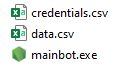
Welcome to Urgableh’s Scuffed-Twitch-Channel-Point-Redemption-OBS Bot.

Last update: 03/07/2020 with OBS 25.0.8

The following guide is a high-level usage of the bot. Source code is attached in the appendix for your perusal if you don’t trust me (I wouldn’t trust me).



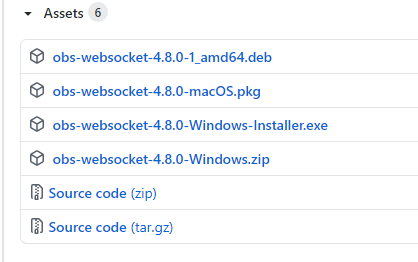
credentials.csv contains the twitch access keys for your channel and your bot.

data.csv contains the channel point redemption to OBS source linkages and timing/activation commands.

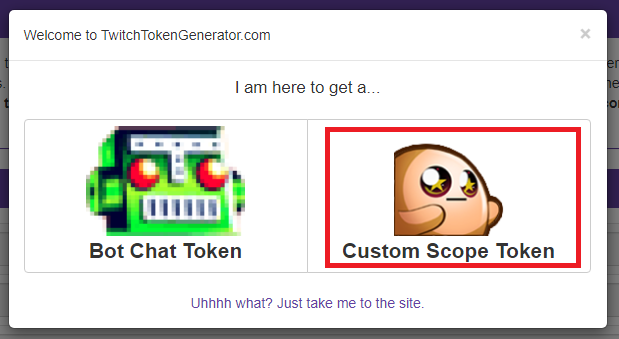
mainbot.exe is the packaged executable running the source code attached at the appendix.

# Setup

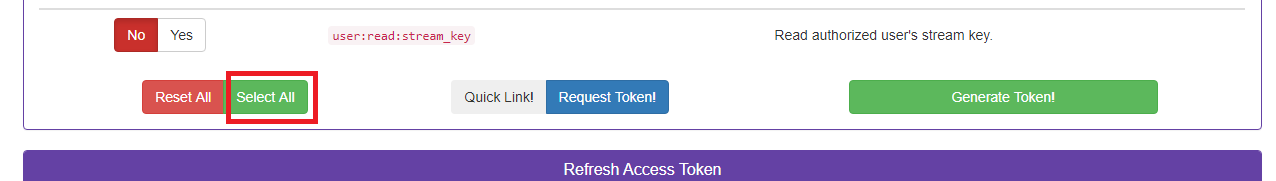
1. **If you haven’t already, download and install obs-websocket.**
   1. Go to <https://github.com/Palakis/obs-websocket/releases/tag/4.8.0>
   2. Download and install depending on your operating system



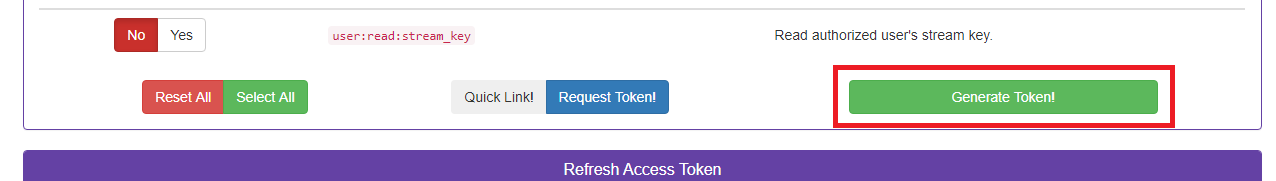
1. **Inputting credentials.csv values.**
   1. Go to <https://twitchtokengenerator.com/> and click Custom Scope Token



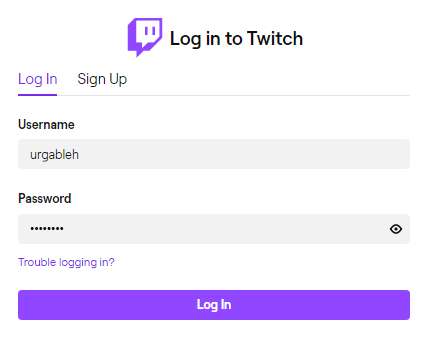
* 1. Scroll to the bottom and Select All



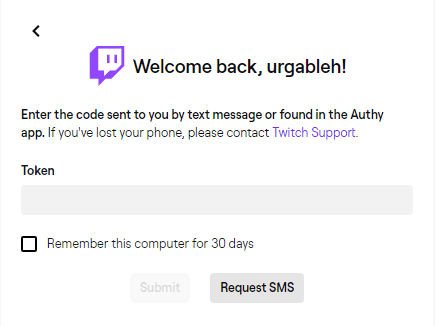
* 1. Click Generate Token!



