Engineering

Nielsen App SDK Android

Developer's Guide

Revision History

Revision	Date	Change Made	Responsible Engineer
1	04/14/2014	Android App SDK version 1.0	Fabio Milan
2	06/17/2014	Android App SDK version 1.1	Fabio Milan
3	09/29/2014	Android App SDK version 1.2	Suraj Jagga

Contents

1. Introduction 4
2. Concept of Operations 4
2.1. Purpose 4
2.2. Description of the System 4
2.3. How Does it Work? 6
2.4. Data Transfer to the Nielsen Collection Facility 7
3. Quick Start Guide 8
3.1. Setting up your Development Environment 8
3.2. Step 1: Initializing the Nielsen App SDK Object 11
3.3. Step 2: Making Connections to the App SDK Object 13
3.4. Step 3: Nielsen App SDK Streaming Sessions 13
4. Nielsen App SDK On-Boarding Process 15
4.1. Steps to Releasing a Nielsen-enabled Application 15
5. SDK Library and API 16
5.1. Purpose 16
5.2. Overview of the Nielsen App SDK 16
5.3. Variable Name Mapping 17
5.4. Interfaces Provided by the SDK 18
5.4.1. Nielsen App SDK Class Properties 19
5.4.2. Nielsen App SDK Methods 19
6. Constants/Enumerations 29
6.1. App SDK Error Code 30
6.2. App SDK Event Codes 32
7. About Nielsen Measurement and Opt-Out 32
7.1.1. Providing Nielsen Measurement Information to the User 32
7.1.2. Opt-Out of Nielsen Measurement 32
7.1.3. Sample Implementation 33
8. Nielsen Sample Application 34
8.1. Implementing Your App 35
8.1.1. Initializing the App SDK 35
8.1.2. Start Playing 36
8.1.3. Pause Playing 37
8.1.4. Sending ID3 Tags 37
8.1.5. Sending the Position of the Playhead 38
8.1.6. Detecting the App in the Background 39
8.1.7. AppSDK EventListener/Notifier 39
8.2. Android Native Media Player 40
8.2.1. Retrieving ID3 Tags 40
8.3. NexStream Player 40
8.3.1. Start Metering Video 40

8.3.2. Retrieving ID3 Tags 41
8.3.3. Stop Metering Video 41
8.4. VisualOn Player 41
8.4.1. Initializing the Player 42
8.4.2. Start Metering Video 42
8.4.3. Retrieving ID3 Tags 42
8.4.4. Retrieving Playhead Position 43
8.4.5. Stop Metering Video 43
8.5. Brightcove Player 43
8.5.1. Initializing the Player 44
8.5.2. Start Metering Video 44
8.5.3. Retrieving ID3 Tags 44
8.5.4. Retrieving Playhead Position 45
8.5.5. Stop Metering Video 45
8.6. Adobe PrimeTime Player 45
8.6.1. Initializing the Player 45
8.6.2. Start Metering Video 46
8.6.3. Retrieving ID3 Tags 46
8.6.4. Retrieving Playhead Position 46
8.6.5. Stop Metering Video 47
Appendix A – JSON String Formats 48
JSON Schemas for NSString Objects 48
JSON Schema for the AppInfo String 48
JSON Schema for the Program Info String 49
Nielsen IAG Support 50
Nielsen OCR (Online Campaign Ratings) 51
OCR Tags 51
SDK Integration 51
Nielsen DPR (Digital Program Ratings) 53
Program name 53
Client Defined 53
NCIS 53
JSON Schemas for VC / IAG loadMetadata Methods 55
Nielsen mTVR (Mobile TV Ratings) 56
Nielsen DRM (Digital Radio Measurement) 57
Appendix B – Data Collected from SDK 59
Appendix C – channelInfo Free-Form Text Field 61
Setting the channelInfo Field 61
Nielsen App SDK Revision 2.0 channelInfo Definition 61
Glossary 62

List of Figures

Figure 1 – Nielsen App SDK High Level Components 4
Figure 2 – Nielsen App SDK Key Components (not using the Media Player Extension) 6
Figure 3 – Adding the Google Play Services to the Project Properties 10
Figure 4 – Android's Application Life Cycle 12
Figure 5 – Nielsen DPR/DRM Streaming Session Lifecycle 14
Figure 6 – Nielsen On-Boarding Process Timeline 15
Figure 7 – Variable Name Mapping 18

List of Tables

Table 1 – Data Elements 8
Table 2 – Online Campaign Ratings Variables 52
Table 3 – Nielsen Digital Program Ratings Parameters 53
Table 4 – Nielsen Mobile TV Ratings Fields 56
Table 5 – Content Metadata data JSON at play: 57
Table 6 – Nielsen Identifiers Collected 59
Table 7 – App-Supplied Information Collected 59
Table 8 – CMS Metadata Collected 59
Table 9 – OS Info Collected 60
Table 10 – Nielsen Watermark Data Collected 60
Table 11 – channelInfo Freeform Text Field 61

1. Introduction

Nielsen’s App SDK (Software Development Kit) is a framework for measuring linear (live) and on-demand TV and audio viewing using Android mobile devices such as tablets and phones. The Nielsen App SDK (or SDK) leverages Nielsen’s audio watermark technologies for TV audience measurement and the industry supported ID3 metadata tag specification. Nielsen’s App SDK is integrated into MVPD/CDN/CO (Content Originator) client’s player applications so that streaming measurement can be provided on mobile devices.

This document covers the following:

- Nielsen App SDK Concept of Operations
- Nielsen App SDK Quick Start Guide
- Nielsen App SDK Developer On-Boarding Process
- Nielsen App SDK Library & API
- Nielsen App SDK Sample Application
- Nielsen Privacy Requirements

2. Concept of Operations

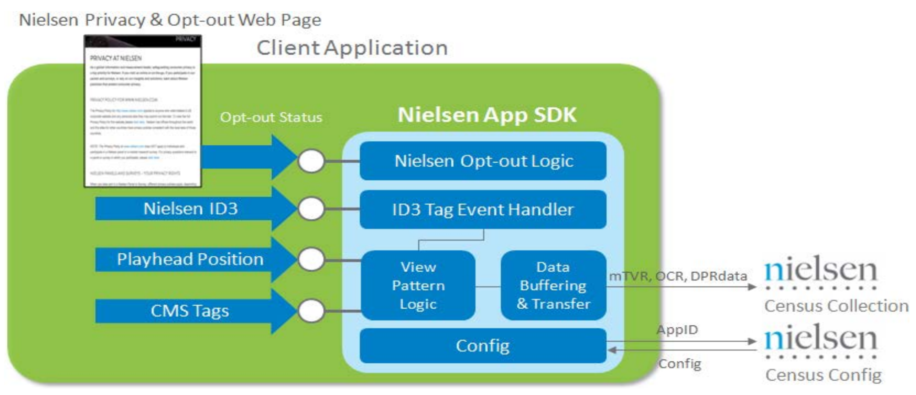
2.1. Purpose

The purpose of this section is to explain the concept of operations behind Nielsen’s App SDK measurement framework and provide integration details of software components found within the SDK.

2.2. Description of the System

Figure 1 shows the high-level components that comprise Nielsen’s App SDK software.

Figure 1 – Nielsen App SDK High Level Components



The Nielsen App framework does the following things:

- Retrieves a configuration file from Nielsen configuration servers based on the Application ID (AppID) assigned to the application.
- Processes ID3 tags, playhead position (DRM or DPR) and CMS tags received during audio or video streaming sessions. The App SDK expects to receive the Nielsen ID3 tags only, for example, the ID3 tags starting with Nielsen's domain (www.nielsen.com). All other non-Nielsen specific ID3 tags should be discarded (for example, 3rd-party ID3 tags).
- Buffers Nielsen ID3 / CMS tags and transfers them to Nielsen's collection facility.
- Supports ID3 / CMS tag caching when users view downloaded content in situations where there is no network connectivity. Cached ID3 / CMS tags are transferred to Nielsen's collection facility when connectivity returns, however, only ID3 tags will not be carried over to the next viewing session.
- Supports user opt-out selection – If the user opts out of measurement, the SDK will notify Nielsen's collection facility that the user has opted out of measurement and will stop sending measurement data to Nielsen.
- App disable/enable – Provides the ability to the client's hosting application to turn the SDK on or off.
- Support generation of location-aware configuration with a client specified precision. Please, note that the App SDK does not track the user's location in a continuous basis, nor provides a very precise location of the user by design.

2.3. How Does it Work?

The client's media player application the Nielsen App SDK integrates is basically of one of these two types:

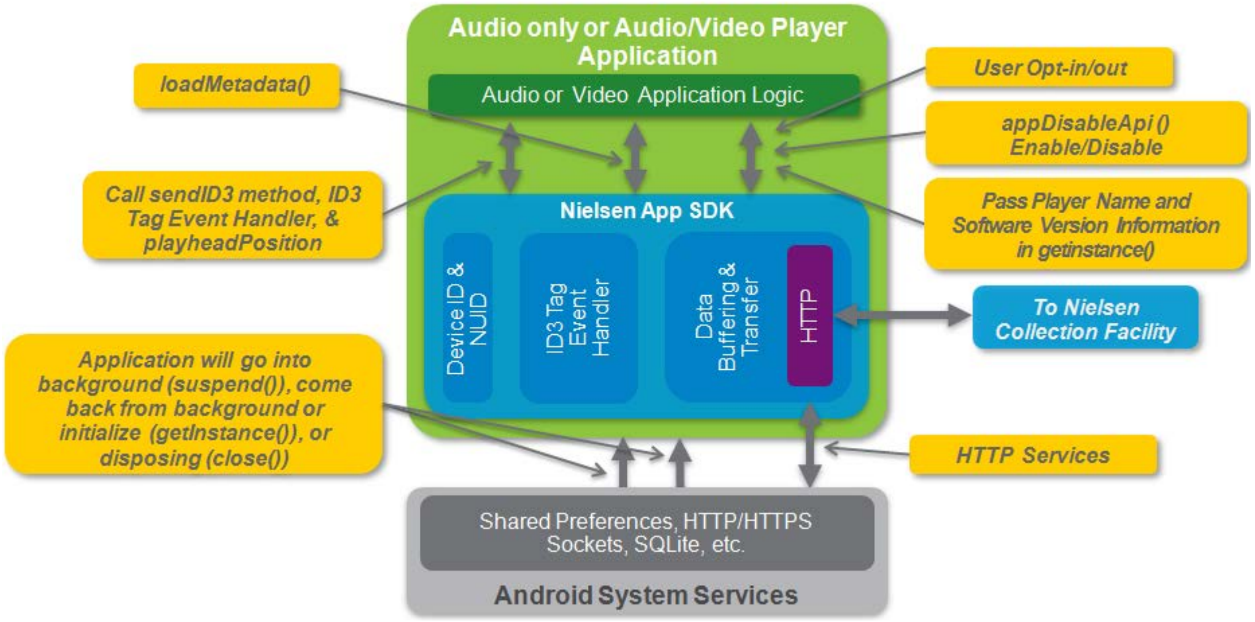
- Video Player Application (VPA) – which normally plays audio and video back to the user
- Audio Player Application (APA) – which normally plays only audio back to the user

From this point on, the client's media player application will be referred as the Application, or App, the terms VPA or APA will only be used when necessary to describe a behavior particular to that type of application.

Once integrated, the Nielsen App SDK execution will flow as follows:

- Configuring the App for Nielsen measurement:

Figure 2 – Nielsen App SDK Key Components (not using the Media Player Extension)


Audio only or Audio/Video Player Application
Audio or Video Application Logic
loadMetadata()
User Opt-in/out
appDisableApi () Enable/Disable
Call sendID3 method, ID3 Tag Event Handler, & playheadPosition
Pass Player Name and Software Version Information in getinstance()
Nielsen App SDK
Device ID & NUID
ID3 Tag Event Handler
Data Buffering & Transfer
HTTP
To Nielsen Collection Facility
Application will go into background (suspend()), come back from background or initialize (getInstance()), or disposing (close())
HTTP Services
Shared Preferences, HTTP/HTTPS Sockets, SQLite, etc.
Android System Services

- Handling playhead position, Nielsen ID3 tags and viewing events:
  - Upon each launch, the App must initialize a Nielsen App SDK object.
  - The SDK component collects general application information and content position (playhead and ID3 tag) to ensure accurate metering. The App must pass the following information to Nielsen's metering framework (a.k.a. App SDK):
    - The name of the App.
    - A unique application ID (AppID) assigned to you by Nielsen for your player app. Nielsen will assign two AppIDs for your application – a test appID and a production appID.

- See the App Supplied Info table in Appendix B for a list of other parameters the client may need to supply when launching the App SDK.

Note	The application must use the test AppID during the development, test, and certification processes. You should only use the production appID when the app has completed the certification process with Nielsen and is ready to submit to the App Store.

