|----------|-----------------------------|-----------|
| <b>1</b> | <b>Installation</b>         | <b>3</b>  |
| <b>2</b> | <b>Support</b>              | <b>5</b>  |
| <b>3</b> | <b>Usage</b>                | <b>7</b>  |
| <b>4</b> | <b>Logging</b>              | <b>9</b>  |
| <b>5</b> | <b>Indices and tables</b>   | <b>11</b> |
| 5.1      | twitchAPI.twitch . . . . .  | 11        |
| 5.2      | twitchAPI.webhook . . . . . | 42        |
| 5.3      | twitchAPI.pubsub . . . . .  | 47        |
| 5.4      | twitchAPI.oauth . . . . .   | 51        |
| 5.5      | twitchAPI.types . . . . .   | 53        |
| 5.6      | twitchAPI.helper . . . . .  | 56        |
|          | <b>Python Module Index</b>  | <b>59</b> |
|          | <b>Index</b>                | <b>61</b> |



This is a full implementation of the Twitch Helix API, its Webhook and PubSub 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

---

### Support

---

For Support please join the [Twitch API Discord server](#).



## CHAPTER 3

---

### 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']))
```



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

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



---

## Indices and tables

---

- `genindex`
- `modindex`
- `changelog`

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

## 5.1 twitchAPI.twitch

### 5.1.1 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 don't need it since the rate limits 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.

### Authentication refresh callback

Optionally you can set a callback for both user access token refresh and app access token refresh.

```
from twitchAPI.twitch import Twitch

def user_refresh(token: str, refresh_token: str):
    print(f'my new user token is: {token}')

def app_refresh(token: str):
    print(f'my new app token is: {token}')

