Creating your own Javascript plugin for the Streamhub Analytics platform

21 Feb 2017	Updates on onMediaLoaded, onMediaBufferedComplete, Ad tracking

Overview

We provide a javascript library that you can use to create your own plugin bridge between a particular type of video player you use and our analytics platform.
The Streamhub Analytics JavaScript library ease that development task by exposing a simple and comprehensible set of predefined JavaScript methods for sending video usage reporting data to the Streamhub platform.

Integration

Load the JavaScript library

First, you need to load the StreamhubAnalytics JavaScript library in the HTML page where you want to add your plugin:

http://static.streamhub.tv/analytics/streamhub-analytics.min.js

Create an instance of the StreamhubAnalytics prototype

The next step is to create a StreamhubAnalytics object.

```
var streamhub = new StreamhubAnalytics(partnerId, endPoint, playerId, isLive, userId, analyticsId);
```

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 methods within the library to track events, ticks and ads

Once a library instance has been created, you can access the API methods.

```
Example: streamhub.onMediaStart();
```

API Reference

As a minimum, we require you to call the onMediaReady, onMediaStart, onTick and onMediaComplete API functions for each viewing of a video.

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*.

Javascript

```
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.

Javascript

```
streamhub.setDuration(seconds);
```

onMediaReady (required)

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

Javascript

```
streamhub.onMediaReady("sara-screen-recording");
```

onMediaLoaded (optional)

onMediaLoaded tracks the player starting to load video bytes, e.g. when the network request is being made but the player's internal buffer is not filled enough to allow playback start. Use the onMediaLoaded api in conjunction with the onMediaBufferedComplete api to track "buffering before video start"

Javascript

```
streamhub.onMediaLoaded();
```

onMediaBufferedComplete (optional)

onMediaBufferedComplete tracks the delays that occurs during playback to refill the player's buffer. Call this api at the end of the buffering sequence, e.g. when the buffer has been refilled enough to allow the playback to restart.
This api method accept 1 parameter: the length of the buffering sequence in seconds.

Javascript

```
streamhub.onMediaBufferedComplete( 2.45 );
```

onMediaStart (required)

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

Javascript

```
streamhub.onMediaStart();
```

onTick (required)

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

Javascript

```
streamhub.onTick(seconds);
```

setPaused (optional)

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

Javascript

```
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.

Javascript

```
streamhub.onMediaComplete()
```

Code sample

Assuming the StreamhubAnalytics library has been loaded into your HTML page :
<script src="https://static.streamhub.tv/analytics/streamhub-analytics.min.js"></script>

```
var partnerId     = "some-partner";
var endPointUrl = "https://stats.streamhub.io";
var playerId     = "some-player-identifier";
var isLive       = true; // whether your player is providing a live service or a onDemand
service
var userId       = undefined; // can be used to track your logged users if your service
requires authentication
var analyticsId   = "your-analytics-id";
var firstplay       = true; // a convenient variable to track the first time the media is
played and call the onMediaStart function accordingly


var shAnalytics = new StreamhubAnalytics( partnerId, endPointUrl, playerId, isLive,
userId, analyticsId );

/**
 * Here we handle a fictional mediaLoaded event callbacks for a fictional video player
accessible via the js variable player
 */
player.on( 'mediaLoaded', function( metadata ){
    shAnalytics.setChannelId( metadata.channelId );
    // if the show is a live-dvr and the duration is known, provide it through the
setDuration to enable completion rates reporting
    shAnalytics.setDuration( metadata.duration );
    shAnalytics.onMediaReady( metadata.videoId );
});

/**
 * Here we handle a fictional playStateChange event callback. We track usage of play and
pause event and call the onMediaStart function if the media is played for the first time.
 */
player.on( 'playStateChange', function( isPlaying ){
    if(isPlaying == true){
    if(firstplay){
        firstplay = false;
        shAnalytics.onMediaStart();
    }
        shAnalytics.setPaused(false);
    }
    else
        shAnalytics.setPaused(true);
});

/**
 * Here we handle the fictional event callback 'timeUpdate' providing the current time in
the video playback
 * make sure currentTimeInSecond has a second resolution (some players API provide this
value in milliseconds)
 * This is the most important part of the integration as it will keep track of the video
views and engagement of the person watching the video.
 */
```

```
player.on( 'timeUpdate', function( currentTimeInSecond /* usually a float value */ ){
    shAnalytics.onTick( currentTimeInSecond );
});

player.on( 'complete', function( event ){
    shAnalytics.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** object which contains the identifiers for your ad and the percentage of the ad that has been watched as an Integer value.

The **adInfo** object should contains the Ad-id which is defined in the VAST specification as **universalAdId**

Property	Comments
id	Required, String

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.

Javascript

```
streamhub.trackAd(adInfo /* Object */, percentile /* integer */ )

examples:
streamhub.trackAd({ id: 'x.y.z' }, 25);
streamhub.trackAd({ id: 'x.y.z', advertiserId: '489' }, 100);
```

onAdClick

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

Javascript

```
streamhub.onAdClick(adInfo /* Object */, percentile /* integer */ )

examples:
streamhub.trackAd({ id: ‘x.y.z’, advertiserId: ‘489’, campaignId: ‘3516’ }, 75);
```