- Software revision of the App.
- If the video content (programming or advertising) supports the use of CMS tags, these tags are to be supplied to the SDK using the SDK's loadMetadata method.
- Streaming session start events, referred to as "play" events.
- Playhead position must be sent to SDK at a recommended rate of once every two seconds if you are using SDK for Nielsen's DPR / DRM products.
- Nielsen ID3 tags must be sent to SDK whenever received if you are using SDK for Nielsen's MTVR, DPRID3 or raw ID3 products.
- Streaming session stop events, referred to as "stop" events.

o The minimum Android version supported on an integration to the SDK library will depend on two factors:

- What use the client will make of the App SDK (audio/video or audio only).
- What media player the App will use (native Android Media Player or a 3rd party media player).

On an application, the minimum Android version supported is 2.3 that is the minimum version supporting the Google Services SDK.

2.4. Data Transfer to the Nielsen Collection Facility

The transfer of viewing data to Nielsen's collection facility is done using HTTP sockets and HTTPS secure sockets connections. A complete description of the data transferred to Nielsen's collection facility (including Nielsen ID3 tags, channelName descriptive information including CMS tags for DRM, DPR, VC, IAG, or OCR, viewing patterns, App name, software revision, Nielsen App SDK revision, etc.) can be found in Appendix B.

All data collected and transferred to Nielsen's collection facility is carefully protected to ensure the anonymity of the user and the integrity of the Nielsen sample is maintained.

3. Quick Start Guide

There are 3 steps you must go through in order to deploy a Nielsen-enabled app using the Nielsen App SDK. For more detailed technical information, see Section 5 of this document.

The following data elements must be in place before your application can send data to Nielsen.

Table 1 – Data Elements

Data Item	Description
appid	The `appid` is provided to you by Nielsen's Technical Account Manager. You will receive two appIDs; a test appID and a production appID. The developer must use the test AppID during the development, test, and certification process.
sfcode	`sfcode` specifies which Nielsen collection facility the app should connect to. During development you will be connecting to a development server.
dma	If your app is supplying the `dma`, it should be done after consultation with your Technical Account Manager.
ccode	If your app is supplying the country code, it should be done after consultation with your Technical Account Manager.
CMS Metadata	Unique CMS metadata details must be discussed with your Technical Account Manager prior to launch.

3.1. Setting up your Development Environment

The Nielsen App SDK library is composed of two parts:

- The Java AppSdk.jar library that runs on the Android's Dalvik Virtual Machine.
- The C/C++ libAppSdk.so native library that runs directory on the device's hardware.

The requirement for the Java AppSdk.jar library and the libAppSdk.so native library will depend on the type of the host application that will make use of them.

- For VPA applications:

  The Android OS hosting the App SDK should use a media player supporting HLS streaming (Android 3.0 and later will support it natively).

If the VPA uses a 3rd party media player implementing its own HLS, than the minimum Android version will be limited to version 2.3, since the SDK depends on Google Play support to work properly.

- For APA applications:

  The Android OS hosting the App SDK should be at version 2.3 and later since the SDK depends on the Google Play support to work properly.

Ensure that you unzip the Nielsen App SDK sample app and copy the **AppSdk.jar** into the **libs/** folder on the App's Eclipse project. Also, copy the **libAppSdk.so** file under **libs/armeabi/** folder into the same Eclipse project. AppSDK 1.2 provides support for x86, mips, and armeabi-7a architecture also; the respective **libAppSdk.so** can be found under the **libs/x86/**, **libs/mips/**, and **libs/armeabi-7a/** folders.

On the project's AndroidManifest.xml file, add the following permissions:

```
<uses-permission android:name=
```

"android.permission.ACCESS_COARSE_LOCATION" android:required="false" />

```
<uses-permission android:name=
"android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name=
"android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name=
"android.permission.INTERNET"/>
```

The client must include the Google Play service in order to use the App SDK, which requires to download the **google-play-services_lib** and include it into the App's project.

If the developer does not have the Google Play included in the project, then the library will make reference to missing imports and the app will not compile. However, the Google Play library is only necessary for development. On a live environment the library will normally be there, provided by the OS. The App SDK checks to see if there is a Google service available and updated, if not it will simply not use it when executing its functions.

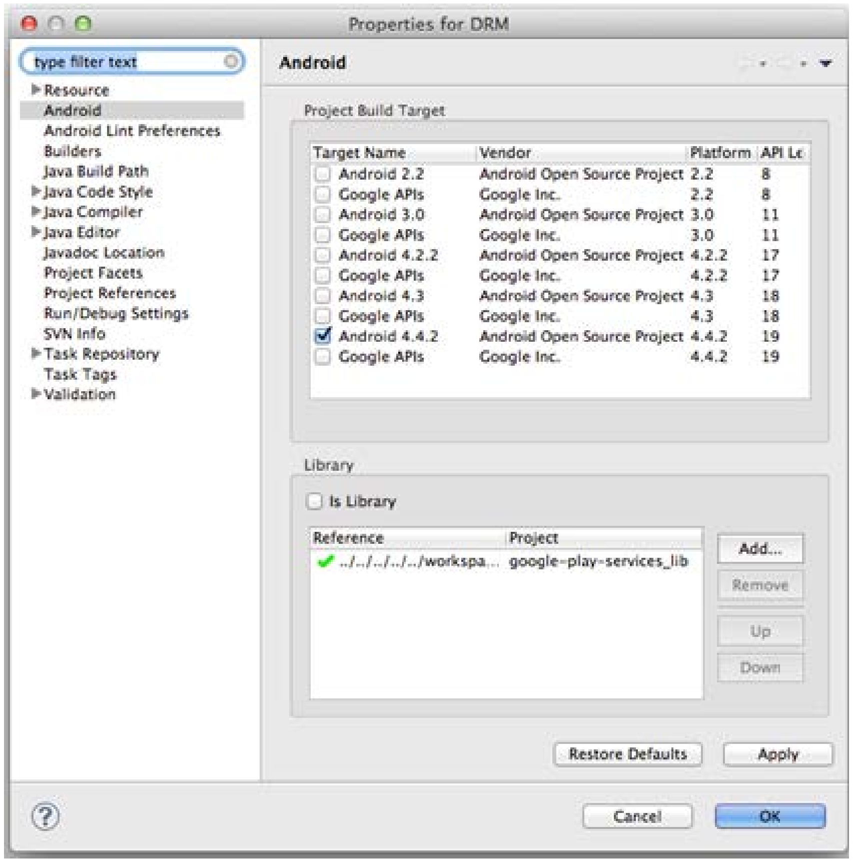
To learn about and download the Google Play library, the client can access:

http://developer.android.com/google/play-services/setup.html

To include the Google Play library in the media player project, the client must copy the **google-play-services_lib** folder into the same location where the client has his/her project.

1. Access File > Import.
2. Select Existing Android Code into Workspace and click Next.
3. Click Browse and navigate to the **google-play-services_lib** to include it into the projects. See Figure 3.

Figure 3 – Adding the Google Play Services to the Project Properties



Target Name	Vendor	Platform	API Le
☐ Android 2.2	Android Open Source Project	2.2	8
☐ Google APIs	Google Inc.	2.2	8
☐ Android 3.0	Android Open Source Project	3.0	11
☐ Google APIs	Google Inc.	3.0	11
☐ Android 4.2.2	Android Open Source Project	4.2.2	17
☐ Google APIs	Google Inc.	4.2.2	17
☐ Android 4.3	Android Open Source Project	4.3	18
☐ Google APIs	Google Inc.	4.3	18
☑ Android 4.4.2	Android Open Source Project	4.4.2	19
☐ Google APIs	Google Inc.	4.4.2	19

☐ Is Library

Reference	Project
✓ ../../../../workspa...	google-play-services_lib

Once the **google-play-services_lib** is included on the App project, the client must include on the AndroidManifest.xml under the <application> node:

```
<meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version"/>
```

Also, the client should include the **version.xml** file that comes with the **google-play-services_lib** under the **res/values** directory on the **res/values** folder of the media player project.

Once the files are in place, import **com.nielsen.app.sdk** to your java source code and start accessing the public interface.

Nielsen AppSDK is uses the following packages/classes from the Google Play service:

- Library: google-play-services_lib
- Classes/package:
  - com.google.android.gms.ads.identifier.AdvertisingIdClient;
  - com.google.android.gms.ads.identifier.AdvertisingIdClient.Info;
  - com.google.android.gms.common.ConnectionResult;
  - com.google.android.gms.common.GooglePlayServicesUtil;

- com.google.android.gms.common.GooglePlayServicesRepairableException;
- com.google.android.gms.common.GooglePlayServicesNotAvailableException;

3.2. Step 1: Initializing the Nielsen App SDK Object

First, you must instantiate a Nielsen App SDK object:

```
String config = "{"+ "\"appName\" : \"" + "AppName" + "\","
                    + "\"appVersion\" : \"" + "1.0" + "\","
                    + "\"sfcode\" : \"" + "uat-cert" +
"\","
                    + "\"appId\" : \"" + "TXXXXXX-XXXX-XXX-
XXXXXXX" + "\""
                    + "}";
AppSdk mAppSdk = AppSdk.getInstance(context, config);
if (!AppSdk.isValid())
{
Log.e(TAG, "Failed in creating the App SDK framework");
return;
}
```

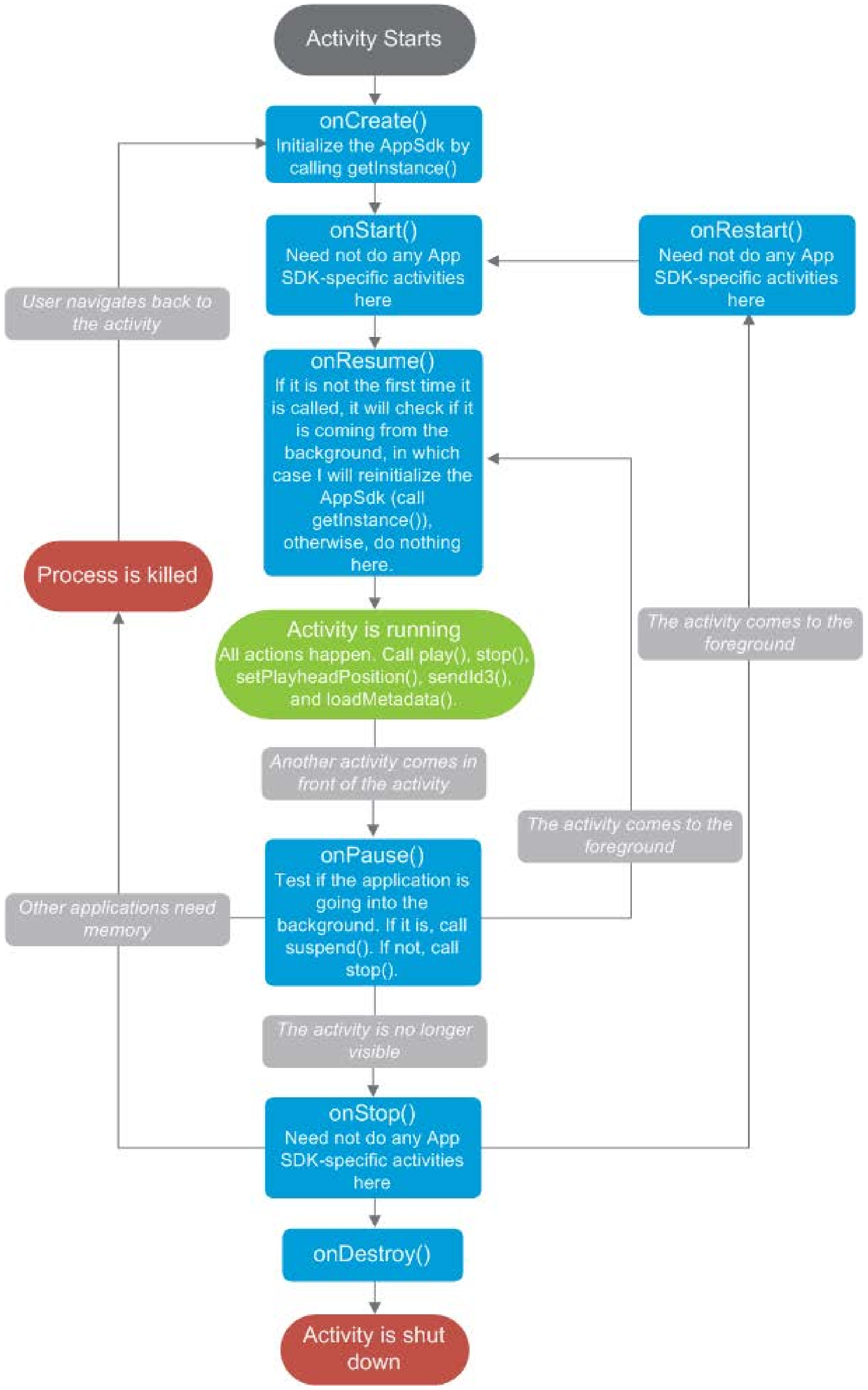
Note

The `appID` is provided to you by Nielsen's Technical Account Manager (TAM) team. The appID is a GUID data type and is specific to your application. The appID has the following type of format: `XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX`. The developer must use the test `appid` during the development, test, and certification process.

