Harman International
Rev. 0.1
May 15, 2015

HK WirelessHD SDK Programming Guide (Android)

(Draft)

Version 0.1

HARMAN

Table of Contents

1. Overview

1.1. Introduction

1.1.1. HKWirelessHD

HKWirelessHD SDK is a set of header files, libraries and documentations to help developers create iOS apps that can control Harman Kardon Omni speakers to play audio wirelessly

In this document, we describe only about Android version of HKWirelessHD SDK.

1.1.2. Key Features of HKWirelessHD

- Wirelessly stream audio to HK Omni Speakers
  - Omni speakers are attached to a WiFi network. You can can manage and control speakers in the same WiFi network to play audio wirelessly.
- Support multi-room audio streaming
  - You can play audio with multiple Omni speakers with synchronization.
- Support party mode audio streaming
  - You can switch the speakers from a room (a set of speakers) to party mode (all speakers in all rooms), or vice versa.
- Support multi-channel audio streaming
  - If you have two Omni speakers, then you can send different channel of a single audio stream to speakers. For example, to play a music in stereo mode, you can assign a speaker for left channel, and assign the other for right channel.
  - Currently, stereo (2 channel) mode is supported. 5.1 channel mode is on schedule.
- Support mp3, wav, flac, sac, m4a and ogg formats, and the sample rate above 44100.

1.2. SDK information

1.1.1. Version

This document complies with the HKWirelessHD SDK Version 0.1.

1.1.2. Requirements & Android App Project Setup

Refer to Getting Started Guide for the description on how to setup an Android project with HKWirelessHD SDK.

2. Architecture Overview

1.1. Overview

1.1.1. Overall Configuration

There are two kind of entities in HKWirelessHD audio streaming - one source device and one or more destination devices. Source device sends audio stream to destination devices (speakers), and destination devices receive the audio stream from source and play it. Therefore, audio streaming is done in one-to-many way. That is, there is one single source device streaming an audio file, and multiple destination devices receive the audio stream with synchronization with each other.

In case of multi-channel streaming, each speaker is assigned with a role to process a dedicated audio channel. For example, a speaker can take left channel or right channel in stereo mode.

Source device can be Android device and destination devices are Harman Kardon Omni speakers (Omni Adapt, Omni 10, Omni 20, Omni Bar, etc.)

1.1.2. Use of HKWirelessHD API to stream audio to Omni Speakers

To send audio stream to destination devices, an App in Android device should use HKWirelessHD API. HKWirelessHD SDK provides the library of the APIs for arm32/64bit architecture. The version is must above Android 4.1(API 16) or later.

1.1.3. Communication channels between source and destinations

As shown in the figure above, there are two kind of communications between source device and (multiple) destination devices.

- Channel for audio streaming (one way communication from a source to multiple destinations)
  - This channel is used for transmitting audio data to destination speakers

- Channel for control commands and device status (bidirectional communication)
  - This channel is used to send command from source to destinations to control the device, like volume, etc.
  - A destination device can also send command to the source device in some cases. For example, a speaker which is not belonging to the current playback session can send a command to the source to add itself to the current playback session and play audio on it.
    - User can add Omni 10 or Omni 20 speaker to the on-going playback session by long-pressing the Home button on the control panel. Please refer to Omni 10 or 20 User's manual for more information.
  - This channel is also used to send device information and status data of destination speakers to the source device.
    - Device information includes the speaker name, the group name, IP address and port number, firmware version, etc.
    - Device status information includes the status about the device availability and change of its attributes, whether or not it is playing music, the Wi-Fi signal strength, volume change, etc.

1.1.4. Asynchronous Communication

The communication between the source device and the destination speakers are done in asynchronous way. Asynchronous behavior is expected because all the commands and status updates are executed in a way like RPC (Remote Procedure Call) or something similar. Even more, audio streaming always involves some amount of buffering of audio data, so, the timing gap between the source and the destinations can be larger.

Below are some examples of asynchronous communications.

- Device availability
  - When a speaker is turned on, the availability of the speaker is reflected to the source device a few second later. Likewise, if a speaker is turned off or disconnected from the network, its unavailability is reflected to the source device a few second later.

- Playback control
  - When the source starts music playback to destination speakers, actual playback in destination speakers starts a few hundreds milliseconds later. Similar things occur when the source pauses or stops the current audio streaming, although stop or pause requires much less time.