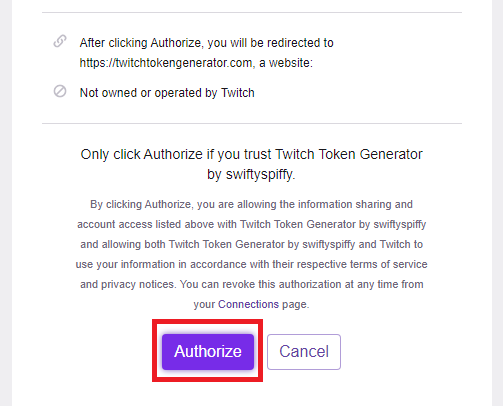
* 1. Login with your channel information as prompted



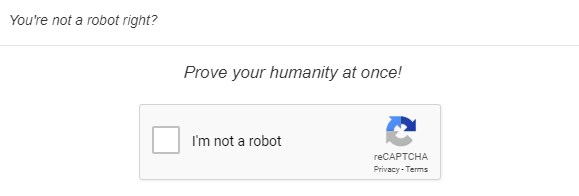
* 1. It may prompt you for 2FA if you have it activated



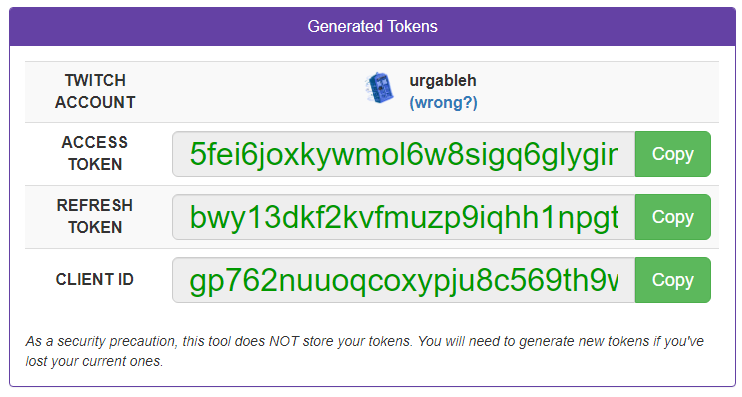
* 1. Click Authorize.



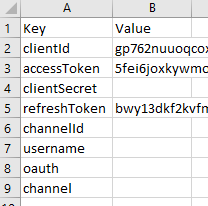
* 1. Fill out the Captcha



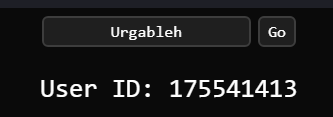
* 1. The generated tokens will then be displayed.



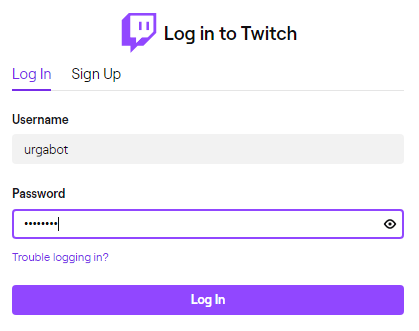
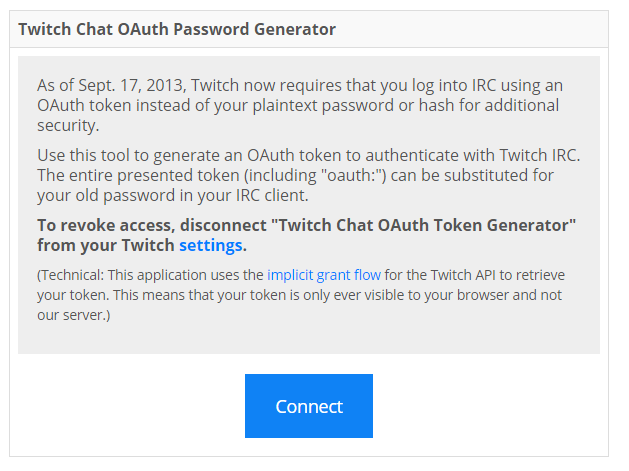
* 1. Locate credentials.csv and copy + paste these values into their associated columns.