Where `context` is the App context object and `config` is a string holding the parameters (`appid, appversion, appname, sfcode`) the App must pass down to the Nielsen App SDK via a JSON string described in Appendix A. The reference value is obtained from Nielsen operational support and is unique to your app.

Figure 4 shows the lifecycle of a typical App SDK enabled application.

Figure 4 – Android's Application Life Cycle


Activity Starts
onCreate()
Initialize the AppSdk by calling getInstance()
onStart()
Need not do any App SDK-specific activities here
onRestart()
Need not do any App SDK-specific activities here
User navigates back to the activity
onResume()
If it is not the first time it is called, it will check if it is coming from the background, in which case I will reinitialize the AppSdk (call getInstance()), otherwise, do nothing here.
Process is killed
Activity is running
All actions happen. Call play(), stop(), setPlayheadPosition(), sendId3(), and loadMetadata().
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
onPause()
Test if the application is going into the background. If it is, call suspend(). If not, call stop().
Other applications need memory
The activity is no longer visible
onStop()
Need not do any App SDK-specific activities here
onDestroy()
Activity is shut down

3.3. Step 2: Making Connections to the App SDK Object

The integration of the Nielsen App SDK will depend on the client’s type of App. The client must ensure that the SDK files (AppSdk.jar and libAppSdk.so) are included under the App’s project and the App SDK is connected to the App (the setting to connect the App SDK to the App can be found on the App’s project’s property page).

3.4. Step 3: Nielsen App SDK Streaming Sessions

After ensuring that the SDK object has been initialized, the client must connect the streaming session APIs.

The next steps are:

1. Call `play()` App SDK method when starting or resuming a streaming session. See Appendix A for more information about JSON string formats.
2. Loading the CMS metadata by calling the loadMetadata() method on the SDK object. See Appendix A for more information about JSON string formats.
3. Calling the stop() SDK method when ending or pausing a viewing session.
4. During session playback, call the SDK `setPlayheadPosition()`and/or `sendID3()`methods:
    a. `setPlayheadPosition()`should be called once every two seconds until the stream is stopped or paused. Normally this happens on DPR and DRM measurements.
    b. `sendID3()`should be called if the client relies on the Nielsen ID3 tags for its measurements; this call should happen whenever a new Nielsen ID3 metadata is available for processing. Normally this happens on mTVR and DPRID3 measurements.

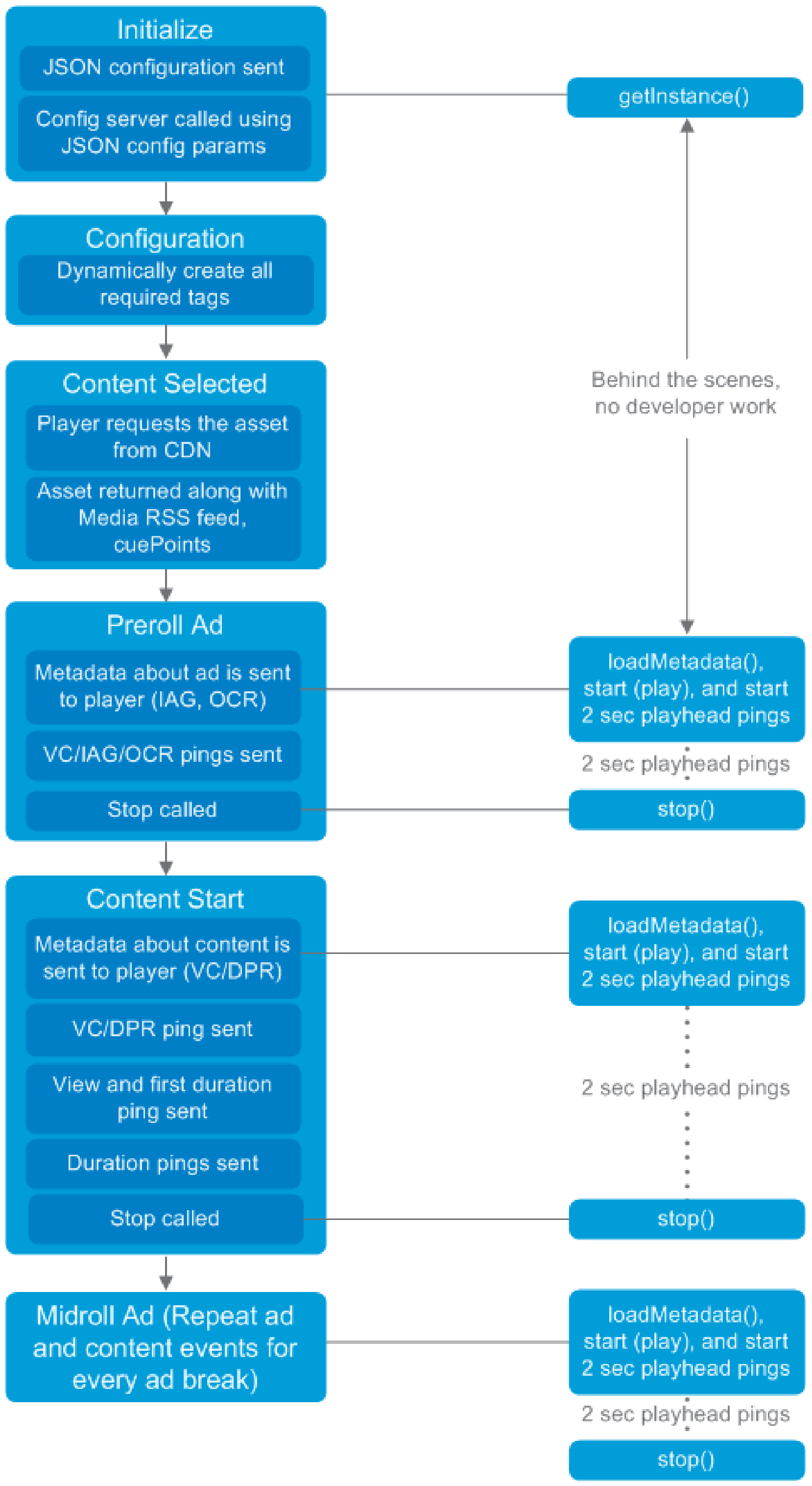
Note

The client’s app must clearly identify the mode of operation (live vs. VOD) and stick to the type of play head coordinates until the playback is completed. The client must reliably provide the appropriated playhead position value depending on the type of content streamed:

If streaming live content, the client must pass the current UTC time in seconds.

If streaming VOD (video on demand), the client must stream the offset from the beginning of the file.

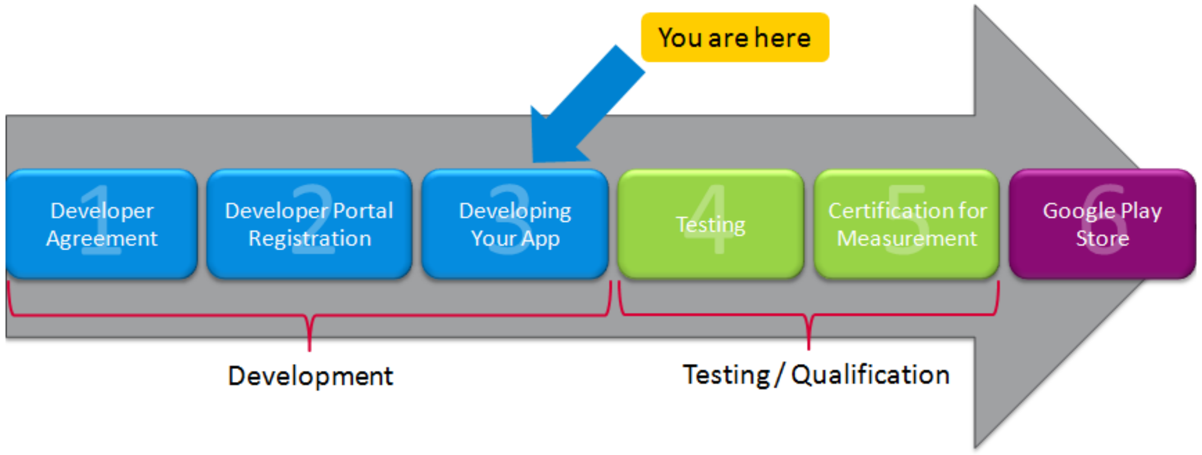
Figure 5 – Nielsen DPR/DRM Streaming Session Lifecycle


Initialize
JSON configuration sent
Config server called using JSON config params
getInstance()
Configuration
Dynamically create all required tags
Content Selected
Player requests the asset from CDN
Asset returned along with Media RSS feed, cuePoints
Behind the scenes, no developer work
Preroll Ad
Metadata about ad is sent to player (IAG, OCR)
VC/IAG/OCR pings sent
Stop called
loadMetadata(), start (play), and start 2 sec playhead pings
2 sec playhead pings
stop()
Content Start
Metadata about content is sent to player (VC/DPR)
VC/DPR ping sent
View and first duration ping sent
Duration pings sent
Stop called
loadMetadata(), start (play), and start 2 sec playhead pings
2 sec playhead pings
stop()
Midroll Ad (Repeat ad and content events for every ad break)
loadMetadata(), start (play), and start 2 sec playhead pings
2 sec playhead pings
stop()

4. Nielsen App SDK On-Boarding Process

4.1. Steps to Releasing a Nielsen-enabled Application

Figure 6 – Nielsen On-Boarding Process Timeline



The following steps are necessary to deploy Nielsen-enabled applications:

1. Client signs the Nielsen App SDK Evaluation Agreement and sets up a developer account via Nielsen's Mobile Measurement Developer Registration Portal.
2. Client fills out form on Registration Portal and submits it to Nielsen.
3. Nielsen's Mobile Measurement team verifies that legal paperwork is in order and provides FTP account information (username & password) to you for downloading the Nielsen App SDK. The Mobile Measurement team also provides Nielsen-assigned appids to you for use in your Android player application.

Note Clients only need a single agreement and single download of the Nielsen App SDK. Each application that integrates the Nielsen App SDK, however, will need to have its own set of Nielsen assigned `appIDs`; a Test `appID` and Production `appID`.

4. Client develops the player app with the App SDK and sets the `appID`, the application name (`appName`), and assigns a software version (`appVersion`) to the app. Client uses the test appID during development, test, and certification.
5. Once the application is certified and CMS variable mapping for ID3, mTVR, DRM, DPRID3, DPR, VC, IAG, and OCR data has been confirmed by your Nielsen Technical Account Manager, your app is ready for the measurement. Change the `appID` to the production `appID` and prepare to submit your application to the App Store for distribution.
6. Submit player application to the Google Play Store.

5. SDK Library and API

5.1. Purpose

This section of the Nielsen App SDK Developer's guide describes the API used to interface to Nielsen's App SDK. This framework provides Nielsen measurement services to client's streaming applications. By integrating the SDK into the clients' apps, the clients will be able to collect streaming media consumption data for Nielsen mobile device measurement.

5.2. Overview of the Nielsen App SDK

The Nielsen App SDK allows the calling application to pass data that Nielsen uses to accurately measure audience viewing. Additionally, VPA applications must also pass ID3 tags and application information, such as App's name, version, and `appid`. The Nielsen App SDK object automatically relays this information to Nielsen's collection facility.

The Nielsen App SDK compatibility to the Android OS platform will depend on the use the client makes of the SDK and the media player used.

Since the SDK uses the Google Play services, the minimum Android version supported is 2.3. The App SDK can also work on systems that do not support Google Play services (for example, Amazon Kindle).

The following is an example of how the Nielsen App SDK is used:

1. Instantiate the SDK object within the Activity- or Service-derived classes on the Android application framework using the `AppSdk.getInstance()` method. To the `AppSdk.getInstance()`method is passed the `appname`, `appid`, `appversion`, and `sfcode` values as arguments via the `appInfo` JSON string schema.
2. When starting or resuming a streaming session, call the `play()`method and use the `channelInfo` parameter to pass channel descriptor information. The channel name field is a 32-character free-form text field containing the name of the program or feed being sent (such as ESPN2, Food Network, Breaking Bad, etc.) which must be inserted on a JSON string. See Section 5.4.3 and Appendix C for more information. There can be other parameters passed, depending on how the SDK is used.
3. If you are using a CMS tagging-based Nielsen solution like DPR or OCR, use the `loadMetadata()` method at the start of each asset being played to pass necessary metadata information to the SDK.
4. During a timed metadata event, use the `sendID3()` method to pass the data object to the Nielsen SDK object for processing.
5. As long as there is a session play back, call `playheadPosition()` once every two seconds to pass the content view position to the SDK object.

6. When ending or pausing each streaming session, channel changing, or stopping playing for any reason, call the `stop()` method. The app must notify the SDK object via `stop()` and `play()` method calls if the player is playing back or not.
7. When exiting the application, call the stop() method (if there is a content playback) followed by `close()` method to completely release and close all resources taken by the app.

