MMTwitchBot Basic Setup and Usage Guide

This is a simple guide to help you set up the Twitch.tv IRC extension for GameMaker Studio 2, by MaddeMichael
Setup for GameMaker: Studio 1.4 is very similar, but not covered in this guide.
See the TwitchQuickReference.txt file for a quick overview of all user scripts

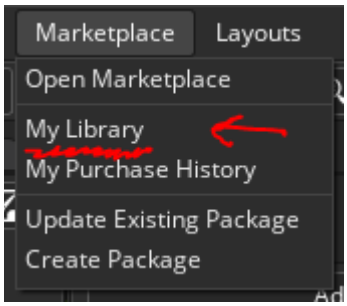
CONTENTS

INSTALLATION

There are two methods of installation: Installing via the Marketplace Library and Installing via the local asset pack (if you downloaded from itch.io)

1) Install Via The GameMaker Marketplace

After claiming the Asset from the Marketplace (https://marketplace.yoyogames.com/assets/6649/mm-s-twitch-tv-irc), open your project in GameMaker, and open Marketplace>My Library from the navbar



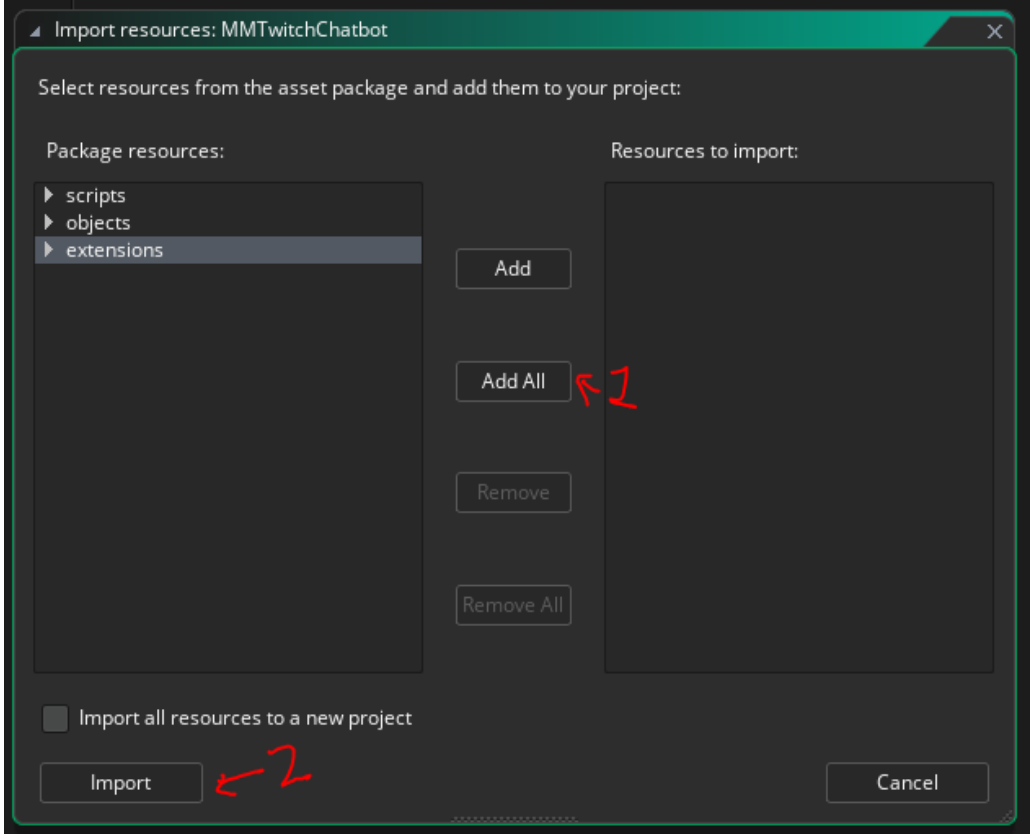
Then, simply locate the asset “MM's Twitch.tv IRC” and hit the download button. (It should be located under “Scripts>Networking” if you need to find it easier)
Once the download is complete (or if you already downloaded it) click the button to import it into your project. See “3) Actually importing the asset” for the next steps.

2) Install via the local asset pack

Locate the “MMTwitchChatbot.yymp” file you probably already downloaded. Open your project, and simply drag the file onto the IDE.
It will ask if you want to import Marketplace assets instead of a datafile. Click yes!
See “3) Actually importing the asset” for the next steps.

3) Actually importing the asset

After following the previous steps, you should be greeted with this window:



You will want to (1) click “Add All” to select all assets, then (2) click “Import” to import the files

And that's it! Installation is complete. You should now have one new object, a group of TwitchScripts and an extension “MMTwitchBot” in your project. Now you can move on to the next steps!