- Volume Control

- When the source changes the volume level of all speakers or selected individual speakers, actual volume changes occur a few millisecond later.

1.1.5. Speaker Management

Whenever a speaker updates its status, the latest status information should be updated on the source device side as well. HKWirelessHD API manages the latest device status information inside of DeviceInfo instances. HKWControlHandler instance maintains a list (table) of DeviceInfo objects, each of which corresponds to a speaker found in the network.

The detailed description on each attribute in the device are described in DeviceObj.java.

1.2. Visibility of Speakers

Any speakers in a network are visible to source devices (Android devices) if a source device successfully initializes the HKWirelessHandler when it starts up. Source devices can be multiple. This means, even in the case that a speaker is being used by a source device, the status of each speaker is also visible to all other source devices in the network, once they are successfully initialized with HKWirelessHandler.

For example, as described in the figure below, let's assume that Speaker-A and Speaker-B are being used by Source A, and Speaker-D and Speaker-E are being used by Source B. Once Source A and Source B initialize HKWirelessHandler, then all the speakers from Speaker-A to Speaker-E are also visible both to Source A and Source B. Therefore, it is possible for Source A to add Speaker-D to its on-going playback session, even if it is being used by Source B. In this case, Speaker-D stops playing the audio stream from Source B, and join the on-going playback audio stream from Source A.

There is an API, called isPlaying(), in DeviceObj.java to return a boolean value indicating if the speaker is being playing audio or not, regardless that which source audio stream comes from.

1.3. Controlling Speakers and Handling the Events from Speakers

1.1.1. Controlling speakers

Speaker controls, like start/pause/resume/stop audio streaming, change volume level, etc. are done by calling APIs provided by the **AudioCodecHandler** object. The app just needs to acquire the AudioCodecHandler object, initialize the HKWirelessHandler object, and then use the AudioCodecHandler to control the speakers. For example, as shown in the figure above, the app can call playCAF() with the AudioCodecHandler to start to play an audio file. The control APIs are described in AudioCodecHandler.java.

1.1.2. Handling events from speakers

On the other hand, the events from speakers are sent to the app through **HKWirelessListener** interface APIs. By implementing the event handler interface callback functions, you can receive and handle the events from speakers. Whenever an event occurs from speakers, the corresponding handler is called and the event information is passed to the handler as parameter.

The SDK provides one listener interface:

- HKWirelessListener (defined in HKWirelessListener.java)
    - To register an object as the listener, do as below
        - device status updated
        - error occurred
        - play ended
        - playback state changed
        - playback time changed
        - volume changed

3. Programming Guide

In this document, we explain how to use HKWirelessHD APIs to create an app controlling HK Omni speakers. The sample codes explained in this section are copied from WirelessOmni app.

All APIs can be accessed through the object pointer of HKWirelessHandler and AudioCodecHandler. Only you have to do is create a HKWirelessHandler object and a AudioCodecHandler object use them to invoke the APIs you want to use.

For setting up a project with HKWirelessHD SDK, please refer to Getting Started Guide.

1.1. Initialization

Controlling HK Omni speakers are done by calling APIs provided by HKWControlHandler. So, the first thing to do to use HKWirelessHD APIs is to acquire the singleton object of HKWControlHandler and initialize it.

Once you acquire the HKWControlHandler object, you need to initialize it by calling initializeHKWirelessController() with a license key value as parameter. Every developer who signed up to Harman developer web site will receive a license key code.

```
  // Create a HKWControlHandler instance
HKWirelessHandler hControlHandler = new HKWirelessHandler();

  // Initialize the HKWControlHandler and start wireless audio
hControlHandler.initializeHKWirelessController("");
```

Note that intializeHKWirelessController() is a blocking call. So, it will not return until the caller successfully initializes HKWireless Controller. If the phone is not connected to a Wi-Fi network, or any other app on the same phone is already using the HKWireless controller, then the call will wait until the app releases the controller. If you want on-blocking behavior, you should call this function asynchronously by running it in a separate thread, not main thread.

Note that even after a successful call to initializeHKWirelessController(), it takes a little time (a few hundreds milliseconds to a couple of seconds) to get the information about the speakers available for streaming audio.