5.3. Variable Name Mapping

Several SDK methods take JSON formatted strings as an input parameter. This means that the data is formatted as key-value pairs: the key being the variable name and the value being the string the key is set to. For example, a JSON string for setting variables "title" and "program length" to values MyEpisodeTitle and 3600 respectively would be as follows:

```
{\"title\" : \"MyEpisodeTitle\", \"length\" : \"3600\"}";
```

Now, consider a case where the player application was implemented using different variable names due to different naming conventions used within your CMS system. As an example, the player app might have implemented the example above as:

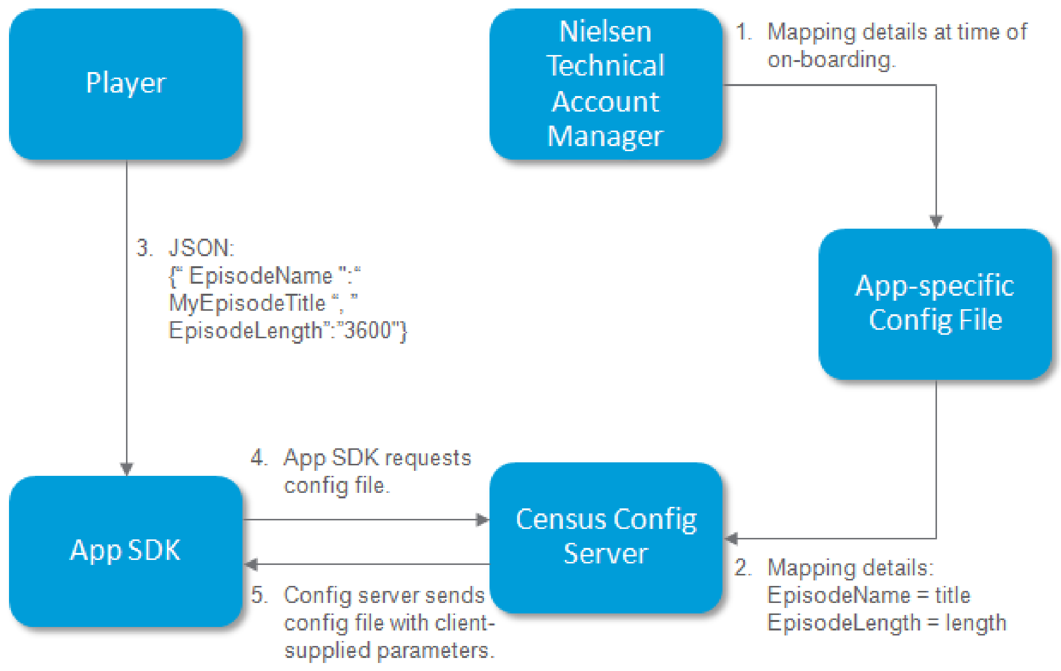
```
{\"EpisodeName\" : \"MyEpisodeTitle\", \"EpisodeLength\" :
\"3600\"}";
```

It is possible to support this option without modifying application code by using Variable Name Mapping. The SDK has functionality for supporting different variable names, but the mapping information must be provided to Nielsen through your Technical Account Manager (TAM).

The new variable name mapping will be updated in Nielsen's configuration server and passed to the SDK via an updated config file. Once changes are made, the SDK will know that "EpisodeName" is synonymous with "title" and "EpisodeLength" actually is the same as "length".

Variable name mapping gives flexibility to the client side implementation. It is recommended, however, that you use the Nielsen-defined names whenever possible to simplify long-term maintainability.

Figure 7 – Variable Name Mapping


Player
Nielsen Technical Account Manager
1. Mapping details at time of on-boarding.
App-specific Config File
3. JSON: {" EpisodeName ":" MyEpisodeTitle ", " EpisodeLength":"3600"}
App SDK
4. App SDK requests config file.
Census Config Server
5. Config server sends config file with client-supplied parameters.
2. Mapping details: EpisodeName = title EpisodeLength = length

1. Mapping details provided to Nielsen Technical Account Manager.
2. Config server handles the URL separately for each application based on initial parameter mapping.
3. Player App uses custom names in the JSON strings with whatever key values agreed upon during Nielsen's on-boarding process.
4. SDK requests config file upon initialization.
5. The config file is customized for each app.

For more information, contact your Nielsen Technical Account Manager.

5.4. Interfaces Provided by the SDK

This section provides a general description of the Nielsen App SDK public class and interface.

The Nielsen App SDK (located on the com.nielsen.app.sdk package) class is the primary application interface to the Nielsen App SDK on Android. For example, after an instance of the AppSdk class is created and initialized, it can be used to collect the HLS timed metadata exposed by the media player currently in use by calling the SDK's `sendID3()` method.

5.4.1. Nielsen App SDK Class Properties

The Nielsen App SDK class is defined as the only public class belonging to the com.nielsen.app.sdk package. It inherits from the closeable interface and exposes the public methods the client's app will use. Below is the declaration of the AppSdk class:

```
public class AppSdk implements Closeable
```

Optionally, you can also implement an object derived from the interface:

```
public interface IAppNotifier
{
public void onAppSdkEvent(long timestamp, int code, String
description);
}
```

This object will allow you to listen for events happening inside the App SDK. These are the two events currently exposed on the App SDKL

```
public static final int EVENT_STARTUP = 2000;
```

The App SDK has just started up. It will happen only after the App SDK has received a valid config file from Nielsen.

```
public static final int EVENT_SHUTDOWN = 2001;
```

The App SDK is shutting down. It will happen only when the App SDK is destroyed.

All configuration and operational settings are made through methods calls described in the section below.

5.4.2. Nielsen App SDK Methods

Overview

The Nielsen App SDK methods provide the interface to the Nielsen Android device metering framework. By calling the getInstance() method, the client will be returned a reference to the only instance possible for the App SDK. Any call to that method made at a later moment will deliver a reference to the same object.

The App SDK class is implemented as a singleton object that allows you to enable and disable the meter, start and stop viewing sessions, and access status events or error information from the metering framework.

Tasks

Initializing and destroying the Nielsen App SDK Object

```
public static AppSdk getInstance(Context context,
 String appInfo);
public AppSdk suspend();
public static boolean isValid();
public void close() throws IOException;
```

Configuring the Nielsen App SDK Object

```
public AppSdk appDisableApi(boolean disabled);
public AppSdk userOptOut(String optOut);
public String userOptOutURLString();
```

Working with a Nielsen App SDK Object

```
public AppSdk setPlayheadPosition(long position);
public AppSdk loadMetadata(String jsonMetadata);
public AppSdk play(String channelInfo);
public AppSdk sendID3(String payload);
public AppSdk stop();
```

Accessing Nielsen App SDK Log Information

```
public static String getLastEvent();
public static String getLastError();
```

Collecting Information About the App SDK Version and Device IDs

```
public static String getMeterVersion();
public String getNielsenId();
public String getDeviceId();
```

Static Methods

public static String getLastEvent()

Queries the Nielsen AppSdk framework for the last event occurred.

Parameters

None

Return Value

Returns an JSON string object with the following event information:

Key	Value
eventCode	Integer
eventDescription	String
Timestamp	Timestamp

`eventCode`

An integer which specifies the Nielsen AppSdk event code and when it occurred. See Section 6 for more information on the `AppSdk EventCode` enumeration types.

`eventDescription`

A String object that describes the actual event.

`timestamp`

An timestamp object containing a time stamp at which the last event occurred.

Discussion

The `getLastEvent` method allows the player application to query the Nielsen App SDK for details on the last event that occurred.

Declared In

`AppSdk.jar`

public static String getLastError()

Queries the App SDK framework for the last error that occurred.

Parameters

None

Return Value

Returns an JSON string describing the last error occurred information:

Key	Value
lastErrorOccurred	JSON string
Timestamp	Event Timestamp

`lastErrorOccurred`

A JSON string object is returned that provides information on the error domain, the error code, and details on the specific error. If the error was caused by throwing an exception, it will print stack information describing the file, line and reason of the exception. It also has an Nielsen App SDK error code enumeration type, see Section 6 for more information.

`timestamp`

An object containing the time stamp when the last event occurred.

Discussion

The `getLastError` method allows the player application to query the Nielsen App SDK class for details on the last error that occurred.

Declared In

`AppSdk.jar`

public static String getMeterVersion()

Returns version of Nielsen App SDK framework being used.

Parameters

None

Return Value

Returns a string object containing the App SDK version. This allows the developer to display version information such as, “Nielsen AppSdk version aa.1.0.2.10”, in the About section of the application’s settings view.

Discussion

This API call returns the current version of the Nielsen App SDK.

Declared In

`AppSdk.jar`

public static boolean isValid()

Returns if the App SDK was successfully instantiated or there was a problem during instantiation. It can also be called to test the error condition on any other AppSdk non-static public method.

Parameters

None

Return Value

It will return true if the App SDK object instantiation was successful, otherwise false.

Discussion

This API tests if the App SDK object is in a valid state or not.

Declared In

`AppSdk.jar`

public static AppSdk getInstance(Context context, String appInfo)

Obtains the only instance of App SDK metering library. If it is call two or more times, it will always return the reference of the object created on the first call.

Parameters

`context`

The context from the App

`appInfo`

The parameter `appInfo` is a string object that defines the application information in JSON string format. The required items are application name (`appName`), application ID (`appId`), application version (`appVersion`), designated market area (`dma`), Nielsen-assigned data node (`sfcode`), and the country code (`ccode`). The following is an example of a properly formed appInfo string (note that all quotes inside the string are escaped with a backslash):

```
”{\"appName\":\"PlayerApp\",\"appId\":\"F1CFBFE7-C1BB-4B86-
87BD-46DB0D0A3604\",\"appVersion\":\"2.0.0.4
\",\"sfcode\":\"us\"}”
```

Return Value

Initializes and returns a reference to the only instance allowed of the App SDK. Returns a null object if the initialization failed. The application should check if the App SDK was correctly started by calling the static method AppSdk.isValid().

Discussion

This method is the initialization method for the App SDK metering library. All parameters in the `getInstance()`method are mandatory and should not be NULL values.

The reference to the App SDK object returned by the `getInstance()` is a singleton object, i.e., there can be no other instance of the same type of object in memory. The method should never return null. As mentioned above, the Nielsen's App SDK utilizes functionality associated with Google Play Services, which mandates that the Android OS should be at least version 2.3 for audio only applications. For audio and video applications other limiting factors take place.

Declared In

`AppSdk.jar`

Instance Methods

public String getNielsenId()

Returns string defining the Nielsen ID (NUID) number assigned to the current Android device.

Parameters

None

Return Value

The current NUID number for the device.

Discussion

This API gets the unique NUID number identifying the Android device.

Declared In

`AppSdk.jar`

public String getDeviceId()

Returns string defining the Device ID number assigned to the current Android device.

Parameters

None

Return Value

The current DeviceId number for the device.

Discussion

This API gets the unique number identifying the Android device, which will be the Google Ad-Id, if there is a Google Play Service installed and updated, or the ANDROID_ID, if there is no Google Play installed and updated.

Declared In

`AppSdk.jar`

public AppSdk setNotifier(IAppNotifier notifier)

Parameters

Client provided object implementing the IAppNotifier interface.

Return Value

None

Discussion

Sets up the reference to the notification object from client implementing actions to take whenever the App SDK events are fired. It can be changed whenever needed or passed as null, if the client does not want to treat any of the events fired.

Declared In

`AppSdk.jar`

public AppSdk loadMetadata(String jsonMetadata)

The app uses this method to send CMS metadata to the SDK. This is done by constructing a JSON formatted string of key-value pairs and calling the `loadMetadata()` method.

Parameters

`jsonMetadata`

The metadata string should contain at least the video type "`type`" and the asset id "`assetid`". If type is marked as "content" the whole JSON dictionary gets saved as new CMS metadata inside the SDK. DPR or OCR content can be flagged with the "tv" and "ocrflag" respectively. For instance, the following is an example of a jsonMetadata parameter string:

```
"{\"tv\" : \"true\",\"category\" :
\"MyProgramCategory\",\"assetid\":\"MyContentAssetId\",
\"title\" : \" MyEpisodeTitle \", \"length\" : \"6000\",
\"type\":\"content\", \"dataSrc\":\"id3\",
\"adModel\":\"1\"}"
```

For more information, see Appendix A.

Return Value

A null reference if the method failed, otherwise it returns a reference to the only AppSdk object

Declared In

`AppSdk.jar`

public AppSdk setPlayheadPosition(long position)

For situations where you are using CMS data in conjunction with Nielsen's DPR or DRM products. For proper reporting, the SDK should be receive a playhead position update every 2 seconds on the playback of content. It should not receive playhead updates when the content playback is paused or stopped. When streaming VOD (Video On Demand), the position is the current location of the player from the beginning of the asset in seconds; when streaming live content, the position passed to the AppSdk is the current UTC time in seconds.

Parameters

position

An Integer value representing the time in content in seconds (VOD – Video On Demand) or the current UTC time in seconds (live).

Return Value

A null reference if the method failed, otherwise it returns a reference to the only App SDK object.