twitch = Twitch('my_app_id', 'my_app_secret')
twitch.app_auth_refresh_callback = app_refresh
twitch.user_auth_refresh_callback = user_refresh
```

### 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
- **user\_auth\_refresh\_callback** (*Callable[[str, str], None]*) – If set, gets called whenever a user auth token gets refreshed. Parameter: Auth Token, Refresh Token  
None
- **app\_auth\_refresh\_callback** (*Callable[[str, str], None]*) – If set, gets called whenever a app auth token gets refreshed. Parameter: Auth Token  
None

**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

**block\_user** (*target\_user\_id: str, source\_context: Optional[twitchAPI.types.BlockSourceContext] = None, reason: Optional[twitchAPI.types.BlockReason] = None*) → dict

Blocks the specified user on behalf of the authenticated user.

Requires User Authentication with `twitchAPI.types.AuthScope.USER_MANAGE_BLOCKED_USERS`

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

#### Parameters

- **target\_user\_id** (*str*) – User ID of the user to be blocked.
- **source\_context** (*BlockSourceContext*) – Source context for blocking the user.  
Optional None
- **reason** (*BlockReason*) – Reason for blocking the user. Optional. None

#### Raises

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid

**Return type** dict

**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**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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\_custom\_reward** (*broadcaster\_id: str, title: str, prompt: str, cost: int, is\_enabled: Optional[bool] = True, background\_color: Optional[str] = None, is\_user\_input\_required: Optional[bool] = False, is\_max\_per\_stream\_enabled: Optional[bool] = False, max\_per\_stream: Optional[int] = None, is\_max\_per\_user\_per\_stream\_enabled: Optional[bool] = False, max\_per\_user\_per\_stream: Optional[int] = None, is\_global\_cooldown\_enabled: Optional[bool] = False, global\_cooldown\_seconds: Optional[int] = None, should\_redemptions\_skip\_request\_queue: Optional[bool] = False*) → dict

Creates a Custom Reward on a channel.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_MANAGE_REDEMPTIONS`

For detailed documentation, see here: <https://dev.twitch.tv/docs/api/reference#create-custom-rewards>

**Parameters**

- **broadcaster\_id** (*str*) – ID of the broadcaster, must be same as user\_id of auth token
- **title** (*str*) – The title of the reward
- **prompt** (*str*) – The prompt for the viewer when they are redeeming the reward
- **cost** (*int*) – The cost of the reward
- **is\_enabled** – Is the reward currently enabled, if false the reward won't show up to viewers. `true`
- **background\_color** (*str*) – Custom background color for the reward. Format: Hex with # prefix. Example: #00E5CB. `None`
- **is\_user\_input\_required** (*bool*) – Does the user need to enter information when redeeming the reward. `false`
- **is\_max\_per\_stream\_enabled** (*bool*) – Whether a maximum per stream is enabled. `false`
- **max\_per\_stream** (*int*) – The maximum number per stream if enabled `None`
- **is\_max\_per\_user\_per\_stream\_enabled** (*bool*) – Whether a maximum per user per stream is enabled. `false`
- **max\_per\_user\_per\_stream** (*int*) – The maximum number per user per stream if enabled `None`
- **is\_global\_cooldown\_enabled** (*bool*) – Whether a cooldown is enabled. `false`
- **global\_cooldown\_seconds** (*int*) – The cooldown in seconds if enabled `None`
- **should\_redemptions\_skip\_request\_queue** (*bool*) – Should redemptions be set to FULFILLED status immediately when redeemed and skip the request queue instead of the normal UNFULFILLED status. `false`

**Raises**

- ***TwitchAPIException*** – if the request was malformed
- ***ValueError*** – if `is_global_cooldown_enabled` is `True` but `global_cooldown_seconds` is not specified
- ***ValueError*** – if `is_max_per_stream_enabled` is `True` but `max_per_stream` is not specified
- ***ValueError*** – if `is_max_per_user_per_stream_enabled` is `True` but `max_per_user_per_stream` is not specified
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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
- ***TwitchAPIException*** – if a Query Parameter is missing or invalid
- ***TwitchAPIException*** – if Channel Points are not available for the broadcaster

**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 or invalid
- **TwitchAuthorizationException** – if the the authentication token became invalid and a re authentication failed
- **TwitchAPIException** – if the request was malformed
- **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. None

**Raises**

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if user authentication is not set or invalid
- **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**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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\_custom\_reward** (*broadcaster\_id: str, reward\_id: str*)

Deletes a Custom Reward on a channel.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_MANAGE_REDEMPTIONS`

For detailed documentation, see here: <https://dev.twitch.tv/docs/api/reference#delete-custom-rewards>

**Parameters**

- **broadcaster\_id** (*str*) – Provided `broadcaster_id` must match the `user_id` in the auth token
- **reward\_id** (*str*) – ID of the Custom Reward to delete, must match a Custom Reward on `broadcaster_id`’s channel.

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid
- *NotFoundException* – if the broadcaster has no custom reward with the given id

**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**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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\_videos** (*video\_ids: List[str]*) → Union[bool, dict]

Deletes one or more videos. Videos are past broadcasts, Highlights, or uploads. Returns False if the User was not Authorized to delete at least one of the given videos.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_MANAGE_VIDEOS`

