

---

# **Harman Developer Documentation**

***Release 1.0***

**Harman International**

February 27, 2016



<b>1</b>	<b>HDWireless iOS SDK Documentations</b>	<b>3</b>
<b>2</b>	<b>HKWirelessHD Android SDK Documentations</b>	<b>5</b>
<b>3</b>	<b>Pulse2 SDK Documentation</b>	<b>7</b>
<b>4</b>	<b>Demos and 3rd Party Integrations</b>	<b>9</b>
<b>5</b>	<b>Contents</b>	<b>11</b>
5.1	SDK Overview (iOS) . . . . .	11
5.2	Getting Started Guide (iOS) . . . . .	30
5.3	Programming Guide (iOS) . . . . .	42
5.4	Architecture Overview (iOS) . . . . .	53
5.5	API Documentation (iOS) . . . . .	57
5.6	HKWHub Specification (iOS) . . . . .	71
5.7	Troubleshooting (iOS) . . . . .	117
5.8	Version History (iOS) . . . . .	119
5.9	SDK Overview (Android) . . . . .	120
5.10	Getting Started Guide (Android) . . . . .	123
5.11	Programming Guide (Android) . . . . .	130
5.12	API Documentation (Android) . . . . .	141
5.13	Version History (Android) . . . . .	154
5.14	Pulse2 SDK Documentation . . . . .	154
5.15	Demos and 3rd party integrations . . . . .	162
<b>6</b>	<b>Search</b>	<b>163</b>





---

**Note:** This documentation is about Harman Kardon WirelessHD SDK. You can download the SDK at [Harman Developer web site](#).

This documentation is a work in progress.

---

Welcome to the Harman developer documentation. Here you will find information about creating your app using HKWirelessHD SDK.



---

# HDWireless iOS SDK Documentations

---

This documentation is organized into a few different chapters:

**SDK Overview (iOS)** This chapter is the overview of HKWirelessHD SDK and the SDK apps.

**Getting Started (iOS)** This chapter describes how you can quickly get started with the HKWirelessHD SDK. Just follow the instruction to setup your Xcode project, and learn how to create a HKWirelessHD App.

**Programming Guide (iOS)** This chapter explains about the APIs of HKWirelessHD SDK.

**Architecture Overview (iOS)** This chapter explains about the overall architecture of HKWirelessHD SDK.

**API Documentation (iOS)** This chapter describes the API Specification of HKWirelessHD SDK.

**HKWHub Specification (iOS)** This chapter explains about the HKWHub App and its REST API Specification.

**Troubleshooting (iOS)** This chapter describes several common problems that developers can face at the beginning of using HKWirelessHD SDK, and provides solutions for them.

**Version History (iOS)** Version History

---



---

## HKWirelessHD Android SDK Documentations

---

**SDK Overview (Android)** This chapter is the overview of HKWirelessHD SDK and the SDK apps based on the SDK.

**Getting Started (Android)** This chapter describes how you can quickly get started with the HKWirelessHD SDK. Just follow the instruction to setup your Xcode project files, and learn how to create a HKWirelessHD App.

**Programming Guide (Android)** This chapter explains the APIs of HKWirelessHD SDK.

**API Documentation (Android)** This chapter explains the APIs of HKWirelessHD SDK.

**Version History (Android)** Version History



---

## Pulse2 SDK Documentation

---

**Pulse2 API** This chapter describes the Pulse2 SDK/API.





---

## Demos and 3rd Party Integrations

---

**Integration** Demos and 3rd party IoT platform and device integrations.

---



---

## Contents

---

### 5.1 SDK Overview (iOS)

#### 5.1.1 Overview of HKWirelessHD SDK (iOS)

The Harman Kardon WirelessHD (HKWirelessHD) SDK is provided for iOS 3rd party developers to communicate with Harman/Kardon Omni Series audio/video devices. The intent of this SDK is to provide the tools and libraries necessary to build, test and deploy the latest audio applications on the iOS platform.

#### What's Included

- **HKWirelessHD library and header files**
  - We provide two versions of SDK - a normal version and a lightweight version. See below for the information.
  - The reason we support the SDK as two separate version is that we know that many developers want a feature of web streaming. To support this feature, we need to include a version of FFmpeg library inside of the SDK library. But, some developers may want to use their own version of MMPEG to handle audio stream for some particular reasons. We describe more about these two versions in the Getting Started (iOS).
- License Agreement.
- **Sample Application source code Located within the Sample Apps page:**
  - Page app
  - Wake app
  - Timer app
  - HKWPlayer app with Apple Watch app support
  - HKWHub app
  - HKWSimple app

#### Requirements

The HKWirelessHD SDK requires iOS 8.3 or later (8.4 as of writing this) for iPhone or iPad devices. Xcode 6.3 (supporting iOS8.3) or later is required for building HKWirelessHD enabled iOS apps with Apple Watch app support.

---

**Note:** If you have updated iOS to version 8.4, you need to install Xcode6.4.

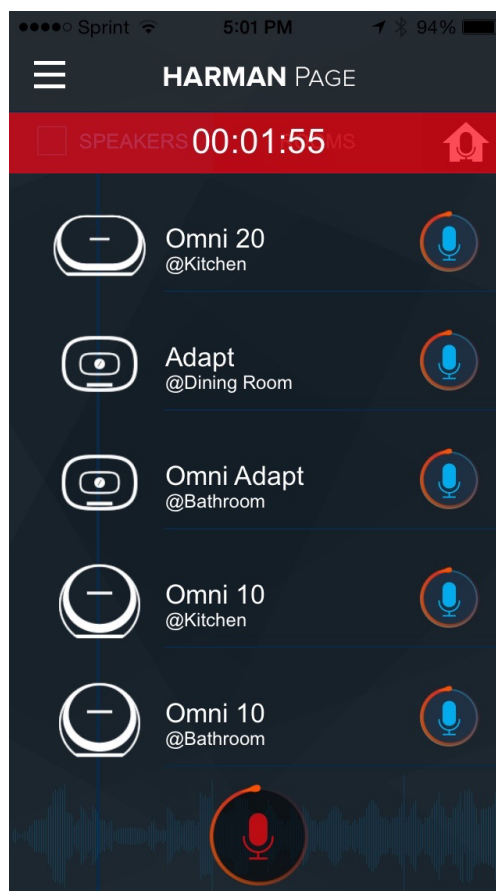
---

## Demo Applications

### HK Page app

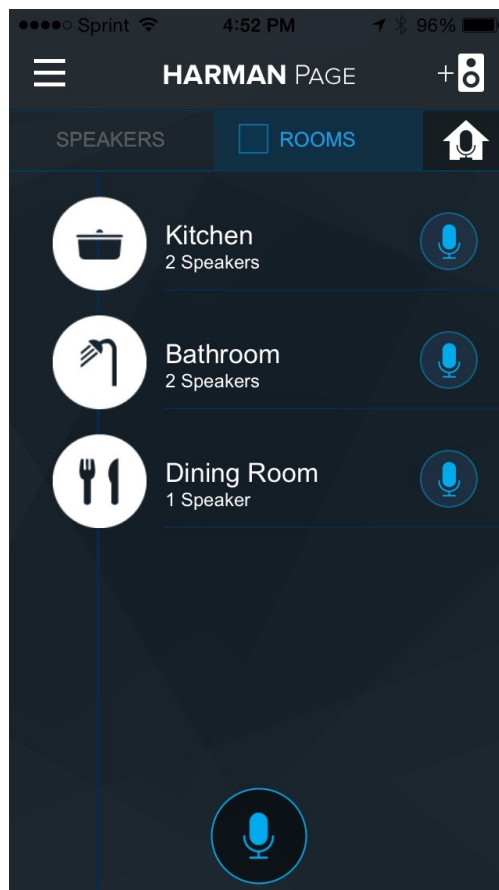
The Page app records user's voice and broadcasts it to a set of selected speakers in the network. User can select speakers individually or select rooms. The Page app also contain House Alert feature that broadcast an alarm sound like siren to all of the speakers in the network.

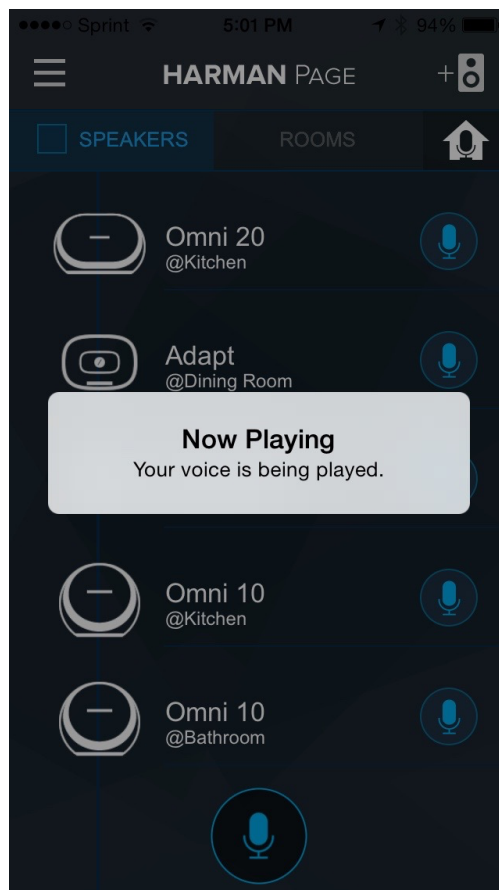
The application also has a feature for changing the speaker information, like speaker name, group name, taking a note, and so on.

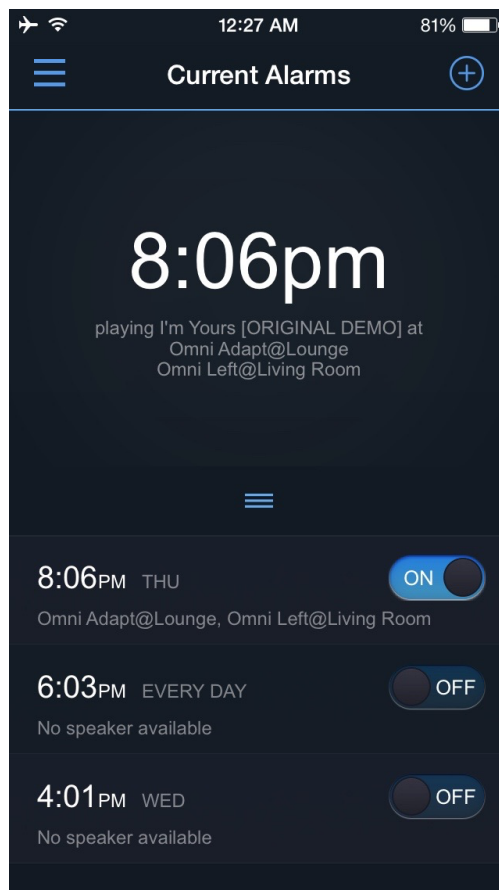


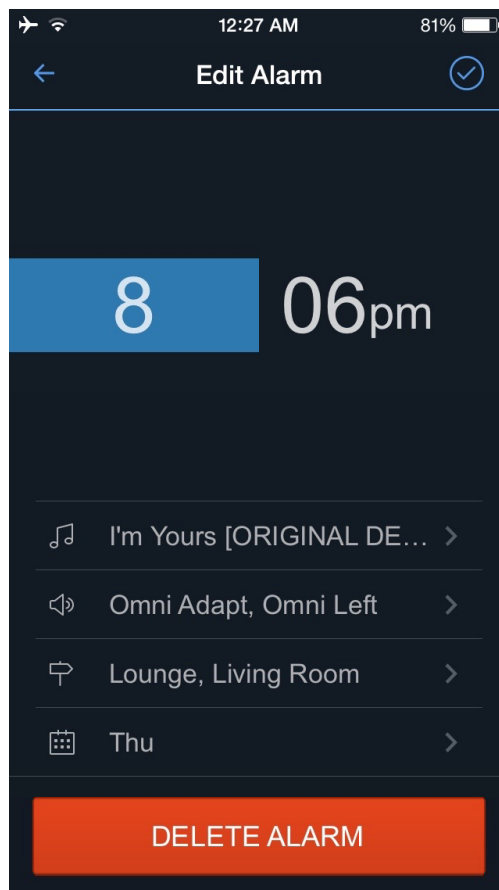
### HK Wake app

Wake app is a kind of Alarm clock. Instead of playing an alarm sound on your phone, the app play a song over the Omni speakers in the network. User can select a set of individual speakers or room to play the alarm sound with.

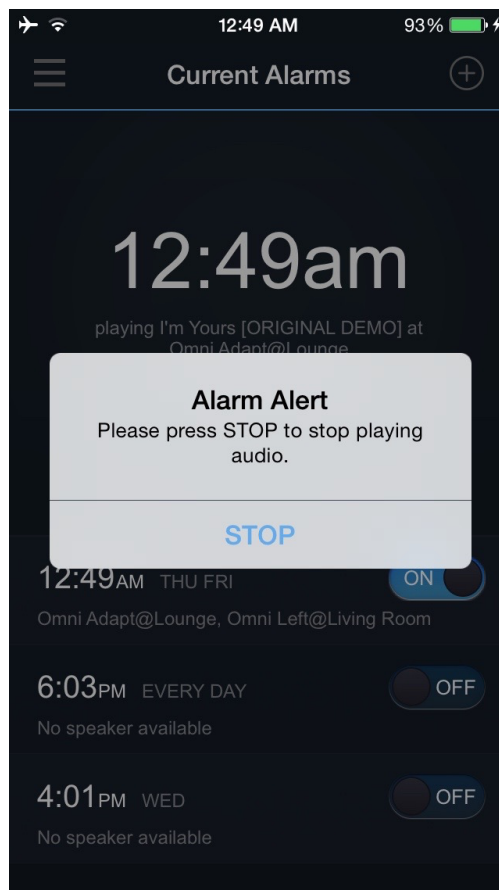






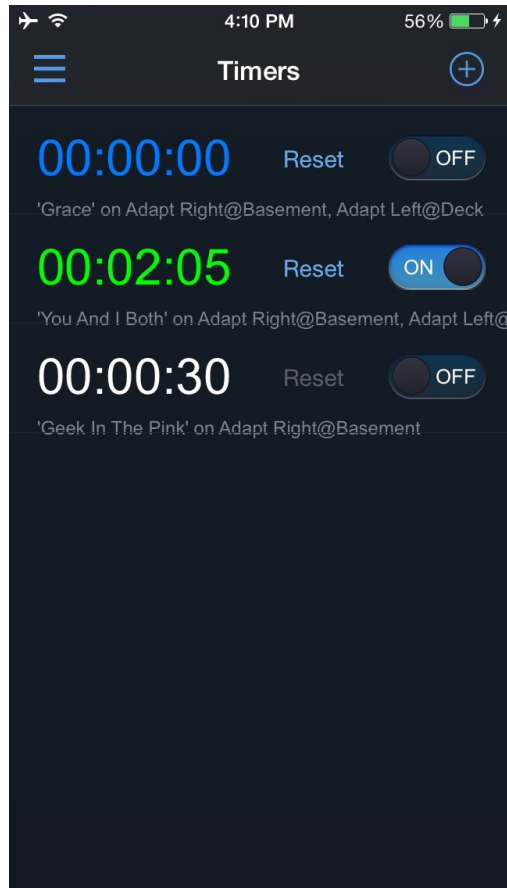






## HK Time app

Time (Timer) app is a kind of timer app. Instead of playing a beep sound on your phone, the app plays a predefined song over the Omni speakers in the network. User can select a set of individual speakers or room to play the alarm sound with.



## HKWPlayer app

HKWPlayer is a sample music player app that plays MP3 audio files with Omni speakers wirelessly. You can create and manage a playlist of MP3 titles from iOS Music app library, and play songs over the Omni speakers in the network. The purpose of the app is to demonstrate the key features of the HKWirelessHD SDK.

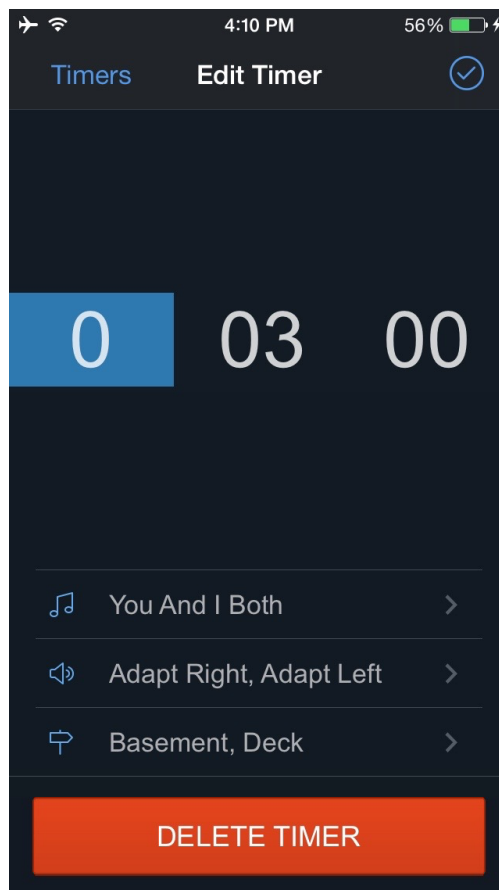
## HKWHub App (Hub for IoT Integration)

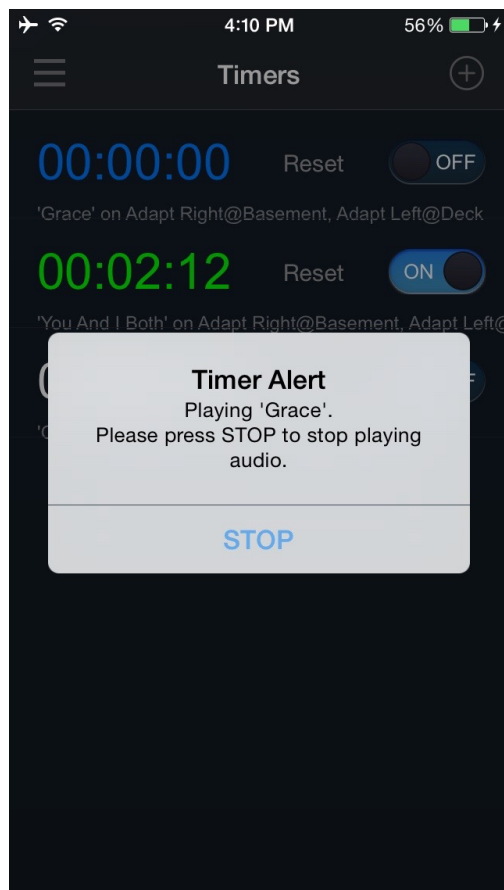
HKWHub app is an iOS app that uses HKWirelessHD SDK and acts as a Web Hub that handles HTTP requests to control speakers and stream music. It enables any types of connected devices (e.g. sensors or smart devices like tablet, smart TV, etc.) and cloud-based services (e.g. SmartThings) to connect HK Omni speakers and stream music. HKWHub runs a web server inside that handles HTTP requests of REST API.

