StreamhubAnalytics generic plugin for Android applications

6th of March 2018	2.0	Standardisation of QoS and metadata Frontloading reporting.
2nd of September 2020	2.1	Replaced internal dependency for HTTP networking (Volley → Retrofit)
6th of March 2022	2.2	Updated play-services-ads dependency version

For all kinds of Android applications (mobile, tablet, etc…), we provide a unique plugin built using the Java programming language and the Android SDK.

Download and extract the plugin from the archive
http://streamhub-static-content.s3.amazonaws.com/plugins/genericplugins/player-plugin-android-generic.zip

The plugin is composed of 3 components:

1. **StreamhubAnalytics.java** which is the only component (class) your app should interact with. It contains the API functions to call from your own Android application's code.
2. **NetworkUI.java** which interact with our remote Backend. You don't need to call its method directly.

Getting Started

To get you started quickly, the downloadable archive contains an Android application which integrates our plugin.

You can browse around the code of the MainActivity.java to get an overview of what can and needs to be implemented, which functions to call and in which order.

Integrate the plugin in your own application

The integration process is straightforward, because the complexity of interacting with our Backend REST API has been abstracted in the plugin logic.

Add the dependency for HTTP I/O requests

We will assume you are using Android studio for this document. If another IDE is being used please just adapt the steps.

1. Open ***build.gradle*** in the Module level
2. Add the following to the "dependencies"

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
```

3. Go to the Android Studio menu and click on File > Sync Project with Gradle Files, and sync the Gradle file

Add the dependency to the play-services-ads SDK

The plugin contains some logic to access the advertisingID (known as AAID) of the user devices for compliance with the global advertisement tracking and opt-out standard.
https://support.google.com/googleplay/android-developer/answer/6048248

In the dependencies section of you build.gradle file, you need to add the following line
`compile 'com.google.android.gms:play-services-ads:12.0.1'`

In order to build the plugin.

Add the plugin files

Locate the 2 plugin files StreamhubAnalytics.java and NetworkUI.java in the archive and copy them into your Android project.

Create an instance of the StreamhubAnalytics class

From an Android activity, create an instance of the StreamhubAnalytics class and save it into a class member.

Java

```
private StreamhubAnalytics streamhub;
...
streamhub   = new StreamhubAnalytics(this, "your-platform", "",
"android-generic-test-player", false, "user-id", "streamhub-5812d");
```

Parameters details

name	type	comments
context	Context	The context of current state of the application/object. Read http://stackoverflow.com/questions/3572463/what-is-context-in-android for details.
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 your programs ? 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

onMediaReady (required, must be called first)

onMediaReady must be called to provide the unique identifier of the media about to be played.
This method provide the media unique identifier to the plugin and must be called before any other calls.

Java
```java
_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)

Java
```java
_streamhub.onMediaLoaded();
```

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 string of individual categories separated by a pipe character |.

Example:

Java
```java
_streamhub.addMediaMetadata("duration", "355");
_streamhub.addMediaMetadata("title", "Big Buck Bunny");
_streamhub.addMediaMetadata("playerTitle", "My Player Friendly Name");
_streamhub.addMediaMetadata("categories", "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.

Java
```java
_streamhub.onMediaMetadata()
```

setChannelId (required for live only)

If you are playing Live video content, then you probably have EPG metadata relating to it. Usually, in the EPG metadata, a program belongs 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*.

Java
```java
_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.

Java
```java
_streamhub.setDuration(seconds);
```

onMediaStart (required)

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

Java
```java
_streamhub.onMediaStart();
```

onTick (required)

onTick is a convenient method that abstract the logic of measuring accurately the video watching time.
The sample app creates a thread to read the video playback current time at regular intervals. 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.

Java
```
_streamhub.onTick(seconds);
```

setPaused (optional)

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

Java
```
_streamhub.setPaused( true );
```

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.

Java
```
_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 resolve the ad - main content relationship (for which main video content a specific ad is playing).

trackAd

Call this method during the Ad playback. You need to provide an **adInfo** object which contains the identifiers for your ad and the percentage of the ad that has been watched.

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 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 an identifier within the adInfo object. Use the ones you depend on and leave the others blank.

Property	**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.

Java
```
streamhub.trackAd(adInfo, percentile);

Examples:
HashMap<String, String> adInfo = new HashMap<>();
adInfo.put("id", "6128");
adInfo.put("campaignId", "484");
adInfo.put("advertiserId", "91");

...later
streamhub.trackAd(adInfo, 25);
```

onAdClick

Call *onAdClick* to report the user clicking on the Ad during playback.

The parameters are the same as the trackAd function.

Java
```
streamhub.onAdClick(adInfo, percentile);

Examples:

HashMap<String, String> adInfo = new HashMap<>();
adInfo.put("id", "6128");
adInfo.put("campaignId", "484");
adInfo.put("advertiserId", "91");

...later
streamhub.onAdClick(adInfo, 25);
```

Reporting QoS events

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 time it happens.

Java
… some logic to compute the buffering time
```
float bufferingTime = 1.05f;
streamhub.onMediaBufferedComplete( bufferingTime, false );
```