iOS Developer Guide

(Build Version: 1.0.0.0)

Index

1. Introduction
2. Loading Stringflow SDK
3. User and Groups
4. Messaging
5. User Presence
6. Media Support
7. Advanced Concepts
8. Appendix

Prerequisite

It is recommended to read through “*Stringflow SDK Architecture & Design*” document for overall understanding of SDK Concepts. A high-level understanding of XMPP protocol concepts will also be helpful for developers (XMPP RFC <https://xmpp.org/rfcs/rfc6120.html>).

Who can read this document?

This document is intended to be used by iOS application developers. The documents assumes that, as a developer, you have familiarity with iOS programming model along with SWIFT language constructs. Basic understanding of XMPP protocol will also be helpful.

Introduction

Overview

Stringflow is a XMPP compliant server technology with multiple proprietary extensions such as Stringflow Content Model (SFCM) for media transfer across the network. While Stringflow server is available for download for on-premise installations, it is also available as a service on cloud. The on-cloud service has various convenience offerings on top of Stringflow server technology. Stringflow is a XMPP compliant service, therefore it is possible to access it using any third party XMPP client library, but in that case, you won’t be able to leverage Stringflow proprietary extensions. Therefore, it is recommended to use Stringflow Client Library.

Stringflow iOS SDK has been designed for seamless communication with Stringflow server with minimum effort from developer. There is a well-thought architecture underneath making it robust and predictable. Below is a high-level architecture of Stringflow iOS SDK-

Application

Platform

Event Bus

User Manager

Chat Manager

Presence Manager

XMPP Stream Manager

Connection Manager

XMPP Packet Reader

XMPP Packet Writer

XMPP Connection

TCP Connection

Network

As shown in the above diagram, Stringflow SDK has a layered architecture in which whole SDK has been divided into various layers communicating to each other; each layer consists of a set of entities.

Compatibility

Stringflow iOS SDK is written in SWIFT 4 and is compatible to all the iOS platform versions (8, 9, 10, and 11).

How to link within an application?

The SDK can be downloaded from github as well as from COCOAPOD (Give details for github as well as COCOAPOD)

Loading Stringflow iOS SDK

*ConnectionManager* is responsible for managing life-cycle of a network connections within SDK. By design, all the entities which require a network connection borrow it from *ConnectionManager.* As there is only one XMPP connection, the same instance is returned in the response of a borrow request. Due to restricted access,Network connections (*TCPConnection/XMPPConnection*) can’t be instantiated outside of network Layer. Connection managers ensures that there is only ONE connection alive at any given point in time for XMPP stream packet exchange (although there may be other connections created to support Stringflow Content Model aka SFCM). It also ensures that connections are cleaned-up when application shuts down SDK.

With unstable network such as mobile devices, network connections are bound to break frequently. In such cases client needs to detect broken connections, re-establish the connection with server and ensure redelivery of the user packets which may have been written to the network socket buffer but couldn’t be transmitted to server. In SDK, *ReconnectionManager* is responsible for making attempts to re-establish network connection with server. There are various Reconnection Policies within SDK which determines the frequency at which *ReconnectionManager* will attempt to reconnect with server.

IO Layer (Input-Output Layer)

IO Layer sits on the top of network layer and performs following tasks-

* Reading bytes from network
* Generating Packets
* Writing User Packets on to network
* Processing packet level acknowledgements

*Reader*, *Writer* and *XMPPStreamManager* are the major stakeholders in IO layer. *Reader* and *Writer* are continuously running threads sharing the same underlying connection for reading and writing bytes respectively. Reader continuously reads bytes from network, generates packets and forwards them to its subscribers (for more information please refer to Stringflow Collector/Forwarder framework). *XMPPStreamManager* is always subscribed to *Reader* for packet collection.

Similarly *Writer* receives all the packets which needs to be written on to the wire (connection). As the underlying connection is not always in CONNECTED state (subjected to network availability on device), these packets are queued up in *Writer* thread which are written to network as and when *Writer* thread can do so.

*XMPPStreamManager* is the entity which does the orchestration among *Reader*, *Writer* and *ConnectionManager*.

Application Layer

Application layer sits on top of the IO layer in SDK. It consists of the classes which are accessed by application to communicate with SDK. Following are the major entities which application developers are likely to use while integrating Stringflow SDK-

* Chat Manager
* Presence Manager
* User Manager

Each of these entities have clearly defined responsibilities; for example *UserManager* deals with managing user level transactions within SDK such as fetching user Roster from server, updating user roster, fetching groups that user is part of etc. Similarly, as the name suggests, *PresenceManager* deals with managing user presence and receiving presence broadcasts from server for other users. *ChatManager* has all the convenience methods to send/receive chat messages.

This document is written for core SDK which is not any platform specific. SDKs such as Android and iOS has extensions of the application layer classes which offers platform specific convenience methods.

Platform

Platform is the custodian for various entities in SDK. It keeps instances of all the application layer classes (*ChatManager*, *PresenceManager* and *UserManager*) along with user session data and Event Bus.

As soon as SDK is loaded (using *SDKLoader*) into memory, various instances are registered within Platform which application developers can access at will.

Event Bus

Stringflow SDK has a concept of events where events can be raised at any of layers within SDK. These events are pushed to Event Bus which maintains active subscribers/handlers for each of the event types. As soon as an event is placed onto Event Bus, it executes the registered handlers for the event. The events raised within core SDK are consistent across all the implementations (Java SDK, C# SDK, Android SDK etc).

Additionally, application developers can define custom events and handlers. These events can be generated at any place and time during execution and pushed on to the Event Bus allowing developers to leverage reactive programming model.

Appendix

Stringflow github account (<https://github.com/AlterBasics>) has lot of other documentations and samples.

Following repositories are available on githib-

* A sample Android application which uses Stringflow Android SDK <https://github.com/AlterBasics/BEACH_Android>
* A sample iOS application which uses Stringflow iOS <https://github.com/AlterBasics/BEACH_iOS>
* A command line Java client which uses Stringflow Core-Java SDK

<https://github.com/AlterBasics/sf_clc_java>

* iOS SDK artifacts, documentation and samples

<https://github.com/AlterBasics/sf_sdk_iOS>

* Android SDK artifacts, documentation and samples

<https://github.com/AlterBasics/sf_sdk_android>

* Java SDK artifacts, documentation and samples

<https://github.com/AlterBasics/sf_sdk_java>