Twitch Plays Documentation

Welcome to the Twitch Plays documentation, this asset package will help you set up a connection with Twitch chat and allow player input to affect your game.

Setup

To use the TwitchPlays system, you will need to set up two things.

1. A way for players to log into their twitch channel, so they can connect to their twitch chat and viewers

2. Twitch commands that can be entered by viewers and their corresponding actions in the game

Twitch Login

While this package comes with a drag and drop prefab to allow simple connection to a twitch account, you may want to create your own custom login panel to fit your game style.

Inside TwitchPlays/UI/UI Scripts you will find a TwitchLogin.cs file that has everything you need to setup a login system for twitch. All you need to provide are two *InputFields* and an optional login panel *GameObject* to disable once login is successful. The script also saves login information and will try to login automatically on startup if an account has been used before.

To fire the *login* command, create a button and connect it (using `OnClick()`) to TwitchLogin.cs' `Submit()` method. Everything else will be handled for you. If connection fails, it will be logged to the console (replace the `Debug.log()` with your own error handling).

Once you are successfully logged in, you'll be all set to start receiving commands from chat!

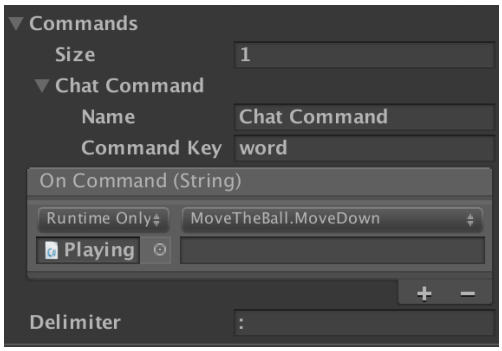
Command Setup

It's not much good to be connected to chat if you're not going to use viewer input, so here we'll talk about setting up chat commands so viewers can interact with your game world.

To start, drag in the TwitchPlays Client prefab from the TwitchPlays/Prefabs folder.
This prefab has two scripts attached to it, the Twitch IRC, and the Play commands. You don't need to touch anything in the Twitch IRC commands, as it handles the nuts and bolts of sending/receiving chat messages.

In the command function we can see a number of fields which we will go over now.

- Commands - The list of commands available to the Twitch chat
- Name - Internal name for command (allows you to remember what each command does)
- Command Key - single word that will be captured from twitch chat message
- OnCommand() - Method that will be called (just like OnClick() for buttons)
- Delimiter - Optional command delimiter to allow command 'options'



The Commands list and Name should be self explanatory, but we will go into a bit more detail for the Command Key, OnCommand(), and Delimiter options.

Command Key

The Command Key will be what the twitch viewers need to say to trigger an event within the game. For gameboy-like games, the command keys could be Up, Down, Left, Right, A, B, Select, and Start. If the game sees a message where the first word is a Command Key, it will fire the command. All commands are **not** case-sensitive (although you can change this in the code quite easily).

OnCommand()

The OnCommand() field is where you link the events you would like to trigger when a chat command is found. It works exactly like the UnityUI's event system for button clicks. You can use dynamic methods that take an input variable, but it must be a string. This means if you want to allow a command to take a number as an option, you will need to parse the number from the string.

Note: Make sure to use the Dynamic version of the method when selecting it from the dropdown menu, that way it will take the chat message option instead of an empty string.

Delimiter

The Delimiter is an optional character or string that signifies that there are options for a Command Key. This is useful if you wanted to allow twitch viewers to specify a value with a command.
For instance, if you were creating a 'Twitch plays chess', then a Command Key would be 'Move' as it is static. However, you would also need to include a value for the move (ie pawn at 2a move to 4a). You could set the Delimiter to "-", and the resulting command might look like:

"Move - p2a to 4a"

Your `HandleMove(string move)` command would be given the string *"p2a to 4a*" where it could be parsed to a computer-readable command (using `move.split("to")`). Invalid moves

can simply be ignored as many users are vying for input, so ignoring input is necessary for the system to work at scale.

TwitchIRC.cs

The TwitchIRC.cs file has additional public methods for developers comfortable with C# scripting.

`SendMsg(string msg)`

Send a message from the game to Twitch Chat. This could allow the game to prompt viewers for input at key moments in the game, or even allow for the player to chat with Twitch Chat without tabbing out of the game.

`messageRecievedEvent`

Any script that has a reference to the TwitchIRC (connected to twitch chat) can add a listener to the messageRecievedEvent. This listener will be called any time a message is received from twitch chat in its raw form.

```
twitch = GetComponent<TwitchIRC>();
twitch.messageRecievedEvent.AddListener(getMessage);

void getMessage(string msg){
     //Handle Message
     //Form: "PRIVMSG #nickName: message"
}
```

The raw message must be handled by your own code to parse out user and message (look at the PlayCommand.cs' `getCommand()` method to see an implementation).