LEARNING twitch

#twitch

Table of Contents

About

Chapter 1: Getting started with twitch

Versions

Version	Release Date
1.0.0	2016-04-14

Examples

Requesting a token

The Implicit Grant flow is best suited for Web applications. It's easily integrated into a website using JavaScript and doesn't require a server to store the authorization code to retrieve a token.

You'll first send the user to the Twitch authorization endpoint. This URL is made up of a the base authorization URL (`https://api.twitch.tv/kraken/oauth2/authorize`) and query string parameters that define what you're requesting. The required parameters are `response_type`, `client_id`, `redirect_uri`, and `scope`.

For the Implicit Grant flow, the `response_type` parameter is always set to `token`. This signifies that you're requesting an OAuth token directly.

The `redirect_uri` is where the user will be redirected after they approve the scopes your application requested. This must match what you registered on your Twitch account Connections page.

The `client_id` is a unique identifier for your application. You can find your client ID on the Connections page, too.

The `scope` parameter defines what you have access to on behalf of the user. You should only request the minimum that you need for your application to function. You can find the list of scopes on the Twitch API GitHub.

The `state` parameter is also supported to help protect against cross-site scripting attacks. When the user is redirected after authorization, this value will be included on the `redirect_uri`.

Redirect the user to this URL:

```
https://api.twitch.tv/kraken/oauth2/authorize
    ?response_type=token
    &client_id=[your client ID]
    &redirect_uri=[your registered redirect URI]
    &scope=[space separated list of scopes]
    &state=[your provided unique token]
```

Get the OAuth token from the URL fragment

If the user authorizes your application, they will be redirected to the following URL:

```
https://[your registered redirect URI]/#access_token=[an access token]
        &scope=[authorized scopes]
```

Note that the access token is in the URL fragment and not the query string. This means the value will not show up in HTTP requests to your server. URL fragments can be accessed from JavaScript with `document.location.hash`.

Read Getting started with twitch online: https://riptutorial.com/twitch/topic/464/getting-started-with-twitch

Chapter 2: Calling Twitch APIs

Remarks

This topic is meant to show a general way to call the Twitch API without OAuth. You can call any APIs found in the Twitch REST API documentation using this pattern. You would simply change the URL to the correct endpoint.

A Client-ID is required for all calls to the Twitch API. In these examples, the Client-ID is added as a header to each call. You can also add it with the `client_id` query string parameter. If you use an OAuth token, the Twitch API will automatically resolve the Client-ID for you.

You can register a developer application at the new client page on Twitch.

Examples

PHP

The following will retrieve a `channel` object for the `twitch` channel and echo the response.

```
 $channelsApi = 'https://api.twitch.tv/kraken/channels/';
  $channelName = 'twitch';
  $clientId = '...';
  $ch = curl_init();

  curl_setopt_array($ch, array(
     CURLOPT_HTTPHEADER=> array(
     'Client-ID: ' . $clientId
     ),
     CURLOPT_RETURNTRANSFER=> true,
     CURLOPT_URL => $channelsApi . $channelName
  ));

  $response = curl_exec($ch);
  curl_close($ch);
  echo $response;
```

JavaScript

The following will log the JSON response from the API to the console if the request was successful, otherwise it will log the error.

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.twitch.tv/kraken', true);

xhr.setRequestHeader('Client-ID', '...');

xhr.onload = function(data){
  console.log(data);
};

xhr.onerror = function(error){
  console.log(error.target.status);
};

xhr.send();
```

jQuery

The following will retrieve a `channel` object for the `twitch` channel. If the request was successful the `channel` object will be logged to the console.