CONNECTING TO TWITCH CHAT IRC

Connecting to Twitch Chat couldn't be easier with this asset.
You'll need 3 things first – Your username, an oauth key generated for that user account with chat permissions and the name of the channel you wish to connect to (If it's your own channel, this is just your username)
To get an oauth key, the easiest method is to go to this website: https://twitchapps.com/tmi/

Then, all you do is simply call the script “twitch_connect” like so:

```
CONNECT_SUCCESS = twitch_connect(oauthkey, username, channelname)
```

oauthkey is the full key you generated **including** the starting “oauth:” bit
username is your username (as a string)
channelname is an optional argument, and is simply the channel name. If this argument is not provided, you will connect to the chat of your own channel

The script returns a value, which we store in “`CONNECT_SUCCESS`”. If the connection was successful, a positive number will be returned. If it failed, zero or a negative number will be returned. Note that if non-blocking connection is enabled, the script may return a positive number before connection is complete. A callback script will need to be used to determine if the connection was truly a success (covered in “Other functions and their usage)

Assuming all your information is correct, you should see a bunch of output in GameMaker's console. A successful connection should have output that appears similar to that seen here:

```
[TWITCH]: > :tmi.twitch.tv 001 maddemichael :Welcome, GLHF!
[TWITCH]: Successfully Connected as maddemichael
[TWITCH]: > :tmi.twitch.tv 002 maddemichael :Your host is tmi.twitch.tv
[TWITCH]: > :tmi.twitch.tv 003 maddemichael :This server is rather new
[TWITCH]: > :tmi.twitch.tv 004 maddemichael :-
[TWITCH]: > :tmi.twitch.tv 375 maddemichael :-
[TWITCH]: > :tmi.twitch.tv 372 maddemichael :You are in a maze of twisty passages, all alike.
[TWITCH]: > :tmi.twitch.tv 376 maddemichael :>
```

(Of course, 'maddemichael' is all replaced with whatever username you use)

NOTE: If formatted correctly, we hide your oauth key from the debug console. This is for when you're streaming dev to reduce risk of having your key stolen (unless you accidentally show it in code)

That's it for connecting! In the default state, the Twitch Manager object will now handle PINGS and PONGS, attempt reconnection if connection is lost, and parse received messages into something you can use easily (more on this in “Receiving and Processing a Command”)

When you've finished connection, you should call “`twitch_disconnect()`”. You can provide an optional argument “reason” that just adds to the debug output to help track disconnections.

When connecting to your own channel, the script assumes operator status for your game.

SENDING A SIMPLE MESSAGE

Once connected to the IRC server, sending a message to chat is super simple. You can just use the script “`twitch_send_chat_message(message)`”
It takes one argument, “message”, which is simply your chat message. It then queues and sends this to the chat of the channel you connected to. Super easy!
If you are not currently connected to chat, this message will be queued until you regain connection (assuming `twitch_connect` was previously called)
Long messages will be split up automatically (50 characters if your 'bot' is not a mod, otherwise 500)
Chat messages are also automatically rate limited (1 every 1.5 seconds for non-mods, 100 per 30 seconds for mods) to reduce risk of your game accidentally getting an 8 hour global ban.

So, sending a chat message is super simple, but you may want to use other commands seen on https://dev.twitch.tv/docs/irc
For this, you have two options:
`twitch_queue_command(command)` – queues a command to be sent when connection is established
`twitch_send_command(command)` – immediately sends a command, but does not care if connection is established.
Commands can be grouped by separating them with newlines (“`\r\n`”)
These scripts can open up some pretty nifty possibilities, but for most basic applications, they won't have much use to you.

RECEIVING AND PROCESSING A COMMAND

So, sending a message is super simple. But what do we do when we receive a message or command from the server? Well, this is also fairly simple.
Some commands are automatically handled by the Twitch Manager Object (PINGs mainly), but you'll need to handle specifics.

The first thing you need to do is create a “handler” script. This is how you'll be able to process all incoming data. Handler scripts must have one argument – this will be a “`twitch_object`” that contains the data to be processed.

“`twitch_object`”s are arrays that contain useful data that can be accessed easily via the enum “`public_twitch_message_struct`”. Possible values are:

```
message
username
channel (note, this usually starts with a '#')
parameters
command
tags
original
MAX
```

For a super basic example, we'll make a handler script that outputs chat messages into the console. This code should work in our handler script:

```
/// @arg twitch_object
var twitch_object = argument0;
switch(twitch_object[public_twitch_message_struct.command])
{
    case "PRIVMSG":
        show_debug_message("[TWITCH CHAT] " +
twitch_object[public_twitch_message_struct.username] + ": " +
twitch_object[public_twitch_message_struct.message]);
    break;
    default: break;
}
```

We'll call it something like "`my_twitch_handler`"

Now, there are numerous commands that can be received, so it would take a while to cover them all. You can always check https://dev.twitch.tv/docs/irc to see what kind of commands there are to process and how they should be handled.
You can also expand what kind of commands and information the client can receive with the "`twitch_set_cap_requests`" script (which should ideally be used before `twitch_connect`)

Now we just need to make the Twitch Manager use our callback script. This is pretty simple. We just call "`twitch_set_callback_script(script_index)`". In this case, we just call it like this:

```
twitch_set_callback_script(my_twitch_handler);
```

This can go before or after twitch_connect, but it is best to place it before to make sure any received messages are handled properly.