1.2. Getting a list of speakers available in the network

HKWControlHandler maintains the list of speakers available in the network. The list changes every time a speaker is added to the network or removed from the network. Whenever a speaker is added to or removed from the network, the **onDeviceStateUpdated** interface function is called. So developer needs to check which speaker has been added or removed, and handle the case accordingly, such as update the UI of speaker list, and so on.

1.1.1. Device Information

Each speaker information contains a list of attributes that specify its static information such as speaker name, group name, IP address, etc. and also dynamic information such as volume level, boolean value indicating if it is playing or not, wifi signal strength, etc.

You can retrieve all the attributes of device (speaker) information through DeviceObj object, and it is specified in DeviceObj.java. The following table shows the list of information that DeviceObj provides.

In the table, "Fixed/Variable" column means if the attribute value is a fixed value during the execution, or can be changed by itself or by calling APIs. "Set by API" column means whether the value of the attribute can be changed by API calls.

Note that all the attributes in DeviceObj.java are "readonly". So, you can only read the value of each attribute. You need to use corresponding API functions to change the values of the attributes.

Attribute	Type in Swift	Description	Fixed/Variable	Set by API
deviceId	long	the unique ID of the speaker	Fixed (in manufacturing)	No
deviceName	String	the name of the speaker	Variable	Yes
ipAddress	String	the IP address as String	Fixed (when network setup)	No
port	int	the port number	Fixed (when network setup)	No
macAddress	String	the mac address as String	Fixed (in manufacturing)	No
groupId	long	the unique ID of the group that the speaker belongs to	Variable (set when a group is created)	No
groupName	String	the name of the group that the speaker belongs to	Variable (set when a group is created)	Yes
modelName	String	the name of the Model of the speaker	Fixed (in manufacturing)	No
volume	Int	the volume level value (0 to 50)	Variable	Yes
active	boolean	the boolean value indicating the speaker is added to current playback session	Variable	Yes
wifiSignalStrength	Int	the value of WiFi signal strength in dBm scale (normally ranging between -100 and 0)	Variable	No
role	Int	the role definition of the speaker (stereo or 5.1 channel)	Variable	Yes
version	String	the firmware version number as String	Fixed (when firmware update)	No

macAddress	String	the mac address of the speaker	Fixed (in manufacturing)	No
balance	Int	the balance value in stereop mode. The value range from -6 to 6, 0 is neutral.	Variable	Yes
isPlaying	boolean	the boolean value that indicates whether the speaker is playing or not, regardless of the source.	Variable	No
channelType	Int	the channel tyle: 1 is stero.	Variable	Yes
isMaster	boolean	the boolean value that indicates whether the speaker is the master in stereo or group mode.	Variable	Yes

The following is an example of retrieving some of attributes of a speaker information.

```
    DeviceObj DeviceInfo = hControlHandler.getDeviceInfoFromTable(groupIndex, deviceIndex);
        Log.d(LOG_TAG, "name :" + DeviceInfo.deviceName);
        Log.d(LOG_TAG, "ipAddress :" + DeviceInfo.ipAddress);
        Log.d(LOG_TAG, "volume :" + DeviceInfo.volume);
        Log.d(LOG_TAG, "port :" + DeviceInfo.port);
        Log.d(LOG_TAG, "role :" + DeviceInfo.role);
        Log.d(LOG_TAG, "modelName :" + DeviceInfo.modelName);
        Log.d(LOG_TAG, "zoneName :" + DeviceInfo.zoneName);
        Log.d(LOG_TAG, "active :" + DeviceInfo.active);
        Log.d(LOG_TAG, "version :" + DeviceInfo.version);
        Log.d(LOG_TAG, "wifi :" + DeviceInfo.wifiSignalStrength);
        Log.d(LOG_TAG, "groupID :" + DeviceInfo.groupId);
        Log.d(LOG_TAG, "balance :" + DeviceInfo.balance);
        Log.d(LOG_TAG, "isPlaying :" + DeviceInfo.isPlaying);
        Log.d(LOG_TAG, "channelType :" + DeviceInfo.channelType);
        Log.d(LOG_TAG, "isMaster :" + DeviceInfo.isMaster);
...
```

1.1.2. Getting a speaker (device) information

HKWControlHandler maintains the list of speaker internally. Each speaker information can be retrieved by specifying the index in the table, or by specifying the index of group and the index of member inside of the group.