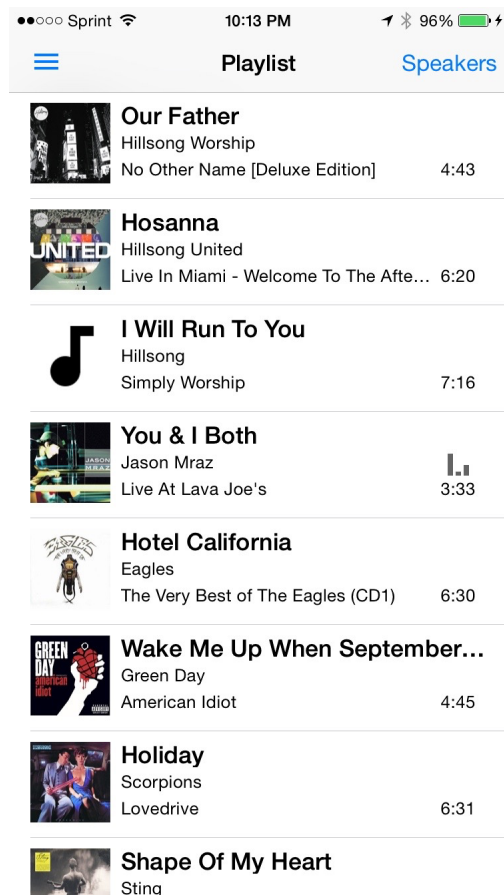
---

**Note:** Please refer to [HKWHub App](#) section of the documentation page for the REST API specification and other information.

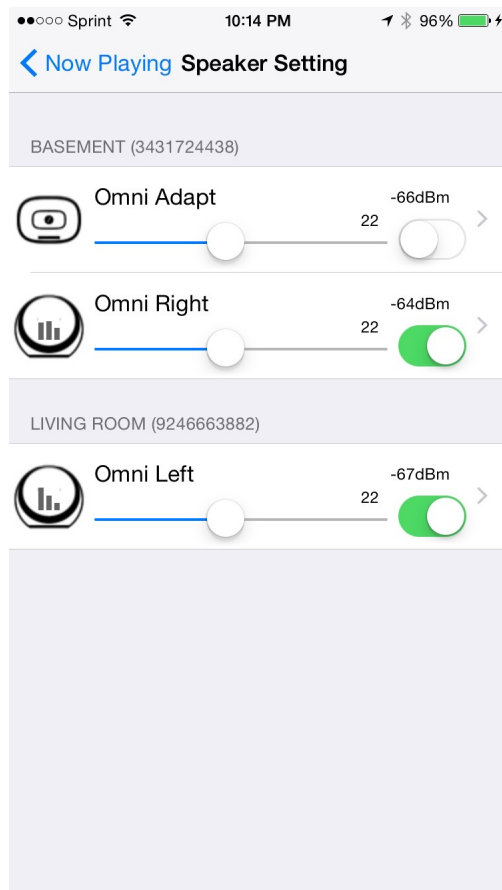
---





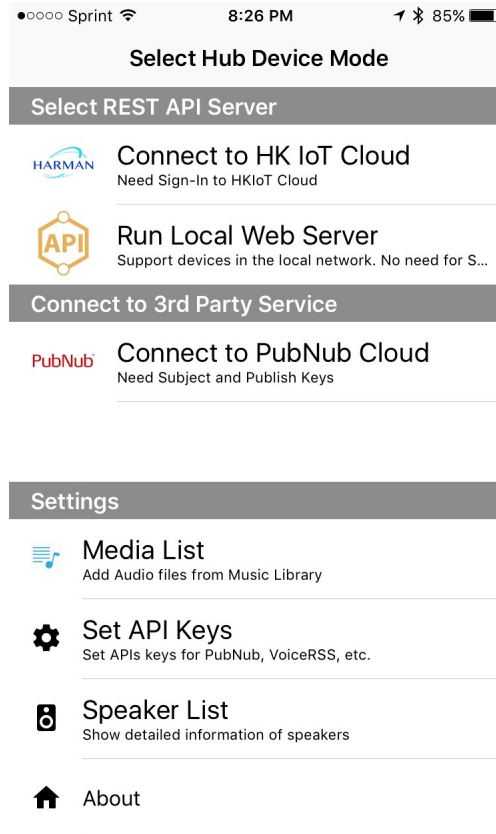






User can add songs or sound file from iOS Music app to the media list, so that client devices can access the list and play media in the list remotely by sending REST API request to the Hub app. For example, a door open/close sensor can send REST API request to play 'dog-barking' sound in the media list of the HKWHub app.

The following images are the screen captures of HKWHub app.



We also created a sample HTML5 app working as a client of the HKWHub app. The HTML5 app uses AJAX to send REST API requests to the HKWHub app to control speakers and stream music. The UI of the HTML5 app is based on Google's Polymer v0.5 (<https://www.polymer-project.org/0.5/>).

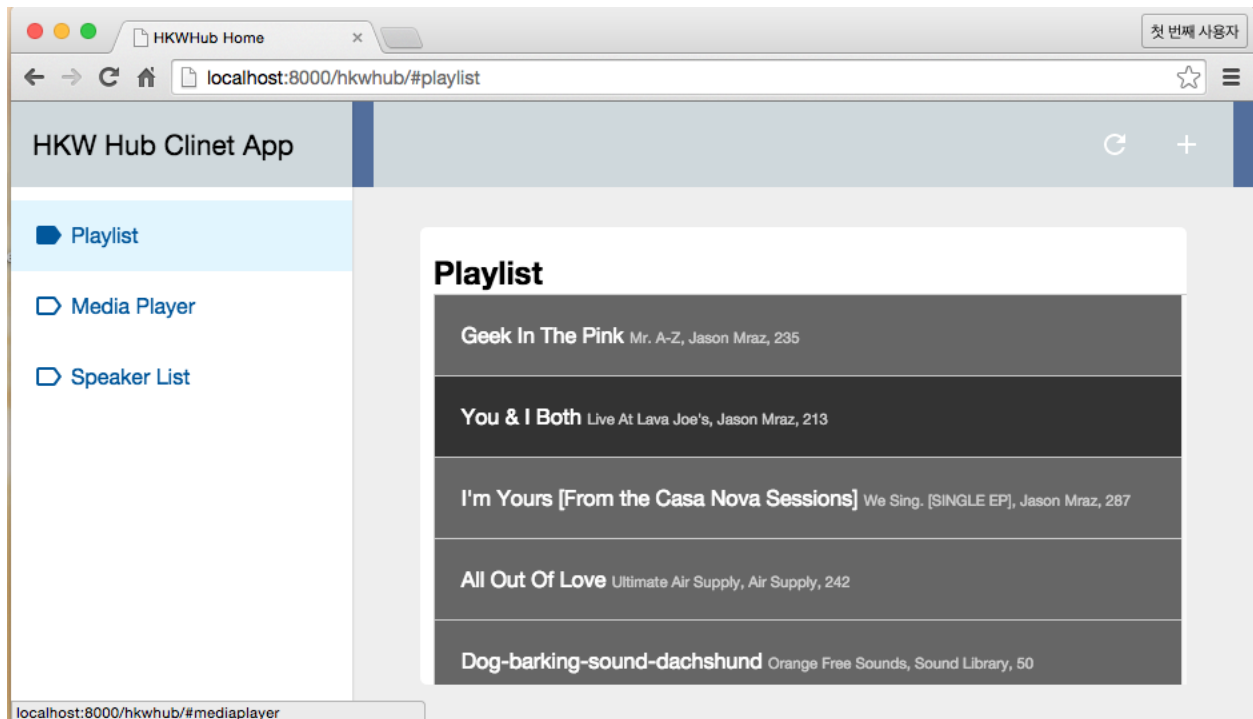
The following images are the screen captures of the HTML5 app.

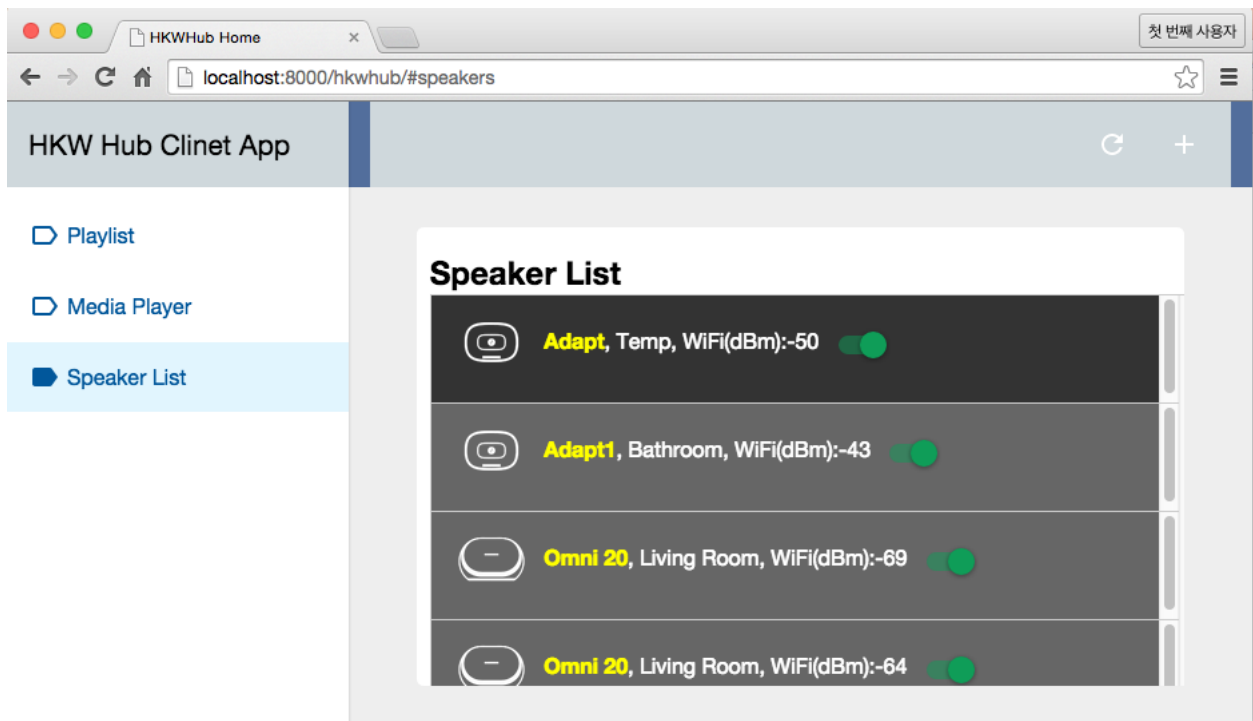
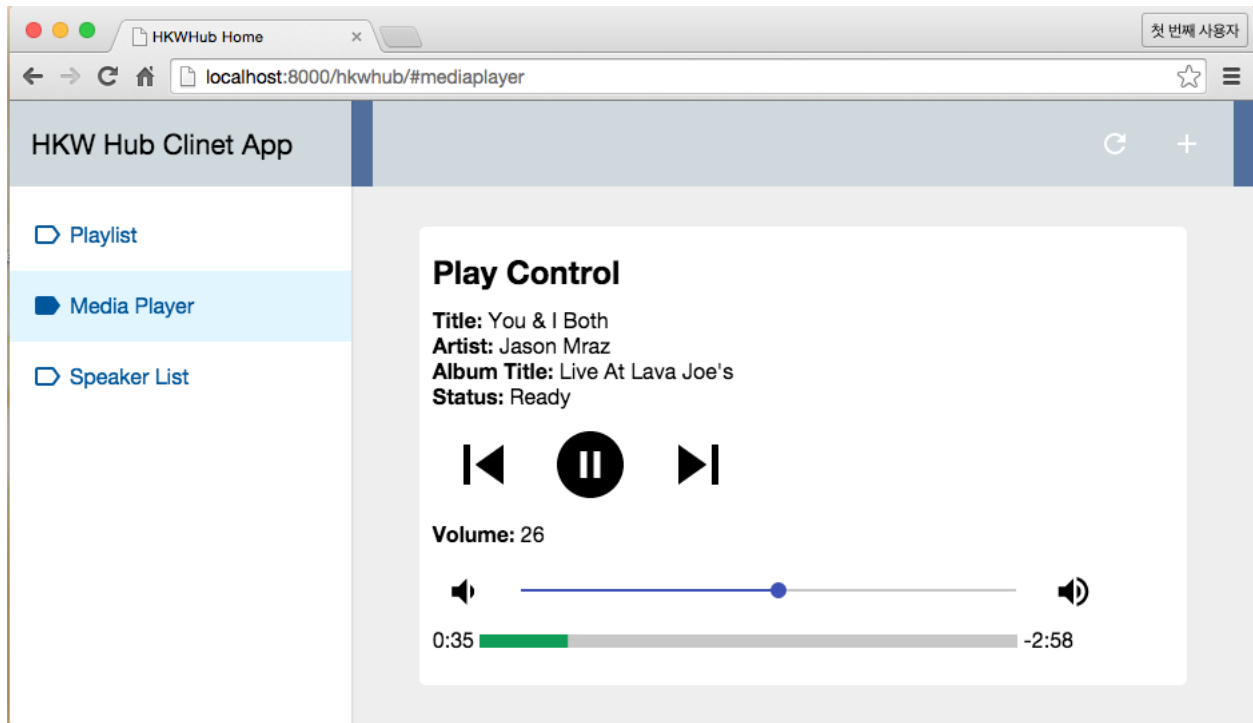
### HKWSimple (a very simple music player for getting started with HKWirelessHDSDK)

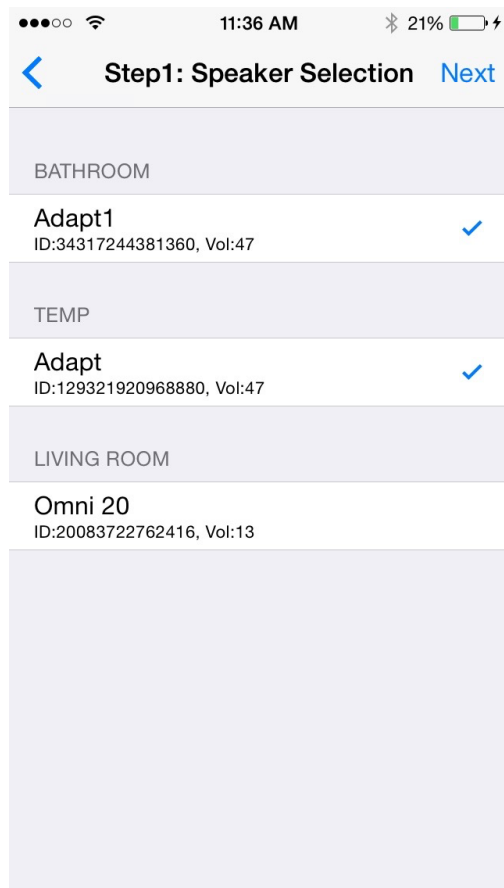
HKWSimple app is a simple music player that was created to explain how to create an app with HKWirelessHD SDK. This app is very simple, but contains key features of HKWirelessHDSDK, such as, manage speakers, control audio playback and volume, play local media files and also web streaming audio, and so on.

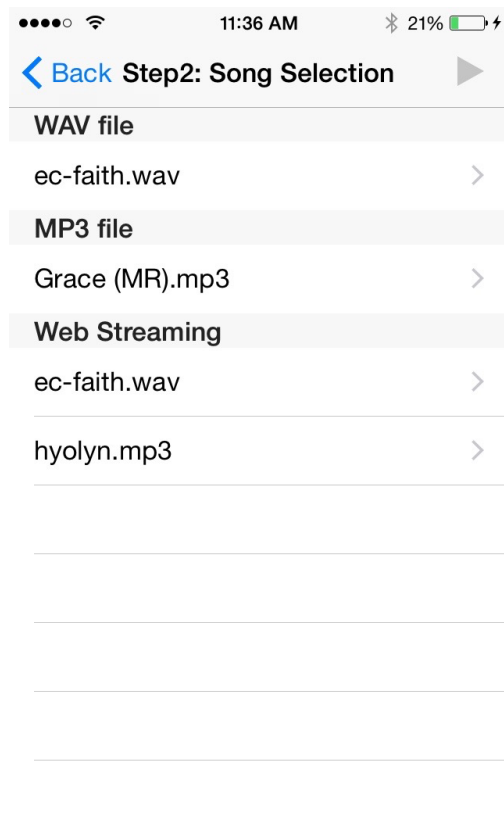
Just get started with the HKWSimple app to quickly build your own HKWirelessHD app!

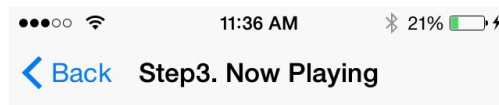












hyolyn.mp3

Now Playing

Play

Volume: 35

Up

Down

## 5.2 Getting Started Guide (iOS)

### 5.2.1 Getting Started Guide (iOS)

HKWirelessHD SDK supports both Objective-C and Swift. This document assumes that developer creates his/her app using the Swift language.

There are two versions of SDK - a normal version: HKWirelessHDS SDK - a lightweight version: HKWirelessHDSDKlw

Most of the features are common in the two version. The only difference is that the normal version includes an API for playing web streaming audio, while the lightweight version does not.

The reason we support the SDK as two separate versions is that we know that many developers want APIs for web streaming. To support this feature, we had to link a version of FFMPEG library with the SDK library. But, some developers may not want to link ffmpeg.

Please see the descriptions of each version below and make a proper choice for your app.

- **HKWirelessHD (normal version)**
  - Support web streaming music playback (streaming music from HTTP server, etc.)
  - need ffmpeg library (SDK contains a prebuilt ffmpeg library)
  - libz.dylib and libbz2.dylib are required when linking.
- **HKWirelessHDlw (lightweight version)**
  - **Do not support web streaming music playback**
  - No other library required

So, if you do not need web streaming music playback for your app, you may use HKWirelessHDlw (lightweight) version. Otherwise, you should use HKWirelessHD version.

In the section, we will use HKWSimple app as an example of using the normal version and HKPage app as an example of using the lightweight version.

### Project Setup with HKWirelessHDS SDKlw (lightweight version)

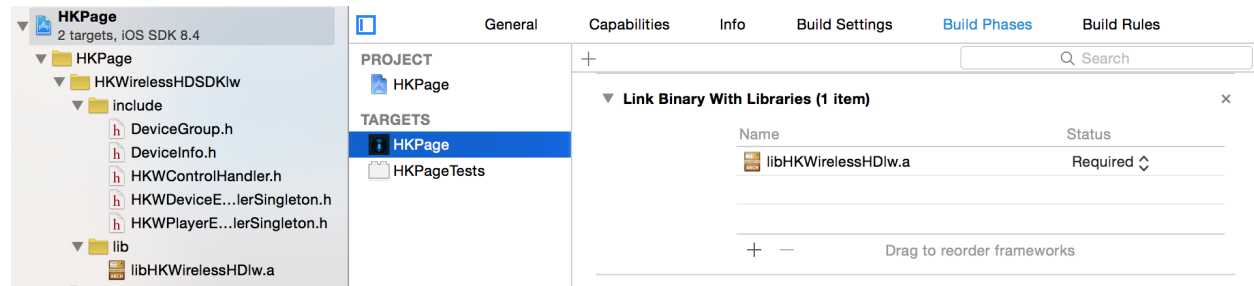
#### Include HKWirelessHDS SDKlw into your project

- Add *HKWirelessHDS SDKlw* to your project by dragging and dropping the HKWirelessHDS SDKlw folder into the project navigator.
- **When you get a dialog saying *Choose options for adding these files* :,**
  - Check the checkmark of *Copy items if needed*
  - Select *Create groups*, and click finish.
- By doing this, the include headers and libraries for HKWirelessHDS SDKlw are added to your project.