```
$.ajax({
  type: 'GET',
  url: 'https://api.twitch.tv/kraken/channels/twitch',
  headers: {
    'Client-ID': '...'
  },
  success: function(data) {
    console.log(data);
  }
});
```

Read Calling Twitch APIs online: https://riptutorial.com/twitch/topic/760/calling-twitch-apis

Chapter 3: Getting an OAuth token using the Authorization Code Flow

Examples

Send the user to the authorize endpoint to get the authorization code

You'll first send the user to the Twitch authorization endpoint. This URL is made up of a the base authorization URL (`https://api.twitch.tv/kraken/oauth2/authorize`) and query string parameters that define what you're requesting. The required parameters are `response_type`, `client_id`, `redirect_uri`, and `scope`.

For the Authorization Code flow, the `response_type` parameter is always set to `code`. This signifies that you're requesting an authorization code from the Twitch API.

The `redirect_uri` is where the user will be redirected after they approve the scopes your application requested. This must match what you registered on your Twitch account Connections page.

The `client_id` is a unique identifier for your application. You can find your client ID on the Connections page, too.

The `scope` defines what you have access to on behalf of the user. You should only request the minimum that you need for your application to function. You can find the list of scopes on the Twitch API GitHub.

The `state` parameter is also supported to help protect against cross-site scripting attacks. The `state` parameter will be included on the `redirect_uri` when the user authorizes your application.

```
https://api.twitch.tv/kraken/oauth2/authorize
  ?response_type=code
  &client_id=[your client ID]
  &redirect_uri=[your registered redirect URI]
  &scope=[space separated list of scopes]
  &state=[your provided unique token]
```

Get the authorization code from the query string

When the user goes to the authorization endpoint, they will be asked to give your application permission to the scopes that you've requested. They can decline this, so you must make sure to take that into consideration in your code. After they've allowed your application access, the user will be redirected to the URL you specified in `redirect_uri`. The query string will now have a `code` parameter, which is the authorization code that you can exchange for an OAuth token.

```
<?php
  $authCode = $_GET['code'];
```

```
?>
```

Exchange the code for the OAuth token

Now that you have an authorization code, you can make a POST to the token endpoint (`https://api.twitch.tv/kraken/oauth2/token`) to get an OAuth token. You will receive a JSON-encoded access token, refresh token, and a list of the scopes approved by the user. You can now use that token to make authenticated requests on behalf of the user.

```
<?php
  $authCode = $_GET['code'];

  $parameterValues = array(
    'client_id' => '...',
    'client_secret' => '...',
    'grant_type' => 'authorization_code',
    'redirect_uri' => 'http://localhost/',
    'code' => $authCode
  );

  $postValues = http_build_query($parameterValues, '', '&');

  $ch = curl_init();

  curl_setopt_array($ch, array(
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_URL => 'https://api.twitch.tv/kraken/oauth2/token',
    CURLOPT_POST => 1,
    CURLOPT_POSTFIELDS => $postValues
  ));

  $response = curl_exec($ch);
  curl_close($ch);

  echo $response;
?>
```

Read Getting an OAuth token using the Authorization Code Flow online: https://riptutorial.com/twitch/topic/6624/getting-an-oauth-token-using-the-authorization-code-flow

Chapter 4: Interactive Embed Video Player

Examples

LIVE Streaming Video Player

Basic implementation:

```
<script src= "http://player.twitch.tv/js/embed/v1.js"></script>
<div id="PLAYER_DIV_ID"></div>
<script type="text/javascript">
    var options = {
        width: 854,
        height: 480,
        channel: "monstercat",
    };
    var player = new Twitch.Player("PLAYER_DIV_ID", options);
    player.setVolume(0.5);
</script>
```

With controls hidden:

```
<script src= "http://player.twitch.tv/js/embed/v1.js"></script>
<div id="PLAYER_DIV_ID"></div>
<script type="text/javascript">
    var options = {
        width: 854,
        height: 480,
        channel: "monstercat",
        controls: false,
    };
    var player = new Twitch.Player("PLAYER_DIV_ID", options);
    player.setVolume(0.5);
</script>
```

Recorded (not live) Video Player