1.1.1.1. Get the speaker information from the table

You can retrieve a speaker information (as DeviceInfo object) by specifying the index in the table.

```
DeviceObj getDeviceInfoByIndex(int deviceIndex);
```

Here, the range of deviceIndex is 0 to the number of speakers (deviceCount) minus 1.

This function is useful when you need to show all the speakers in ordered list in list.

1.1.1.2. Get a speaker information from the group list

You can retrieve a speaker information by specifying a group index and the index of the speaker in the group.

```
DeviceObj getDeviceInfoFromTable(int groupIndex, int deviceIndex);
```

Here, groupIndex represents the index of the group where the device belong to. deviceIndex means the index of the device in the group.

This function is useful to find the device information (DeviceInfo object) that will be shown in a ListView. For example, to show a speaker information in two section ListView, the groupIndex can correspond to the section number, and deviceIndex can correspond to the row number.

1.1.3. Get a speaker information with deviceId

If you already knows the deviceId (device unique identifier) of a speaker, then you can retrieve the deviceInfo object with the following function.

```
DeviceObj findDeviceFromList(long deviceId);
```

1.3. Refreshing device status information

If any change happens on the speaker side, the corresponding speaker sends an event with updated information to HKWControlHandler and then the speaker information stored in HKWControlHandler is updated. And then, HKWControlHandler calls corresponding interface functions registered by the app to make it processed by the event handler.

However, in our current implementation, the event dispatching initiated by speaker like explained above takes a little more time than the app polling to check if there is any update on speakers. To reduce the time of status update, we provide a pair of functions to refresh device status, which is a kind of polling to check the update. Especially, if you need to show a list of speakers with the latest information, you'd better force to refresh the speaker information, not just waiting updates from speakers.

To discover and update the status of speakers immediately, you can use the following functions:

```
// start to refresh devices ...
hControlHandler.startRefreshDeviceInfo()

// stop to refresh devices
hControlHandler.stopRefreshDeviceInfo()
```

startRefreshDeviceInfo() will refresh and update every 2 seconds the status of the devices in the current Wi-Fi network.

1.4. Add or remove a speaker to/from a playback session

To play a music on a specific speaker, the speaker should be added to the playback session.

You can check whether or not a speaker is currently added to a playback session by check the "**active**" attribute of DeviceInfo object (in DeviceObj.java).

```
/*! Indicates if the speaker is active (added to the current playback session) */
public boolean active;
```

1.1.1. Add a speaker to a session (to play on)

Use addDeviceToSession() to add a speaker to the current playback session.

```
boolean addDeviceToSession(long id);
```

For example,

```
// add the speaker to the current playback session
hControlHandler.addDeviceToSession(deviceId);
```

If the execution is successful, then the attribute "active" of the speaker is set to "true".

Note that a speaker can be added to the current on-going playback session anytime, even the playback is started already. It usually takes a few seconds for the added speaker to start to play audio.

1.1.2. Remove a speaker from a session

Use removeDeviceFromSession() to remove a speaker from current playback session. The removed speaker will stop playing audio immediately.

```
boolean removeDeviceFromSession(long deviceId);
```

```
// remove a speaker from the current playback session
hControlHandler.removeDeviceFromSession(deviceId);
```

If the execution is successful, then the attribute "active" of the speaker is set to "false".

Note that a speaker can be removed from the current on-going playback session anytime.

Note that after a speaker was removed from the session and there is no speaker remaining in the session, then the current playback stops automatically.

1.5. Play a song

1.1.1. Play MP3 and WAV file

Firstly, you acquire the AudioCodecHandler object.

```
AudioCodecHandler hAudioControl = new AudioCodecHandler();
```

If one or more speakers are added to the session, you can start to play a song. Currently, mp3, wav, flac, sac, m4a and ogg formats are supported, but the sample rate of the song must above 44100. Use playCAF() to play mp3, wav, flac, sac, m4a or ogg file, and playWAV only for WAV file.

```
boolean playCAF(String url, String songName, boolean resumeFlag);
```

To play a song, you should prepare a url using String first. Here is an example:

```
String url = …
String songTitle = …
hAudioControl.playCAF(url, songTitle, false)
```

Here, resumeFlag is false, if you start the song from the beginning. If you want to resume to play the current song, then resumeFlag should be true. 'songTitle' is a String, representing the song name. (This is only internally used as a file name to store converted PCM data in the memory temporarily.)