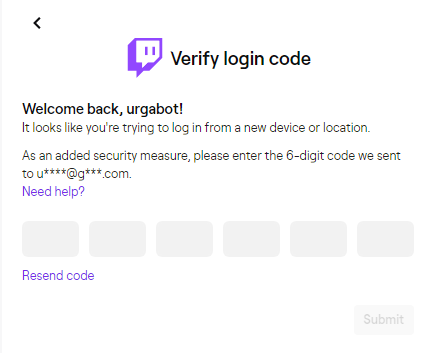


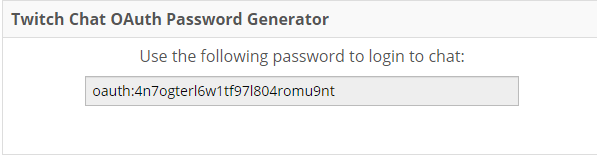
* 1. Retrieve your channelId by going to <https://codepen.io/Alca/pen/yLBdjyb> developed by Alca.  
     Add this to the credentials.csv as well as your channel.



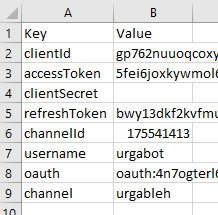
* 1. To generate an oauth password for you bot account, go to <https://twitchapps.com/tmi/>   
     (Ensure that this is NOT the same account as your channel account, unless you want the bot to also be the same account as the channel account.) (You could use an incognito window for a separate login. It may need extra verification.)







* 1. Copy the entire string (including oauth) into the credentials.csv file along with the bot username.



1. **Inputting data.csv values.**
   1. Values vary depending on the alert needing to be activated by channel point redemptions.

If OBS Scenes are included as sources of other scenes, the bot will detect this. It is important to include the scene that it originates from in data.csv.

Scene: OBS Scene

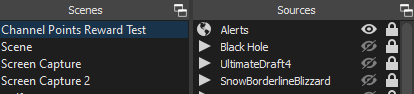
Source: OBS Media Source

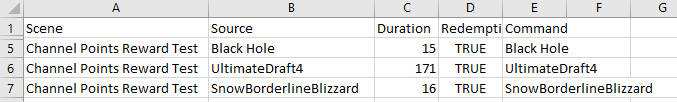
Duration: Duration of full Media Source

Redemption: TRUE or FALSE (different features are implemented, but the focus is channel points)

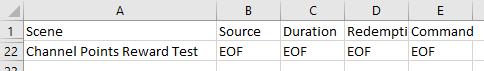
Command: Channel point redemption reward name

For example, the following sources may be entered as:

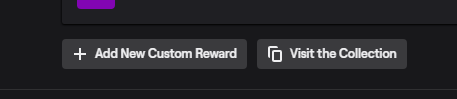




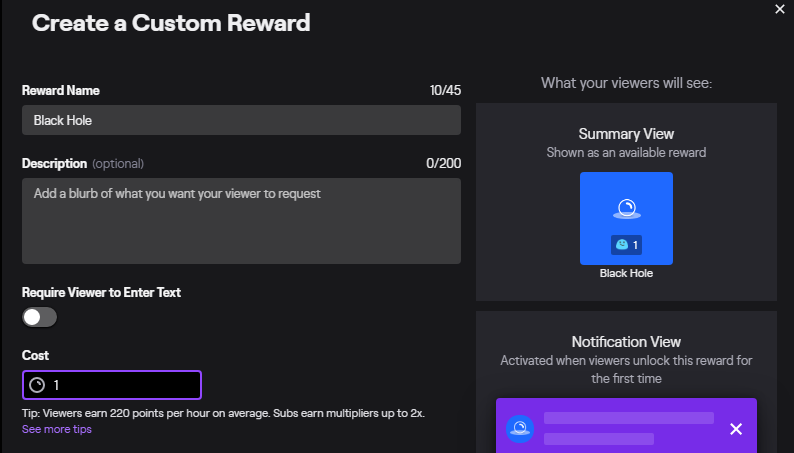
* 1. Ensure the final line is an End of File line for an existing scene (current bug >.<):



