StreamhubAnalytics Swift generic plugin for iOS and tvOS applications

5th of March 2018	2.0	QoS, metadata Frontloading tracking
23rd of April 2018	3.0	Swift 4.0 compatible Updated links
5th of June 2019	4.0	Swift 5.0 compatible version

For all kind of Apple applications (iOS, tvOS, swift app, objective-c app...), we provide a unique plugin built using the Swift programming language.

Download and extract the plugin from the archive at https://s3-eu-west-1.amazonaws.com/streamhub-static-content/plugins/genericplugins/player-plugin-ios-generic.zip

The plugin is composed of the following component:

1. StreamhubAnalytics.swift which is the sole API to integrate with.
2. ShUserData.swift which is responsable for tracking users.
3. NetworkUI.swift which interact with our remote Backend.

The downloadable .zip file contains many files but the 2 main entry points are :

1. **iosGenericPlugin.xcworkspace** is the Xcode project file for the Swift plugin and Swift sample video player application provided as an example of plugin integration
2. **swiftObjcHybridApp/bridgeApp/bridgeApp.xcworkspace** is the Xcode project file that provide an example of hybrid application written in objective-c and using our Swift plugin

Getting started

Both applications are configured via cocoapod.

To get you started quickly we've configured the 2 applications using cocoapod.
Open a terminal and navigate to the folder where you have extracted the .zip archive.
There, type 'pod install' which will install the unique required dependency to be able to compile and run the project in Xcode.

Follow the same step in the **swiftObjcHybridApp/bridgeApp** to be able to compile and run the hybrid objective-c/swift project in Xcode.

Browse around the code to get familiar with the plugin APIs.

Integrate the plugin in your own application

The integration process is quite easy and straightforward, because the complexity of interacting with our Backend REST API has been abstracted in the plugin logic.
If your application is a Swift application, you will be able to complete the integration very quickly.
If your application is a Hybrid application (written in Objective-C with a Swift plugin), then there will be a slight Xcode configuration overhead that we will cover below.

Add the Alamofire library dependency

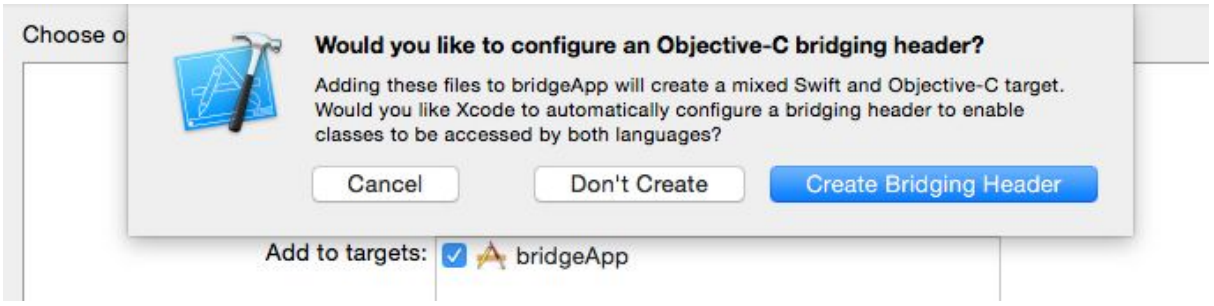
First, we need to install the Alamofire library which is a required dependency for the plugin. Just copy the Podfile from the .zip archive, or add the following line in your own Podfile:
pod 'Alamofire', '~> 3.0'

Then run ‘pod install’ in a terminal to install the library.
From now on, you have to open your application using the project file with extension .xcworkspace, no longer the .xcodeproj project file, as usual in the Xcode ecosytem.

If your application is written in Objective-C

In the case of an hybrid integration, there is a few extra steps you need to do before you can start using the plugin API.

First, you have to make Xcode generate an ***{applicationModuleName}*-Swift.h** header that will let you access the Swift plugin classes from your objective-c code.
In order to do that, just copy-paste the 3 Swift plugin files into your objective-c application ; NetworkUI.swift, ShUserData.swift, StreamhubAnalytics.swift.
Xcode will then prompt you with



As a result, you should now be able to import the ***{applicationModuleName}*-Swift.h** header file in your Objective-C own files and access the Swift plugin files.

Another alternative is to add a temporary Swift file by right-clicking your project structure > new file > Swift file, which will prompt the same message as above.
After that, just delete the temporary Swift which sole purpose is to force Xcode to generate the ***{applicationModuleName}*-Swift.h** header file.

