Twitcher

A Twitch Chat / Vote Plugin For Unity

Overview

Ever want to add features to your game that make for a more entertaining engagement for streamers? Ever want to build a game where streamers can allow their audience to interact with the game?

Well Twitcher might be for you.

This plugin aims to provide you an easy to use, and extendable system for adding that kind of feature to your next project. It is all based around a system that manages the communication with the Twitch IRC services and provides you easy ways to receive and send messages easily.

On top of the basic chat system is a voting system that comes with several vote types out of the box. These votes can be used to add new levels of viewer engagement with streamed games, and can be easily extended to add your own custom vote types!

Using Twitcher - Chat

You can set up the very basics of twitcher in just a few lines of code. Create a TwitchController, give it a callback for the messages, and you're ready to start receiving the chat feed.

NOTE: If you want to test your integration, you will need an oauth token for a twitch account to log in as, these can be generated from the following url: https://twitchapps.com/tmi/ I am in no way affiliated with this service, but It's a simple and in my experience safe means of generating oAuth tokens.

```
private void Awake()
{
    twitch = TwitchController.Create("oauth_token", "twitch_channel");
    twitch.Client.onMessageReceived += OnMessageReceived;
}

private void OnMessageReceived(Message message)
{
    if (message.Command == TwitchClient.Commands.PRIVMSG)
    {
        Debug.Log($"{message.Sender}: {message.ChatMessage}");
    }
}
```

The onMessageReceived event will be called for any message that the twitch connection receives, with an exception of the PING message which is handled within the Twitcher plugin.

So that's receiving messages, but what about sending them? Well that's just as simple. The TwitchController class is a MonoBehaviour that contains the TCP connection to twitch you can send messages through this connection with a single line of code.

```
twitch.Client.SendPrivMessage("Your message here");
```

That's all there is to it. This message will be sent as the user who's OAuth token was used to create the TwitchController object.

Message

Property	Comments
RawMessage	The raw message as received through the tcp connection.
Info	Meta information about the message and sender. This includes information such as the chat colour for the users name, user status such as subscriber, mod, admin, etc. And several other useful bits of information.
Command	The command type for the message, these can be compared with constant values stored in TwitchClient.Commands, PRIVMSG will be the most common message type that you will be interested in and dealing with.
Sender	Automatically extracted from the parameters of the message, contains the id of the sender of the message, In cases of PRIVMSG types, this will be the username of the person who sent the message, in cases of message types like NOTICE this will be the name of the twitch server.
ChatMessage	In the case of PRIVMSG types this will contain the message the message that the sender sent to the chat. In the case of other command types, it will be null.
Parameters	The broken down parameters of the message as a string array.

Using Twitcher - Votes

Now for the fun stuff, votes.
Twitcher comes with three types of votes already supplied, those vote types are as follows:

- PersistentVote: This vote only ends when you want it to, once started it will continue to rack up votes for as long as you want, this would be used for much longer time period votes.

- TimedVote: Like it says on the tin, this vote has a time limit, once that time expires the vote will automatically end and provide you with the results to act upon.

- FirstToVote: This vote ends when one of the options reaches the set amount of votes.

You can create a vote instance at any time using the vote types constructor, but a vote will not start until you assign it to run on a Twitcher instance. This is done by simply passing the vote to the StartVote method on your TwitchController instance. The vote will then immediately become active, register itself for message events, and start tracking votes.

Vote types will track the username of the users who are casting votes, if a user votes multiple times within a given votes duration, then only their latest vote will be counted. When creating a vote you can provide a method to call when the vote completes, this will be given a list of the votes results, however at any time you can manually check the votes current result tally and progress status using the Results, and Progress properties respectively.

PersistentVote

Property	Comments
votePrefix	This prefix value must appear at the start of a message for it to be considered a vote.
options	Case-insensitive options that can be voted for when typed as the only other part of a message that is prefixed by the assigned votePrefix.
onComplete	Callback that will be called when this vote concludes, and given the votes final results as its only parameter

TimedVote

Property	Comments
votePrefix	*(See persistent vote)*
options	*(See persistent vote)*
duration	Time in seconds that the vote will be active for, this duration will start from when you pass the vote into the StartVote method of a Twitcher instance.
onComplete	*(See persistent vote)*

FirstToVote

Property	Comments
votePrefix	*(See persistent vote)*
options	*(See persistent vote)*
target	The number of votes an option requires to be the winning option of the vote.
onComplete	*(See persistent vote)*

Ending Votes

All votes can be ended at any time by calling the EndVote method on it. This will mark the vote as completed, and it will trigger the normal events when it’s assigned Twitcher updates.