#### Make sure if libHKWirelessHDlw.a was added to your *Link Binary With Libraries*

- **Project Setting > Your Targets > Build Phases > Link Binary With Libraries**
  - Check if *libHKWirelessHDlw.a* was added to the list.
  - If it was not added, add it by clicking + and *Add Other...*

After adding the HKWirelessHDSDKIw folder into your project and adding libHKWirelessHD.a, the project navigator will look like as below:



### Add Swift Bridging Header

HKWirelessHD SDK has been written in C++ and Objective-C. Therefore, when you use the SDK in Swift, you should include Swift Bridging Header in your project. To do this:

1. Go to Project Setting > Build Setting > Swift Compiler - Code Generation > Objective-C Bridging Header



2. Add [My-Project-Name]/[My-Project-Name]-Bridging-Header.h
  - In our example, it looks like : HKPage/HKPage-Bridging-Header.h
3. Add the following lines in the header file.

```
#import "HKWControlHandler.h"
#import "HKWPlayerEventHandlerSingleton.h"
#import "HKWDeviceEventHandlerSingleton.h"
```

### Add -lstdc++ as linker flag

- Project Setting > Your Targets > Build Settings > Linking > Other Linker Flags
  - Add “-lstdc++” in the text field

### Project Setup with HKWirelessHDSDK (normal version)

#### Include HKWirelessHDSDK into your project

- Add *HKWirelessHDSDK* to your project by dragging and dropping the HKWirelessHDSDK folder into the project navigator.
- When you get a dialog saying *Choose options for adding these files* :,
  - Check the checkmark of *Copy items if needed*
  - Select *Create groups*, and click finish.

## ▼ Linking

Setting	HKWake
Bundle Loader	
Compatibility Version	
Current Library Version	
Dead Code Stripping	Yes ↕
Display Mangled Names	No ↕
Don't Create Position Independent Executables	No ↕
Don't Dead-Strip Inits and Terms	No ↕
Dynamic Library Install Name	
Dynamic Library Install Name Base	
Exported Symbols File	
Initialization Routine	
Link With Standard Libraries	Yes ↕
Mach-O Type	Executable ↕
Order File	
Other Librarian Flags	
▼ Other Linker Flags	-lstdc++
Debug	-lstdc++
Release	-lstdc++

- By doing this, the include headers and libraries for HKWirelessHDS SDK are added to your project.

Make sure if `libHKWirelessHD.a` was added to your *Link Binary With Libraries*

- **Project Setting > Your Targets > Build Phases > Link Binary With Libraries**
  - Check if `libHKWirelessHD.a` was added to the list.
  - If it was not added, add it by clicking '+' and *Add Other...*

Add `libz.dylib` and `libbz2.dylib` to your *Link Binary With Libraries*

- **Project Setting > Your Targets > Build Phases > Link Binary With Libraries**
  - Click '+' and find and add `libz.dylib` and `libbz2.dylib` to the list

After adding the HKWirelessHDS SDKw folder into your project and adding `libHKWirelessHD.a`, `libz.dylib` and `libbz2.dylib`, the project navigator will look like as below:

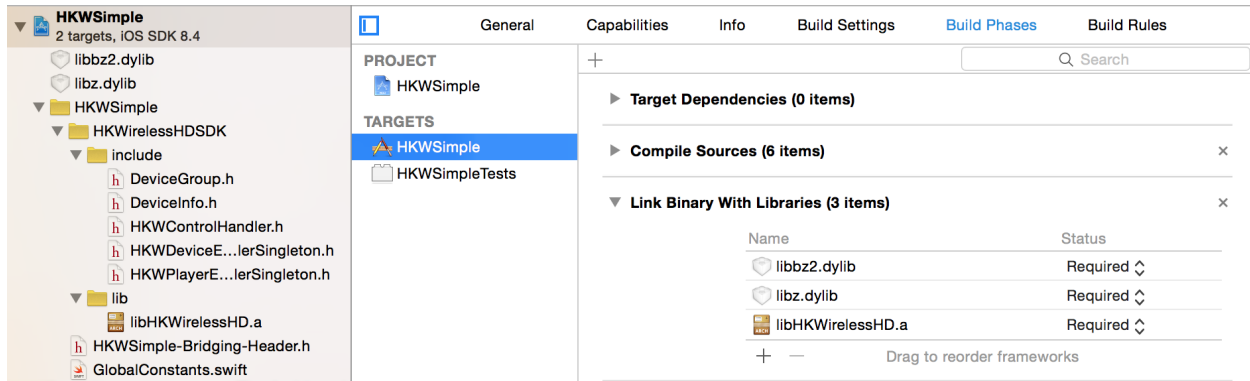
Add Swift Bridging Header and `-lstdc++` as linker flag

Follow the instruction for adding Swift bridging header and `-lstdc++` linker flag as described in the previous section.

## Creating a Sample Application (HKWSimple)

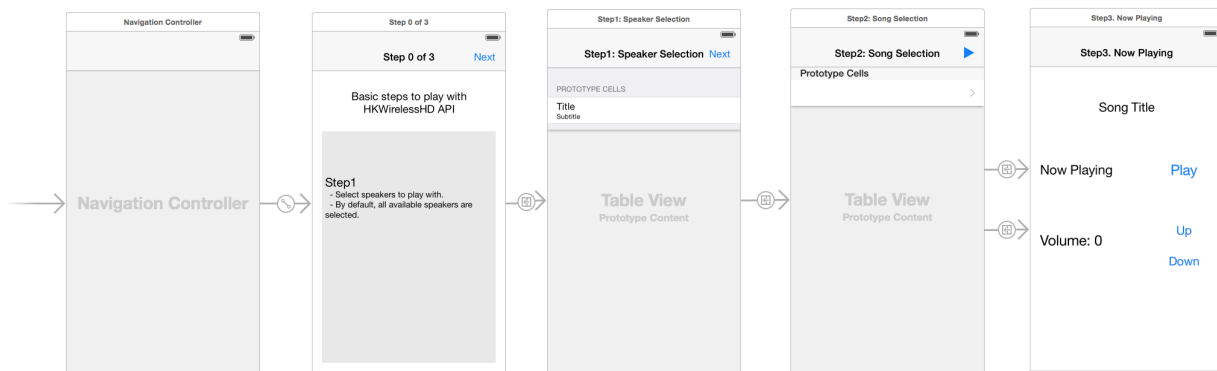
In this section, we explain how to create a HKWirelessHD iOS App. We will create a simple iOS app called **HK-WSimple** that can play WAV or MP3 file, and also play Web-based streaming music with HTTP protocol.





This app is so simple, so we highly recommend you to start with this app to understand how HKWirelessHD is working.

As shown in the figure, the app is composed of a sequence of UIViewController starting from a TableViewController showing a list of available speakers, and then a TableViewController showing a list of songs to play, and then finally a ViewController that shows a playback control panel with Play/Stop buttons and Volume control buttons.



## 1. Project Setup

For the project setup, please refer to the previous session of **Project Setup with HKWirelessHDS SDK (normal version)**.

## 2. Initialize HKWirelessHD Controller

In HKWSimple app, the initialization of HKWirelessHD Controller is done in the first ViewController called MainVC. When the app is launched, if HKWControlHandler is not initialized, then the app shows a dialog saying it is about to initialize the HKWControlHandler. This is done in `viewDidLoad()`. After that, in `viewDidAppear()`, the app actually tries to initialize HKWControlHandler. And it is successful, it dismisses the dialog. If not, it keeps showing the dialog so that the user can take an action.

As shown in the dialog message, if the app cannot initialize HKWControlHandler, then the reason would be one of the followings:

- The phone is not in Wi-Fi network.
- An app using HKWControlHandler is running on the same phone.

```

class MainVC: UIViewController {
    var g_alert: UIAlertController!

    override func viewDidLoad() {
        super.viewDidLoad()

        if !HKWControlHandler.sharedInstance().isInitialized() {
            // show the network initialization dialog
            g_alert = UIAlertController(title: "Initializing", message: "If this dialog does not disappear, please check the network connection")

            self.presentViewController(g_alert, animated: true, completion: nil)
        }

        override func viewWillAppear(animated: Bool) {

            if !HKWControlHandler.sharedInstance().initializing() && !HKWControlHandler.sharedInstance().isInitialized() {
                dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), {
                    if HKWControlHandler.sharedInstance().initializeHKWirelessController(kLicenseKeyGlobal) {
                        println("initializeHKWirelessControl failed : invalid license key")
                        return
                    }
                    println("initializeHKWirelessControl - OK");

                    // dismiss the network initialization dialog
                    if self.g_alert != nil {
                        self.g_alert.dismissViewControllerAnimated(true, completion: nil)
                    }
                })
            }
        }
    }
}

```

### 3. Get the list of available speakers

The list of speakers are presented in SpeakerSelectionTVC TableViewController. In order to show the list of speakers with the latest status information, the ViewController should receive events about device status. So it implements the delegate functions defined in HKWDeviceEventHandlerDelegate.

First, SpeakerSelectionTVC class should have HKWDeviceEventHandlerDelegate in its class declaration to be a delegate object of it.

```

class SpeakerSelectionTVC: UITableViewController, HKWDeviceEventHandlerDelegate {

```

In viewDidLoad(), the class will set the delegate of HKWDeviceEventHandler instance as itself. And then, it starts to refresh the device information, by calling startRefreshDeviceInfo().

```

    override func viewDidLoad() {
        super.viewDidLoad()
        HKWDeviceEventHandlerSingleton.sharedInstance().delegate = self
        HKWControlHandler.sharedInstance().startRefreshDeviceInfo()
    }

```

If the SpeakerSelectionTVC disappears, for example, by clicking **Back** button of Navigation Controller, it should stop

refreshing the device info, so it calls `stopRefreshDeviceInfo()` in `viewDidDisappear()`.

```
override func viewDidDisappear(animated: Bool) {
    super.viewDidDisappear(animated)
    HKWControlHandler.sharedInstance().stopRefreshDeviceInfo()
}
```

The follow codes are all about listing the speakers with their detailed information in the TableView. If a speaker is active, that is, the speaker belongs to playback session, then it shows a checkmark in the cell.

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    return HKWControlHandler.sharedInstance().getGroupCount()
}

override func tableView(tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
    return HKWControlHandler.sharedInstance().getDeviceCountInGroupIndex(section)
}

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("Speaker_Cell", forIndexPath: indexPath)
    cell.selectionStyle = UITableViewCellSelectionStyle.None
    var deviceInfo: DeviceInfo = HKWControlHandler.sharedInstance().getDeviceInfoByGroupIndexAndIndex(indexPath.section, indexPath.row)
    cell.textLabel?.text = deviceInfo.deviceName;
    var uniqueId: NSString = NSString(format: "ID:%llu, Vol:%d", deviceInfo.deviceId, deviceInfo.volume)
    cell.detailTextLabel?.text = uniqueId as String

    // Show the checkmark if the speaker is active
    if deviceInfo.active {
        cell.accessoryType = UITableViewCellAccessoryType.Checkmark
    } else {
        cell.accessoryType = UITableViewCellAccessoryType.None
    }
    return cell
}

override func tableView(tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    var header = HKWControlHandler.sharedInstance().getDeviceGroupNameByIndex(section);
    return header
}

override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    let cell = tableView.dequeueReusableCellWithIdentifier("Speaker_Cell", forIndexPath: indexPath)
    var deviceInfo: DeviceInfo = HKWControlHandler.sharedInstance().getDeviceInfoByGroupIndexAndIndex(indexPath.section, indexPath.row)
    if deviceInfo.active {
        HKWControlHandler.sharedInstance().removeDeviceFromSession(deviceInfo.deviceId)
        cell.accessoryType = UITableViewCellAccessoryType.Checkmark
    } else {
        HKWControlHandler.sharedInstance().addDeviceToSession(deviceInfo.deviceId)
        cell.accessoryType = UITableViewCellAccessoryType.None
    }
}
```

The follow codes are for handling events from Device Handler. In this example, it just redraw the table when it receives any device update events from the HKWControlHandler.

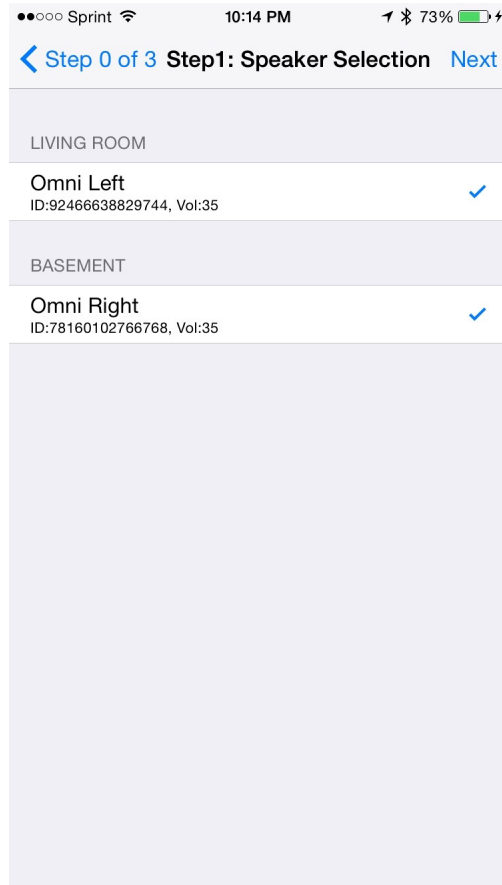
```
func hkwDeviceStateUpdated(deviceId: Int64, withReason reason: Int) {
    self.tableView.reloadData()
    nextBBI.enabled = !(HKWControlHandler.sharedInstance().getActiveDeviceCount() == 0)
}
```

```

func hkwErrorOccurred(errorCode: Int, withErrorMessage errorMesg: String!) {
    println("Error: \(errorMesg)")
}

```

The following figure shows a screen of the speaker list.



#### 4. Create the playlist to play

SongSelectionTVC shows the list of songs available for playback. It searches for the songs included in the app as bundle, and show the list of the songs in TableViewController. To be bundled within an app, mp3 or wav files should be included in the project setting. It also adds a list of songs with the URL information for web-based streaming.

In SongSelectionTVC, there is no use of HKWirelessHD APIs, because it is all about listing songs to play.

```

class SongSelectionTVC: UITableViewController {
    var g_wavFiles = [String]()
    var g_mp3Files = [String]()
    var curSection = 0
    var curRow = 0
    let serverUrlPrefix = "http://seonman.github.io/music/";
    var songList = ["ec-faith.wav", "hyolyn.mp3"]
    @IBOutlet var bbiNowPlaying: UIBarButtonItem!
}

```

```

override func viewDidLoad() {
    super.viewDidLoad()

    var bundleRoot = NSBundle.mainBundle().bundlePath
    var dirContents: NSArray = NSFileManager.defaultManager().contentsOfDirectoryAtPath(bundleRoot)
    var fltr: NSPredicate = NSPredicate(format: "self ENDSWITH '.wav'")
    g_wavFiles = dirContents.filteredArrayUsingPredicate(fltr) as! [String]

    for var i = 0; i < g_wavFiles.count; i++ {
        println("wav file: \(g_wavFiles[i])")
    }

    var fltr2: NSPredicate = NSPredicate(format: "self ENDSWITH '.mp3'")
    g_mp3Files = dirContents.filteredArrayUsingPredicate(fltr2) as! [String]

    for var i = 0; i < g_mp3Files.count; i++ {
        println("mp3 file: \(g_mp3Files[i])")
    }

    bbiNowPlaying.enabled = HKWControlHandler.sharedInstance().isPlaying()
}

override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    return 3
}

override func tableView(tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
    if section == 0 {
        return g_wavFiles.count
    } else if section == 1 {
        return g_mp3Files.count
    } else if section == 2 {
        return songList.count
    } else {
        return 0
    }
}

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("SongTitle_Cell", forIndexPath: indexPath)
    if indexPath.section == 0 {
        cell.textLabel?.text = g_wavFiles[indexPath.row]
    } else if indexPath.section == 1 {
        cell.textLabel?.text = g_mp3Files[indexPath.row]
    } else {
        cell.textLabel?.text = songList[indexPath.row]
    }
    return cell
}

override func tableView(tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    if section == 0 {
        return "WAV file"
    } else if section == 1 {
        return "MP3 file"
    } else {
        return "Web Streaming"
    }
}

```

```

    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if segue.identifier == "Song_Cell" {
            let section = self.tableView.indexPathForSelectedRow()?.section
            curSection = section!
            let row = self.tableView.indexPathForSelectedRow()?.row
            curRow = row!

            let destTVC:NowPlayingVC = segue.destinationViewController as! NowPlayingVC
            destTVC.section = curSection
            destTVC.row = curRow
            if curSection == 0 {
                destTVC.songTitle = g_wavFiles[curRow]
            } else if curSection == 1 {
                destTVC.songTitle = g_mp3Files[curRow]
            } else {
                destTVC.songTitle = songList[curRow]
                destTVC.songUrl = serverUrlPrefix + songList[curRow]
                destTVC.serverUrl = serverUrlPrefix
            }

            destTVC.viewLoadByCellSelection = true
            destTVC.nsWavPath = NSBundle.mainBundle().bundlePath.stringByAppendingPathComponent(destTVC.songTitle)
            destTVC.songSelectionTVC = self
        }
        else if segue.identifier == "NowPlaying_BBI" {
            let destTVC:NowPlayingVC = segue.destinationViewController as! NowPlayingVC
            if curSection == 0 {
                destTVC.songTitle = g_wavFiles[curRow]
            } else if curSection == 1 {
                destTVC.songTitle = g_mp3Files[curRow]
            } else {
                destTVC.songTitle = songList[curRow]
                destTVC.songUrl = serverUrlPrefix + songList[curRow]
                destTVC.serverUrl = serverUrlPrefix
            }

            destTVC.viewLoadByCellSelection = false
            destTVC.nsWavPath = NSBundle.mainBundle().bundlePath.stringByAppendingPathComponent(destTVC.songTitle)
            destTVC.songSelectionTVC = self
        }
    }
}

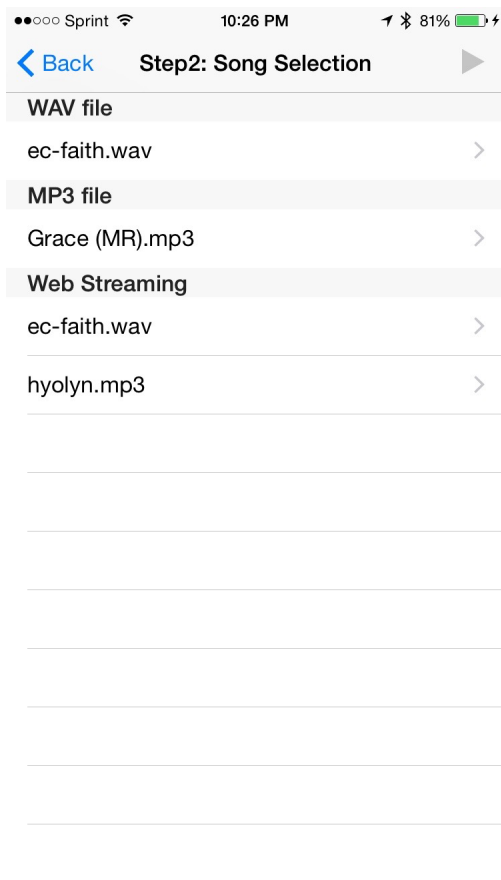
```

The following figure shows an example of the SongSelectionTVC screen.

## 5. Playback and Volume Control

NowPlayingVC controls the playback and volume level, so we have to use the related HKWirelessHDSDK APIs. All playback and volume control related events are sent via HKWPlayerEventHandlerDelegate, so NowPlayingVC class should implement the delegate.

Add HKWPlayerEventHandlerDelegate in the class definition.