You can see this step in action in the provided hybrid sample app's ViewController.m file at the top:
#import "bridgeApp-Swift.h"

Make sure your **Build Settings** parameters have the expected values:

- **Product Module Name :** myproject
- **Defines Module :** YES
- **Embedded Content Contains Swift :** YES

Create an instance of the StreamhubAnalytics class

Swift code:

```
var _streamhub:StreamhubAnalytics = StreamhubAnalytics(partnerId: "jwplatform",
endPoint: "http://stats.streamhub.io", playerId: "ios-swift-generic-test-player",
isLive: false, userId: "", analyticsId: "streamhub-5812d")
```

Objectice-C code:

```
StreamhubAnalytics* streamhub = [[StreamhubAnalytics alloc] initWithPartnerId:@"jwplatform"
endPoint:@"http://stats.streamhub.io" playerId:@"ios-swift-generic-test-player" isLive:NO
userId:@"" analyticsId:@"streamhub-5812d"];
```

Parameters details

name	type	comments
partnerId	String	provided to you by your Streamhub integration engineer
endPoint	String	http://stats.streamhub.io is the production endPoint for submitting stats. However, you could choose to use your own mockup server during the plugin development phase.
isLive	Boolean	specify if the video content is either a Live or a VoD program. If the content is Live, then, we usually require that you provide a channel identifier for the programs being

		played by your video player. See the API method **setChannelId** below. For any details, please contact your Streamhub integration engineer.
playerId	String	unique identifier of the player playing the video. It is a unique identifier for the player in both your system and ours. If any clarification is needed, please contact your Streamhub integration engineer.
userId	String	is used if you are able to track your user via a user identifier. If so, you will be able to see user-specific analytics (How old is the public watching the program XYZ ? Which gender ? Which location ? so forth...)
analyticsId	String	The main tracking code that has been provided to you by your Streamhub integration engineer.

Call the API methods

setUserAgent (required)

The plugin being generic for all kind of Apple devices, there is cases where the user agent can't be detected automatically at runtime.
As a result, we ask you to hardcode it, according to your specific device.

Swift:

```
_streamhub?.setUserAgent( "My-Platform-User-Agent" )
```

Objectice-C:

```
[streamhub setUserAgent:@"My-Platform-User-Agent"];
```

setChannelId (required for live only)

If your are playing Live video content, then you probably have EPG metadata relating to it. Usually, in the EPG metadata, a program belong to a channel. We require you to provide the channel identifier associated with the playing program if you have initialized the StreamhubAnalytics library with *isLive: true*.

Swift

```
_streamhub?.setChannelId("a-channel-unique-identifier-mapping-to-some-epg-content")
```

Objective-C

```
[streamhub setChannelId: @"a-channel-unique-identifier-mapping-to-some-epg-content"];
```

setDuration (optional, vod or live with time-shifting)

If your video content has a known duration, then you should provide it to the plugin so that it will be able to report stats with completion rates events. Completion rates will give you insight on how much of your videos are actually watched by your end users.

Swift

```
_streamhub?.setDuration(seconds)
```

Objective-C

```
[streamhub setDuration:68.9f];
```

onMediaReady (required)

onMediaReady must be called to provide the unique identifier of the media about to be played.

Swift

```
_streamhub?.onMediaReady("sara-screen-recording")
```

Objective-C

```
[streamhub onMediaReady:@"sara-screen-recording"];
```

onMediaLoaded

The onMediaLoaded() API signals the beginning of a playback session.
That is the first event your application needs to send and that needs to happen *before* the app is requesting the first video bytes from the network (in order to measure some QoS metrics accurately)

Swift

```
_streamhub?.onMediaLoaded();
```

Objective-C

```
[streamhub onMediaLoaded];
```

onMediaBufferedComplete

The onMediaBufferedComplete(bufferingTime: Int, prebuffering: bool) API signals that a buffering window has been detected.
That API needs to called at the end of the buffering sequence (e.g. when the playback has resumed) and it must be sent with a bufferingTime parameter and a boolean signaling if the buffering has occured before (prebuffering: true) the video's first frame has been played (e.g

before calling the *onMediaStart()* API) or after (prebuffering: false)the playback has started (e.g signaling a rebuffering)
In the case of a rebuffering, the event should be called each times it happens.