To see a more interesting implementation, check out the video guide which shows how chat can be used to manipulate stuff in-game.

OTHER IMPORTANT FUNCTIONS AND THEIR USAGE

Aside from what has already been covered, there are a few other useful scripts to use. These are all located in the QuickReference file too.

```
twitch_set_cap_requests(cap1, cap2, cap3, ...)
```

Used to setup capability requests - certain capabilities are needed for a Twitch-bot to receive certain commands, data and perform certain operations (More info: "https://dev.twitch.tv/docs/irc#twitch-irc-capability-tags")
Capabilities can be grouped into one string, separated by spaces, however, ALL must pass for the capability to be acknowledged.

It is useful to include the "membership" capability to determine the operator (or Mod) status of our game.

This script should be used **BEFORE** `twitch_connect();`

I've provided some basic macros to be supplied as arguments for most cases:

```
PUBLIC_TWITCH_CAP_MEMBERSHIP
PUBLIC_TWITCH_CAP_TAGS
PUBLIC_TWITCH_CAP_COMMANDS
```

```
PUBLIC_TWITCH_CAP_ALL
PUBLIC_TWITCH_CAP_CHATROOM
```

To clear the capability requests, simply pass '`undefined`' or an empty string as a single argument.

```
twitch_get_property(twitch_object, property_name)
```

If your game has been granted the "tags" capability, it can receive special user tags marked with the "@" symbol (More info: "https://dev.twitch.tv/docs/irc#twitch-irc-capability-tags")
Simply pass the `twitch_object` (usually within your command callback script) as well as the property name you wish to read.
If the property exists, the value is returned as a STRING, otherwise, '`undefined`' is returned. If the tag exists but has no value or an empty value assigned, the empty string "" is returned.
The script `twitch_unescape_string(string)` is also included to allow you to unescape properties such as “display-name”

```
twitch_set_non_blocking_connect_callback(script_index)
```

When using non-blocking connections (or HTML5), you'll need to specify a callback script to handle whether a connection was successful or not. Handler scripts will be passed a single argument, with the value 1 if connection was successful, or 0 if it failed.
This script should be used BEFORE twitch_connect();

```
twitch_enable_debug_output(enable)
```

Used to enable or disable Twitch debug messages which can show useful information about the current statuses.
By default, this is enabled.

This script should be used BEFORE `twitch_connect();`

```
twitch_enable_auto_reconnect(enable)
```

Used to enable or disable automatic reconnection if connection was lost after initial establishment (e.g, no server response for 12 minutes, internet connection lost)
By default, this is enabled.

This script should be used BEFORE `twitch_connect();`

HTML5 NOTES

I feel that it is slightly important to note that the HTML5 versions has some minor quirks due to the lack of native support for networking on HTML5.
To work around this, I built a small extension to provide a way of interfacing with Twitch chat via WebSockets, as well as some `gmcallback_*` scripts to allow async events.
This only has one issue worth worrying about – all connection requests are non-blocking, so you MUST specify a non-blocking connect callback, else twitch_connect will fail, telling you that you need a callback specified (for safety reasons).
Also note that `network_set_config` seems to be unimplemented on HTML5 as of writing, so it is important you DON'T call it on that platform, else your game will crash.
Other than that, the HTML5 version should work the same as all other platforms.

OTHER IMPORTANT NOTES

This bundle of scripts has been created for using *specifically* with Twitch.tv chat, with *one* user connected to *one* channel. While it is actually possibly to connect to multiple channels with this, it's not really designed for it – this can confuse the rate limiting (it only considers the channel specified in `twitch_connect`) and could lead to an unwanted 8 hour global ban.
It is probably also possible the modify the scripts to connect to other IRC clients fairly easily, however, the scripts aren't optimised for this sort of thing and I have little interest in supporting or fixing issues relating to this, as it isn't the goal of the extension (it just wants to connect to Twitch and allow chat to play your game)
There are also some “internal” globals and macros that can change the behaviour of the system (allowing things such as altered rate limiting, different servers, disabled automatic command handling, etc.). While you can play with these as much as you want, changing these values falls outside of what the extension is designed to do, so I can't really help you with that. Just be careful, okay!

IT'S OVER! YOU DID IT!

Thanks for checking this out, and I hope you have fun playing with Twitch!