If you want to specify a starting point of the audio stream, then you can use playCAFFromCertainTime() to start the playback from a specified time.

boolean playCAFFromCertainTime(String url, String songName, int startTime);

Here, startTime is in second.

playCAF() and playCAFFromCertainTime() can play mp3, wav, flac, sac, m4a or ogg audio file. In case of playWAV(), it is played without conversion. In case of playCAF(), it is converted to PCM format first, and then played.

To play WAF audio file, use playWAV().

boolean playWAV(String url);

The following example shows how to play a WAV file stored in the application bundle.

```
String wavPath = …

hAudioControl.playWAV(wavPath);
```

Note that playCAF() and playCAFFromCertainTime() cannot play an audio file in service in currently. Songs should reside locally on the device for playback. So, it would be nice to check if the song resides on the devic.

You can check the playback status anytime, that is, before and after as well as in the middle of the playback. You can get the player status by calling getPlayerState(). (HKPlayerState is defined in HKPlayerState.java)

HKPlayerState getPlayerState();

If you just want to check if the player is playing audio now, then use isPlaying().

boolean isPlaying();

1.1.2. Playback controls

1.1.1.1. Stop playback

To stop the current playback, use stop(). As a result. the playback status is changed to EPlayerState_Stop, and onPlaybackStateChanged() delegate protocol is called if implemented.

void stop();

```
hAudioControl.stop();
```

It is safe to call stop() even if there is no on-going playback. Actually, we recommend to call stop() before you start to play a new audio stream.

1.1.1.2. Pause playback

To pause the current play, use pause(). As a result, the playback status is changed to EPlayerState_Pause, and onPlaybackStateChanged() delegate protocol is called if implemented.

void pause();

```
hAudioControl.pause();
```

1.1.3. Volume Control

You can set volumes in two ways – one is set volume for an individual speaker, and the other is set volume for all speakers with the same volume level. The volume level ranges from 0 (mute) to 50 (max).

Note that volume change functions are all asynchronous call. That is, it takes a little time (a few milli second) for a volume change to take effect on the speakers.

Note also that when setVolumeDevice() is called, the average volume can be also changed. So, it is safe to retrieve the speaker volumes using VolumeLevelChanged callback (explained later) when your app calls volume control APIs.

1.1.1.1. Set volume to all speakers

Use setVolumeAll() to set the same volume level to all speakers.

void setVolumeAll(int volume);

```
// set volume level to 25 to all speakers
int volume  = 25;
hAudioControl.setVolumeAll(volume);
```

1.1.1.2. Set volume to a particular speaker

Use setVolumeDevice() to set volume to a particular speaker. You need to specify the deviceId for the speaker.

void setVolumeDevice(long deviceId, int volume);

```
// set volume level to 25 to a speaker
int volume  = 25;
hAudioControl.setVolumeDevice(deviceId volume:volume);
```

1.1.1.3. Get volume of all speakers

Use getVolume() to get the average volume level fro all speakers.

int getVolume();

```
int averageVolume = hAudioControl.getVolume();
```

1.1.1.4. Get volume of a particular speaker

Use getDeviceVolume() to get the volume level of a particular speaker.

```
int volume = hAudioControl.getDeviceVolume(deviceId);
```

1.6. Speakers and Groups

In HKWirelessHD SDK, a group is a collection of speakers. A group is defined as below:

- The group of a speaker is defined by specifying a group name in the speaker information as attribute.
- A speaker can join only one group at a time. The meaning of "joining a group" is to have the group name in its attribute.
- All the speakers with the same group name belong to the same group associated with the group name.
- The group ID is determined by following the device ID of the initial member of a group. For example, there is no group with the name "Group-A", and Speaker-A sets the group as "Group-A", then the GroupID is created with the deviceID of Speaker-A. After that, if Speaker-B joins Group-A, then the group name and the group ID are set by the ones that Speaker-A has.

1.1.1. Change speaker name

Use setDeviceName() to change the speaker name. Note that you cannot set the device name by setting “deviceName” property value directly. The property is read-only.

void setDeviceName(long deviceId, String deviceName);

For example,

```
hAudioControl.setDeviceName(deviceId, ”My Omni10”);
```

Be careful that while a speaker is playing audio, if the name of the speaker is changed, then the current playback is interrupted (stopped) with error. The error code and message are returned by onErrorOccurred(), a delegate defined in HKWirelessListener interface.