Discussion

When playing back content, the player application must send the value of the playhead position to the SDK once every 2 seconds. Likewise, the playhead position should not be updated when there is no content being played back OR if the playback is paused. The client app can schedule a 2-second-period timer to send the playhead position in the following manner:

```
class PlayerPositionTracker extends Thread implements
Runnable
{
   public static final int POSITON_INTERVAL = 2000;

   @Override
   public void run()
   {
      while (true)
      {
         long t = 0;
         if (isPlaying)
         {
            if (isLive) // Live – dependent on media player
            {
               t = (c.getTimeInMillis()/ 1000;
               mAppSdk.setPlayheadPosition(t);
            }
            else // VOD
            {
               t = mMediaPlayer.getCurrentPosition()
                   / 1000;

               mAppSdk.setPlayheadPosition(t);
            }
         }
         Thread.sleep(POSITON_INTERVAL);
      }
   }
}
```

Declared In

`AppSdk.jar`

public AppSdk sendID3(String payload)

This method will consume data from the timed metadata events (ID3 tags) provided by the video player.

This method will not filter out non-Nielsen-specific ID3 tags, it will assume that the data received are ID3 tags; it will perform only a quick validation of the data delivered and cache all valid data for later transfer to Nielsen’s collection facility. The SDK will process only the Nielsen ID3 tags, and ignore non Nielsen ID3 tags.

Parameters

`payload`

A string of 249 characters in length containing the Nielsen ID3 tag starting with the "www.nielsen.com" string.

Return Value

A null reference if the method failed, otherwise it returns a reference to the only AppSdk object.

Discussion

During the HLS timed metadata event, only the PRIV frame's owner ID of the payload (or a copy of the payload) from the media player shall be passed into this method. This method ignores any owner ID that does not pertain to Nielsen. Since ID3 tags are continuously streamed, every timed metadata event must be captured, stored, and transferred for accuracy of metering.

Note It is not necessary for the player application to enable/disable the Nielsen App SDK component depending on whether Nielsen ID3 tags are present in the stream. This is handled automatically by the SDK.

Declared In

`AppSdk.jar`

public AppSdk appDisableApi (Boolean-disabled)

Enable or disable meter functions.

Parameters

`disabled`

A Boolean value set by the app developer to enable or disable Nielsen measurement.

Return Value

A null reference if the method failed, otherwise it returns a reference to the only AppSdk object.

Discussion

To disable meter, the `appDisableApi()` method should be called with the value set to TRUE.

It is not necessary for the App to call the appDisableAPI method to enable/disable Nielsen metering. We recommend that you only call this method when you want to completely disable measurement.

Declared In

`AppSdk.jar`

public AppSdk play (String channelInfo)

The AppSdk is started by calling the `play()` method with the `channelInfo` parameter. Normally this method is called when the user presses the play button on the player.

Parameters

`channelInfo`

The `channelInfo` string objects contains the channel name (channelName) defined in JSON string format. For example:

```
"{\"channelName\":\"TheMovieTitle\"}"
```

Return Value

A null reference if the method failed, otherwise it returns a reference to the only AppSdk object.

Discussion

This method must be called at the start of every streaming session. The maximum length of this string is 32 characters. For more information, see Appendix B.

Note | The text format of the `channelInfo` string description is free-form.

Declared In

`AppSdk.jar`

public AppSdk stop()

Records a channel `stop` event.

Parameters

None

Return Value

A null reference if the method failed, otherwise it returns a reference to the only AppSdk object.

Discussion

Indicates the end of a streaming session. This method is to be called when the user either stops or suspends the streaming content or switches to a different HLS stream, in which case, the app should call the `play()` method when starting the new stream.

Declared In

`AppSdk.jar`

public String userOptOutURLString()

Queries the metering framework for the Nielsen opt-out URL web page.

Parameters

None

Return Value

Returns a string object with the URL of Nielsen opt-out page.

Discussion

Get URL of web page that is used to give the user a chance to opt out from the Nielsen measurement.

Declared In

`AppSdk.jar`

public AppSdk userOptOut (String optOut)

Enable or disable the App for Nielsen measurement.

Parameters

`optOut`

String value received back from the Nielsen Opt-out web page. This value is set by the web page and the application is responsible for passing this value back to the SDK without modification.

Return Value

A null reference if the method failed, the command response or URL was not intended for the Nielsen SDK. Retrieve the next URL if requested. Otherwise this command/URL was handled by Nielsen SDK and returns a reference to the only App SDK object.

Note The value of the `userOptOut` state is persisted across application launches on the shared preferences object local to the App SDK.

Discussion

This method is to be called when the user has selected to Opt-in or Opt-out of Nielsen measurement.

Declared In

`AppSdk.jar`

6. Constants/Enumerations

The following are the Nielsen App SDK constants identifying the errors and events on the Android side:

6.1. App SDK Error Code

Constants with predefined error codes which the AppSdk object can generate:

```
public class AppSdk implements Closeable
{
      // failed generating ping string due to error on
parsing
// description – include last error message from URL parser
      public static final int
ERROR_FAILED_CREATE_URL_STRING = 1;

      // failed to receive configuration file from Census
// description – on 5th time, it will log event and keep
// requesting config 10 min apart
      public static final int ERROR_FAILED_RECEIVE_CONFIG =
2;

// failed parsing the config file JSON string
      // description - include json error number/short
message
// from iOS or Android
      public static final int ERROR_FAILED_PARSING_CONFIG =
3;

// failed parsing the play() JSON string
// description - include json error number/short message
// from iOS or Android
      public static final int ERROR_FAILED_PARSING_PLAY =
4;

// failed parsing the play() JSON string
// description - include JSON error number/short message
// from iOS or Android
      public static final int ERROR_FAILED_PARSING_METADATA
= 5;

// failed creating ping before adding it to the UPLOAD
table)
// description - include ping nol_url index, cadence to id
ping
      public static final int ERROR_FAILED_GENERATING_PING
= 6;

// failed starting processor thread. Normally, it means a
product
// description - include processor that failed to start
```

```
      public static final int ERROR_FAILED_PROCESSOR_START
= 7;

// failed processing data on a data processor
// normally, it means the input to a product
      // description - include processor and data that
failed to
// process (ID3 tag on a MTVR impression, for example)
      public static final int ERROR_FAILED_PROCESS_ID3 = 8;

// failed sending HTTP or HTTPS requests
// description - include HTTP error number
      public static final int ERROR_FAILED_HTTP_SEND = 9;

// failed sending pings (on ANDROID, pings on the UPLOAD
table)
      // description - include ping up to 80 char from the
end
      public static final int ERROR_FAILED_SENDING_PING =
10;

// failed sending TSV requests
      // description - include TSV request message
      public static final int ERROR_FAILED_SENDING_TSV =
11;

// failed sending StationId requests
// description - include Station ID request message
      public static final int
ERROR_FAILED_SENDING_STATION_ID = 12;

// failed read/write from/to database table
// description - with SQL statement, data, SQLite error
message
      public static final int ERROR_FAILED_ACCESSING_DB =
13;

// device ID changed
// description - none
public static final int ERROR_CHANGED_DEVICE_ID = 14;

// NUID changed
// description - none
      public static final int ERROR_CHANGED_NUID = 15;

// any other exception or error
      // description - class.method name + shor exceotion
description
// (up to 80 characters in total)
```

```
                public static final int ERROR_EXCEPTION = 16;
    }
```

6.2. App SDK Event Codes

These are the constants predefining the event codes which the App SDK object can generate:

```
    public class AppSdk implements Closeable
    {
          /**
           * all possible event codes
           */

    // App SDK has started up. It happens after App SDK has received a valid config
    public static final int EVENT_CODE_STARTUP = 2000;
    // App SDK is shutting down. It will happen only when App SDK is destroyed
    public static final int EVENT_CODE_SHUTDOWN = 2001;
}
```

7. About Nielsen Measurement and Opt-Out

7.1.1. Providing Nielsen Measurement Information to the User

In accordance with Nielsen’s SDK licensing agreement, developers are required to provide basic informational data to users about Nielsen’s Privacy Policy, and where additional information on Nielsen measurement can be obtained.

In the App store DESCRIPTION, include a short description about Nielsen measurement.

Refer to the Nielsen Privacy Requirements document for more information.

7.1.2. Opt-Out of Nielsen Measurement

The application must provide a user interface for the user to opt-out from Nielsen measurement.

In the App Privacy Policy settings, a link or button named “About Nielsen Measurement” should be provided. When the user clicks on the link/button, the App makes a call to the SDK, and the SDK returns the privacy/opt-out URL. This URL must open in a web view from within the App, not from within Safari or Chrome, for user to select opt-out or opt-in.

The opt-out happens by opening a Nielsen-defined web page and passing the user choice from the web view commands. In order to do this, the application needs to implement the following web view where the Nielsen Privacy web page will be displayed to the user, capture the user selection, and pass it back to the SDK via the userOptOut method.

When the web view is closed, the status returned from the web view must be passed to the SDK within the app. The SDK will manage the user’s choice (opt-out/opt-in) so this status is not required or needed to be managed by the app.

7.1.3. Sample Implementation

1. Create a web view with the Nielsen opt-out URL

```
if(optOutUrl){
optOutUrl = mAppSdk.userOptOutURLString();
mWebView = (WebView) findViewById(R.id.webView);
mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.getSettings().setBuiltInZoomControls(true);
mWebView.getSettings().setDisplayZoomControls(false);
mWebView.getSettings().setLoadWithOverviewMode(true);
mWebView.getSettings().setUseWideViewPort(true);
mWebView.getSettings().setLayoutAlgorithm(LayoutAlgorithm.S
INGLE_COLUMN);
mWebView.setWebViewClient(new MonitorWebView());
mWebView.setWebChromeClient(new WebChromeClient());
mWebView.loadUrl(optOutUrl);
} else{
//Handle it gracefully and Retry later
}
```

2. Capture and forward user selection

```
private class MonitorWebView extends WebViewClient
{
      public void onPageFinished(WebView view, String url)
      {
            cancelDialog();
      };
      public void cancelDialog()
      {
            if (dialog != null)
            {
                  dialog.cancel();
                  dialog = null;
            }
      }
      @Override
public boolean shouldOverrideUrlLoading(WebView view,
String url)
      {
            if (url.indexOf("nielsen") == 0)
            {
                  mAppSdk.userOptOut(url);
                  finish();
                  return false;
            }
            else
            {
                  dialog =
ProgressDialog.show(OptOutActivity.this,
                  "OptOut", "Loading...");
                  return true;
            }
      }
}
```

The web view should display the call the Nielsen opt-out URL method every time the user wishes to change its opt-out state.

Refer to the Nielsen Privacy Requirements document for more information.

8. Nielsen Sample Application

The Android Nielsen App SDK includes a fully-documented sample application to demonstrate the use of all of the features included within the SDK.

The Nielsen Android sample player application consists of a sample video player (Android native media player or a third party media player) integrated with the SDK framework. The player demonstrates all the supported functions of the SDK.

The Android sample players will all share the same UI interface, with the following functions / components:

- APA:
  - Android App SDK + Native Android Media player
- VPA:
  - Android App SDK + Native Android Media Player
  - Android App SDK + Native Android Media Player + eXtension
  - Android App SDK + NexStream media player
  - Android App SDK + VisualOn media player
  - Android App SDK + Brightcove media player
  - Android App SDK + AdobePrimeTime media player

The UI components of the Android App SDK sample applications are common to all. See below:

- From the Channel Selection buttons, the user will be able select the channels to stream or add new ones.

- The Info button displays more detailed information of the SDK version, current Nielsen ID used for the device, and the option for opt-out/opt-in.
- The Play and Pause buttons will control the streaming of the selected channel.
- The area at the bottom of the window displays the current ID3 tags and stream status.
- The Clear button clears out the status window.
- The Email button can be used to email the tags and status in a text file to Nielsen.

While importing sample application's code in IDE, it is recommended to add latest android support library android-support-vX.jar in the sample application project.

Note The sample video player app (MPX) uses the android native player and the Nielsen-developed custom utility to extract the ID3 tags. Clients should use the sample app only for reference purpose as the custom utility is not production ready, and should not be used with the client's app.

8.1. Implementing Your App

Nielsen created multiple apps based on the players available at the time. One of the fundamental elements of the design was to create an interface/protocol such that using the same UI that could handle different players with their intrinsic differences and implementations. Basic methods such as play, pause, and change channel were created to ease the multi-player implementation. At the same time, the App SDK was integrated to the associated behaviors of the player, these being play and pause.

As follows, these are the methods to have a UI, App SDK, and integrated player. Furthermore, there are other functionalities that are also integrated into the app in order to meet the App SDK requirements, such as the position of the playhead, optionally ID3 tags, and detection of the app being in background.

This section explains the App SDK interfacing with various media players: Android Media Player, NexStream, and VisualOn.

In summary, the app has to initialize the App SDK and the player, monitor and notify the App SDK of the current activity of the video, and do the necessary housekeeping once the app is terminated or goes into background.