If you want to end any and all active votes for any reason, say the game has ended, rendering the votes redundant, you can call ClearAllVotes on the TwitchController instance. By default this will silently end all votes, however if passed the parameter ‘true’ then it will broadcast all the normal events as if the votes had ended normally before clearing them from the active votes list.

Chat Extras

There is a whole lot of extra information that Twitcher gets for messages that you have access to should you want to use it. Things like the users chat colour, badges, status values for admin, mod, subscriber, etc. These can all be accessed via the Info property of a Message instance. Below is a brief overview of the values you can access from this:

Property	Comments
displayName	*Display name of the user sending the message.*
userId	*UserId of the sending user.*
admin *(deprecated)*	True if user has an admin badge, false otherwise.
subscriber *(deprecated)*	*True if user has a subscriber badge, false otherwise.*
moderator *(deprecated)*	*True if user has a moderator badge, false otherwise.*
staff *(deprecated)*	*True if user has a staff badge, false otherwise.*
broadcaster *(deprecated)*	*True if user has a broadcaster badge, false otherwise.*
permissions	*Enum flag for admin, subscriber, moderator, broadcaster, staff, viewer.*
colourHex	*Hex string for the users chat colour*
colour	*Unity Color struct for the users chat colour.*
timestamp	*The timestamp at which the twitch service received the message.*
bits	*The number of bits associated with this message.*
id	*ID assigned to this specific message by twitch.*
badges	*All badges associated with the user, each badge contains a badge id (string) and a version (integer)*
emotes	*A dictionary of all emotes contain in this message. The key is the ID of the emote in question, the List<Vector2Int> contains the indexes of the message that would otherwise be replaced with the emote.*

Emotes

Since it is intended for you to manage your own display for chat if you chose to display it, you must also handle any replacement of emotes in said display, however Twitcher does provide a method for downloading emotes.

TwitcherUtils.DownloadEmote can be called, with a given emote ID, and a size value (twitch offers emote images in 3 sizes). This is an async download using UnityWebRequest and you can provide a callback method which will be called with the resulting image once downloaded.

Chat Commands

New in version 1.0.3, ChatCommands are a simple to extend the behaviour of Twitcher.

ChatCommands are a way of combining a normal onMessageReceived callback with both a permission check, and an initial message filter. Commands can be constructed independently like the Votes, and then are assigned to a specific twitch client. Below is example code showing a very basic implementation of a chat command:

```
public void CreateCommand()
{
        var permissions = Message.Permission.Admin | Message.Permission.Broadcaster;
        var command = new ChatCommand("!ping", OnCommandReceived, permissions);
        command.AssignToClient(twitch.Client);
}

public void OnCommandReceived(Message message)
{
        twitch.Client.SendPrivMessage("Pong!");
}
```

This example creates a command that responds to the chat message of “!ping” but only when the user sending that message is the broadcaster, or an admin of the channel.

By default, all messages are flagged with the Permission.Viewer flag, but based on the presence of relevant badge data can also be set as Admin, Broadcaster, Moderator, Subscriber, and Staff. Whilst you can use the permission value itself, or the TagInfo.HasPermission method to check these permissions, the ChatCommand feature will take care of that for you, only calling the callback when the appropriate message is sent by an appropriate user.

Method	Comments
AssignToClient	Assigns this chat command to a specific twitch client. Note that if this command is already assigned to another client, it will be removed before being reassigned.
RemoveFromClient	Removes a command from a client if it is assigned.
SetPermissions	Updates the permissions associated with this command.

> The addition of chat commands can significantly increase the scope of Twitcher, not only can it be used to facilitate chat and voting features within a game you’re developing, but it can now be utilized to power more traditional ChatBot style features such as quotes, giveaways, auto-responses, and more.