```
class NowPlayingVC: UIViewController, HKWPlayerEventHandlerDelegate {
```

In `viewDidLoad()`, the `NowPlayingVC` should set itself to the delegate attribute of `HKWPlayerEventHandlerSingleton` object, so that all delegate functions implemented in this class can be referenced by the `HKWPlayerEventHandler`.

```
override func viewDidLoad() {
    super.viewDidLoad()
    HKWPlayerEventHandlerSingleton.sharedInstance().delegate = self

    labelSongTitle.text = songTitle
    curVolume = HKWControlHandler.sharedInstance().getVolume()
    labelAverageVolume.text = "Volume: \(curVolume)"

    if viewLoadByCellSelection {
        playCurrentTitle()
    } else {
        if HKWControlHandler.sharedInstance().isPlaying() {
            btnPlayStop.setTitle("Stop", forState: UIControlState.Normal)
            labelStatus.text = "Now Playing"
        } else {
            btnPlayStop.setTitle("Play", forState: UIControlState.Normal)
            labelStatus.text = "Play Stopped"
        }
    }
}
```

The followings are several variable to show the list of songs in the `ViewController`.

```
var row = 0
var section = 0
var songTitle = ""
var nsWavPath = ""
var viewLoadByCellSelection = false
var songSelectionTVC: SongSelectionTVC!
var curVolume: Int = 50
var songUrl = ""
var serverUrl = ""

var g_alert: UIAlertController!

@IBOutlet var labelSongTitle: UILabel!
@IBOutlet var btnPlayStop: UIButton!
@IBOutlet var labelAverageVolume: UILabel!
@IBOutlet var btnVolumeDown: UIButton!
@IBOutlet var btnVolumeUp: UIButton!
@IBOutlet var labelStatus: UILabel!
```

The following codes are to play and pause the media file.

```
@IBAction func playOrStop(sender: UIButton) {
    if HKWControlHandler.sharedInstance().isPlaying() {
        HKWControlHandler.sharedInstance().pause()
        labelStatus.text = "Play Stopped"

        btnPlayStop.setTitle("Play", forState: UIControlState.Normal)
    }
}
```



```

    else {
        playCurrentTitle()
        labelStatus.text = "Now Playing"
    }
}

func playCurrentTitle() {
    // just to be sure that there is no running playback
    HKWControlHandler.sharedInstance().stop()

    if section == 0 {
        if HKWControlHandler.sharedInstance().playWAV(nsWavPath) {
            // now playing, so change the icon to "STOP"
            btnPlayStop.setTitle("Stop", forState: UIControlState.Normal)
        }
    } else if section == 1 {
        let assetUrl = NSURL(fileURLWithPath: nsWavPath)

        if HKWControlHandler.sharedInstance().playCAF(assetUrl, songName: songTitle, resumeFlag: false) {
            // now playing, so change the icon to "STOP"
            btnPlayStop.setTitle("Stop", forState: UIControlState.Normal)
        }
    } else {
        playStreaming()
    }
    songSelectionTVC.bbiNowPlaying.enabled = true
}

func playStreaming() {
    HKWControlHandler.sharedInstance().playStreamingMedia(songUrl, withCallback: {(bool result) -> Void} {
        if result == false {
            println("playStreamingMedia: failed")
            self.btnPlayStop.selected = false

            self.g_alert = UIAlertController(title: "Warning", message: "Playing streaming media failed", preferredStyle: UIAlertControllerStyle.Alert)
            self.g_alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.Default, handler: nil))
            self.presentViewController(self.g_alert, animated: true, completion: nil)
        } else {
            println("playStreamingMedia: successful")
            self.btnPlayStop.setTitle("Stop", forState: UIControlState.Normal)
        }
    })
}

```

The following codes are to control volumes, e.g up and down.

```

@IBAction func volumeUp(sender: UIButton) {
    curVolume += 5
    if curVolume > 50 {
        curVolume = 50
    }
    HKWControlHandler.sharedInstance().setVolume(curVolume)
    labelAverageVolume.text = "Volume: \(curVolume)"
}

@IBAction func volumeDown(sender: UIButton) {
    curVolume -= 5
    if curVolume < 0 {

```

```
        curVolume = 0
    }
    HKWControlHandler.sharedInstance().setVolume(curVolume)
    labelAverageVolume.text = "Volume: \(curVolume)"
}
```

The following codes implements the delegate functions defined by HKWPlayerEventHandlerDelegate. Here, we can add codes to handle the events of PlayEnded and VolumeChanged.

```
func hkwPlayEnded() {
    btnPlayStop.setTitle("Play", forState: UIControlState.Normal)
    songSelectionTVC.bbiNowPlaying.enabled = false
}

func hkwDeviceVolumeChanged(deviceId: Int64, deviceVolume: Int, withAverageVolume avgVolume: Int) {
    println("avgVolume: \(avgVolume)")
    curVolume = avgVolume
}
}
```

## 5.3 Programming Guide (iOS)

### 5.3.1 Programming Guide (iOS)

In this chapter, 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 HKWPlayer app.

All APIs can be accessed through the singleton object pointer of HKWControlHandler. The first thing you have to do is create an instance of HKWControlHandler and use it to invoke the APIs you want to use.

In HKWirelessHD SDK, all the APIs are defined in Objective-C. However, the examples in this document are written in Swift. In case you use Swift for app development, you need to create a Swift bridging header file to make Objective-C APIs visible to Swift code.

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

#### Initialization

##### Initialize HKWirelessHD Control Handler and start the Wireless Audio

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(). initializeHKWirelessController() takes a string of license key as parameter. Every developer who signs up for Harman developer community will receive a license key. If you have not received it yet, then just use the license key specified in the sample code for temporary use until your own license key is delivered to you.

This API requires a boolean parameter that specifies if all speakers in the network will be added right after initialization. If it is **true**, then all speakers will be added to session, so you can play music without adding speakers separately. If it is **false**, none of speakers will be added to playback session, so you need to add speakers separately to session to play music using APIs like AddDeviceToSession().

```

if HKWControlHandler.sharedInstance().isInitialized() == false {
    // HKWControlHandler has not been initialized before. Initialize it now.
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), {
        if HKWControlHandler.sharedInstance().initializeHKWirelessController(g_licenseKey, w
            println("initializeHKWirelessControl failed")
            return
        }
        ...
    })
}

```

**Note:** Note that `intializeHKWirelessController()` is a blocking call. So, it will not return until the caller 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 non-blocking behavior, you should call this function asynchronously by running it in a separate thread, not within the main thread. So, it would be nice to present a dialog to user before calling `initializeHKWirelessController()` to notice that the app will wait until HKWirelessHD network is available.

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

## Getting the information of speakers

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 `hkwDeviceStateUpdated()` delegate 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.

## 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, Wi-Fi signal strength, etc.

You can retrieve all the attributes of device (speaker) information through DeviceInfo object, and it is specified in DeviceInfo.h. The following table shows the list of information that DeviceInfo 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 DeviceInfo.h 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	CLong-Long	the unique ID of the speaker	Fixed (in manufacturing)	No
deviceName	String	the name of the speaker	Variable	Yes
groupId	CLong-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
ipAddress	String	the IP address as String	Fixed (when network setup)	No
port	String	the port number	Fixed (when network setup)	No
macAddress	String	the mac address as String	Fixed (in manufacturing)	No
volume	Int	the volume level value (0 to 50)	Variable	Yes
active	Bool	indicates if added to the current playback session	Variable	Yes
wifiSignal-Strength	Int	Wi-Fi strength in dBm scale, -100 (low) to 0 (high)	Variable	No
role	Int	the role definition (stereo or 5.1 channel)	Variable	Yes
version	String	the firmware version number as String	Fixed (when firmware update)	No
balance	Int	the balance value in stereo mode. -6 to 6, 0 is neutral	Variable	Yes
isPlaying	Bool	indicates whether the speaker is playing or not	Variable	No
channelType	Int	the channel type: 1 is stereo.	Variable	Yes
isMaster	Bool	indicates if it is the master in stereo or group mode	Variable	Yes

As shown in the table above, some of the attributes can be set by the APIs. And some attributes change during the runtime, so the app should keep the latest value of the attributes by calling corresponding APIs or by callback functions.

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

```
let deviceInfo: DeviceInfo = HKWControlHandler.sharedInstance().getDeviceInfoByGroupIndexAndDeviceIndex(deviceIndex, groupIndex)

println("deviceName: \(deviceInfo.deviceName)")
println("groupName: \(deviceInfo.groupName)")
println("volume: \(deviceInfo.volume)")
println("deviceId: \(deviceInfo.deviceId)")
println("deviceActive: \(deviceInfo.active)")
println("deviceModel: \(deviceInfo.modelName)")
...
```

**Getting a speaker (device) information** HKWControlHandler maintains the list of speaker internally. Each speaker information can be retrieved either by specifying the index in the table, or by specifying the index of group and the index of member inside of the group.

#### Get the speaker information from the table

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

```
- (DeviceInfo *) getDeviceInfoByIndex:(NSInteger)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 `TableViewCell`.

### 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.

```
- (DeviceInfo *) getDeviceInfoByGroupIndexAndDeviceIndex:(NSInteger) groupIndex
                                     deviceIndex:(NSInteger) 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 `TableViewCell`. For example, to show a speaker information in two section `TableView`, the `groupIndex` can correspond to the section number, and `deviceIndex` can correspond to the row number.

### 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.

```
- (DeviceInfo *) getDeviceInfoById:(long long) deviceId;
```

## Refreshing device status information

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

However, in our current implementation, the event dispatching initiated by speaker 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 speakers 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 ...
g_HKWControlHandler.startRefreshDeviceInfo()

... ..

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

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

## 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 current playback session.

You can check whether or not a speaker is currently added to the current playback session by check the “active” attribute of `DeviceInfo` object (in `DeviceInfo.h`).

```
/*! Indicates if the speaker is active (added to the current playback session) */
@property (nonatomic, readonly) BOOL active;
```

**Add a speaker to a session (to play on)** Use `addDeviceToSession()` to add a speaker to the current playback session.

```
- (BOOL) addDeviceToSession:(long long) deviceId;
```

For example,

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

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

---

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

---

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

```
- (BOOL) removeDeviceFromSession:(long long) deviceId;
```

For example,

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

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

---

**Note:** A speaker can be removed from the current on-going playback session anytime.

---

---

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

---

## Play Audio File

### Play CAF file (MP3, WAV, etc.)

If one or more speakers are available to the session, or if there is at least one speaker active, you can start to play a song.

The playback is based on the Apple Core Audio framework. So, the supported audio file and data formats by HK-WirelessHDS SDK are the same as those supported by Apple’s Core Audio framework. The detailed information is available at [iOS audio file formats](#). According to the Apple developer documentation, CAF supports AIFF (.aif, .aiff), CAF (.caf), MPEG-1, Layer 3 (.mp3), MPEG-2 or MPEG-4 ADTS (.aac), MPEG-4 Audio (.mp4, .m4a), and WAVE (.wav).

```
- (BOOL) playCAF:(NSURL *)assetURL songName:(NSString*)songName resumeFlag:(BOOL)resumeFlag;
```

To play a song, you should prepare a `AssetURL` using `NSURL` first. Here is an example:

```
let assetUrl = NSURL(fileURKWithPath: nsPath)

HKWControlHandler.sharedInstance().playCAF(assetUrl, songName: songTitle, resumeFlag: 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.

```
- (bool) playCAFFromCertainTime:(NSURL *)assetURL
                                songName:(NSString*) songName
                                startTime:(NSInteger) startTime;
```

Here, `startTime` is in second.

`playCAF()` and `playCAFFromCertainTime()` can play both WAV and MP3 audio file. In case of WAV, it is played without conversion. In case of MP3, it is converted to PCM format first, and then played.

To play WAF audio file, use `playWAV()`.

```
- (bool) playWAV:(NSString*) wavPath;
```

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

```
nsWavPath = NSBundle mainBundle().bundlePath.stringByAppendingPathComponent(songTitle)

HKWControlHandler.sharedInstance().playWAV(nsWavPath)
```

**Note:** `playCAF()` and `playCAFFromCertainTime()` cannot play an audio file in Apple's iTunes Match service. Songs should reside locally on the device for playback. So, it would be nice to check if the song resides on the device when your app gets the `MPMediaItem` of a song from `MediaPicker`.

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 `HKWControlHandler.h`)

```
- (HKPlayerState) getPlayerState;
```

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

```
- (bool) isPlaying;
```

## Play Web Streaming Music

**Note:** This API is **NOT** supported by `HKWirelessHDSDKlw` (lightweight version of `HKWirelessHDSDK`). It is only supported by `HKWirelessHD SDK`.

Use `playStreamingMedia()` to play a streaming media. It uses a parameter of `streamingMediaUrl` to specify the URL of the media file in the streaming service. It starts with a protocol name, such as "`http://`" or "`rtsp://`". Currently, `http:`, `rtsp:`, and `mms:` are supported. The supported file format is `mp3`, `m4a`, `wav`.

`completedCallback` is a callback that returns the result of the call, that is, if the call is successful or not. If it cannot find the media file on the server or some other errors occur, then it return `false`.

```
- (void)playStreamingMedia:(NSString *)streamingMediaUrl withCallback:(void (^)(bool result))complete
```

---

**Note:** When you stop playing the streaming music, you must use `stop()`, not `pause()`.

---

### Playback controls Stop playback

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

```
- (void) stop;
```

For example,

```
HKWControlHandler.sharedInstance().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.

---

**Note:** If the current playback is stopped by calling `stop()`, the playback cannot be resumed. Resuming is only possible when the playback is paused by calling `pause()`.

---

### Pause playback

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

The playback paused by calling `pause()` can be resumed.

```
- (void) pause;
```

For example,

```
HKWControlHandler.sharedInstance().pause()
```

### Resume playback

To resume the paused playback, use `playCAF()` with the parameter `resumeFlag` set to `true`. The API is described earlier in this section.

### 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:** 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:** 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.

---



**Set volume to all speakers** Use `setVolume()` to set the same volume level to all speakers.

```
- (void) setVolume:(NSInteger)volume;
```

For example,

```
// set volume level to 25 to all speakers
var volume = 25
HKWControlHandler.sharedInstance().setVolume(volume)
```

**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 long)deviceId volume:(NSInteger)volume;
```

For example,

```
// set volume level to 25 to a speaker
var volume = 25
HKWControlHandler.sharedInstance().setVolumeDevice(deviceId volume:volume)
```

**Get volume of all speakers** Use `getVolume()` to get the average volume level from all speakers.

```
- (NSInteger) getVolume;
```

For example,

```
var averageVolume = HKWControlHandler.sharedInstance().getVolume()
```

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

For example,

```
var volume = HKWControlHandler.sharedInstance().getDeviceVolume(deviceId)
```

**Mute and Unmute** Use `mute()` to mute the current playback. The volume level turns to 0.

Use `unmute()` to unmute the current muted playback. After `unmute()`, the volume level returns to the original volume level before `mute()` is called.

Use `isMuted()` to check if the current playback is muted or not.

## 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.

### Change speaker name

Use `setDeviceName()` to change the speaker name.

---

**Note:** You cannot set the device name by setting `deviceName` property value directly. The property is read-only.

---

```
- (void) setDeviceName:(long long)deviceId deviceName:(NSString *)deviceName;
```

For example,

```
HKWControlHandler.sharedInstance().setDeviceName(deviceId, deviceName:"My Omni10")
```

---

**Note:** 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 `hkwErrorOccurred()` delegate defined in `HKWDeviceEventHandlerDelegate` protocol.

---

### 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 long)deviceId groupName:(NSString *)groupName;
```

For example,

```
HKWControlHandler.sharedInstance().setDeviceGroupName(deviceId, groupName:"Living Room")
```

---

**Note:** If you change the group name of a speaker, then the list of speakers of the group automatically changes.

---

### Remove a speaker from a group

Use `removeDeviceFromGroup()` to remove the speaker from the currently 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 long)deviceId;
```

For example,

```
HKWControlHandler.sharedInstance().removeDeviceFromGroup(deviceId)
```

## Delegate APIs for events handling

In HKWirelessHD, the communication between user's phone and speakers are done in asynchronous way. Therefore, some API calls from HKWControlHandler can take a little time to take effects on the speaker side. Similarly, 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 **delegate protocols**. So, your app needs to implement the delegate protocols accordingly to receive and handle the events from HKWControlHandler.

There are two kinds of delegate protocols for event handling.

- **HKWDeviceEventHandlerDelegate (defined in HKWDeviceEventHandlerSingleton.h)**
  - This delegate protocol defines several APIs for receiving events about status changes or error from speakers.
- **HKWPlayerEventHandlerDelegate (defined in HKWPlayerEventHandlerSingleton.h)**
  - This delegate protocol defines several APIs for receiving events about playback status and volume control.

### HKWDeviceEventHandlerDelegate

#### hkWDeviceStateUpdated (required)

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 HKWControlHandler.h.

Note that volume level change does not trigger this call. The volume update is reported by `hkWVolumeLevelChanged()` callback.

```
-(void) kWDeviceStateUpdated:(long long)deviceId withReason:(NSInteger) 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.

A most common usage for this function is to invoke `tableView.reloadData()` to update the list of speakers in a table view controller, as shown below.

```
func kWDeviceStateUpdated(deviceId: CLongLong, withReason reason: Int) {
    self.tableView.reloadData()
}
```

#### hkWErrorOccurred (required)

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 HKWControlHandler.h.

```
-(void) kWErrorOccurred:(NSInteger)errorCode withErrorMessage:(NSString*)errorMsg;
```

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

### HKWPlayerEventHandlerDelegate kWPlayEnded (required)

This function is invoked when the current playback has ended.

```
-(void) hkwPlayEnded;
```

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

#### **hkwDeviceVolumeChanged (optional)**

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) hkwDeviceVolumeChanged:(long long)deviceId deviceVolume:(NSInteger)deviceVolume withAverageVo
```

---

**Note:** When speaker volume is changed by a call to `setVolume()`, then all the speakers are set to a new volume level, and this function can be used to get the new volume level value.