1.1.2. Set the group for a speaker

To set a group for a speaker (in other words, to join a speaker to a group), use setDeviceGroupName as below:

void setDeviceGroupName(long deviceId, String groupName);

For example,

```
hAudioControl.setDeviceGroupName(deviceId, ”Living Room”);
```

Note that if you change the group name of a speaker, then the list of speakers of the group automatically changes.

1.1.3. Remove a speaker from a group

Use removeDeviceFromGroup() to remove the speaker from the belonged group. After being removed from a group, the name of group of the speaker is set to "**harman**", which is a default group name implying that the speaker does not belong to any group.

void removeDeviceFromGroup(long groupId, long deviceId);

For example,

```
hControlHandler.removeDeviceFromGroup(groupId, deviceId);
```

1.7. Interface APIs for events handling

In HKWirelessHD, the communication between user's phone and speakers are done in asynchronous way. Therefore, some API calls from HKWirelessHandler can take a little time to take effects on the speaker side. Similarity, any change of status on the speaker side are reported to the phone a little time later. For example, the status of availability of a speaker can be updated a few seconds later after a speaker turns on or off.

All the status update from the speaker side are reported to the phone via **HKWirelessListener interface**. So, your app needs to implement the interface accordingly to receive and handle the events from HKWirelessHandler.

1.1.1. HKWirelessListener

1.1.1.1. onDeviceStateUpdated

This function is invoked when some of device information have been changed on a particular speaker. The information being monitored includes device status (active or inactive), model name, group name, and wifi signal strengths, etc. The parameter 'reason' specifies what the update is about. The reason code is defined in HKDeviceStatusReason.java.

Note that volume level change does not trigger this call. The volume update is reported by VolumeLevelChanged callback.

```
void onDeviceStateUpdated(long deviceId, int reason);
```

This callback is essential to retrieve and update the speaker information in timely manner. If your app has a screen that shows a list of speakers available in the network with latest information, you can receive the event via this function and update the list.

1.1.1.2. onErrorOccurred

This function is invoked when an error occurs during the execution. The callback returns the error code, and also corresponding error message for detailed description. The error codes are defined in HKErrorCode.java.

```
void onErrorOccurred(int errorCode, String errorMesg);
```

A most common usage of this function is to show an alert dialog to notice the user of the error.

1.1.1.3. onPlayEnded

This function is invoked when the current playback has ended.

```
void onPlayEnded();
```

This function is useful to take any action when the current playback has ended.

1.1.1.2. onVolumeLevelChanged

This function is invoked when volume level is changed for any speakers. It is called asynchronously right after any of SetVolume APIs are called by apps.

The function delivers the device ID of the speaker with volume changed, a new device volume level, and average volume level value, as below:

```
void onVolumeLevelChanged(long deviceId, int deviceVolume, int avgVolume);
```

Note that when speaker volume is changed by a call to "setVolumeAll()", then all the speakers are set to a new volume level, and this function can be used to get the new volume level value.

1.1.1.3. onPlaybackStateChanged

This function is invoked when playback state is changed during the playback. The callback delivers the playState value as parameter.

```
void onPlaybackStateChanged(int playState);
```

1.1.1.4. onPlaybackTimeChanged

This function is invoked when the current playback time is changed. It is called every one second. The function parameter timeElapsed returns the time (in second) elapsed since the start of the playback. This function is useful when your app update the progress bar of the current playback.

```
void onPlaybackTimeChanged(int timeElapsed);
```

1.1.3. How to implement the HKWirelessListener interface

Your class has to implement the listener by specifying the interface name in the class definition as below:

```
public class YourClass implements HKWirelessListener {
……
}
```

And then, your class should register the listener as below:

HKWirelessHandler hWirelessController = new HKWirelessHandler();
hWirelessController.registerHKWirelessControllerListener(this);

At last, your class should implement the interfaces as below:

```
public void onDeviceStateUpdated(long deviceId, int reason){
   …
}
public void onPlaybackStateChanged(int playState){
   …
}
public void onVolumeLevelChanged(long deviceId, int deviceVolume, int avgVolume){
   …
}
public void onPlayEnded(){
   …
}
public void onPlaybackTimeChanged(int timeElapsed){
   …
}
public void onErrorOccurred(int errorCode, String errorMesg){
   …
}
```

Revision History

Rev	Description	Date
r01	First draft (SDK v0.1)	05/15/2015