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

**Parameters** **video\_ids** (*list (str)*) – ids of the videos, Limit: 5 ids

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid
- *ValueError* – if `video_ids` contains more than 5 entries or is a empty list

**Return type** dict or False**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 None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20
- **tag\_ids** (*list [str]*) – IDs of tags. Maximum 100 entries None

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if app authentication is not set or invalid
- *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 None
- **after** (str) – Cursor for forward pagination None
- **first** (int) – Maximum number of objects to return. Maximum: 100. 20

#### Raises

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if user authentication is not set or invalid
- **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\_banned\_users** (broadcaster\_id: str, user\_id: Optional[str] = None, after: Optional[str] = None, first: Optional[int] = 20, 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. None
- **after** (str) – Cursor for forward pagination None
- **before** (str) – Cursor for backward pagination None
- **first** (int) – Maximum number of objects to return. Maximum: 100. 20

**Raises**

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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\_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. None
- **user\_id** (*str*) – ID of the user whose results are returned; i.e., the person who paid for the Bits. None

**Raises**

- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***MissingScopeException*** – if the user authentication is missing the required scope
- ***TwitchAuthorizationException*** – if the used authentication token became invalid and a re authentication failed
- ***TwitchAPIException*** – if the request was malformed
- ***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, *after*: Optional[str] = None, *first*: Optional[int] = 20) → 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 None
- **after** (*str*) – Cursor for forward pagination. None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

**Raises**

- **UnauthorizedException** – if user authentication is not set or invalid
- **MissingScopeException** – if the user authentication is missing the required scope
- **TwitchAuthorizationException** – if the used authentication token became invalid and a re authentication failed
- **TwitchAPIException** – if the request was malformed
- **TwitchBackendException** – if the Twitch API itself runs into problems
- **ValueError** – if user\_ids has more than 100 entries
- **ValueError** – if first is not in range 1 to 100

**Return type** dict

**get\_channel\_editors** (*broadcaster\_id: str*) → dict

Gets a list of users who have editor permissions for a specific channel.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_READ_EDITORS`

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

**Parameters** **broadcaster\_id** (*str*) – Broadcaster’s user ID associated with the channel

**Raises**

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if user authentication is not set or invalid
- **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
- **TwitchAPIException** – if a Query Parameter is missing or invalid

**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**

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if app authentication is not set or invalid

- *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**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if app authentication is not set or invalid
- *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. None
- **game\_id** (*str*) – ID of the game for which clips are returned. None
- **clip\_id** (*list[str]*) – ID of the clip being queried. Limit: 100. None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20
- **after** (*str*) – Cursor for forward pagination None
- **before** (*str*) – Cursor for backward pagination None
- **ended\_at** (*datetime*) – Ending date/time for returned clips None
- **started\_at** (*datetime*) – Starting date/time for returned clips None

**Raises**

- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- **TwitchAPIException** – if the request was malformed
- **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 or invalid
- **TwitchAuthorizationException** – if the used authentication token became invalid and a re authentication failed
- **TwitchAPIException** – if the request was malformed
- **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\_custom\_reward** (*broadcaster\_id: str, reward\_id: Optional[List[str]] = None, only\_manageable\_rewards: Optional[bool] = False*) → dict

Returns a list of Custom Reward objects for the Custom Rewards on a channel. Developers only have access to update and delete rewards that the same/calling client\_id created.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_READ_REDEMPTIONS`

For detailed documentation, see here: <https://dev.twitch.tv/docs/api/reference#get-custom-reward>

#### Parameters

- **broadcaster\_id** (*str*) – Provided broadcaster\_id must match the user\_id in the auth token
- **reward\_id** (*list[str]*) – When used, this parameter filters the results and only returns reward objects for the Custom Rewards with matching ID. Maximum: 50 None
- **only\_manageable\_rewards** (*bool*) – When set to true, only returns custom rewards that the calling client\_id can manage. false

**Return type** dict

#### Raises

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if app authentication is not set or invalid

- *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 if reward\_id is longer than 50 entries