---

**hkwPlaybackStateChanged (optional)** This function is invoked when playback state is changed during the playback. The callback delivers the `playState` value as parameter.

```
-(void) hkwPlaybackStateChanged:(NSInteger)playState;
```

#### **hkwPlaybackTimeChanged (optional)**

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) hkwPlaybackTimeChanged:(NSInteger)timeElapsed;
```

**How to implement the delegate protocols** To make your ViewController as delegate, that is, to receive the events from HKWControlHandler in your ViewController, your ViewController class has to implement the protocol by specifying the delegate protocol name in the class definition as below:

```
class SpeakerListViewController: UIViewController, UITableViewDataSource,
    SpeakerTableViewCellDelegate, HKWDeviceEventHandlerDelegate {
    ...
}
```

And then, the ViewController class should set itself to `delegate` attribute of the handler singleton class as below:

```
HKWDeviceEventHandlerSingleton.sharedInstance().delegate = self
```

Note that during the runtime, only one instance of the event handler for HKWControlHandler is instantiated, so it is designed as a singleton. It can be retrieved by calling `sharedInstance()` to the class.

#### **For device event handler**

```
HKWDeviceEventHandlerSingleton.sharedInstance().delegate
```

#### **For player event handler**

```
HKWPlayerEventHandlerSingleton.sharedInstance().delegate
```

---

**Note:** Since there is only one instance of delegator for each delegate, if you set `delegate` in several different places of your app, then the latest setting will override the `delegate` value, and the previous setting will be overridden. So,

you need to set the delegate every time you show a ViewController, and one right place to set the delegate is inside `viewDidAppear()` in ViewController class as shown below.

```
override func viewDidAppear(animated: Bool) {
    HKWDeviceEventHandlerSingleton.sharedInstance().delegate = self
    HKWPlayerEventHandlerSingleton.sharedInstance().delegate = self

    ...
}
```

## 5.4 Architecture Overview (iOS)

### 5.4.1 Architecture Overview (iOS)

#### Overview

#### Overall Configuration

There are two types of entities in HKWirelessHD audio streaming - one is source device and the other is destination device. Source device sends audio stream to destination devices (speakers), and destination devices receive the audio stream from source and play it. In HKWirelessHD audio, audio streaming is in a one-to-many way. That is, there is one single source device sending an audio stream, and multiple destination devices receive the audio stream by synchronizing with other speakers.

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

Source device can be iOS device, such as iPhone and iPad, and destination devices are Harman Kardon Omni speakers (Omni Adapt, Omni 10, Omni 20, Omni Bar, etc.) You can find more information on HK Omni speakers at <http://www.harmankardon.com/content?ContentID=omni-v2>.

#### Use of HKWirelessHD API to stream audio to Omni Speakers

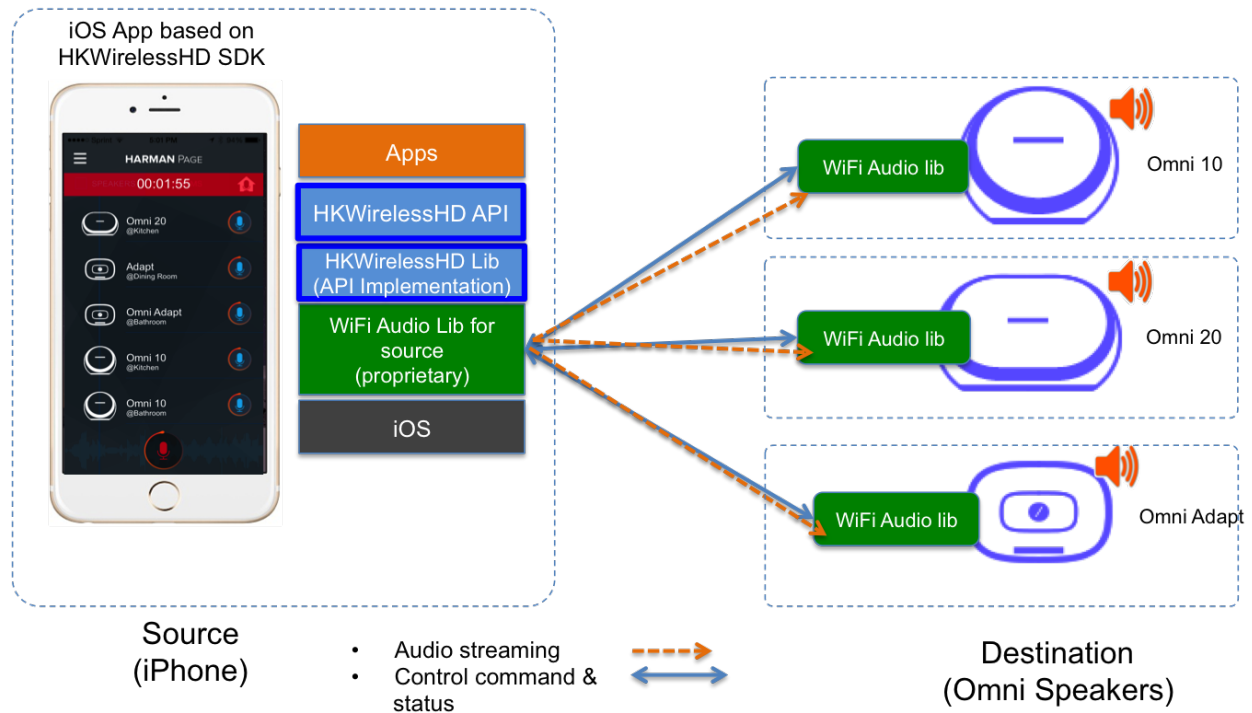
To send audio stream to destination devices, an App has to use HKWirelessHD API. HKWirelessHD SDK provides the library of the APIs for arm32/64bit architecture.

Note that as of date of writing, no x86 architecture is supported, so developer cannot use iOS Simulator for running the app with HKWirelessHD API.

#### Communication channels between source and destinations

As shown in the figure above, there are two kind of communications between a 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 commands from the source to the destinations to control the device, like volume control, etc.



– **Destination device can also send commands 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.
- \* User can add Omni 10 or Omni 20 speaker to the current running 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 the device information and the status data of a 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 changes of its attributes, whether or not it is playing music, Wi-Fi signal strength, volume change, etc.

### Asynchronous Communication

The communication between the source device and the destination speakers are accomplished in asynchronous way. Asynchronous behavior is natural because all the commands and status updates are executed in a way like RPC (Remote Procedure Call). Even more, audio streaming always involves some amount of buffering of audio data packets, so the timing gap between the source and the destinations is inevitable.

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. This is because several steps are involved to be attached the network and become discoverable by other speakers in the same network. 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 device starts music playback and streaming to destination speakers, actual playback in destination speakers occurs 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 speakers, actual volume changes occur a few millisecond later.

## 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 DeviceInfo instances. HKWControlHandler instance maintains a list of DeviceInfo objects, each of which corresponds to a speaker found in the network.

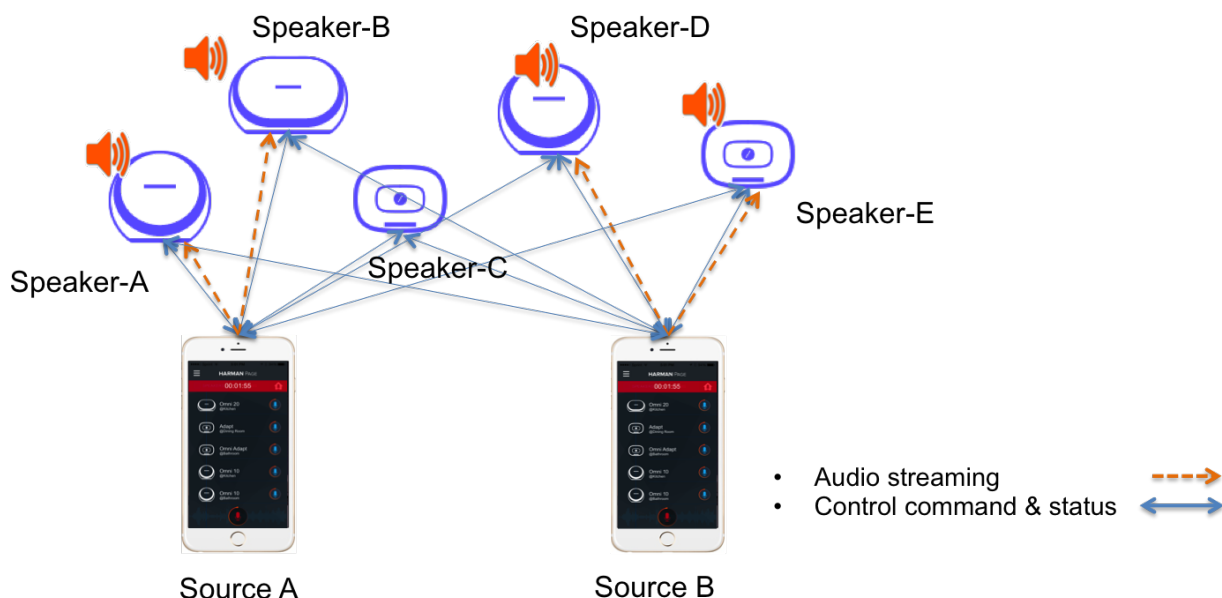
The detailed description on each attribute in the device are described in DeviceInfo.h.

## Visibility of Speakers

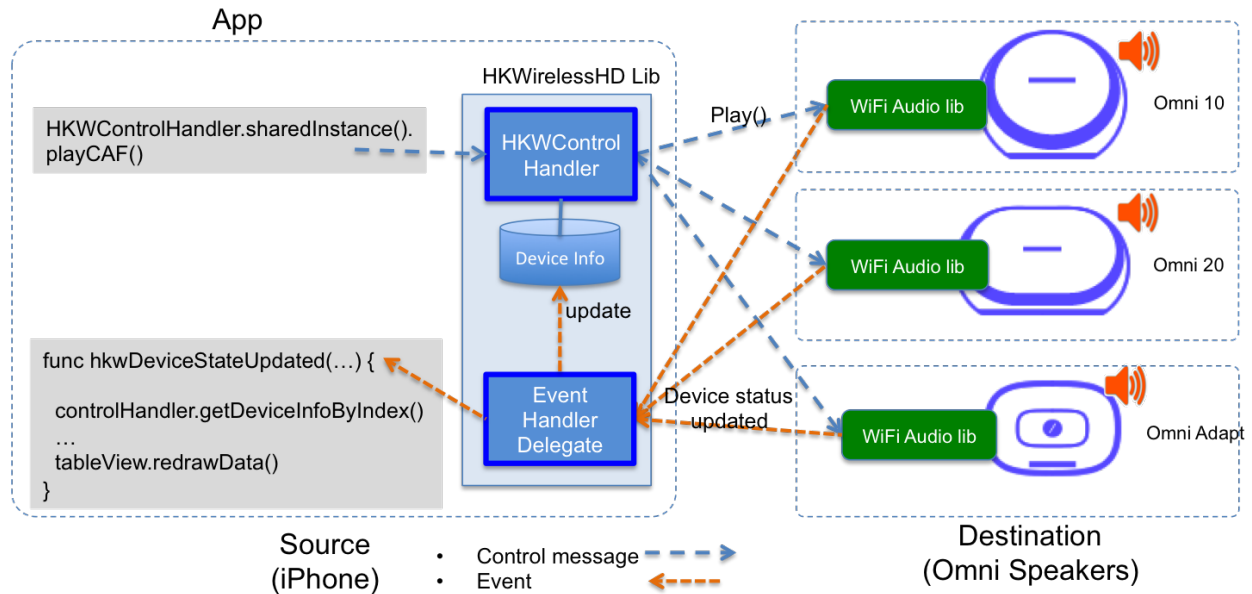
Any speakers in a network are visible to source devices (iOS devices) if a source device successfully initializes the HKWControlHandler when it starts up. Source device can be multiple. This means, even in the case that a speaker is being used by a source device, the status of each speaker is also visible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

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 HKWControlHandler, 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 DeviceInfo.h to return a boolean value indicating if the speaker is being playing audio or not, regardless that which source audio stream comes from.



## Controlling Speakers and Handling the Events from Speakers



### Controlling speakers

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

`HKWControlHandler` is a singleton object, and it can be acquired by calling as below. Note that there is no initializer API for the object. The `sharedInstance()` singleton API will create and initialize it if there is no instance in the runtime.

```
HKWControlHandler.sharedInstance()
```

### Handling events from speakers

On the other hand, the events from speakers are sent to the app through Delegate protocol APIs. By implementing the event handler delegate functions (in `ViewController` class in most cases), 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 two delegate protocols:

- **HKWDeviceEventHandlerDelegate (defined in `HKWDeviceEventHandlerSingleton.h`)**
  - All the events related to the status of speakers. There are two cases the event is sent:
    - \* device status updated
    - \* error occurred
  - To register an object as the delegate, do as below (self is the object that implements the delegate protocols):



```
* HKWDeviceEventHandlerSingleton.sharedInstance().delete = self
```

- **HKWPlayerEventHandlerDelegate (defined in HKWPlayerEventHandlerSingleton.h)**

- **All the events related to play music.**

- \* play ended
    - \* playback state changed
    - \* playback time changed
    - \* volume changed

- **To register an object as the delegate, do as below (self is the object that implements the delegate protocols):**