```
<script src= "http://player.twitch.tv/js/embed/v1.js"></script>
<div id="PLAYER_DIV_ID"></div>
<script type="text/javascript">
    var options = {
        width: 854,
        height: 480,
        video: "v53336925",
    };
    var player = new Twitch.Player("PLAYER_DIV_ID", options);
    player.setVolume(0.5);
</script>
```

The above snippet will play the following video: twitch.tv/general_mittenz/**v/53336925**

Start with a Muted Player

```
<script src= "http://player.twitch.tv/js/embed/v1.js"></script>
<div id="{PLAYER_DIV_ID}"></div>
<script type="text/javascript">
    var options = {
        width: 854,
        height: 480,
        channel: "{CHANNEL}"
    };
    var player = new Twitch.Player("{PLAYER_DIV_ID}", options);
    player.setMuted(true);
</script>
```

Read Interactive Embed Video Player online: https://riptutorial.com/twitch/topic/470/interactive-embed-video-player

Chapter 5: Lists of Streamers by Game

Examples

Getting the First Page in Ruby

This Ruby example uses Mechanize, a library to automate web interactions.

`client_id` is an OAuth client_id.

`game` is the game directory to list.

```
require 'mechanize'
master_agent = Mechanize.new

client_id = "123"
game = "Minecraft"

url = "https://api.twitch.tv/kraken/streams?game=#{game}&client_id=#{client_id}"
final_list = []
master_agent.get(url) do |page|
    master_list = JSON.parse(page.body)
    master_list["streams"].each do |stream|
        final_list << stream["channel"]["name"]
    end
end
```

Read Lists of Streamers by Game online: https://riptutorial.com/twitch/topic/552/lists-of-streamers-by-game

Chapter 6: Twitch Chat (IRC) Bot

Remarks

Twitch Chat is a simple IRC chat. For any serious development, there are multiple documents for it, including the most comprehensive and general ressource: http://ircdocs.horse/

Connection, Handshake

IRC is a basic, plaintext based TCP protocol. Connecting to Twitch works just like any regular IRC service with a difference in authenticating:

Connection Initiation > Handshake > Usage

The handshake is regularly the hardest part to get right:

After building up the connection to the server, you are required to provide `PASS` and **then** a `NICK`, where `PASS` is an OAuth-Token (which you can generate here) and `USER` being the username to this OAuth token.

The handshake is then as following (`<` being sent from client to server, `>` being sent from server to client):

```
< PASS oauth:your_oauth_token
< NICK your_username
> :tmi.twitch.tv 001 your_username :connected to TMI
> :tmi.twitch.tv 002 your_username :your host is TMI
> :tmi.twitch.tv 003 your_username :this server is pretty new
> :tmi.twitch.tv 004 your_username tmi.twitch.tv 0.0.1 w n
> :tmi.twitch.tv 375 your_username :- tmi.twitch.tv Message of the day -
> :tmi.twitch.tv 372 your_username :- not much to say here
> :tmi.twitch.tv 376 your_username :End of /MOTD command
```

Once you received either any of these `MODE`, `376` or `422`, you're good to go and can send the twitch server any commands, like:

```
> JOIN :#gamesdonequick
> PRIVMSG #gamesdonequick :Hello world!
```

A more throughout guide to client-server commands can be found here.

Twitch-specific Capabilities

While Twitch uses a standard IRC service, there are some events seen on the IRC service which correlate to activity in a channel on the Twitch website. Examples here are slowmode being enabled or disabled, subscriber-only mode being enabled/disabled on a streamer's chat, hosting activity, and bits/cheer activity, among others.

Details on these Twitch-specific capabilities are listed in the GitHub documentation for Twitch IRC, which can be found here.

Examples

Python

Here is a simple Python command-line program which will connect to a Twitch channel as a bot and respond to a few simple commands.

Dependencies:

- `irc` Python lib (`pip install irc` or `easy_install irc`)

Source: https://gist.github.com/jessewebb/65b554b5be784dd7c8d1