Swift

```
_streamhub?.onMediaBufferedComplete(bufferingTime: Int(timeInterval), prebuffering: true)
```

Objective-C

```
[streamhub onMediaBufferedComplete:3 prebuffering:true];
```

addMediaMetadata, onMediaMetadata (required)

The addMediaMetadata() and onMediaMetadata() API function works together so that your app can provide media metadata to StreamhubAnalytics.
First you need to call addMediaMetadata(String key, String value) to provide the metadata field name and field value.
Valid keys are ; `duration`, `title`, `playerTitle`, `categories`.
All values must be provided as strings. You should not care about providing `duration` explicitly since this metadata will be added to the metadata object when you call the setDuration API. However, should you wish to provide that metadata explicitly, you have to provide the `duration` parameter as an **Integer**
The API also takes care of url-encoding the values to use them in HTTP requests.
Finally, the `categories` metadata should be provided as a comma-delimited string of category.

Example:

Swift

```
var duration = 4
var added = _streamhub?.addMediaMetadata( key: "title", value: "SARA Screen Recording" )
added = _streamhub?.addMediaMetadata( key: "duration", value: String(_duration) )
added = _streamhub?.addMediaMetadata( key: "playerTitle", value: "My Beautiful Player" )
added = _streamhub?.addMediaMetadata( key: "categories", value:
"some,meaningful,categories" )
```

The addMediaMetadata method returns a boolean value (true, if the metadata has been correctly added. false otherwise)

Once you have provided all the metadata you wanted, you will need to call the onMediaMetadata() API method to actually send a metadata event to our collector service.

Swift

```
_streamhub?.onMediaMetadata();
```

Objective-C

```
[streamhub onMediaMetadata];
```

onMediaStart (required)

onMediaStart must be called the first time your video content starts to play.

Swift

```
_streamhub?.onMediaStart()
```

Objective-C

```
[streamhub onMediaStart];
```

onTick (required)

onTick is a convenient method that abstract the logic of measuring accurately the video watching time.
the AVFoundation framework provides different way to read the video playback current time (see sample apps).
This is the perfect place to call the *onTick* api method**.**
You should call *onTick* during the whole duration of the video playback.
The unique parameter has to be provided in seconds.

Swift

```
_streamhub?.onTick(seconds)
```

Objective-C

```
[streamhub onTick:1.0f];
[streamhub onTick:2.33f];
[streamhub onTick:3.4f];
[streamhub onTick:5.7f];
```

setPaused (optional)

Call this method when media is paused or unpaused, providing the appropriate boolean value.

Swift

```
_streamhub?.setPaused( true )
```

Objective-C

```
[streamhub setPaused:YES];
```

onMediaComplete (required)

Call *onMediaComplete* to report a video completed event.
The state of the plugin is reset when you call *onMediaComplete*.
As a result, *onMediaReady, onMediaStart, setDuration* should be called again to process the next video if any.

Swift

```
_streamhub?.onMediaComplete()
```

Objective-C

```
[streamhub onMediaComplete];
```

Tracking Ads

If you need to track Ads into our system to get Ad specific reports, we provide 2 utility functions that you can call to report the viewing of ads in your video player and user engagement.

Note: you must call onMediaReady before you can call any of the ad tracking utility function. This is too ensure that the system can compute for which main content you are tracking ads.

trackAd

Call this method during the Ad playback. You need to provide an **adInfo** of type Dictionary<String, String> which contains the identifiers for your ad and the percentage of the ad that has been watched as an Integer value.

adInfo depending on how your Ad metadata are stored, you might need to provide 1 or more identifier to track your ad.
At the minimum, you could provide an id following the Ad-id standard (http://www.ad-id.org/) but, if your ad id is not universal unique identifier, then you could also rely on a couple of other fields to identify your ad like a combination of an Ad id + campaignId + advertiserId.

Below is the list of fields you can provide as key within the adInfo dictonary. Use the ones you depend on.

Key	**Comments**
id	Required, String
campaignId	Optional, String
distributorId	Optional, String
advertiserId	Optional, String
code	Optional, String
customField	Optional, String. Use this field to add any other identifiers not listed above

Percentile how much of your ad has been watched. The standard in the ad industry is to send an event every time 25% of the Ad has been watched.
Possible values are then integer values: 0, 25, 50, 75, 100.

Swift

```
public func trackAd(adInfo: [String: String], percentile: Int)

Examples:

let adInfo:[String: String]  = [ "id": "6128", "advertiserId": "487" ]

... and later...
_streamhub?.trackAd(adInfo, percentile: 0)
```

onAdClick

Call *onAdClick* to report the user clicking on the Ad during playback.
The parameters are the same as the trackAd function.

Swift

```
_streamhub?.onAdClick(adInfo:Dictionary<String, String>, percentile:int )

examples:

let adInfo:[String: String]  = [ "id": "6128", "advertiserId": "487" ]

... and later...
_streamhub?.trackAd(adInfo, percentile: 0)
```