```
* HKWPlayerEventHandlerSingleton.sharedInstance().delete = self
```

## 5.5 API Documentation (iOS)

### 5.5.1 API Documentation (iOS)

#### Initialization

Refer to HKWControlHandler.h in the SDK.

#### **initializeHKWirelessController()**

Initializes and starts HKWirelessHD controller. This API requires a boolean parameter that specifies if all the speakers in the network will be added right after initialization. If it is **true**, then all speakers will be added to session, so you can play music without adding speakers. If it is **false**, none of speakers will be added to playback session, so you need to add speakers separately to session to play music.

This API also requires a license key as string. If the input license key fails in key validation, then the API returns -1. If it is successful, return 0.

The license key will be delivered to you once you sign on to developer.harman.com. Until it is delivered, you may use the license key included in the sample apps.

---

**Note:** This is a blocking call, and 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 using the HKWireless controller, it waits until the other app releases the controller.

If you needs non-blocking behavior, you should call this API asynchronously using other thread.

---

**Signature:** - (NSInteger) initializeHKWirelessController:(NSString\*)licenseKey  
withSpeakersAdded: (BOOL)withSpeakersAdded;

#### **Parameters:**

- (NSString\*) licenseKey - a string containing the license key
- (BOOL) withSpeakersAdded - boolean value specifying if all speakers are added to session after initialization

**Returns:** `NSInteger` - an integer indicating success (0 or `HKW_INIT_SUCCESS`) or failure (-1 or `HKW_INIT_FAILURE_LICENSE_INVALID`)

---

### `isInitialized()`

Checks if `HKWirelessHD` controller is already initialised.

**Signature:** - (BOOL) `isInitialized`;

**Returns:** `BOOL` - boolean value indicating if `HKWirelessController` has been initialized or not.

---

### `initializing()`

Checks if `HKWirelessHD` controller is being initialised.

**Signature:** - (BOOL) `initializing`;

**Returns:** `BOOL` - boolean value indicating if `HKWirelessController` is being initialized

## Refreshing Speaker Information

### `refreshDeviceInfoOnce()`

Refreshes the device status one time. The device information will be refreshed and updated to `DeviceInfo` objects (defined in `DeviceInfo.h`). If there is any update on `DeviceInfo`, then `hkwDeviceStateUpdated` delegate function defined by `HKWDeviceEventHandlerDelegate` will be called. From there, you can see which device has been updated and what was the reason of the update.

**Signature:** - (void) `refreshDeviceInfoOnce`;

**Returns:** `void`

---

### `startRefreshDeviceInfo()`

Starts to keep refreshing `DeviceInfo` every two seconds. It continues until `stopRefreshDeviceInfo()` is called.

**Signature:** - (void) `startRefreshDeviceInfo`;

**Returns:** `void`

---

### `stopRefreshDeviceInfo()`

Stops refreshing `DeviceInfo`, which was initiated by `startRefreshDeviceInfo()`.

**Signature:** - (void) `stopRefreshDeviceInfo`;

**Returns:** `void`

---

## Playback Control

### playCAF()

Plays a CAF audio file in local storage. If it is successful, `hkwPlaybackStateChanged()` delegate function will be called and return the status value of `HKPlayerState.EPlayerState_Play`.

The playback uses Apple Core Audio framework. So, the types of supported audio files and data formats by HK-WirelessHDSDK are the same as those supported by Apple's Core Audio framework. The detailed information is available at [iOS audio file formats](#). According to the Apple developer documentation, CAF supports AIFF (.aif, .aiff), CAF (.caf), MPEG-1, Layer 3 (.mp3), MPEG-2 or MPEG-4 ADTS (.aac), MPEG-4 Audio (.mp4, .m4a), and WAVE (.wav).

**Signature:** - (BOOL) playCAF:(NSURL \*)assetURL songName:(NSString\*)songName  
resumeFlag:(BOOL)resumeFlag;

#### Parameters:

- (NSURL\*) assetURL - NSURL to the audio file
- (NSString\*) songName - the song name to be played. The songName is used internally to save the temporary PCM file generated from the original audio file.
- (BOOL) resumeFlag - a boolean specifying if the playback should resume from the point where it was paused in the previous playback. When you want to start a song from the beginning, resumeFlag must be false. If the previous playback was stopped not paused, then the playback will start from the beginning even with resumeFlag true.

**Returns:** BOOL - boolean value indicating success or failure

---

### playCAFFromCertainTime()

Plays a CAF audio file beginning from a certain time of playback specified by `startTime`. For example, you can start a song from the point of 10 seconds of the song. `hkwPlaybackStateChanged()` delegate function will be called and return the status value of `HKPlayerState.EPlayerState_Play`.

**Signature:** - (BOOL) playCAFFromCertainTime:(NSURL \*)assetURL  
songName:(NSString\*)songName startTime:(NSInteger)startTime;

#### Parameters:

- (NSURL \*)assetURL - NSURL to the audio file.
- (NSString\*) songName - the song name to be played. The songName is used internally to save the temporary PCM file generated from the original audio file.
- (NSInteger) startTime - time in second that specifies the start time.

**Returns:** BOOL - boolean value indicating success or failure

---

### playWAV()

Plays a WAV file. `hkwPlaybackStateChanged` delegate function will be called and return the status value of `HKPlayerState.EPlayerState_Play`.

**Signature:** - (BOOL) playWAV:(NSString\*)wavPath;

**Returns:** BOOL - boolean value indicating success or failure

### **playStreamingMedia()**

Plays a streaming media from web server. Because this API takes a little while to get the result of play because of all networking stuffs, the API just returns immediately, and instead it takes a callback to be called when the result is ready. You should take the result of the call inside of the callback.

**Signature:** - (void)playStreamingMedia:(NSString \*)streamingMediaUrl  
withCallback:(void (^)(bool result))completedCallback;

#### **Parameters:**

- (NSString\*)streamingMediaUrl - a string that specifies the URL of the streaming media source. It starts with a protocol name, such as “[http://](#)” or “[rtsp://](#)”. Currently, http, rtsp, and mms are supported. The supported file format is mp3, m4a, and wav.
- (void (^)(bool result))completedCallback - a callback that returns the result of the playback

**Returns:** void

---

### **pause()**

Pauses the current playback. hkwPlaybackStateChanged() delegate function will be called and return the status value of HKPlayerState.EPlayerState\_Pause. Once the playback is paused, it can resume by calling playCAF() with resumeFlag true and the same songName. A playback by playStreamingMedia() cannot be paused. If pause() is called, then the playback will stop.

**Signature:** - (void) pause;

**Returns:** void

---

### **stop()**

Stops the current playback. hkwPlaybackStateChanged() delegate function will be called and return the status value of HKPlayerState.EPlayerState\_Stop.

**Signature:** - (void) stop;

**Returns:** void

---

### **isPlaying()**

Checks if the player is playing some audio or not.

**Signature:** - (bool) isPlaying;

**Returns:** BOOL - boolean value indicating if the player is playing or now.

---

### **getPlayerState()**

Gets the current state of playback.

**Signature:** - (HKPlayerState) getPlayerState;

**Returns:** HKPlayState - indicates the current player state.

---

## **Volume Control**

### **setVolume()**

Sets a volume level to all speakers. The same volume level is set to all speakers.

The range of volume level is 0 to the maximumVolumeLevel (currently, 50) defined by getMaximumVolumeLevel().

Setting volume is asynchronous call. So, the effect of the API call will occur after a few milliseconds. The hkwDeviceVolumeChanged() delegate function defined in HKWPlayerEventHandlerDelegate (in HKWPlayerEventHandlerSingleton.h) will be called when the volume level of the specified speaker has actually changed.

If the volume is being muted by calling mute(), the volume level is changed from the original volume level before mute.

**Signature:** - (void) setVolume:(NSInteger)volume;

**Parameters:**

- (NSInteger) volume - the volume level to set

**Returns:** void

---

### **setVolumeDevice()**

Set a volume level to a speaker specified by device ID. The range of volume level is 0 to the maximumVolumeLevel (currently, 50) defined by getMaximumVolumeLevel(). setVolume is asynchronous call. So, the effect of the API call will occur after a few milliseconds. The hkwDeviceVolumeChanged() delegate function defined in HKWPlayerEventHandlerDelegate (in HKWPlayerEventHandlerSingleton.h) will be called when the volume level of the specified speaker has actually changed.

If the volume is being muted by calling mute(), the volume level is changed from the original volume level before mute.

**Signature:** - (void) setVolumeDevice:(long long)deviceId volume:(NSInteger)volume;

**Parameters:**

- (long long) deviceId - the device ID of the speaker
- (NSInteger) volume - the volume level to set

**Returns:** void

---

### `getVolume()`

Gets the average volume level for all devices.

**Signature:** - (NSInteger) getVolume;

**Returns:** NSInteger - the average volume level of all speakers

---

### `getDeviceVolume()`

Gets the volume level of the specified speaker.

**Signature:** - (NSInteger) getDeviceVolume:(long long)deviceId;

**Parameters:** - (long long) deviceId - the deviceId of the speaker inquired.

**Returns:** NSInteger - the device volume level

---

### `getMaximumVolumeLevel()`

Returns the maximum volume level that the system provides. Currently, it is 50.

**Signature:** - (NSInteger) getMaximumVolumeLevel;

**Returns:** NSInteger - the maximum volume level

---

### `mute()`

Mutes the current volume of all speakers.

**Signature:** - (void) mute;

**Returns:** void

---

### `unmute()`

Unmute the volume. It returns the previous volume level before mute.

**Signature:** - (void) unmute;

**Returns:** void

---

### `isMuted()`

Check if volume is muted or not.

**Signature:** - (bool) isMuted;

**Returns:** BOOL - the Boolean value indicating if mute is on or not.

---

## Device (Speaker) Management

### addDeviceToSession()

Adds a speaker to the current playback session. The added speaker will start playing audio. This can be done during the audio playback.

**Signature:** - (BOOL) addDeviceToSession:(long long) deviceId;

**Parameters:**

- (long long) deviceId - The ID of the device to add

**Returns:** BOOL - boolean value indicating whether the addition is successful or not.

---

### removeDeviceFromSession()

Removes a speaker from the current playback session. The removed speaker will stop playing audio. This can be done during the audio playback.

**Signature:** - (BOOL) removeDeviceFromSession:(long long) deviceId;

**Parameters:**

- (long long) deviceId - The ID of the device to remove

**Returns:** BOOL - boolean value indicating whether the removal is successful or not.

---

### getDeviceCount()

Gets the number of all speakers in the HKWirelessHD network.

**Signature:** - (NSInteger) getDeviceCount;

**Returns:** NSInteger - the number of speakers.

---

### getGroupCount()

Gets the number of the groups of speakers.

**Signature:** - (NSInteger) getGroupCount;

**Returns:** NSInteger - the number of the groups

---

### `getDeviceCountInGroupIndex()`

Gets the number of the speakers that belongs to a group specified by the index.

**Signature:** - (NSInteger) getDeviceCountInGroupIndex:(NSInteger) groupIndex;

**Parameters:**

- (NSInteger) groupIndex - the index of the group looking for. It starts from 0 to (GroupCount-1).

**Returns:** NSInteger - the number of speakers

---

### `getDeviceInfoByGroupIndexAndDeviceIndex()`

Returns the DeviceInfo object (defined in DeviceInfo.h) pointed by groupIndex and deviceIndex. This API is useful to find a DeviceInfo that will be shown in a TableViewCell. For example, to show a speaker information in two section TableView, the groupIndex can correspond to the section number, and deviceIndex can correspond to the row number.

**Signature:** - (DeviceInfo \*) getDeviceInfoByGroupIndexAndDeviceIndex:(NSInteger) groupIndex deviceIndex:(NSInteger) deviceIndex;

**Parameters:**

- (NSInteger) groupIndex - The index of the group where the speaker belongs to.
- (NSInteger) deviceIndex - The index of the speaker in the group.

**Returns:** DeviceInfo\* - the DeviceInfo object

---

### `getDeviceInfoByIndex()`

Returns the DeviceInfo object pointed by deviceIndex from the table containing all speakers. The range of deviceIndex will be 0 to (deviceCount - 1).

**Signature:** - (DeviceInfo \*) getDeviceInfoByIndex:(NSInteger) deviceIndex;

**Parameters:** - (NSInteger) deviceIndex - The index of the device from the table with all devices.

**Returns:** DeviceInfo\* - the DeviceInfo object

---

### `getDeviceGroupByDeviceId()`

Returns the object of the DeviceGroup that a speaker belongs to.

**Signature:** - (DeviceGroup \*) getDeviceGroupByDeviceId:(long long) deviceId;

**Parameters:**

- (long long) - the ID of the speaker that belongs to a DeviceGroup

**Returns:** DeviceGroup\* - the DeviceGroup object

---



**getDeviceInfoById()**

Finds a DeviceInfo from the table by device Id. It is useful to retrieve DeviceInfo with a particular deviceId.

**Signature:** - (DeviceInfo \*) getDeviceInfoById:(long long) deviceId;

**Parameters:**

- (long long) deviceId - the ID of the device we are looking for.

**Returns:** DeviceInfo\* - The DeviceInfo object

---

**isDeviceAvailable()**

Checks whether the specified speaker is available on the network or not.

**Signature:** - (BOOL) isDeviceAvailable:(long long) deviceId;

**Parameters:** - (long long) deviceId - The ID of the speaker

**Returns:** (BOOL) - boolean indicating if the speaker is available or not.

---

**isDeviceActive()**

Checks whether the speaker is active (added to the current playback session) or not.

**Signature:** - (BOOL) isDeviceActive:(long long) deviceId;

**Parameters:** - (long long) deviceId - The ID of the speaker

**Returns:** (BOOL) - boolean indicating if the speaker is active or not.

---

**removeDeviceFromGroup()**

Removes (ungroup) the specified speaker from the currently belonged group. It is done internally by setting the GroupName as “harman” (which is factory default device name, and implies Not-Assigned.).

**Signature:** - (void) removeDeviceFromGroup:(long long) deviceId;

**Parameters:** - (long long) deviceId - The ID of the speaker to ungroup.

**Returns:** void

---

### `getDeviceGroupByIndex()`

Gets the DeviceGroup object by index.

**Signature:** - `(DeviceGroup *)getDeviceGroupByIndex:(NSInteger)groupIndex;`

**Parameters:**

- `(NSInteger)groupIndex` - the index of the group

**Returns:** `DeviceGroup*` - The object of DeviceGroup

---

### `getDeviceGroupByGroupId()`

Gets DeviceGroup by group ID.

**Signature:** - `(DeviceGroup *)getDeviceGroupByGroupId:(long long)groupId;`

**Parameters:**

- `(long long)groupId` - the ID of the group

**Returns:** `DeviceGroup*` - the object of device group.

---

### `getDeviceGroupNameByIndex()`

Gets the name of the DeviceGroup by index.

**Signature:** - `(NSString *)getDeviceGroupNameByIndex:(NSInteger)groupIndex;`

**Parameters:**

- `(NSInteger)groupIndex` - the index of the group in the group table.

**Returns:** `NSString*` - the string of group name

---

### `getDeviceGroupIdByIndex()`

Gets the ID of the DeviceGroup by index.

**Signature:** - `(long long)getDeviceGroupIdByIndex:(NSInteger)groupIndex;`

**Parameters:**

- `(NSInteger)groupIndex` - the index of the group in the table

**Returns:** `long long` - the group id

---

### setDeviceName()

Sets device name to a speaker. Note that you cannot set the device name by setting “deviceName” property directly. The property is read-only.

---

**Note:** If you set a device name while the a playback is running, the speaker stops playing and return error.

---

**Signature:** - (void) setDeviceName:(long long)deviceId deviceName:(NSString \*)deviceName;

**Parameters:**

- (NSInteger) deviceId - The ID of the device
- (NSString\*) deviceName - The name of the device to set

**Returns:** void

---

### setDeviceGroupName()

Sets device group to a speaker with Group name. Note that you cannot set the group name by setting “groupName” property directly. The property is read-only.

---

**Note:** If you set a group name while the a playback is running, the speaker stops playing and return error.

---

**Signature:** - (void) setDeviceGroupName:(long long)deviceId groupName:(NSString \*)groupName;

**Parameters:**

- (NSInteger) deviceId - The ID of the device
- (NSString\*) groupName - The name of the group name to set

**Returns:** void

---

### setDeviceRole()

Sets the role for the speaker. The role information is used to define which part of audio channel the speaker takes for the playback.

**Signature:** - (void) setDeviceRole:(long long)deviceId role:(int)role;

**Parameters:**

- (long long) deviceId - The id of the device
- (int) role - the interger value indicating the role of the speaker. The possible options are listed in the HKRole enumeration type. The default value is EMono (21).

**Returns:** void

---

### **getActiveDeviceCount()**

Gets the number of active speakers (the speakers that are being added to the current playback session.)

**Signature:** - (NSInteger) getActiveDeviceCount;

**Returns:** NSInteger - the number of active devices

---

### **getActiveGroupCount()**

Gets the number of active groups. An active group is defined as a group that all the speakers that belongs to a group are active. If even one of the speakers in the same group is inactive, then the group is inactive.

**Signature:** - (NSInteger) getActiveGroupCount;

**Returns:** NSInteger - the number of active groups

---

### **refreshDeviceWiFiSignal()**

Refreshes the device's Wifi Signal strength value. This is asynchronous call, and the result of refreshing will come a few milliseconds later. The new Wifi signal strength value will be reported by `hkwDeviceStateUpdated()` delegate function defined in `HKWDeviceEventHandlerDelegate` (in `HKWDeviceEventHandlerSingleton.h`). In `hkwDeviceStateUpdated()`, you should get the wifi signal value by getting `wifiSignalStrength` attribute of `DeviceInfo` object.

**Signature:** - (void)refreshDeviceWiFiSignal:(long long)deviceId;

**Parameters:**

- (long long)deviceId - The ID of the device

**Returns:** void

---

### **getWifiSignalStrengthType()**

Gets Wifi signal strength type by signal value.

**Signature:** - (HKWifiSingalStrength)getWifiSignalStrengthType:(NSInteger)wifiSignal;

**Parameters:**

- (NSInteger)wifiSignal - the wifi signal value

**Returns:** HKWifiSingalStrength - a value of HKWifiSignalStrength type

---

## HKWDeviceEventHandlerSingleton

### sharedInstance()

Returns the singleton object to HKWDeviceEventHandler.

**Signature:** +(HKWDeviceEventHandlerSingleton\*)sharedInstance;

---

### delegate

The delegate attribute to be set by an object to be a HKWDeviceEventHandlerDelegate.

```
@property (nonatomic, weak) id<HKWDeviceEventHandlerDelegate> delegate;
```

---

## HKWDeviceEventHandlerDelegate

### hkwdDeviceStateUpdated() - required

Invoked when some of device information have been changed for any speakers. It is also invoked when the network is disconnected and no speakers are available any longer, or when the network becomes up from down, so speakers in the network become added to the HKWirelessHD network. <p>The information monitored includes device status (active or inactive), model name, group name, and wifi signal strengths.<p>Volume change does not trigger this call. The volume update is reported by CallbackVolumeLevelChanged.

**Signature:** -(void)hkwdDeviceStateUpdated:(long long)deviceId withReason:(NSInteger)reason;

**Parameters:**

- (long long)deviceId - the deviceId of the speaker
- (NSInteger)reason - the reason code about the updated status

**Returns:** void

---

### hkwdErrorOccurred() - required

Invoked when an error occurs.

**Signature:** -(void)hkwdErrorOccurred:(NSInteger)errorCode withErrorMessage:(NSString\*)errorMesg

**Parameters:**

- (NSInteger)errorCode - an integer value indicating error code.
- (NSString\*)errorMesg - a string value containing a description about the error.

**Returns:** void

---

## HKWPlayerEventHandlerSingleton

### sharedInstance()

Returns the singleton object to HKWPlayerEventHandler.

**Signature:** +(HKWPlayerEventHandlerSingleton\*) sharedInstance;

---

### delegate

The delegate attribute to be set by an object to be a HKWPlayerEventHandlerDelegate.