1. **Twitch channel point redemption rewards.**
   1. Navigate to your Channel Points > Manage Rewards & Challenges on Twitch (e.g. <https://dashboard.twitch.tv/u/urgableh/community/channel-points/rewards>)
   2. Click Add Custom Reward



* 1. Enter a Reward Name (**which will be the same as the Command value in data.csv**) and a Cost.
  2. Ensure that it does NOT require viewer to enter text.
  3. Click Create.



# Operation

1. Open OBS.
2. Ensure credentials.csv and data.csv are in the same file directory as mainbot.exe.
3. Open mainbot.exe.

# Appendix

/\*

In cmd, install the following packages:

npm install tmi.js

npm install obs-websocket-js

npm install --save twitch twitch-pubsub-client

Twitch chatbot followed this guide: https://dev.twitch.tv/docs/irc

package using 'pkg mainbot.js --targets node10-win-x64'

\*/

const fs = require('fs');

const util = require('util');

const tmi = require('tmi.js');

const OBSWebSocket = require('obs-websocket-js');

const PubSubClient = require('twitch-pubsub-client').default;

const request = require("request");

const TwitchClient = require('twitch').default;

var obsData = [];

var credentialsData = [];

function readData() {

// Reads a file in the same directory

  var text1 = fs.readFileSync('data.csv', 'utf8');

  processData(text1, obsData);

  var text2 = fs.readFileSync('credentials.csv', 'utf8');

  processData(text2, credentialsData);

}

// Processes a csv into "lines" variable by splitting

function processData(allText, lines) {

  allText = allText + '';

  var allTextLines = allText.split(/\r\n|\n/);

  var headers = allTextLines[0].split(',');

  for (var i=1; i<allTextLines.length; i++) {

      var data = allTextLines[i].split(',');

      if (data.length == headers.length) {

          var tarr = [];

          for (var j=0; j<headers.length; j++) {

              tarr.push([headers[j],data[j]]);

          }

          lines.push(tarr);

      }

  }

  //console.log(lines);

}

// Execute data reading and storage

readData();

// Define keys for pubsub

const clientId = credentialsData[0][1][1];    //https://twitchtokengenerator.com/

const accessToken = credentialsData[1][1][1]; //https://twitchtokengenerator.com/

const clientSecret = credentialsData[2][1][1];

const refreshToken = credentialsData[3][1][1];  //https://twitchtokengenerator.com/

const channelId = credentialsData[4][1][1];      // https://codepen.io/Alca/pen/yLBdjyb

// Define configuration options

const opts = {

  identity: {

    username: credentialsData[5][1][1],

    password: credentialsData[6][1][1]    //from https://twitchapps.com/tmi/

  },

  channels: [

    credentialsData[7][1][1]

  ],

  connection: {

    server: 'irc-ws.chat.twitch.tv',

    port: 80

  }

};

// Initiate Queue features

var queue = require('queue');

var q = queue();

var results = [];

var temp1 = [], j = 0, temp2;

// Redemption obs function

function runningQueue(redemptionName, sceneName, timeS, redeemerName) {

  var sceneMatch = false, sourceMatch = false;

  q.stop();

  q.timeout = 99999;

  console.log(redemptionName);

  obs.send('GetSceneList')

  .then(data => {

    obs.send('GetCurrentScene')

    .then(data => {

        if (data.name == sceneName) {

          sceneMatch = true;

        }

        for (i=0; i< data.sources.length - 1; i++) {

          if (data.sources[i].name == redemptionName) {

            sourceMatch = true;

          }

        }

    })