```
get_custom_reward_redemption (broadcaster_id: str, reward_id: str, id:
                               Optional[List[str]] = None, status: Op-
                               tional[twitchAPI.types.CustomRewardRedemptionStatus]
                               = None, sort: Optional[twitchAPI.types.SortOrder] = <Sor-
                               tOrder.OLDEST: 'OLDEST'>, after: Optional[str] = None,
                               first: Optional[int] = 20) → dict
```

Returns Custom Reward Redemption objects for a Custom Reward on a channel that was created by the same client\_id.

Requires User Authentication with twitchAPI.types.AuthScope.CHANNEL\_READ\_REDEMPTIONS

For detailed documentation, see here: [https://dev.twitch.tv/docs/api/reference#  
get-custom-reward-redemption](https://dev.twitch.tv/docs/api/reference#get-custom-reward-redemption)

#### Parameters

- **broadcaster\_id** (*str*) – Provided broadcaster\_id must match the user\_id in the auth token
- **reward\_id** (*str*) – When ID is not provided, this parameter returns paginated Custom Reward Redemption objects for redemptions of the Custom Reward with ID reward\_id
- **id** (*list (str)*) – When used, this param filters the results and only returns None Custom Reward Redemption objects for the redemptions with matching ID. Maximum: 50 ids None
- **status** (*CustomRewardRedemptionStatus*) – When id is not provided, this param is required and filters the paginated Custom Reward Redemption objects for redemptions with the matching status. None
- **sort** (*SortOrder*) – Sort order of redemptions returned when getting the paginated Custom Reward Redemption objects for a reward. *SortOrder.OLDEST*
- **after** (*str*) – Cursor for forward pagination. None
- **first** (*int*) – Number of results to be returned when getting the paginated Custom Reward Redemption objects for a reward. Limit: 50 20

**Return type** dict

#### Raises

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if app authentication is not set or invalid
- *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 id has more than 50 entries
- **ValueError** – if first is not in range 1 to 50

**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 None
- **user\_id** (*str*) – A Twitch User ID None
- **game\_id** (*str*) – A Twitch Game ID None
- **after** (*str*) – The cursor used to fetch the next page of data. None
- **first** (*int*) – Maximum number of entitlements to return. Maximum: 100 20

#### Raises

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if app authentication is not set or invalid
- **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 None
- **extension\_id** (*str*) – If this is specified, the returned URL points to an analytics report for just the specified extension. None
- **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. None

- **started\_at** (*datetime*) – Starting date/time for returned reports, if this is provided, *ended\_at* must also be specified. *None*
- **report\_type** (*AnalyticsReportType*) – Type of analytics report that is returned *None*

**Return type** *dict*

**Raises**

- **UnauthorizedException** – if user authentication is not set or invalid
- **MissingScopeException** – if the user authentication is missing the required scope
- **TwitchAuthorizationException** – if the used authentication token became invalid
- **TwitchAPIException** – if the request was malformed 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. *None*
- **after** (*str*) – cursor for forward pagination *None*
- **first** (*int*) – Maximum number of objects returned, range 1 to 100, 20

**Raises**

- **UnauthorizedException** – if app authentication is not set or invalid
- **TwitchAuthorizationException** – if the used authentication token became invalid and a re authentication failed
- **TwitchAPIException** – if the request was malformed
- **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 `None`
- **first** (*int*) – Maximum number of objects returned, range 1 to 100, 20
- **game\_id** (*str*) – Game ID `None`
- **ended\_at** (*datetime*) – Ending date/time for returned reports, if this is provided, *started\_at* must also be specified. `None`
- **started\_at** (*datetime*) – Starting date/time for returned reports, if this is provided, *ended\_at* must also be specified. `None`
- **report\_type** (`AnalyticsReportType`) – Type of analytics report that is returned. `None`

#### Raises

- **`UnauthorizedException`** – if user authentication is not set or invalid
- **`MissingScopeException`** – if the user authentication is missing the required scope
- **`TwitchAuthorizationException`** – if the used authentication token became invalid and a re authentication failed
- **`TwitchAPIException`** – if the request was malformed
- **`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 `None`
- **names** (*list [str]*) – Game Name `None`

#### Raises

- **`UnauthorizedException`** – if app authentication is not set or invalid
- **`TwitchAPIException`** – if the request was malformed
- **`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 None
- **cursor** (*str*) – Cursor for forward pagination None

#### Raises

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if app authentication is not set or invalid
- ***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, after: Optional[str] = None, first: Optional[int] = 20*) → 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 None
- **after** (*str*) – Cursor for forward pagination None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

#### Raises

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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
- ***ValueError*** – if first is not in range 1 to 100

**Return type** dict

**get\_moderators** (*broadcaster\_id: str, user\_ids: Optional[List[str]] = None, first: Optional[int] = 20, 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 None
- **after** (*str*) – Cursor for forward pagination None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