```
@property (nonatomic, weak) id<HKWPlayerEventHandlerDelegate> delegate;
```

## HKWPlayerEventHandlerDelegate

### hkwPlayEnded() - required

This is called when the current playback has ended.

**Signature:** -(void) hkwPlayEnded;

---

### hkwDeviceVolumeChanged() - optional

This is called when volume level has changed for any spekaers.

**Signature:** -(void) hkwDeviceVolumeChanged: (long long) deviceId  
deviceVolume: (NSInteger) deviceVolume withAverageVolume: (NSInteger) avgVolume;

**Parameters:**

- (long long) deviceId - the device unique ID (long long type)
- (NSInteger) deviceVolume - the volume level of the device (speaker)
- (NSInteger) avgVolume - the average volume level

### hkwPlaybackStateChanged() - optional

This is called when player state has changed during the playback.

**Signature:** -(void) hkwPlaybackStateChanged: (NSInteger) playState;

**Parameters:**

- (NSInteger) playState - The player state
-

**hkwPlaybackTimeChanged() - optional**

This is called when the current time of playback has changed. It is called every one second.

**Signature:** – (void)hkWPlaybackTimeChanged: (NSInteger)timeElapsed;

**Parameters:**

- (NSInteger)timeElapsed - the time (in second) passed since the beginning of the playback.

## 5.6 HKWHub Specification (iOS)

### 5.6.1 HKWHub App - Connecting your Omni speakers to any 3rd party services or devices (iOS)

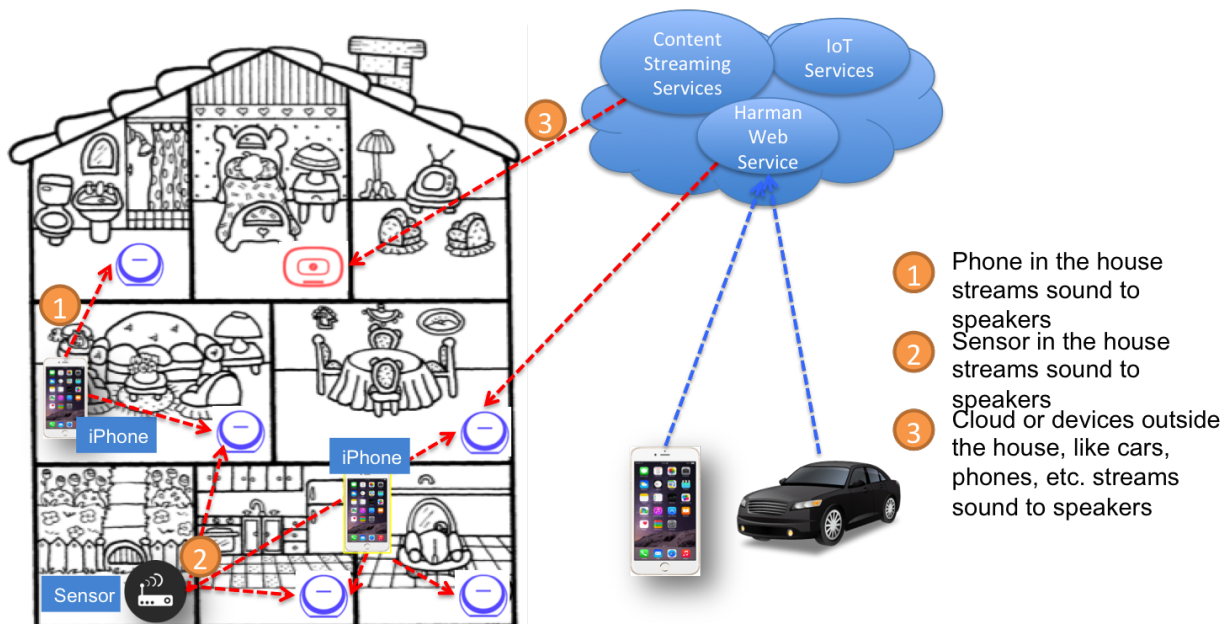
#### Overview of HKWHub

HKWHub app is an iOS that that uses HKWirelessHD SDK and acts as a Web Hub that handles HTTP requests to control speakers and stream music. It enables any type of “connected” devices (like sensors or smart devices) and cloud-based service to connect HK Omni speakers and stream music.

**Note:** Please note that HKWHub app is an on-going project, and not yet ready for production. We hope developers play around with this app and implement their own apps or services by integrating many other IoT devices or services and adding intelligence to the Hub app.

Any feedback or request for new APIs and features are always welcome.

#### Use Cases



## HKWHub App

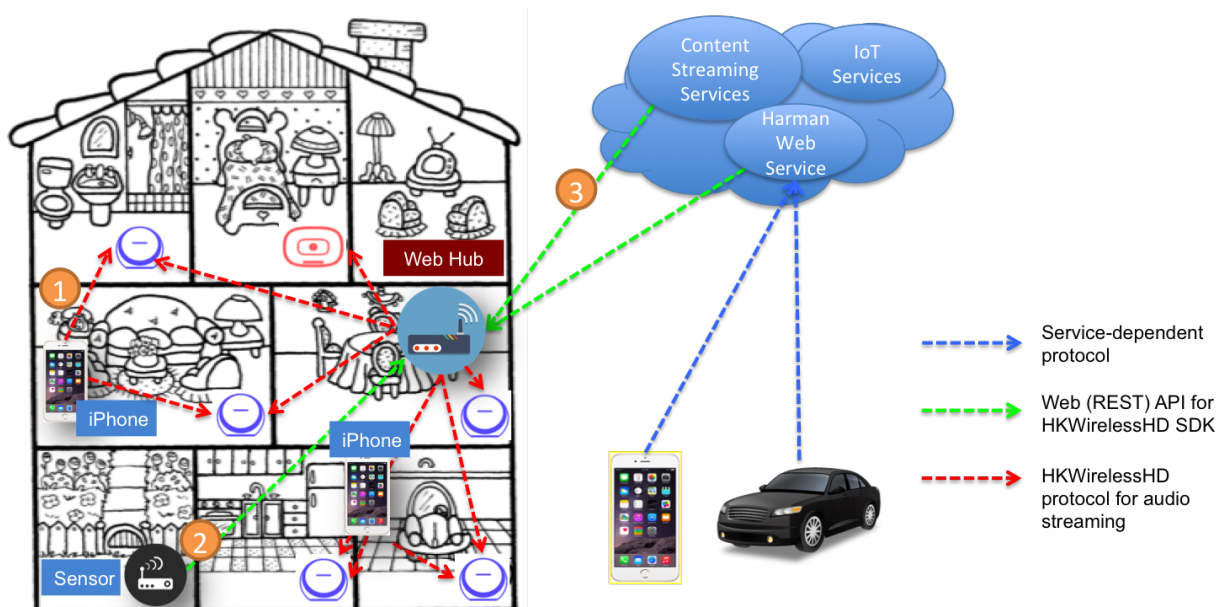
Web Hub handles all the requests from and response to smart devices, sensors or clouds to control audio playback with wireless speakers in the house.

### Features

- **Supports integration with cloud-based services, smart devices or sensors**
  - Receives the requests from clouds (web service) outside or sensors in the house
  - Translates the requests into HKWirelessHD commands and controls the speakers based on the requests.
  - Sends response with status of speakers to the cloud if necessary
- **Central Music Playlist manager**
  - Maintain user's media list from iOS local music library or streaming services, like MixRadio, etc.
  - Maintain a collection of sound files used for IoT use cases, like door bell, etc.

### Usage

- User may place an iOS device on the cradle and run WebHub app. Then the app acts as Web Hub. (A stationary device in a house like AppleTV can be a nice iOS device for WebHub.)



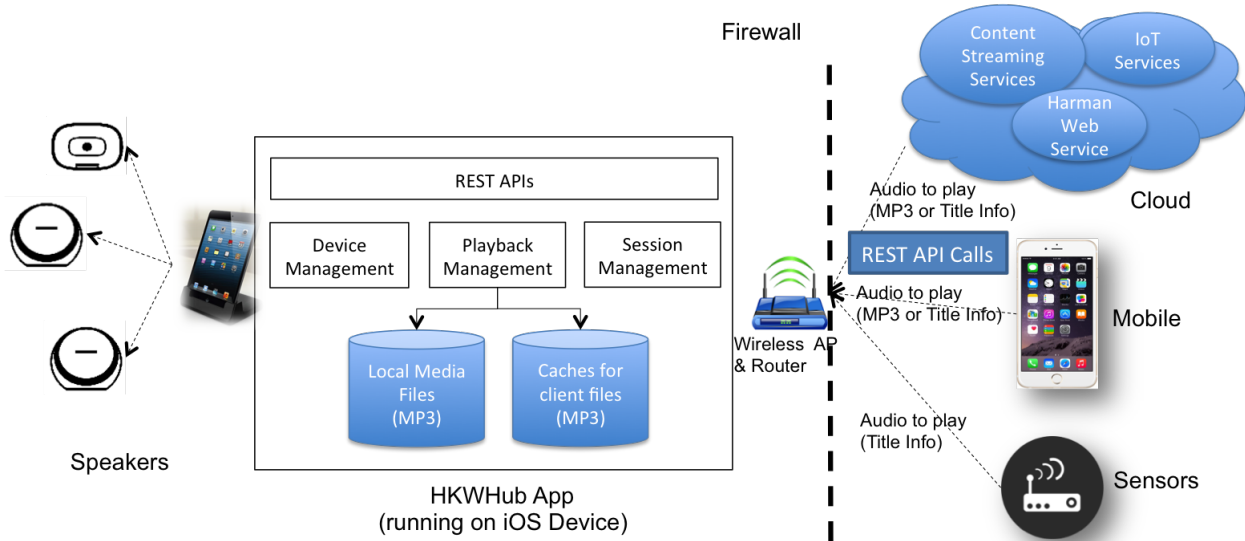
### Overall Architecture

HKWHub App handles requests from and sends responses to sensors, smart devices or cloud-based services to control audio playback with wireless speakers in the house.

The latest version of HKWHub app supports the following three modes:

- **Cloud mode (HKIoTCloud)**





- HKWHub app communicates with HKIoTCloud to receive speaker control commands by REST API call from 3rd party services or clients.
- HKIoTCloud handles the REST API request from any clients in the Internet. The clients can be 3rd party apps or services or devices like smartphone or sensors.
- In this mode, any 3rd party services or clients in the Internet can reach out to HKWHub app and then control speakers and playback of audio.
- All the 3rd party apps or services should be authorized with OAuth2 to get access token. An access token is required when 3rd party apps call the REST APIs. The detailed information about OAuth2 is available at [this link](#).

#### • Local Server Mode

- HKWHub app launches a web server internally, and then handles the REST API requests for speaker control and playback from devices, sensors or applications in the same local network.
- HKWHub app opens a HTTP port in the local network, so if devices or services outside of the local network want to reach out to HKWHub (and then speakers) then user needs to configure the route so that a request coming from outside can be routed to HKWHub app accordingly, such as firewall, etc.

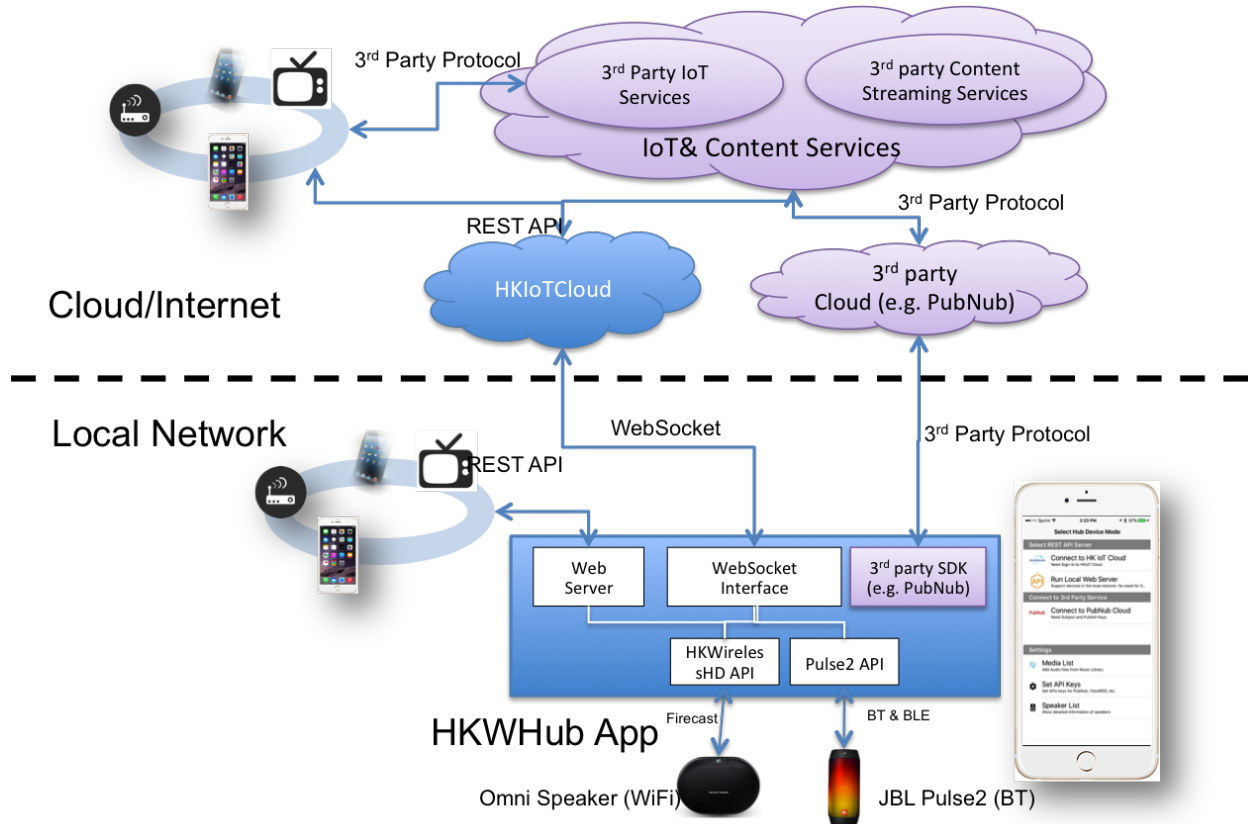
#### • PubNub Cloud mode

- HKWHub app uses PubNub API/SDK to connect to PubNub server and communicate with it to receive commands from other PubNub clients, and also sends events to other PubNub client, through a common PubNub channel.
- By setting the same PubNub channel, any client devices or services can communicate with the HKWHub app, and then control speakers and playback of audio.

The following figure shows how HKWHub app handles three modes.

## Quick Getting Started Guide to HKWHub App

### Overview of HKWHubApp



**Note:** You can download the HKWHub app from App Store ([click here](#))

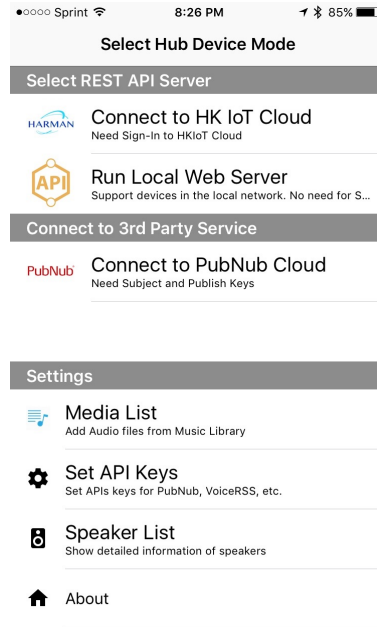
**Main Screen** The main screen is composed of two parts - **Select Server mode** (amongst HKIoTCloud, Local Server and PubNub Cloud), and **Settings**.

The **Select Server mode** has three options:

- **Connect to HK IoT Cloud**
  - HKWHub app connects to HKIoTCloud, and communicate with it with WebSocket to receive REST API commands from and send the responses back to the Cloud.
- **Run Local Web Server**
  - HKWHub app runs a local web server and processes incoming REST requests to control speakers and playback of audio
- **Connect to PubNub Cloud**
  - HKWHub app uses PubNub APIs to connect PubNub server and communicate with other PubNub client through a common channel.

The **Settings** menu has four sub menus:

- **Media List**
  - User can maintain the list of audio files for audio playback.
  - User can add audio from iOS Media Library.




---

**Note:** Note that only the media file available offline and not from Apple Musica can be added. The music file that came from Apple Music cannot be added by DRM issue.

---

- **Set API Keys**

- To use PubNub mode, user needs to enter PubNub API keys. It requires Publish Key and Subscribe Key. And also, user needs to set the channel where it exchanges the command and events with other clients.
- If user (or developer) wants to use TTS APIs such as **play\_tts**, then user needs to enter VoiceRSS (<http://www.voicerss.org>) API keys. You can get a free API key.

- **Speaker List**

- You can see the list of speakers available in the current local network.
- You can also change the device name or group name from this screen.

- **About**

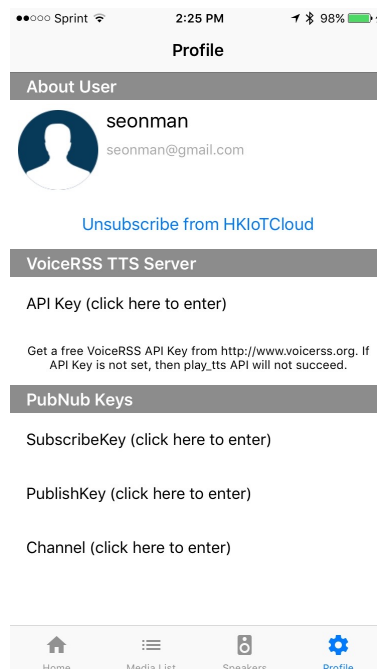
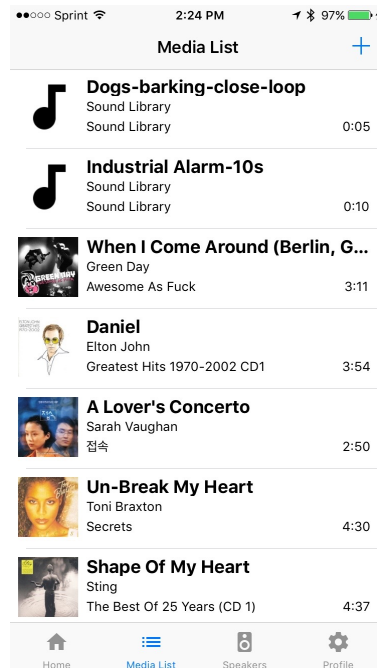
- The information of the app and the links to Harman developer documentation site.

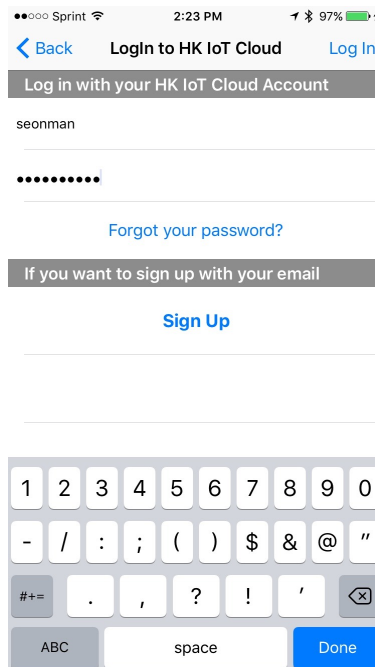
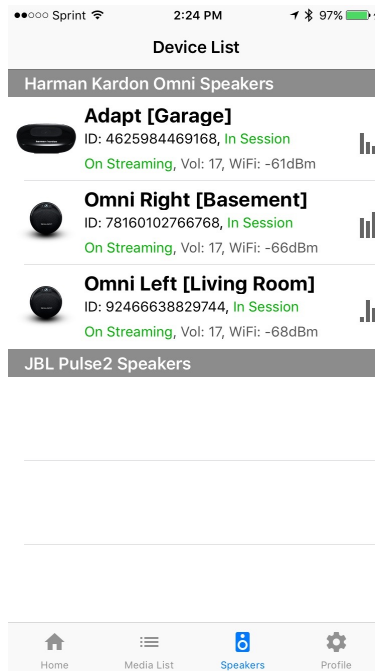
From now on, we will explain a little more detail about each server mode.

---

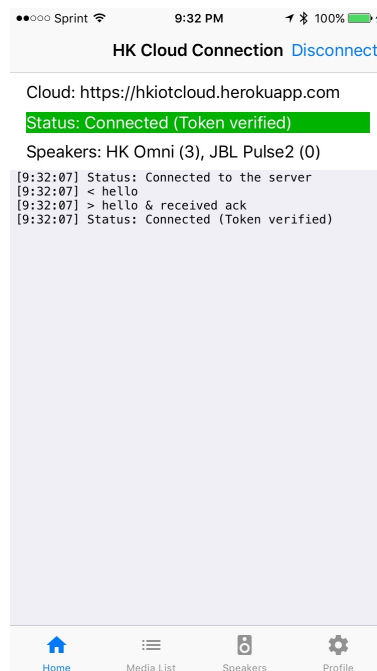
## HKIoTCloud Mode

**Connecting to HKIoTCloud** In HKIoTCloud demo, 3rd party clients can connect to HKIoTCloud (<http://hkiotcloud.herokuapp.com>) and send REST requests to control speakers and play audio. In order to use HKIoT-Cloud mode, user needs to sign up to the cloud with username, email address and password. Once sign up is done, user need to sign in to the server. User sign-up and sign-in can be done within the HKWHub app, as shown below.





Once the HKWHub app successfully signs in to HKIoTCloud, the screen will be switched to Log screen, like shown as below. You can see all the message logs received from or sent to the cloud. Each log contains a JSON data, so you can see what information is being sent and received between the server.



If you want to disconnect the server and return to the main screen, press **Disconnect** button on the top righthand corner.

**Sending REST Requests to HKIoTCloud** Once the HKWHub App is running, you can now connect a client to HKIoTCloud and send REST requests to the server. We will explain about the REST APIs supported with a little more detailed example of **curl** commands in the next section.

---

**Note:** For a client to connect to HKIoTCloud, the same username and password are required.

---

As an example of client, HKIoTCloud hosts a Web-based client app, at <http://hkiotcloud.herokuapp.com/webapp/>. The following is a screenshot of the web app.

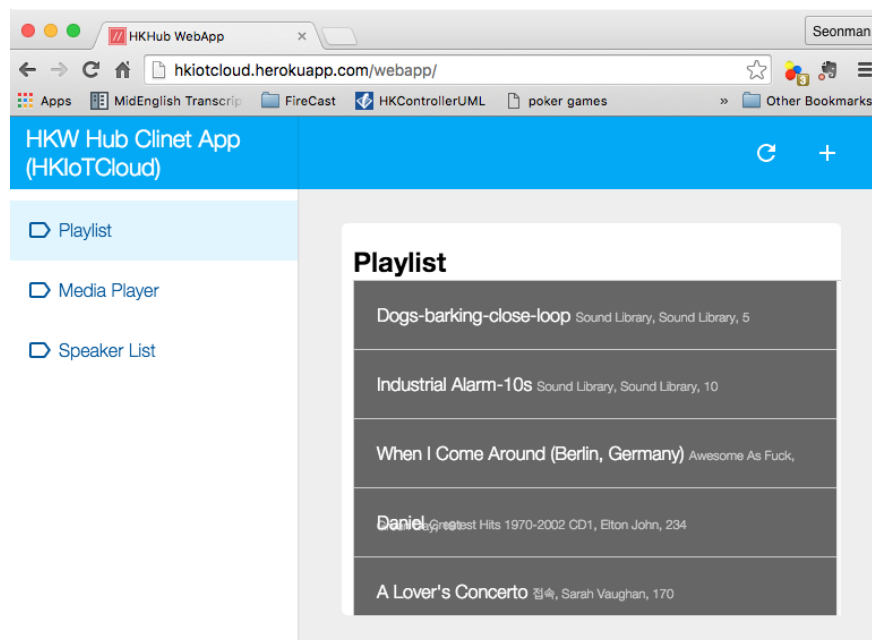
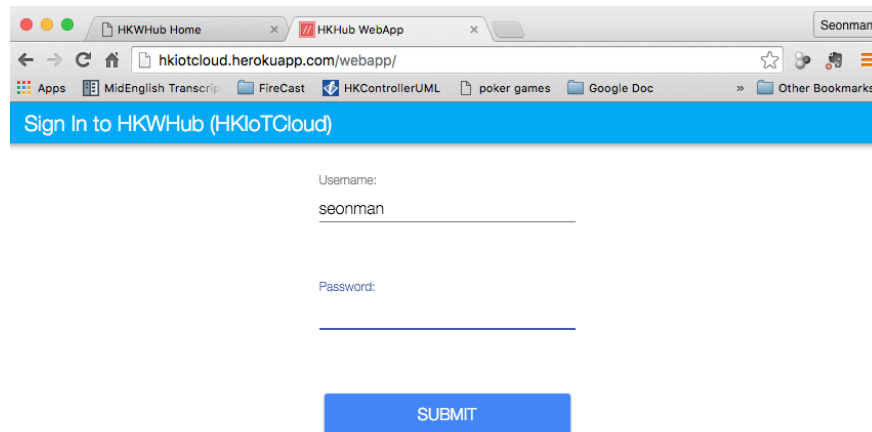
Once user authentication is done successfully, the Web app will switch the screen to the Playlist screen.

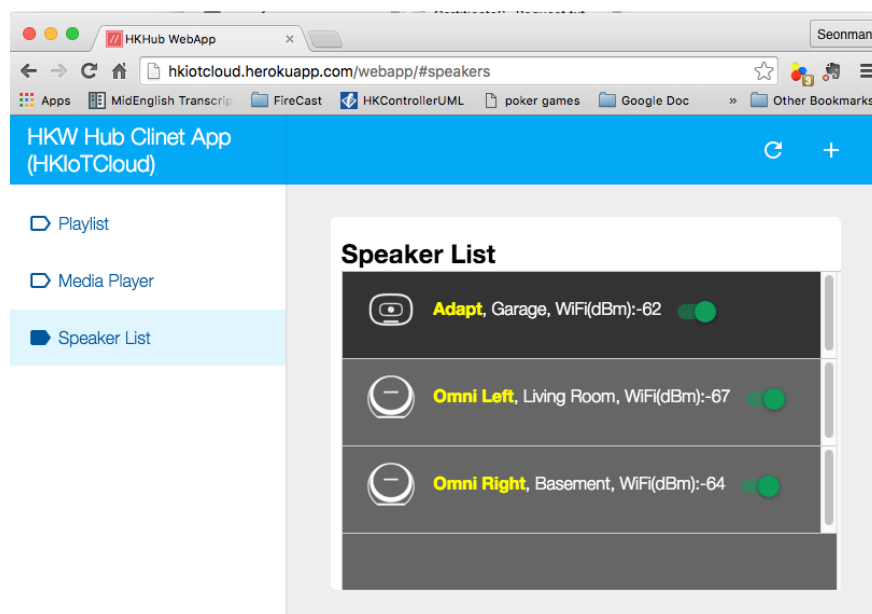
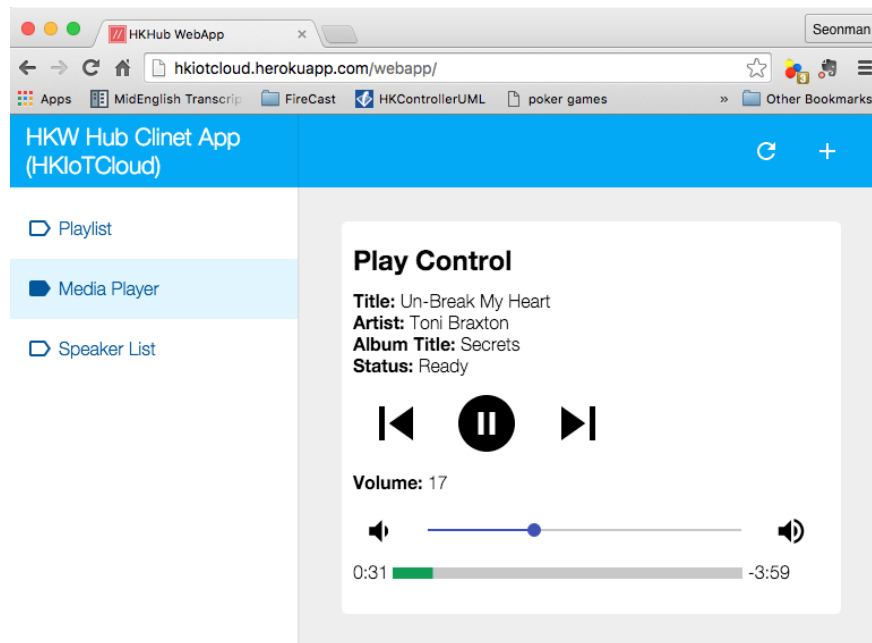
Now, you can click one of the titles in the list, and see how the web app is playing the title, showing the information of the title, volume, and playback time, and so on.

If you click **Speaker List** menu on the left, you can see more detailed information of speakers like below, and can control speakers, like remove a speaker from the current playback session or add a speaker to playback.

### Local Server Mode

**Running Local Server** Local Server Mode is almost the same as HKIoTCloud, except that HKWHub app runs a web server inside, instead connecting to HKIoTCloud. Therefore, HKWHub app can receive REST requests directly from clients in the same network. If you want to connect speakers from any type of devices in the same local network, then Local Server mode can be easier solution.

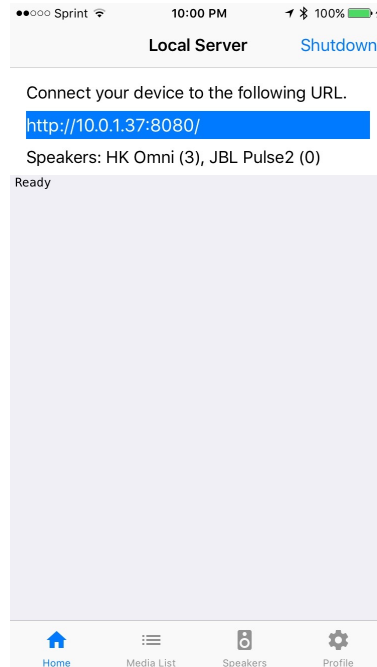






Once you click **Run Local Web Server** menu, then you will see the following screen. From the screen, you can see a URL indicating where a client should connect to. In this example, the client should enter the URL **http://10.0.1.37:8080/** followed by REST command and parameters.

The RESI APIs are almost the same as the ones of HKIoTCloud mode.



**Sending REST Requests to LocalServer** As a sample client app, you can use **WebHubWebApp** that you can download from Harman Developer web site (<http://developer.harman.com>) or directly from [here](#). The Web app is created using Polymer v0.5 (<https://www.polymer-project.org/0.5/>).

Once you download the app, unzip it. You will see the following sub directories.

- bower\_components: This is the folder where polymer libraries are located.
- hkwhub: this is the folder containing the WebHubApp source code.