8.1.1. Initializing the App SDK

The App SDK has to be initialized with information pertaining to your application. This information is supplied to it in the form of a JSON string followed by the initialization invocation as described below:

```
private void appIntializeAppSdk()
{
String config = "{"
            + "\"appName\"      : \"" + appName + "\","
            + "\"appVersion\"   : \"" + appVersion + "\","
            + "\"appId\"        : \"" + appId + "\""
            + "}";
mAppSdk = AppSdk.getInstance(this, config);

if (!AppSdk.isValid()) {
            Log.e(TAG, "Failed creating App SDK instance");
            return;
}
}
```

In the above snippet, `mAppSdk` is the App SDK interface to the client application. All the interactions between the player and the App SDK take place through this object.

The config parameter passed to the constructor of the App SDK is a JSON string that contains information regarding the host application.

`getInstance()` should be called when the application is being initialized (preferably in `onCreate()` or `onResume()`).

Once the method is executed successfully, the App SDK object is initialized, the next step is to notify the station it is playing, the position of playhead, and optionally, the ID3 payload.

8.1.2. Start Playing

To start playing the App SDK, two steps are necessary. First, send the name of the channel by calling `play()`, then send the metadata associated with the stream being played back by calling `loadMetadata()`. See the following example for these two steps:

```
private void appStartMeteringVideo()
{
String tag = "{ \"channelName\" : \"" +
currentMovie.getName() + "\" }";
mAppSdk.play(tag);
mAppSdk.loadMetadata(metaData);
}
```

In the above method:

`metadata` is the JSON string includes metadata of either the content or the ad. The available metadata should come from the video/audio or audio-only server or ad manager, when the content is selected and while content is playing.

A sample JSON string for a MTVR application:

```
metadata = "{\"type\" : \"content\","
+ "\"assetId\"  : \"" + movie.getName() + "\", "
+ "\"tv\"       : \"true\","

+ "\"title\"    : \"MyEpisode\","
+ "\"category\" : \"MyProgram\",";

if (!dataSrc.equals("") && !adModel.equals(""))
```

```
        {
              metadata += "\"dataSrc\" : \"" + dataSrc +
              "\"," + "\"adModel\"       : \"" + adModel +
              "\",";
        }

        metaData += "\"length\":\"6000\"}";
```

A sample JSON string for a DRM application:

```
        String metadata = "{\"type\" : \"radio\", "
                    + "\"assetid\"         : \"" +
        movie.getName() + "\", "
                    + "\"stationType\"     : \"3\",";

        if (!dataSrc.equals("") && !adModel.equals(""))
              metadata += "\"dataSrc\" : \"" + dataSrc +
              "\"," + "\"adModel\"       : \"" + adModel +
              "\",";

        metaData +=  "\"provider\":\"Clear Channel\" }";
```

`tag` is the JSON string holding channel information (the stream that you want to analyze using the App SDK). This can be the URL of the channel or the name of a station. The following is a sample:

```
        { "channelName": " currentMovie " }
```

The method appStartMeteringVideo() is invoked to start video streaming.

8.1.3. Pause Playing

Pausing and stopping metering the media are treated in the same way. Call this method:

```
  private void appStopMeteringVideo()
  {
        mAppSdk.stop();
  }
```

8.1.4. Sending ID3 Tags

ID3 tags are handled differently between players. Below is the suggested method to have the player interfaced to the App SDK:

```
  public void appProcessID3tag(String id3String)
  {
        mAppSdk.sendID3(id3String);
        uiAppendLog(id3String);
  }
```

8.1.5. Sending the Position of the Playhead

This is timer/thread-based method that based on the type of video (Live/HLS or VOD/Video On Demand) feeds information of the playhead position as described in the following method.

Please note that the following method calls are not player-dependent:

- mPlayer.isPlaying(),
- mPlayer.videoDuration()
- mPlayer.videoPosition()

It is part of the player implementation to supply this information via this "generic method".

```
private void appMonitorPlayHead()
{
   if (monitorHeadTimer != null)
      return;

   monitorHeadTimer = new Timer();
   TimerTask monitorHeadPos = new TimerTask()
   {
      @Override
      public void run()
      {
         runOnUiThread(new Runnable()
         {
           @Override
           public void run()
           {
            if ((mPlayer != null) && (mPlayer.isPlaying()))
            {
               videoDuration  = mPlayer.videoDuration();
               videoPostion   = mPlayer.videoPosition();

               // streaming or there is no duration
               if (videoDuration <= 0)
               {
                  // UTC time in seconds
                  long time = c.getTimeInMillis() / 1000;

                  // if is streaming, there is no duration
                  mAppSdk.setPlayheadPosition(time);
               }
               else // it is playing back a local file
               {
                  // for FILE (not live) PLAYBACK, PLAYHEAD
                  // position must be number of second from
                  // beginning of content
                  int sec = videoPostion / 1000;

                  // seconds since PLAYBACK begin
```

```
                    mAppSdk.setPlayheadPosition(sec);
                 }
                 uiAppendLog(String.format("pos: %d",
                    videoPostion));
              }
            }
         });
      }
   };
   monitorHeadTimer.scheduleAtFixedRate(monitorHeadPos, 0,
      2000);
}
```

8.1.6. Detecting the App in the Background

Normally, the user does not need to notify the App SDK when the host application is going to or is coming back from the background. The App SDK will automatically handle it by refreshing itself every 24hrs and requesting a new configuration file from Nielsen.

If the user needs to destroy the App SDK for any reason, it should call suspend() that disposes it. If the user needs to bring back the App SDK, it should call getInstance() to recreate a new App SDK instance.

This is a very important function of the app in order for the App SDK to be able to perform a number of internal activities. Android, contrary to other mobile platforms, does not have a mechanism to truly notify that the app was sent to background. The onPause and onResume are callbacks at the activity level, not at the application level.

Refer to the sample application for a reference implementation.

8.1.7. AppSDK EventListener/Notifier

Client apps can subscribe to the AppSDK event listener to get notified if the AppSDK is ready. This notification helps client apps to call the AppSDK APIs only when it is ready to process the request.

To set up the notification, clients have to implement interface:

```
com.nielsen.app.sdk.IAppNotifier
```

//subscribe to the eventlistener:

```
mAppSdk.setNotifier(this)
```

The AppSDK will notify apps via following api:

```
onAppSdkEvent(long timestamp, int code, String description)
```

where code = EVENT_STARTUP which notifies when the AppSDK is ready.

8.2. Android Native Media Player

The DRM-MP and mTVR-MPX applications are based on the Native Media Player of Android. The most important difference between the two is that DRM is a service-based app where the App SDK as well as the class "AppService" can continue operating while the player is playing, in this case radio.

On the other hand, mTVR-MPX is a video player app; therefore, once the app goes in background, the video is paused.

The class "AppService" has the core to interface with the player, the App SDK and the UI. However, the UI has access to AppService through exposed methods and vice versa in order to complement each other.

To control the player please refer to the Android documentation at: http://developer.android.com/reference/android/media/MediaPlayer.html

8.2.1. Retrieving ID3 Tags

Because the Android media player does not support ID3, Nielsen has created a library that becomes an extension to the media player, thus MPX. This library extracts the ID3 tags and sends them to the app. For more information on how to use the MPX component, please refer to the Nielsen-supplied sample application.

8.3. NexStream Player

This section is intended to inform the client on how to integrate the App SDK to their applications when using the NexStream as their media player.

The mTVR-NXS application is based on the Android NexStream Media Player.

This section focuses on the details of the NexStream integration into the client's application. It is assumed here that the client knows how the NexStream media player works and how to interface it with the App SDK.

Note	For more details on the NexStream media player capabilities, please refer to NexStream user guide and technical support NexStream provides.

8.3.1. Start Metering Video

After the App SDK is initialized, it can be used during media playback. On the sample application, the method `appStartMeteringVideo()` is invoked to start video streaming:

```
mNexPlayer.open(mCurrentChannel, null, null,
NexPlayer.NEXPLAYER_SOURCE_TYPE_STREAMING,
NexPlayer.NEXPLAYER_TRANSPORT_TYPE_TCP, 0)
```

Where mNexPlayer is the NexStream Player object and mCurrentChannel is the stream that is being opened.

appStartMeteringVideo() can be called before this or inside onAsyncCmdComplete(NexPlayer mp, int command, int result, int param1, int param2) callback, depending on whether the mNexPlayer.open(...) operation was success or failure. If declared inside onAsyncCmdComplete(...), appStartMeteringVideo() should be called immediately before mNexPlayer.start(seektime).

8.3.2. Retrieving ID3 Tags

ID3 tags will be received in the NexStream Player

onTimedMetaRenderRender(NexPlayer mp,NexID3TagInformation metadata) callback method.

A sample implementation for the above callback is shown below:

```
public void onTimedMetaRenderRender(NexPlayer mp,
    NexID3TagInformation m)
{
    text = m.getPrivateFrame();
    if (text != null)
    {
       data = text.getTextData();
       if (data != null)
       {
         // make sure to identify the beginning  of  the
         // Nielsen ID3 tag payload by searching for the
         // “www.nielsen.com” string on the ID3 tag and
         // passing to the App SDK all information that
         // follows. It should be:
         // nlsPayload = “www.nilesen.com“ + dataFollowing
         nlsPayload = getDataAfterWwwNielsenCom(data);
         if (nlsPayload!= NULL)
            mAppSdk.sendID3(nlsPayload);
       }
    }
}
```

The mAppSdk.sendID3() sends the extracted Nielsen ID3 payload to the App SDK for analysis.

8.3.3. Stop Metering Video

When stopping/pausing the player, the client should stop sending the video metering data to the App SDK as well by calling the stop() App SDK API.

8.4. VisualOn Player

This section informs the clients on how to integrate the App SDK to their applications, when using the VisualOn as their media player.

The mTVR-VON application is based on the Android VisualOn Media Player. There are many similarities between the VisualOn sample application implementation and the mTVR-MPX sample application; those similarities are not mentioned here. This section concentrates on the differences of the VisualOn implementation.

It is assumed here that the client knows how the VisualOn media player works and how to interface it with the App SDK.

Note | For more details on the VisualOn media player capabilities, please refer to VisualOn user guide and technical support VisualOn provides.

8.4.1. Initializing the Player

Initialize the player as shown below:

```
VOCommonPlayerImpl m_sdkPlayer = new VOCommonPlayerImpl();
```

Implement the `VOCommonPlayerListener` to listen to the various client/player interactions using callback handlers.

Once the VisualOn player has been initialized, the client should initialize the App SDK in the same fashion as described previously by calling main activity's method appIntializeAppSdk().

8.4.2. Start Metering Video

Once the player is initialized and video content is playing, use the player's playVideo() API to capture the information from VisualOn and pass it along to the App SDK.

Before playVideo(), the client should initialize the App SDK to start metering and then pass the information from the content playback into the App SDK.

The Start Metering process is done as described previously by calling the appStartMeteringVideo().

8.4.3. Retrieving ID3 Tags

While the VisualOn player plays the content, VOCommonPlayerListener triggers an event when an ID3 packet is received (VO_OSMP_SRC_CUSTOMERTAGID_TIMEDTAG).

Upon receiving this event, the client should collect the incoming ID3 tags and only pass the Nielsen payload of the PRIV frame to the App SDK for analysis.

The sample code is presented below:

```
case VO_OSMP_SRC_CB_CUSTOMER_TAG:
{
    VO_OSMP_SRC_CUSTOMERTAGID tag =
VO_OSMP_SRC_CUSTOMERTAGID.valueOf(nParam1);
    switch (tag)
    {
      case VO_OSMP_SRC_CUSTOMERTAGID_TIMEDTAG:
```

```
                // do something with this tag
                int time = nParam2;
                byte[] b = (byte[]) obj;
                String s = new String(b);
                NlsId3Tag nlsID3 = new NlsId3Tag(b);

                // Sent ID3 Tags to App
                appProcessID3tag(nlsID3.NlsPayload);
                if (appId3If != null)
                   appId3If.onId3(nlsID3.NlsPayload);
                break;

             case VO_OSMP_SRC_CUSTOMERTAGID_MAX:
                // ignore this type of tag
                break;

             default:
                break;
          }
          break;
       }
```

8.4.4. Retrieving Playhead Position

As soon as the VisualOn player plays the content, VOCommonPlayerListener triggers an event VO_OSMP_SRC_CB_OPEN_FINISHED.

Upon receiving this event, the client should start collecting the playhead position every two seconds from the player object and pass it to App SDK as below:

```
mAppSdk.setPlayheadPosition(sec);
```

The value of **sec** should be UTC in seconds for Live content or play position for VOD content. You can use getDuration() as an indicator for Live or VOD. When getDuration() returns a value greater than 0, when using VO_OSMP_SRC_CB_OPEN_FINISHED, the play position should be used throughout this content, otherwise, use UTC.

8.4.5. Stop Metering Video

When stopping/pausing the player using player’s **stopVideo()** and **pauseVideo() APIs, t**he client should stop sending the video metering data to the App SDK as well by calling the stop() App SDK API.