#### Raises

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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
- ***ValueError*** – if first is not in range 1 to 100

**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

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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 None
- **before** (*str*) – Cursor for backward pagination None
- **first** (*int*) – Number of values to be returned when getting videos by user or game ID. Limit: 100. 20

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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 that's tags are going to be fetched

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if app authentication is not set or invalid
- *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 None
- **before** (*str*) – Cursor for backward pagination None
- **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. None
- **language** (*list[str]*) – Stream language. You can specify up to 100 languages. None
- **user\_id** (*list[str]*) – Returns streams broadcast by one or more specified user IDs. You can specify up to 100 IDs. None
- **user\_login** (*list[str]*) – Returns streams broadcast by one or more specified user login names. You can specify up to 100 names. None

**Raises**

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if app authentication is not set or invalid
- **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 None
- **before** (*str*) – Cursor for backward pagination None
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

**Raises**

- **TwitchAPIException** – if the request was malformed

- *UnauthorizedException* – if app authentication is not set or invalid
- *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.  
None

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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\_block\_list** (*broadcaster\_id: str, first: Optional[int] = 20, after: Optional[str] = None*) → dict

Gets a specified user's block list. The list is sorted by when the block occurred in descending order (i.e. most recent block first).

Requires User Authentication with `twitchAPI.types.AuthScope.USER_READ_BLOCKED_USERS`

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

**Parameters**

- **broadcaster\_id** (*str*) – User ID for a twitch user
- **first** (*int*) – Maximum number of objects to return. Maximum: 100. 20

- **after** (*str*) – Cursor for forward pagination None

#### Raises

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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
- ***TwitchAPIException*** – if a Query Parameter is missing or invalid
- ***ValueError*** – if first is not in range 1 to 100

**Return type** dict

**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

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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. None
- **logins** (*list[str]*) – User login name. Multiple login names can be specified. Limit: 100. None

#### Raises

- ***TwitchAPIException*** – if the request was malformed

- *UnauthorizedException* – if user authentication is not set or invalid
- *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 None
- **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. None
- **to\_id** (*str*) – User ID. The request returns information about users who are following the to\_id user. None

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if app authentication is not set or invalid
- *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: Optional[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. None
- **user\_id** (*str*) – ID of the user who owns the video. None
- **game\_id** (*str*) – ID of the game the video is of. None

- **after** (*str*) – Cursor for forward pagination `None`
- **before** (*str*) – Cursor for backward pagination `None`
- **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. `None`
- **period** (`TimePeriod`) – Period during which the video was created. `TimePeriod.ALL`
- **sort** (`SortMethod`) – Sort order of the videos. `SortMethod.TIME`
- **video\_type** (`VideoType`) – Type of video. `VideoType.ALL`

**Raises**

- **`TwitchAPIException`** – if the request was malformed
- **`UnauthorizedException`** – if app authentication is not set or invalid
- **`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 `None`

**Raises**

- **`TwitchAPIException`** – if the request was malformed
- **`UnauthorizedException`** – if app authentication is not set or invalid
- **`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 *None*
- **broadcaster\_language** (*str*) – The language of the channel *None*
- **title** (*str*) – The title of the stream *None*

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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 fields are specified: *game\_id*, *broadcaster\_language*, *title*
- *ValueError* – if title is a empty string

**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 or invalid
- *TwitchAPIException* – if the request was malformed
- *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

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if user authentication is not set or invalid
- ***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 None

#### Raises

- ***TwitchAPIException*** – if the request was malformed
- ***UnauthorizedException*** – if app authentication is not set or invalid
- ***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 None
- **live\_only** (*bool*) – Filter results for live streams only. False

**Raises**

- **TwitchAPIException** – if the request was malformed
- **UnauthorizedException** – if app authentication is not set or invalid
- **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*)

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 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**