```
import logging
import sys

from irc.bot import SingleServerIRCBot


# config
HOST = 'irc.twitch.tv'
PORT = 6667
USERNAME = 'nickname'
PASSWORD = 'oauth:twitch_token'  # http://www.twitchapps.com/tmi/
CHANNEL = '#channel'


def _get_logger():
    logger_name = 'vbot'
    logger_level = logging.DEBUG
    log_line_format = '%(asctime)s | %(name)s - %(levelname)s : %(message)s'
    log_line_date_format = '%Y-%m-%dT%H:%M:%SZ'
    logger_ = logging.getLogger(logger_name)
    logger_.setLevel(logger_level)
    logging_handler = logging.StreamHandler(stream=sys.stdout)
    logging_handler.setLevel(logger_level)
    logging_formatter = logging.Formatter(log_line_format, datefmt=log_line_date_format)
    logging_handler.setFormatter(logging_formatter)
    logger_.addHandler(logging_handler)
    return logger_

logger = _get_logger()


class VBot(SingleServerIRCBot):
    VERSION = '1.0.0'

    def __init__(self, host, port, nickname, password, channel):
        logger.debug('VBot.__init__ (VERSION = %r)', self.VERSION)
        SingleServerIRCBot.__init__(self, [(host, port, password)], nickname, nickname)
        self.channel = channel
        self.viewers = []

    def on_welcome(self, connection, event):
        logger.debug('VBot.on_welcome')
        connection.join(self.channel)
        connection.privmsg(event.target, 'Hello world!')

    def on_join(self, connection, event):
        logger.debug('VBot.on_join')
        nickname = self._parse_nickname_from_twitch_user_id(event.source)
        self.viewers.append(nickname)

        if nickname.lower() == connection.get_nickname().lower():
            connection.privmsg(event.target, 'Hello world!')

    def on_part(self, connection, event):
        logger.debug('VBot.on_part')
        nickname = self._parse_nickname_from_twitch_user_id(event.source)
        self.viewers.remove(nickname)

    def on_pubmsg(self, connection, event):
        logger.debug('VBot.on_pubmsg')
        message = event.arguments[0]
        logger.debug('message = %r', message)
        # Respond to messages starting with !
        if message.startswith("!"):
            self.do_command(event, message[1:])

    def do_command(self, event, message):
        message_parts = message.split()
        command = message_parts[0]

        logger.debug('VBot.do_command (command = %r)', command)

        if command == "version":
            version_message = 'Version: %s' % self.VERSION
            self.connection.privmsg(event.target, version_message)
        if command == "count_viewers":
            num_viewers = len(self.viewers)
            num_viewers_message = 'Viewer count: %d' % num_viewers
            self.connection.privmsg(event.target, num_viewers_message)
        elif command == 'exit':
            self.die(msg="")
        else:
            logger.error('Unrecognized command: %r', command)

    @staticmethod
    def _parse_nickname_from_twitch_user_id(user_id):
        # nickname!username@nickname.tmi.twitch.tv
        return user_id.split('!', 1)[0]


def main():
    my_bot = VBot(HOST, PORT, USERNAME, PASSWORD, CHANNEL)
    my_bot.start()


if __name__ == '__main__':
    main()
```

Read Twitch Chat (IRC) Bot online: https://riptutorial.com/twitch/topic/1847/twitch-chat--irc--bot

Credits

S. No	Chapters	Contributors
1	Getting started with twitch	Community, DallasNChains
2	Calling Twitch APIs	DallasNChains, KnottytOmo
3	Getting an OAuth token using the Authorization Code Flow	DallasNChains
4	Interactive Embed Video Player	Community, Tim Penner
5	Lists of Streamers by Game	Christopher Muller, Community, DallasNChains, Josh
6	Twitch Chat (IRC) Bot	awkwardpaws, Jesse Webb, Josh, Mio Bambino, Old Badman Grey