8.5. Brightcove Player

This section informs the clients on how to integrate the App SDK to their applications when using the Brightcove as their media player.

The mTVR-BRC application is based on the Android Brightcove Media Player. There are many similarities between the Brightcove sample application implementation and the mTVR-MPX sample application; those similarities are not mentioned here. This section concentrates on the differences of the Brightcove implementation.

It is assumed here that the client knows how the Brightcove media player works and how to interface it with the App SDK.

Note | For more details on the Brightcove media player capabilities, please refer to Brightcove user guide and technical support Brightcove provides.

8.5.1. Initializing the Player

Initialize the player as shown below:

```
BrightcoveVideoView brightcoveVideoView = (SeamlessVideoView)
findViewById(R.id.brightcove_video_view);
```

Once the BrightcoveVideoView player has been initialized, the client should initialize the App SDK in the same fashion as described previously by calling main activity's method applntializeAppSdk().

8.5.2. Start Metering Video

Once the player is initialized and video content is playing, use the player's playVideo() API to capture the information from Brightcove and pass it along to the App SDK.

Before playVideo(), the client should initialize the App SDK to start metering and then pass the information from the content playback into the App SDK.

The Start Metering process is done as described previously by calling the appStartMeteringVideo().

8.5.3. Retrieving ID3 Tags

While the Brightcove player plays the content, EventListener triggers an event when an ID3 packet is received (SeamlessVideoDisplayComponent.ID3_TAG).

Upon receiving this event, the client should collect the incoming ID3 tags and only pass the Nielsen payload of the PRIV frame to the App SDK for analysis.

The sample code is presented below:

```
brightcoveVideoView.getEventEmitter().on(SeamlessVideoDispl
ayComponent.ID3_TAG, new EventListener()
{

    public void processEvent(Event event)
    {
      NlsId3Tag nlsID3 = new
      NlsId3Tag(event.properties.get(SeamlessVideoDisplayCo
      mponent.ID3_DATA).toString());
      Log.w("ID3", nlsID3.NlsPayload);
      // Sent ID3 Tags to App
      appProcessID3tag(nlsID3.NlsPayload);
      }
   });
}
```

8.5.4. Retrieving Playhead Position

As soon as the Brightcove player plays the content, the client should start collecting the playhead position every two seconds from the player object brightcoveVideoView.getCurrentPosition() and pass it to App SDK as below:

```
mAppSdk.setPlayheadPosition(sec);
```

The value of **sec** should be UTC in seconds for Live content or play position for VOD content. You can use getDuration() as an indicator for Live or VOD. When getDuration() returns a value greater than 0, the play position should be used throughout this content, otherwise, use UTC.

8.5.5. Stop Metering Video

When stopping/pausing the player using player’s **stopVideo()** and **pauseVideo() APIs, t**he client should stop sending the video metering data to the App SDK as well by calling the stop() App SDK API.

8.6. Adobe PrimeTime Player

This section informs the clients on how to integrate the App SDK to their applications, when using the Adobe PrimeTime as their media player.

The mTVR-APT application is based on the Adobe PrimeTime Media Player. There are many similarities between the Adobe PrimeTime sample application implementation and the mTVR-MPX sample application; those similarities are not mentioned here. This section concentrates on the differences of the Adobe PrimeTime implementation.

It is assumed here that the client knows how the Adobe PrimeTime media player works and how to interface it with the App SDK.

Note	For more details on the Adobe PrimeTime media player capabilities, please refer to Adobe PrimeTime user guide and technical support Adobe PrimeTime provides.

8.6.1. Initializing the Player

Initialize the player as shown below:

```
MediaPlyaer mediaPlayer =
DefaultMediaPlayer.create(getApplicationContext());
```

Once the Adobe PrimeTime MediaPlayer has been initialized, the client should initialize the App SDK in the same fashion as described previously by calling main activity’s method appIntializeAppSdk().

8.6.2. Start Metering Video

Once the player is initialized and video content is playing, use the player’s playVideo() API to capture the information from Adobe PrimeTime and pass it along to the App SDK.

Before playVideo(), the client should initialize the App SDK to start metering and then pass the information from the content playback into the App SDK.

The Start Metering process is done as described previously by calling the appStartMeteringVideo().

8.6.3. Retrieving ID3 Tags

While the Adobe PrimeTime player plays the content, MediaPlayer.PlaybackEventListener triggers an callback when an ID3 packet is received (onTimedMetadata).

Upon receiving this callback, the client should collect the incoming ID3 tags and only pass the Nielsen payload of the PRIV frame to the App SDK for analysis.

The sample code is presented below:

```
    public void onTimedMetadata(TimedMetadata id3Metadata) {

      NlsId3Tag nlsID3 = new
NlsId3Tag(id3Metadata.getMetadata().toString());
      Log.w(LOG_TAG, "ID3 Timed Data --> " +
nlsID3.NlsPayload);
      Log.w(LOG_TAG, "PayLoad Size --> " +
nlsID3.NlsPayload.length());

      appProcessID3tag(nlsID3.NlsPayload);
    }
```

8.6.4. Retrieving Playhead Position

As soon as the Adobe PrimeTime player plays the content, the client should start collecting the playhead position every two seconds from the player object mediaPlayer.getPlaybackMetrics().getTime() / 1000 and pass it to App SDK as below:

```
mAppSdk.setPlayheadPosition(sec);
```

The value of **sec** should be UTC in seconds for Live content or play position for VOD content. You can use getDuration() as an indicator for Live or VOD. When getDuration() returns a value greater than 0, the play position should be used throughout this content, otherwise, use UTC.

8.6.5. Stop Metering Video

When stopping/pausing the player using player’s **stopVideo()** and **pauseVideo() APIs, t**he client should stop sending the video metering data to the App SDK as well by calling the stop() App SDK API.

Appendix A – JSON String Formats

JSON Schemas for NSString Objects

The following details provide the JSON schema used in the getInstance(), loadMetadata() and play() methods. See Section 5.3 for information on variable name re-mapping if your company uses different CMS naming conventions to refer to these fields.

JSON Schema for the AppInfo String

```
{
   "title": "App Information",
   "type": "object",
   "properties": {
         "appid": {
         "title": "Unique AppID assigned by Nielsen",
         "type": "string"
         },
         "appversion": {
         "title": "Version of App",
         "type": "string"
         },
         "appname": {
         "title": "Name of App",
         "type": "string"
         },
         "sfcode": {
"title": "Nielsen assigned Collection Facility",
         "type": "string"
         },
"dma": {
         "title": " Nielsen Designated Market Area code",
         "type": "string"
         },
         "ccode": {
         "title": "Country code",
         "type": "string"
         },

   },
   "required": ["appid", "appversion", "appname", "sfcode"]
}
```

Note DMA and Country Code are optional parameters. DMA codes supplied should be standard Nielsen-recognized DMA codes. If an unrecognized `dma` value is used, it will be overwritten by Nielsen specified values.

Note sfcode identifies the Nielsen collection facility to which the SDK should connect. The default `sfcode` is "US". Please consult your Technical Account Manager before using DMA and Country Code to ensure these parameters are applicable for your application.

JSON Schema for the Program Info String

```
{
      "title": "Program Information",
      "type": "object",
      "properties": {
            "assetid": {
            "title": "ID of asset from CMS",
            "type": "string"
            },
            "clientid": {
"title": "Unique Client ID assigned by Nielsen",
            "type": "string"
            },
            "vcid": {
            "title": "VC ID assigned by Nielsen",
            "type": "string"
            },
            "title": {
"title": "Descriptive title value for reporting purposes.
Use URL if title not available",
            "type": "string"
            },
            "length": {
            "title": "Episode Length",
            "type": "integer",
            "minimum": 0
            },
            "type": {
            "title": "Type of content played",
"enum" : [“content”, “preroll”, “midroll”, “postroll”,
“ad”]
            "type": "string"
            },
            "category": {
            "title": "Category of the Episode",
            "type": "string"
            },
            "censuscategory": {
```

```
"title": "Enables client defined reporting in Video
Census",
            "type": "string",
            },
            "ocrtag": {
            "title": "OCR tags"
            "type": "string"
            },
            "tv": {
"title": "Is DPR enabled for this content – true, false"
            "type": "boolean"
            },
            "prod": {
"title": "Type of product. For IAG, set to iag. If both
VideoCensus and IAG are used set the prod value to vc,iag”
            "type": "string",
            },
            "pd": {
"title": "IAG Partner Distribution - Call sign or short
abbreviation of partner"
            "type": "string"
            },
            "tfid": {
"title": "IAG Tag Format ID supplied by Nielsen IAG"
            "type": "string"
            },
            "sid": {
"title": "IAG Source ID supplied by Nielsen IAG"
            "type": "string"
            },
      },
}
```

For more information, consult your Technical Account Manager.

Note

Census Category

`<censuscategory>` specifies whether a play should be counted and surfaced under “client-defined category” in VideoCensus syndicated reporting (the “cg” parameter). Often, this refers to a show (for example, The Simpsons). The <category> value in Video Analytics serves the same purpose. You can specify either parameter or both. For more information, consult your Technical Account Manager.

Nielsen IAG Support

Nielsen’s App SDK can automatically generate parameters needed for Nielsen’s IAG service. Additional IAG parameters can be accepted by the SDK. IAG parameters can also be configured in the Nielsen configuration servers. For more information on IAG, consult your Technical Account Manager.

Nielsen OCR (Online Campaign Ratings)

OCR tags are used to track commercial delivery for campaign ratings. OCR tags are received during the streaming session from the Content Management System (CMS) identifying that the video contains ad content. This information should be passed as a JSON string via the `loadMetadata:` method with OCR contents.

OCR Tags

The Nielsen OCR tag, or Nielsen OCR beacon, may come in different forms from the ad service via VAST XML or the ad service integrated player framework library.

SDK Integration

After getting the string from an ad service, to integrate Nielsen SDK for OCR tag, the app should first identify a Nielsen beacon in the use cases below:

1. If the input string is CDATA string, get the value of the CDATA as a new input string.
2. If the input string (or new input string) starts with http and the hostname ends with "imrworldwide.com", the input string is a Nielsen beacon (case 1 and case 3).
3. If the input string (or new input string) starts with http and the CR field exists:
   a. Get the value of the CR field.
   b. If the CR value starts with http and the hostname of the CR value ends with "imrworldwide.com", the CR value is a Nielsen beacon (case 2 and case 4).
   c. If the Nielsen beacon exists, remove the value of the CR field from the input string and process the updated string for the non-OCR beacon.
4. If the Nielsen beacon exists:
   a. URL decode the Nielsen beacon.
   b. If the c13 field exists in the Nielsen beacon, remove the c13 from the Nielsen beacon.
   c. Create json with the Nielsen beacon:

      Json:

      ```
      {
              "type":"ad",
              "ocrtag":"<Nielsen beacon>"
      }
      ```

      Sample json string with the OCR string (decoded http string without the c13):

      ```
          {
                  "type":"ad",
                  "ocrtag":"http://secure-
      us.imrworldwide.com/cgi-
      bin/m?ci=ent30986&am=22&ep=1&at=view&rt=banner&st=image&ca=
      cmp97144&cr=1186239&pc=3739659&r=2011370876"
      ```

```
        }
```

a. Send the json to the Nielsen SDK using loadMetadata.

5. If the Nielsen beacon does not exist, process the input string for the non-OCR beacon.

Table 2 – Online Campaign Ratings Variables

Usage: Online Campaign		
Variable Name	**Variable Description**	**Example**
type	Type identifies the tag content type. For OCR, the data type should be always set to "ad".	ad
ocrTag	The complete tag/URL is supplied by your Technical Account Manager. This should include the complete URL including the http portion. The ocrtag should be properly URI encoded.	http://secure- uat-cert.imrworldwide.com/cgi-bin/m?ci= ENTXX5&am=3&ep=1&at=view&rt=banner&st= image&ca=XX1717&cr=crvXX35&pc=plc1234

Example

```
metaDatainformation = {
    "type" : "ad", "ocrtag" : "OCR URL received from TAM"
};
```

Description

The ocrtags received from your CMS system should be transformed into a JSON string and passed in via the `loadMetadata` method.

JSON

- type – Type identifies tag content type. For OCR, data type should be always set to "ad".
- ocrtag – The complete tag/URL is supplied by your Technical Account Manager. This should include the complete URL including the http portion. The ocrtag should be properly URI encoded.

Nielsen DPR (Digital Program Ratings)

Nielsen's DPR product requires the following data parameters:

Table 3 – Nielsen Digital Program Ratings Parameters

Usage: TV App			
Variable Name	**Variable Description**	**SDK Generated/Client Defined Value**	**Example**
dataSrc**	Source of the data. For DPR datasrc should be specified as "cms".	Client Defined	cms
type	Type of play event , preroll, midroll, postroll.	Client Defined	Content
assetid	Unique ID of content	Client Defined	vid-123
tv	TV-enabled flag should be set as true if the content relates to any program that has been aired on TV. Note: "True" for demographics data. "False" for non-demographic data Default value: False	Client Defined	true
program/ category	Program name	Client Defined	NCIS
title	Episode title	Client Defined	S2, E1
length	Length of content in sec	Client Defined	3600
** Only applicable for SDK 1.1 and above.			