- **TwitchAPIException** – if the request was malformed

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

**unblock\_user** (*target\_user\_id: str*) → bool

Unblocks the specified user on behalf of the authenticated user.

Requires User Authentication with twitchAPI.types.AuthScope.USER\_MANAGE\_BLOCKED\_USERS

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

**Parameters** **target\_user\_id** (*str*) – User ID of the user to be unblocked.

**Raises**

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid

**Return type** bool

**update\_custom\_reward** (*broadcaster\_id: str, reward\_id: str, title: str, prompt: str, cost: int, is\_enabled: Optional[bool] = True, background\_color: Optional[str] = None, is\_user\_input\_required: Optional[bool] = False, is\_max\_per\_stream\_enabled: Optional[bool] = False, max\_per\_stream: Optional[int] = None, is\_max\_per\_user\_per\_stream\_enabled: Optional[bool] = False, max\_per\_user\_per\_stream: Optional[int] = None, is\_global\_cooldown\_enabled: Optional[bool] = False, global\_cooldown\_seconds: Optional[int] = None, should\_redemptions\_skip\_request\_queue: Optional[bool] = False*) → dict

Updates a Custom Reward created on a channel.

Requires User Authentication with twitchAPI.types.AuthScope.CHANNEL\_MANAGE\_REDEMPTIONS

For detailed documentation, see here: <https://dev.twitch.tv/docs/api/reference#update-custom-rewards>

**Parameters**

- **broadcaster\_id** (*str*) – ID of the broadcaster, must be same as user\_id of auth token
- **reward\_id** (*str*) – ID of the reward that you want to update
- **title** (*str*) – The title of the reward
- **prompt** (*str*) – The prompt for the viewer when they are redeeming the reward

- **cost** (*int*) – The cost of the reward
- **is\_enabled** – Is the reward currently enabled, if false the reward won't show up to viewers. *true*
- **background\_color** (*str*) – Custom background color for the reward. *None* Format: Hex with # prefix. Example: #00E5CB.
- **is\_user\_input\_required** (*bool*) – Does the user need to enter information when redeeming the reward. *false*
- **is\_max\_per\_stream\_enabled** (*bool*) – Whether a maximum per stream is enabled. *false*
- **max\_per\_stream** (*int*) – The maximum number per stream if enabled *None*
- **is\_max\_per\_user\_per\_stream\_enabled** (*bool*) – Whether a maximum per user per stream is enabled. *false*
- **max\_per\_user\_per\_stream** (*int*) – The maximum number per user per stream if enabled *None*
- **is\_global\_cooldown\_enabled** (*bool*) – Whether a cooldown is enabled. *false*
- **global\_cooldown\_seconds** (*int*) – The cooldown in seconds if enabled *None*
- **should\_redemptions\_skip\_request\_queue** (*bool*) – Should redemptions be set to FULFILLED status immediately when redeemed and skip the request queue instead of the normal UNFULFILLED status. *false*

**Raises**

- *TwitchAPIException* – if the request was malformed
- *ValueError* – if *is\_global\_cooldown\_enabled* is *True* but *global\_cooldown\_seconds* is not specified
- *ValueError* – if *is\_max\_per\_stream\_enabled* is *True* but *max\_per\_stream* is not specified
- *ValueError* – if *is\_max\_per\_user\_per\_stream\_enabled* is *True* but *max\_per\_user\_per\_stream* is not specified
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid
- *TwitchAPIException* – if Channel Points are not available for the broadcaster or the custom reward belongs to a different broadcaster
- *ValueError* – if the given *reward\_id* does not match a custom reward by the given broadcaster

**Return type** *dict*

**update\_redemption\_status** (*broadcaster\_id: str, reward\_id: str, redemption\_ids: List[str], status: twitchAPI.types.CustomRewardRedemptionStatus*) → *dict*

Updates the status of Custom Reward Redemption objects on a channel that are in the UNFULFILLED status.

Requires User Authentication with `twitchAPI.types.AuthScope.CHANNEL_MANAGE_REDEMPTIONS`

For detailed documentation, see here: <https://dev.twitch.tv/docs/api/reference#update-redemption-status>

### Parameters

- **broadcaster\_id** (*str*) – Provided broadcaster\_id must match the user\_id in the auth token.
- **reward\_id** (*str*) – ID of the Custom Reward the redemptions to be updated are for.
- **redemption\_ids** (*list (str)*) – IDs of the Custom Reward Redemption to update, must match a Custom Reward Redemption on broadcaster\_id's channel Max: 50
- **status** (*CustomRewardRedemptionStatus*) – The new status to set redemptions to.

### Raises

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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
- *TwitchAPIException* – if a Query Parameter is missing or invalid
- *TwitchAPIException* – if Channel Points are not available for the broadcaster or the custom reward belongs to a different broadcaster
- *ValueError* – if redemption\_ids is longer than 50 entries
- *ValueError* – if no custom reward redemptions with status UNFULFILLED were found for the given ids
- *TwitchAPIException* – if Channel Points are not available for the broadcaster or the custom reward belongs to a different broadcaster

### 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

- *TwitchAPIException* – if the request was malformed
- *UnauthorizedException* – if user authentication is not set or invalid
- *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**

- `TwitchAPIException` – if the request was malformed
- `UnauthorizedException` – if user authentication is not set or invalid
- `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`

## 5.2 twitchAPI.webhook

### 5.2.1 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. You can also hand in a valid ssl context to be used in the constructor.

---

You can check on whether or not your webhook is publicly reachable by navigating to the URL set in `callback_url`. You should get a 200 response with the text `pyTwitchAPI webhook`.

#### 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))
```