    .then(data => {

      if (sceneMatch == true && sourceMatch == true) {

        redeemChat(redemptionName, redeemerName);

        obs.send('SetSceneItemRender', {

          source: redemptionName,

          render: false,          // Disable visibility

          "scene-name": sceneName

        })

        .catch(err => {

          console.log(err);

        });

        wait(100);  // Necessary to wait between setting attributes

        // It was found that it would ignore one of the requests if it was too fast.

        obs.send('SetSceneItemRender', {

          source: redemptionName,

          render: true,           // Enable visibility

          "scene-name": sceneName

        })

        .catch(err => {

          console.log(err);

        });

        temp1[j] = setTimeout(function() {

          obs.send('GetSceneList')

          .then(data => {

            //console.log(data);

            obs.send('SetSceneItemRender', {

              source: redemptionName,

              render: false,          // Disable visibility

              "scene-name": sceneName

            })

            .catch(err => {

              console.log(err);

            });

          })

        }, (timeS + 3)\*1000);

        if (temp2) {

          clearTimeout(temp2);

        }

        temp2 = setTimeout(startqueue, (timeS+3)\*1000);

        j++;

      }

      else {

        console.log(redemptionName + " is not a recognised alert redemption in the current scene.");

        startqueue();

      }

    })

    //console.log(data);

  })

}

// Chatbot to say who's redemption is being fulfilled atm

function redeemChat(redemptionName, redeemerName) {

  client.say(opts.channels[0],`Playing ${redeemerName}'s ${redemptionName}`);

}

// Start queue

function startqueue() {

  q.start();

}

// Redemption internal function

function inRedemption(channelId, message) {

  var timeS;

  var redemptionName = null;

  var redeemerName = message.userDisplayName;

  var sceneName = null;

  var i;

  for (i=0; i < obsData.length - 1; i++) {

    if (message.rewardName == obsData[i][4][1]) {

      redemptionName = obsData[i][1][1];

      sceneName = obsData[i][0][1];

      timeS = (parseFloat(obsData[i][2][1]));

    }

  }

  return [redemptionName, sceneName, timeS, redeemerName];

}

// Redemption from pubsub client

const runRedemption = async () => {

  const twitchClient = TwitchClient.withCredentials(clientId, accessToken, undefined, {clientSecret, refreshToken, onRefresh: async (t) => {

  }});

  const pubSubClient = new PubSubClient();

  await pubSubClient.registerUserListener(twitchClient);

  var temp = false;

  pubSubClient.onRedemption(channelId, (message) => {

    if ( q.length == 0 ) {

      temp = true;

    }

    else {

      temp = false;

    }

    console.log("Redemption received");

    var redeemed = inRedemption(channelId, message);

    q.push(function (run) {

      results.push(runningQueue(redeemed[0], redeemed[1], redeemed[2], redeemed[3]))

      run();

    })

    q.push(function (run) {

      results.push(console.log("Redemption complete"));

      run();

    })

    if (temp){

      q.start();

    }

  })

}

// Run redemption bot

runRedemption();

// Scene and source constants

const sceneMain = 'Screen Capture 2'

const sceneMain2 = 'AndthenScene'

const sourceMain = 'Andthen'

const sourceMainEx = 'NoAndthen'

const waitPeriod = 15           // Global cooldown (s) when triggering alerts to disable again

// Create a client with our options

const client = new tmi.client(opts);

const obs = new OBSWebSocket();

obs.connect()

.then(() => {

    console.log(`Success! We're connected & authenticated to OBS.`);

    return obs.send('GetSceneList');

})

.then(data => {

    //console.log(data);

    console.log(`${data.scenes.length} Available Scenes!`);

})

.catch(err => { // Promise convention dicates you have a catch on every chain.

    console.log(err);

});

// Register our event handlers (defined below)

client.on('message', onMessageHandler);

client.on('connected', onConnectedHandler);

// Connect to Twitch:

client.connect()

.catch(err => {

  console.log(err);

});

// Global variables for functions

var counter1 = 1;

var coolingdown = false;

var subonly = false;

// Called every time a message comes in

function onMessageHandler (target, context, msg, self) {

  // Remove whitespace from chat message and take the first word

  const commandName = msg.trim().split(' ')[0];

  if (coolingdown) {    // If cooling down, keep monitoring chat but do not respond

    console.log('cooling down...');

    return;