Players should retrieve these data items from the Content Management System. A proper JSON string must be created to pass this information via the SDK's `loadMetaData:` method.

Note The episode title will be the episode name used for reporting. It is recommended to use the following naming convention: Episode Name - Season Number - Episode Number - Shortform (SF)/longform (LF) content (*e.g.* Episode Name - S2 - E1 - LF*)*.

Nielsen's DPR product requires the following data parameters:

- assetid – unique ID of asset (content only)
- program / category – the program name

- title – the episode title
- length – the duration of the video
- type – content (must be set for ALL content)
- dprflag – flag used to separate DPR content from non-DPR content

Players should retrieve these data items from the Content Management System. A proper JSON string must be created to pass this information via the SDK’s `loadMetaData` method.

Your CMS system might refer to these data items with alternate names such as:

- videoID – assetID
- program – program
- episode – title
- duration – length
- vidType – type
- tv – dprflag

In the above example, the CMS mappings would need to be captured by Nielsen’s Technical Account Manager during the on-boarding process. The mapping will ensure that the player can create a proper JSON string using your current schema KEYS. In this case, you would send a JSON string of your CMS data (for example, JSON string-ify the metadata object, this would be the easiest implementation):

```
{
    “videoid”:”12345”,
    “program”:”NCIS”,
    “episode”:”Season 7 – Finale”,
    “duration”:”12300”,
    “vidType”:”content”,
    “tv”:”true”,
    “custom1”:”value 1”,
    “custom2”:”value 2”,
    "dataSrc":"cms",
    "adMode":"1"
}
```

The SDK, after the config call, would have the following CMS mapping:

```
{
    “nol_assetid” : “videoid”,
    “nol_category”: “program”,
    “nol_title”:”episode”,
    “nol_length”:”duration”,
    “nol_type”:”type”,
    “nol_dpr”:”tv”
}
```

If you chose to pass the entire CMS metadata object (or implement an auto reduction of Nielsen variables), the addition of tag variables becomes nothing more that adding the values from your CMS system, mapping that information in the CMS map, and then adding that information in the URL. There would be no (player/app) code changes required to extend the tags or add new tags, as Nielsen can support a new tagging schema, as long as the variables are there.

Sample JSON input for DPR:

```
{
"type":content",
"assetid":"MyContentAssetId",
"tv":"true",
"program":"MyProgram",
"title":"My Episode Title",
"category":"testcategory",
"length":"3600"
}
```

Sample

```
// Create the necessary JSON input
dprMetadata = "{\"type\" : \"content\",\"assetid\" :
\"MyContentAssetId\",\"tv\" : \"true\",\"program\" :
\"MyProgram\",\"title\" : \"MyEpisodeTitle\",
\"category\":\"testcat\", \"length\" : \"6000\"}";
```

DPR viewing is driven from the playhead position. The current playhead position (position in the video asset) must be passed to the SDK via the `playheadPosition` method every 2 seconds.

JSON Schemas for VC / IAG loadMetadata Methods

Example for a VC / IAG for Content Tags

```
{
"length":"600".
"title":"TestContentTitle",
"censuscategory":"TestContentCensusCategory",
"type":"content",
"assetid":"245671"
}
```

Example for a VC / IAG Advertisement (Type Preroll) Tags

```
{
“length”:”16”,
“title”:”FOX_StateFarm_SFSSP_Video_VAST_DFA1”,
“iag_epi”:”Season 3 - Guess Who's Back?”,
“iag_seg”:”1”,
“iag_pgm”:”New Girl”,
“category”:”STATE FARM”,
“subcategory”:”2013 State Farm Online Video”,
“type”:”preroll”
}
```

The following is an example of how you might load CMS data into an JSON string to send to the Android Nielsen App SDK using the `loadMetadata()` method:

```
    vcInfoDict = {
        "length":  "16",
         "title":  "FOX_StateFarm_SFSSP_Video_VAST_DFA",
       "iag_epi":  "Season 3 - Guess Who's Back?",
       "iag_seg":  "1",
       "iag_pgm":  "NewGirl",
         ”type” :  “preroll”,
       "catagory": "STATE FARM",
       };
```

Where:

- iag_epi – Episode Name
- iag_seg – Segment Number
- iag_pgm – Program

Nielsen mTVR (Mobile TV Ratings)

Setting the channelName Field

The ***channelInfo*** field is a 32-character free-form text field in the protocol between the device and Nielsen’s Collection Facility. This field is used to convey a text description of the channel or asset being selected by the user for viewing on his or her device. In an effort to enhance this field’s usefulness across different VPA player apps, this field has been sub-divided to support the creation of different channelInfo Types with the goal that MVPDs/CNDs/COs (Content Originators) would select one of these types for use in their particular application. For example, the use of CRIDs for on-demand assets allows Nielsen's collection facility to isolate on-demand content through the use of an asset’s CRID Authority.

Table 4 – Nielsen Mobile TV Ratings Fields

Name	Description	Sample Value
adModel **	Identifier to add model uses by the station. 0 - Default: Crediting is based on ad model break-out from the ID3 tag. 1 – Linear: Content will have the same linear ads 2 – Dynamic: Content will have ads dynamical served and not the same as linear ads	1

Name	Description	Sample Value
** Only applicable for SDK 1.1 and above.		

JSON for channel name at play:

```
{
”channelName”: ”ESPN2”,
”dataSrc”: ”id3”,
”adModel”: ”1”
}
```

Nielsen DRM (Digital Radio Measurement)

The SDK can accept latitude and longitude values in the case of DRM.

Sample JSON input for DPM:

```
{
"sfcode":"drm",
"appid":" XXXXE427-1970-47E1-BF7D-75752353XXXX ",
"appname":"Nielsen SDK Sample QA",
"appversion":"app.3.1.1",
"longitude":"123.223",
"latitude":"-223.323"
}
```

Sample

```
// Create the necessary JSON input
String config = "{"
      + "\"appName\"    : \"" + appName + "\","
      + "\"appVersion\" : \"" + appVersion + "\","
      + "\"sfcode\"     : \"" + sfCode + "\","
      + "\"appId\"      : \"" + appId + "\","
      + "\"latitude\"   : \"" + latitude + "\"," // optional
      + "\"longitude\"  : \"" + longitude + "\"" // optional
+ "}";
```

DRM viewing is driven from the playhead position. The current playhead position (position in the video asset) must be passed to the SDK via the playheadPosition method every 2 seconds.

Load Content Metadata at Playing

Table 5 – Content Metadata data JSON at play:

Name	Description	Sample Value
dataSrc **	Source of the data. For DRM datasrc should be passed as “cms”.	cms

Name	Description	Sample Value
type	Type of content. For DRM mention type as “radio”	radio
assetid	Station identifier. Should include Call letters and Band	WXYZ-FM
stationType	Station type indicating the type of usage and the ad load 0: Custom station built per client 1: Streaming station corresponds to an OTA station and has same ad load as OTA 2: Streaming station corresponds to an OTA but with different ad load 3: Multicast e-station or online station	0, 1, 2, 3
provider	Name of provider **Note** “provider” can be changed as long as the new names are provided to Nielsen.	providerName
** Only applicable for SDK 1.1 and above.		

Typical JSON of Content Metadata

JSON for radio content:

```
{
"type”: ”radio”,
”assetid”: ”WXYZ-FM”,
”stationType”: "3",
”provider”: ”providerName”
"dataSrc":"cms",
"adModel":"1"
}
```

Sample

```
metaTag = "{\"type\"     : \"radio\", "
      + "\"assetid\"     : \"KAAZ-FM\", "
      + "\"stationType\" : \"3\","
      + "\"dataSrc\"    : \"" + dataSrc + "\","  // optional
      + "\"adModel\"    : \"" + adModel + "\","  // optional
      + "\"provider\"    : \"Clear Channel\""
      + "}";
```

Appendix B – Data Collected from SDK

Table 6 – Nielsen Identifiers Collected

Nielsen Identifiers	Description
NUID	A unique ID assigned to each device generated by SDK on the device
App ID	A unique identifier provided to each app by Nielsen

Table 7 – App-Supplied Information Collected

App-Supplied Information	Description
App Version	Version of MVPD app
App Name	Name of MVPD app
App Disabled Flag	A flag indicating whether app has disabled SDK
Opt-Out flag	A flag indicating whether user has opt-out of measurement
sfCode	Country Code indicating the collection server the SDK will connect
Channel Info	A descriptive name provided by player to SDK channel name or URL
DMA	Nielsen Designated Market Area code
ccode	Country code
Play head position	Current play head position
longitude	Approximate geographical coordinates
latitude	Approximate geographical coordinates
Type	Radio or content. Type of play event
stationType	Station type indicating the type of usage and the ad load 0: Custom station built per client 1: Streaming station corresponds to an OTA station and has same ad load as OTA 2: Streaming station corresponds to an OTA but with different ad load 3: Multicast e-station or online station
provider	Name of Provider.

Table 8 – CMS Metadata Collected

CMS Metadata	Description
Program Name	Name of program currently viewed
Episode Name	Episode name of current program

CMS Metadata	Description
Episode Length	Length of current program
DPR enabled flag	Flag indicating whether content is DPR eligible
OCR data	Tags from OCR content
Asset ID	An ID for asset. This is an in-house number (for example, each client will have their own id for each asset) For DRM, asset ID is station identifier and should include call letters and band
Data Source	Defines from where the feeding the App SDK comes from: or ID3 tags (id3) or playhead position (cms)
Advertisement Model	The advertisement model can be: 0: Unknown, or not defined 1: Linear, normally live streaming 2: Non-linear or on demand
IAG Variables	Legacy tag
Video type	Indicates whether content is video or commercial

Table 9 – OS Info Collected

OS Info	Description
OS Version	Version of OS.
Device Type ID	Device Model Details
Device ID (IDFA)	Google Advertising ID or the Android Device ID

Table 10 – Nielsen Watermark Data Collected

Nielsen Watermark Data	Description
ID3 Payload	Encrypted ID3 payload and CIDs

Appendix C – channelInfo Free-Form Text Field

Setting the channelInfo Field

The ***channelInfo*** field is a 32-character free-form text field in the protocol between the device and Nielsen’s Collection Facility. This field is used to convey a text description of the channel or asset being selected by the user for viewing on his or her device. In an effort to enhance this field’s usefulness across different Apps, this field has been sub-divided to support the creation of different channelInfo Types with the goal that MVPDs/CNDs/COs (Content Originators) would select one of these types for use in their particular application. For example, the use of CRIDs for on-demand assets allows Nielsen's collection facility to isolate on-demand content through the use of an asset’s CRID Authority.

Nielsen App SDK Revision 2.0 channelInfo Definition

Table 11 – channelInfo Freeform Text Field

Field SubName	Description	Size
channelInfoType	Allows multiple mappings of different formats into this 32-character field	2 chars
channelInfo	Channel name information as defined by channelInfoType	30 chars

Glossary

A

APA

Audio Player Application. A media player application designed to play only audio (no video). It will host the Android App SDK.

C

CO

Content Originator. Content Originators are Nielsen clients that aggregate content and distribute it to end users, either as a linear feed (examples include traditional cable networks such as A&E, ESPN, TNT, etc.), or as on-demand content (examples include iPad applications such as A&E, WatchESPN, TNT for iPad, etc.).

CMS

Content Management System

D

DPR

Digital Program Ratings

H

HLS

HTTP Live Streaming. An HTTP-based media streaming communications protocol implemented by Apple and proposed as an Internet Draft Standard.

HTTPS

Hypertext Transfer Protocol – Secure. Standard secure connection protocol used on the web.

I

IAG

Nielsen acquired IAG Research in 2008. IAG measures consumer engagement with television programs, national commercials, and product placements.

ID3

A metadata container convention that has been adapted for use with HLS-streamed content.

M

MPX

Nielsen Media Player Extension. The Media Player Extension (MPX) is a Nielsen component that provides the capability to extract ID3 tag metadata data from the Android Native HLS media player.

MVPD

Multichannel Video Programming Distributor. MVPDs are server providers that deliver video programming feeds to end users. MVPDs may be cable television systems (CATV), direct-broadcast satellite provides (DBS), or traditional telcos. Examples include Comcast, Time Warner, DirecTV, DISH, FiOS, A&T U-Verse, etc.

N

Nielsen Watermarkd

A proprietary algorithm developed by Nielsen to insert sub-audible watermarks in audio portion of media files and/or streams. Nielsen's watermarks enable audience measurement.

O

OCR

Online Campaign Ratings

S

SID

Source Identifier. Source IDs and Program IDs are associated with the tags that are injected into the ID3 tags by Nielsen for retrieval by the component

V

VC

Video Census

Viewing Session

The period of time from the moment the App player starts streaming content, to the point where the user either shuts down the application, sends it to the background, or stops streaming content.

VPA

Video Player Application. A media player application designed to play audio and video. It will host the Android App SDK.