(continues on next page)

(continued from previous page)

```

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 default, 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 were left over from a previous run.

## Class Documentation:

```

class twitchAPI.webhook.TwitchWebHook(callback_url: str, api_client_id: str, port: int,
                                     ssl_context: Optional[ssl.SSLContext] = None)

```

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
- **ssl\_context** (*SSLContext*) – optional ssl context to be used `None`

### 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 webhooks. 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** (*Twitich*) – a authenticated instance of *Twitich*

**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

**Raises** **ValueError** – if both *from\_id* and *to\_id* are None

**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.

## 5.3 twitchAPI.pubsub

### 5.3.1 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 twitchAPI.types import AuthScope
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)

# setting up Authentication and getting your user id
twitch = Twitch('my_app_id', 'my_app_secret')
twitch.authenticate_app([])
# you can get your user auth token and user auth refresh token following the example_
↳in twitchAPI.oauth
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']

# starting up PubSub
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_READ_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*

## 5.4 twitchAPI.oauth

### 5.4.1 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 environments 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)
```

(continues on next page)

(continued from previous page)

```
# 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 finished and then return the `access_token` and the `refresh_token`

**Parameters** `callback_func` – Function to call once the authentication finished.

**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`

**Raises**

- *InvalidRefreshTokenException* – if refresh token is invalid



- *UnauthorizedException* – if both refresh and access token are invalid (eg if the user changes their password of the app gets disconnected)

**Return type** (str, str)

## 5.5 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** –
- **CHANNEL\_MANAGE\_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** –
- **CHANNEL\_READ\_EDITORS** –
- **CHANNEL\_MANAGE\_VIDEOS** –
- **USER\_READ\_BLOCKED\_USERS** –
- **USER\_MANAGE\_BLOCKED\_USERS** –

**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.BlockReason

**Variables**

- **SPAM** –
- **HARASSMENT** –
- **OTHER** –

**class** twitchAPI.types.BlockSourceContext

**Variables**

- **CHAT** –
- **WHISPER** –

**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.CustomRewardRedemptionStatus

**Variables**

- **UNFULFILLED** –
- **FULFILLED** –

- **CANCELED** –

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

#### Variables

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

**exception** twitchAPI.types.InvalidRefreshTokenException  
used User Refresh Token is invalid

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

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

#### Variables

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

**exception** twitchAPI.types.NotFoundException  
Resource was not found with the given parameter

**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.SortOrder

#### Variables

- **OLDEST** –

- **NEWEST** –

**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** –

## 5.6 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, enum\_value=True*) → str

Build a valid url string

**Parameters**

- **url** – base URL

- **params** – dictionary of URL parameter
- **remove\_none** (*bool*) – if set all params that have a None value get removed `False`
- **split\_lists** (*bool*) – if set all params that are a list will be split over multiple url parameter with the same name `False`
- **enum\_value** (*bool*) – if true, automatically get value string from Enum values `True`

**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]`

Iterates 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])`  
 Iterates over dict or list recursively 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]`



**t**

twitchAPI.helper, 56  
twitchAPI.oauth, 51  
twitchAPI.pubsub, 47  
twitchAPI.twitch, 11  
twitchAPI.types, 53  
twitchAPI.webhook, 42





**A**

AnalyticsReportType (class in twitchAPI.types), 53  
 authenticate() (twitchAPI.oauth.UserAuthenticator method), 52  
 authenticate() (twitchAPI.webhook.TwitchWebHook method), 44  
 authenticate\_app() (twitchAPI.twitch.Twitch method), 13  
 AuthScope (class in twitchAPI.types), 53  
 AuthType (class in twitchAPI.types), 54

**B**

block\_user() (twitchAPI.twitch.Twitch method), 13  
 BlockReason (class in twitchAPI.types), 54  
 BlockSourceContext (class in twitchAPI.types), 54  
 build\_scope() (in module twitchAPI.helper), 56  
 build\_url() (in module twitchAPI.helper), 56

**C**

check\_automod\_status() (twitchAPI.twitch.Twitch method), 13  
 CodeStatus (class in twitchAPI.types), 54  
 create\_clip() (twitchAPI.twitch.Twitch method), 14  
 create\_custom\_reward() (twitchAPI.twitch.Twitch method), 14  
 create\_entitlement\_grants\_upload\_url() (twitchAPI.twitch.Twitch method), 15  
 create\_stream\_marker() (twitchAPI.twitch.Twitch method), 16  
 create\_user\_follows() (twitchAPI.twitch.Twitch method), 16  
 CustomRewardRedemptionStatus (class in twitchAPI.types), 54

**D**

delete\_custom\_reward() (twitchAPI.twitch.Twitch method), 17

delete\_user\_follows() (twitchAPI.twitch.Twitch method), 17  
 delete\_videos() (twitchAPI.twitch.Twitch method), 18

**E**

extract\_uuid\_str\_from\_url() (in module twitchAPI.helper), 57

**F**

fields\_to\_enum() (in module twitchAPI.helper), 57