  }

  else {

    if (self) { return; } // Ignore messages from the bot

    var subbed = context.subscriber;  // variable to check if message is from a subscriber

    // Toggle submode if channel owner or moderator

    if (context.username === `${opts.channels[0].split("#").pop()}` || context.mod === true) {

      if (commandName === '!submode') {

        submode(target, context, msg, self);

        return;

      }

//      THIS FEATURE CAN BE ABUSED TO BREAK THE TIMEOUT FUNCTIONS

//      if (commandName === '!clearqueue') {

//        q.end();

//        client.say(target, `Redemption queue has been cleared.`);

//        return;

//      }

    }

    // If sub mode is enabled and chatter is not a sub, then return.

    if (subonly && !subbed) {

      return;

    }

    // Deactivates then reactivates the visibility of the Andthen alert in Screen Capture 2

    if (commandName === '!andthen') {

      andthenf(target, context, msg, self, commandName);

      coolingdown = true;   // sets a cooldown variable to true

      setTimeout(cooldown, waitPeriod\*1000);  // calls the function to re-enable commands

      return;

    }

    else {

      //  console.log(`\* Unknown command ${commandName}`);

    }

  }

}

//////// FUNCTION DEFINITIONS BELOW ////////

// Called every time the bot connects to Twitch chat

function onConnectedHandler (addr, port) {

  //  client.say(opts.channels[0],'/me is now running.');

  console.log(`\* Connected to ${addr}:${port}`);

}

// Cooldown function that resets the cooldown and resets invisibility of sources

function cooldown() {

  coolingdown = false;

  obs.send('GetSceneList')

  .then(data => {

    //console.log(data);

    obs.send('SetSceneItemRender', {

      source: sourceMain,

      render: false,          // Disable visibility

      "scene-name": sceneMain

    });

    obs.send('SetSceneItemRender', {

      source: sourceMainEx,

      render: false,          // Disable visibility

      "scene-name": sceneMain

    });

  })

}

// Function to hold the program for ms seconds in milliseconds

function wait(ms){

  var start = new Date().getTime();

  var end = start;

  while(end < start + ms) {

    end = new Date().getTime();

 }

}

// function for andthen

function andthenf(target, context, msg, self, commandName){

  if (counter1%5 !== 0) {     // Activate if counter1 is not divisible by 5.

    client.say(target,`AND THEN?! (${counter1})`);

    obs.send('GetSceneList')

    .then(data => {

      //console.log(data);

      obs.send('SetSceneItemRender', {

        source: sourceMain,

        render: false,          // Disable visibility

        "scene-name": sceneMain

      });

      wait(100);  // Necessary to wait between setting attributes

      // It was found that it would ignore one of the requests if it was too fast.

      obs.send('SetSceneItemRender', {

        source: sourceMain,

        render: true,           // Enable visibility

        "scene-name": sceneMain

      });

    })

    .catch(err => {

      console.log(err);

    });

    counter1++;

    console.log(`\* Executed ${commandName} command`);

  }

  else {     // Activate if counter1 is divisible by 5.

    client.say(target,`NO AND THEN!! (${counter1})`);

    obs.send('GetSceneList')

    .then(data => {

      //console.log(data);

      obs.send('SetSceneItemRender', {

        source: sourceMainEx,

        render: false,          // Disable visibility

        "scene-name": sceneMain

      });

      wait(100);  // Necessary to wait between setting attributes

      // It was found that it would ignore one of the requests if it was too fast.

      obs.send('SetSceneItemRender', {

        source: sourceMainEx,

        render: true,           // Enable visibility

        "scene-name": sceneMain

      });

    })

    .catch(err => {

      console.log(err);

    });

    counter1++;

    console.log(`\* Executed ${commandName} command`);

  }

}

function submode(target, context, msg, self, commandName){

  subonly = !subonly;

  var mode;

  if (subonly === true) {

    mode = 'sub only';

  }

  else {

    mode = 'free for all';

  }

  client.say(target, `/me is in ${mode} mode.`)

  .catch(err => {

    console.log(err);

  });

}