StreamLinked Code Concepts

Created with reference to version 2.3.1

This document is to loosely cover the general ideas surrounding StreamLinked and its intended structural concepts that make it work inside and outside the package.
This will be updated based on user feedback requesting further information about specific ‘flows’ StreamLinked takes to achieve its goals.
Work in progress.

Authentication flow

Automation of Tokens vs Manual Acquisition

The Twitch API Client is built and operates by default on a system of refreshing tokens as soon as possible, if a token is detected to not be present or expired the Twitch API Client will refresh them. The setting to dictate if a token is to be automatically updated is located on the token itself under the tick box named ‘Auto Retrieve New Auth’.

MakeTwitchAPIRequest will make 2 attempts to refresh the token (If ‘Auto Retrieve New Auth’ is set to true);
1st is before the request is sent to Twitch, if the TokenInstance fails the check done by the method in TokenInstance called CheckRefreshNeeded[or *Async*], it will queue up the token to be refreshed by the API. MakeTwitchAPIRequest requests wait for the provided TokenInstance pending Auth requests to be completed before proceeding, so it queues the current token that needs refreshing and then proceeds to wait.
2nd is after the request has been sent to Twitch but has been returned as 401 status, which is the current token is no longer authenticated and a new one is required. MakeTwitchAPIRequest will then queue the token to be refreshed and will then restart the request process from the beginning. The pending token refresh will then like the prior and wait for the Auth requests to complete.

If you were to do this manually in the case of holding back attempted verification of tokens until optional Twitch connections are enabled, monitoring of tokens can be done via the token itself via CheckRefreshNeeded, the client via its events (OnAuthentication*) or its provided methods such as CheckOAuthExistsAndInDate.
Attempt has been made to do everything for you however everything (should) be open enough for if you want to do it manually or even outside of the client you can.

TextMeshPro Twitch Messages

To enable near full presentation of Twitch chat into Unity, TextMeshPro was required.
The structure of process is as such;

- IRC receives message

- Message contains badges or emotes and sends them to managers (if they exist)
- Managers either have them already acquired such as global emotes (from enabled) or will need to download them from the Twitch
- GameObject and Components are created
- Messages request the Atlas manager to produce a TextMeshPro image atlas of the specific images required for that instance along with the text to reference the attached atlas
- The managers return the atlas and the text components in either username and message segments or in segments separated by image position (and the ‘:’)
- The User either adds the message directly into the GameObject or parse it however they wish

Images are cached during a session and will need to be reacquired on restart as with the dynamic nature of Twitchs emotes and badges. Attempt has been made to try and allow saving of messages so that they can just reacquire the message if reopened however this has not been tested and not component has been made to do this.

InternalSettingsStore and Encryption

InternalSettingsStore is a set of methods to store and retrieve string data inside Unity PlayerPrefs. It does this using managed names under the Enum SavedSettings. Due to it being an internal Unity method, these methods must be called on Unitys main thread. These methods handle type conversions out of the string data stored inside them, however it will only convert to the base types for example int, double and bool.

Included also is overloads to provide an encryption key. An Aes class has been included to encrypt and decrypt data locally and also to generate a key to get started using them. Using the integrated methods, when you provide the 32 bit key to EditSetting, the string will be encrypted and then stored into PlayerPrefs. That same key when provided to TryGetSetting will decode the stored string and then convert it.
Please do not store the key in PlayerPrefs, please use a server or another method of storage to keep your keys for your deployed applications.
If you wish to use the encrypter directly, you can access it from the static class AesEncrypter, You can Encrypt providing your string and key, returning the encoded bytes. Then Decrypt the encoded bytes with the key. Finally if you need a key you can use GenerateKey which returns the key as a byte array.