**G**

get\_all\_stream\_tags() (twitchAPI.twitch.Twitch method), 18  
 get\_app\_token() (twitchAPI.twitch.Twitch method), 19  
 get\_banned\_events() (twitchAPI.twitch.Twitch method), 19  
 get\_banned\_users() (twitchAPI.twitch.Twitch method), 19  
 get\_bits\_leaderboard() (twitchAPI.twitch.Twitch method), 20  
 get\_broadcaster\_subscriptions() (twitchAPI.twitch.Twitch method), 20  
 get\_channel\_editors() (twitchAPI.twitch.Twitch method), 21  
 get\_channel\_information() (twitchAPI.twitch.Twitch method), 21  
 get\_cheermotes() (twitchAPI.twitch.Twitch method), 22  
 get\_clips() (twitchAPI.twitch.Twitch method), 22  
 get\_code\_status() (twitchAPI.twitch.Twitch method), 23  
 get\_custom\_reward() (twitchAPI.twitch.Twitch method), 23  
 get\_custom\_reward\_redemption() (twitchAPI.twitch.Twitch method), 24  
 get\_drops\_entitlements() (twitchAPI.twitch.Twitch method), 25

get\_extension\_analytics() (twitchAPI.twitch.Twitch method), 25  
 get\_extension\_transactions() (twitchAPI.twitch.Twitch method), 26  
 get\_game\_analytics() (twitchAPI.twitch.Twitch method), 26  
 get\_games() (twitchAPI.twitch.Twitch method), 27  
 get\_hype\_train\_events() (twitchAPI.twitch.Twitch method), 27  
 get\_json() (in module twitchAPI.helper), 57  
 get\_moderator\_events() (twitchAPI.twitch.Twitch method), 28  
 get\_moderators() (twitchAPI.twitch.Twitch method), 29  
 get\_stream\_key() (twitchAPI.twitch.Twitch method), 29  
 get\_stream\_markers() (twitchAPI.twitch.Twitch method), 30  
 get\_stream\_tags() (twitchAPI.twitch.Twitch method), 30  
 get\_streams() (twitchAPI.twitch.Twitch method), 31  
 get\_top\_games() (twitchAPI.twitch.Twitch method), 31  
 get\_used\_token() (twitchAPI.twitch.Twitch method), 32  
 get\_user\_active\_extensions() (twitchAPI.twitch.Twitch method), 32  
 get\_user\_auth\_scope() (twitchAPI.twitch.Twitch method), 32  
 get\_user\_auth\_token() (twitchAPI.twitch.Twitch method), 32  
 get\_user\_block\_list() (twitchAPI.twitch.Twitch method), 32  
 get\_user\_extensions() (twitchAPI.twitch.Twitch method), 33  
 get\_users() (twitchAPI.twitch.Twitch method), 33  
 get\_users\_follows() (twitchAPI.twitch.Twitch method), 34  
 get\_uuid() (in module twitchAPI.helper), 57  
 get\_videos() (twitchAPI.twitch.Twitch method), 34  
 get\_webhook\_subscriptions() (twitchAPI.twitch.Twitch method), 35

## H

HypeTrainContributionMethod (class in twitchAPI.types), 55

## I

InvalidRefreshTokenException, 55

## L

listen\_bits() (twitchAPI.pubsub.PubSub method), 48

listen\_bits\_badge\_notification() (twitchAPI.pubsub.PubSub method), 48  
 listen\_bits\_v1() (twitchAPI.pubsub.PubSub method), 48  
 listen\_channel\_points() (twitchAPI.pubsub.PubSub method), 49  
 listen\_channel\_subscriptions() (twitchAPI.pubsub.PubSub method), 49  
 listen\_chat\_moderator\_actions() (twitchAPI.pubsub.PubSub method), 50  
 listen\_whispers() (twitchAPI.pubsub.PubSub method), 50

## M

make\_fields\_datetime() (in module twitchAPI.helper), 57  
 MissingScopeException, 55  
 ModerationEventType (class in twitchAPI.types), 55  
 modify\_channel\_information() (twitchAPI.twitch.Twitch method), 35

## N

NotFoundException, 55

## P

PubSub (class in twitchAPI.pubsub), 47  
 PubSubListenTimeoutException, 55  
 PubSubResponseError (class in twitchAPI.types), 55

## R

redeem\_code() (twitchAPI.twitch.Twitch method), 36  
 refresh\_access\_token() (in module twitchAPI.oauth), 52  
 refresh\_used\_token() (twitchAPI.twitch.Twitch method), 36  
 renew\_subscription() (twitchAPI.webhook.TwitchWebHook method), 44  
 replace\_stream\_tags() (twitchAPI.twitch.Twitch method), 36

## S

search\_categories() (twitchAPI.twitch.Twitch method), 37  
 search\_channels() (twitchAPI.twitch.Twitch method), 37  
 set\_user\_authentication() (twitchAPI.twitch.Twitch method), 38  
 SortMethod (class in twitchAPI.types), 55  
 SortOrder (class in twitchAPI.types), 55  
 start() (twitchAPI.pubsub.PubSub method), 51

start () (*twitchAPI.webhook.TwitchWebHook method*), 44  
 start\_commercial () (*twitchAPI.twitch.Twitch method*), 38  
 stop () (*twitchAPI.oauth.UserAuthenticator method*), 52  
 stop () (*twitchAPI.pubsub.PubSub method*), 51  
 stop () (*twitchAPI.webhook.TwitchWebHook method*), 44  
 subscribe\_channel\_ban\_change\_events () (*twitchAPI.webhook.TwitchWebHook method*), 44  
 subscribe\_extension\_transaction\_created () (*twitchAPI.webhook.TwitchWebHook method*), 45  
 subscribe\_hype\_train\_events () (*twitchAPI.webhook.TwitchWebHook method*), 45  
 subscribe\_moderator\_change\_events () (*twitchAPI.webhook.TwitchWebHook method*), 45  
 subscribe\_stream\_changed () (*twitchAPI.webhook.TwitchWebHook method*), 45  
 subscribe\_subscription\_events () (*twitchAPI.webhook.TwitchWebHook method*), 45  
 subscribe\_user\_changed () (*twitchAPI.webhook.TwitchWebHook method*), 46  
 subscribe\_user\_follow () (*twitchAPI.webhook.TwitchWebHook method*), 46  
 unsubscribe\_all () (*twitchAPI.webhook.TwitchWebHook method*), 46  
 update\_custom\_reward () (*twitchAPI.twitch.Twitch method*), 39  
 update\_redemption\_status () (*twitchAPI.twitch.Twitch method*), 40  
 update\_user () (*twitchAPI.twitch.Twitch method*), 41  
 update\_user\_extensions () (*twitchAPI.twitch.Twitch method*), 42  
 UserAuthenticator (*class in twitchAPI.oauth*), 52  
 VideoType (*class in twitchAPI.types*), 56

## T

TimePeriod (*class in twitchAPI.types*), 56  
 Twitch (*class in twitchAPI.twitch*), 12  
 twitchAPI.helper (*module*), 56  
 twitchAPI.oauth (*module*), 51  
 twitchAPI.pubsub (*module*), 47  
 twitchAPI.twitch (*module*), 11  
 twitchAPI.types (*module*), 53  
 twitchAPI.webhook (*module*), 42  
 TwitchAPIException, 56  
 TwitchAuthorizationException, 56  
 TwitchBackendException, 56  
 TwitchWebHook (*class in twitchAPI.webhook*), 43

## U

UnauthorizedException, 56  
 unblock\_user () (*twitchAPI.twitch.Twitch method*), 39  
 unlisten () (*twitchAPI.pubsub.PubSub method*), 51