```
$ cd WebHubWebApp
$ python -m SimpleHTTPServer
```

You will get some log messages like “Serving HTTP on 0.0.0.0 port 8000 ...”

Next, launch your web browser (Chrome, Safari, ...) and go to <http://localhost:8000/hkwhub/>

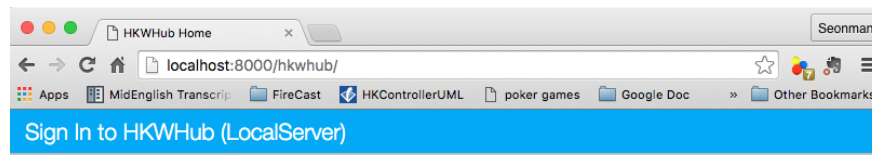
**Note:** Your iOS device running HKWHub app and your Desktop PC running web browser should be in the same network.

At the first screen looking like this:

Enter the URL that the HKWHub app says: <http://10.0.1.37:8080/>, like this:

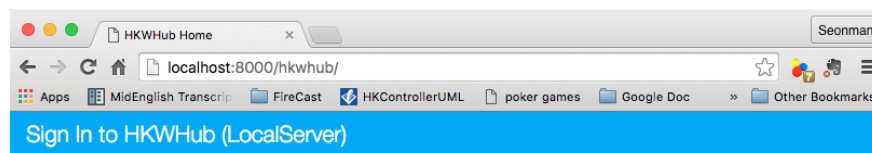
If you press **Submit**, then you will see the first screen like below. This is the list of media items available at the HKWHub app.

The UI of the Web app is exactly the same as HKIoTCloud web app. So, we skip to explain the rest parts of the app.



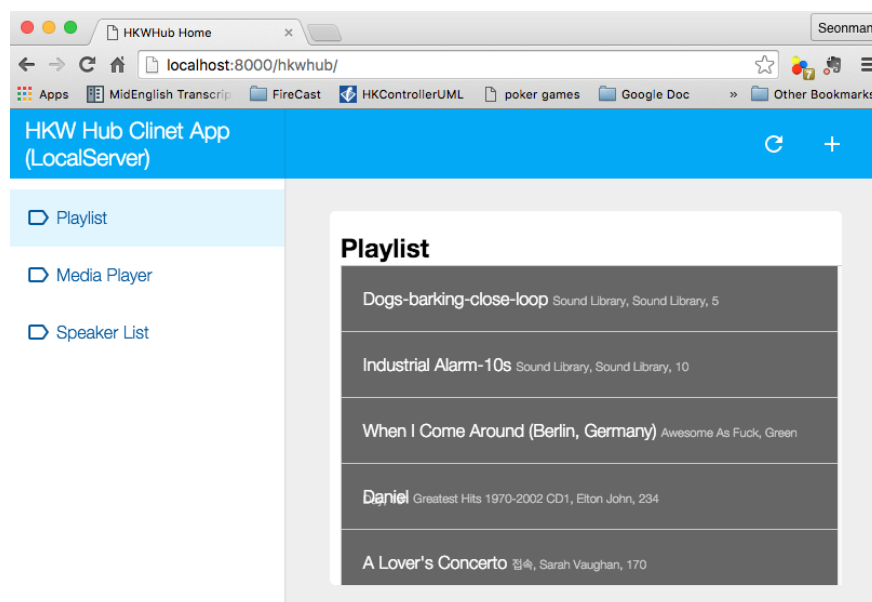
HKWHub URL:

SUBMIT



http://10.0.1.37:8080/

SUBMIT



## PubNub Server Mode

**Connect to PubNub Server** With PubNub server mode, any PubNub client can connect to and control Omni speakes managed by HKWHub app. Just click **Connect to PubNub Cloud** menu in the main screen, then you will see the screen like below. Please check if the logs are saying something like “Received: Hello from HKWHubApp” which is the message sent back from PubNub server after the HKWHub app published the message. This means the app is now connected to PubNub cloud.



Differently from HKIoTCloud or Local Server mode that relies on **REST API** for control and playback of speakers, PubNub is using Publish/Subscribe messaging instead. And in order to route the message among clients, we should set **PubNub Channel** so that all the published messages are correctly routed to subscribed clients of the same channel.

So, for HKWHub app successfully connects to PubNub cloud, user needs to set PubNub **Publish Key**, **Subscribe Key**, and **Channel**. As explained already, user can set these keys in the **Settings/Set API Keys** menu in the main screen.

**Sending REST Requests to PubNub Cloud** Once the HKWHub app is connected to PubNub cloud, a PubNub client can send PubNub message. Even though it does not use REST API, but use PubNub’s Subscribe/Publish messaging instead, the content of the messages are almost the same as the REST APIs, and it is in JSON format.

**Note:** One biggest difference between REST API and Publish/Subscribe messaging is that Pub/Sub messaging does not need to do **Polling** for getting information from the server when an event occurs on the server side, because REST API does not support **callback** mechanism to notify an **event** to clients. However, Pub/Sub messaging is bidirectional, the client can get notified immediately from the server. Either client or server can publish a message to the channel being shared to notify an event to subscribers.

In this reason, the messages of request and response for speaker control are a little different. For a client to send a command to speaker, the client **publish** the command to the channel. Then because HKWHub app is one of the clients, it receives the command, and process the command internally. If the command requires a response, then HKWHub app should send the response back to the client. To that, HKWHub app also needs to **publish** the response to the channel. And, the client will get the response because it subscribed to the channel.

If HKWHub app has some event to report to notify to clients, for example, device status changed, or playback time changed, etc., then HKWHub app publish the events to the channel, then all the client listening to the channel will receive the event.

**Sample Web App** As a sample client app, you can use **WebHubPubNubApp** that you can download from Harman Developer web site (<http://developer.harman.com>) or directly from [here](#).. Likewise, The Web app is created using Polymer v0.5 (<https://www.polymer-project.org/0.5/>).

Once you download the app, unzip it. You will see the following sub directories.

- bower\_components: This is the folder where polymer libraries are located.
- hkwhub: this is the folder containing the WebHubApp source code.

```
$ cd WebHubPubNubApp
$ python -m SimpleHTTPServer
```

You will get some log messages like “Serving HTTP on 0.0.0.0 port 8000 ...”

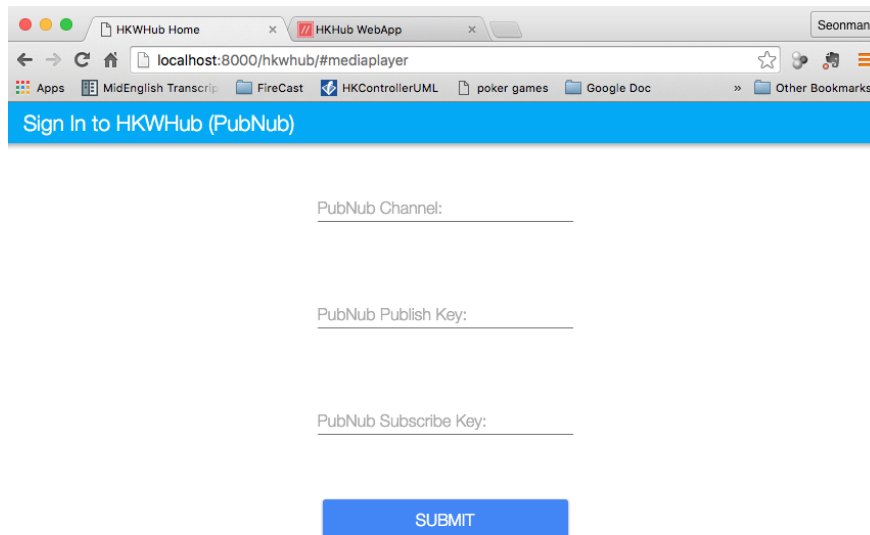
Next, launch your web browser (Chrome, Safari, ...) and go to <http://localhost:8000/hkwhub/>

---

**Note:** Your iOS device running HKWHub app and your Desktop PC running web browser should be in the same network.

---

At the first screen looking like below. Note that it looks different from the screen from Local Server mode, which requires only URL of the web server.

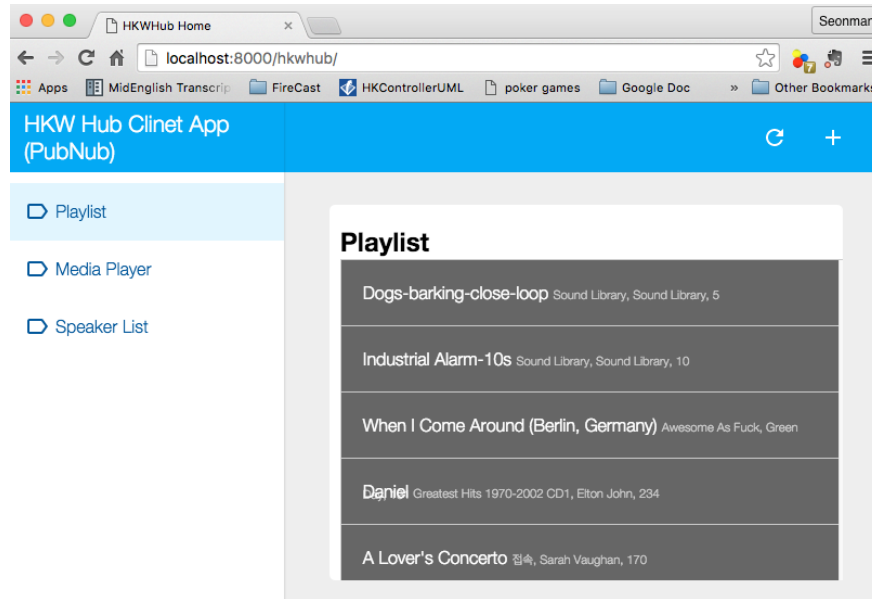


Enter the same PubNub publish key, subscribe key, and channel name that you used for HKWHub app, and click **Submit**, as below.

<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcdcbncdddomop/related?hl=en>

If you press **Submit**, then you will see the first screen like below. This is the list of media items available at the HKWHub app.

The UI of the Web app is exactly the same as HKIoTCloud web app. So, we skip to explain the rest parts of the app.

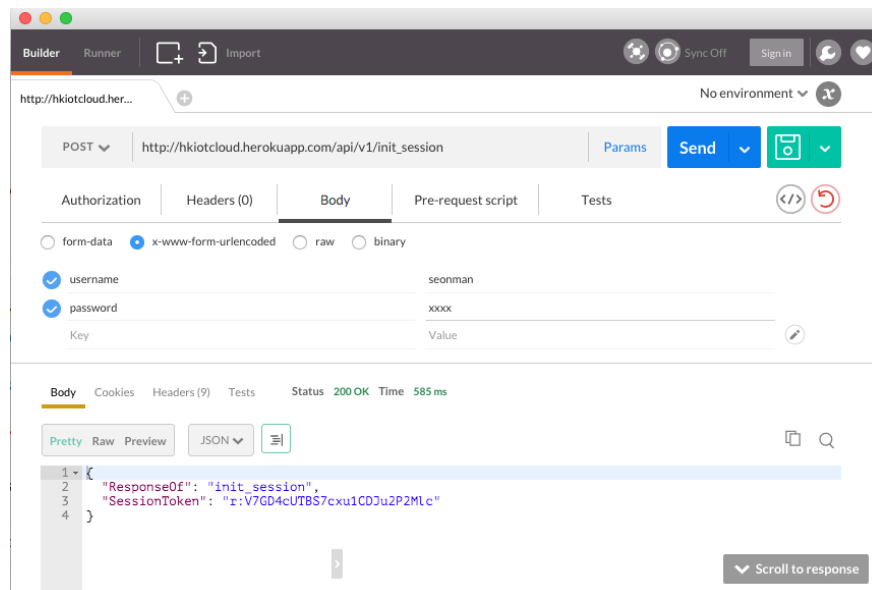


### Use `curl` command to send REST requests

We show how to control Omni speakers by sending REST requests to HKIoTCloud. Sending REST requests to Local Server is almost the same.

You can use **curl** command in your shell to send REST requests.

If you are a chrome browser user, you can use **Postman** (<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcdcbnc>) chrome extension to send HTTP requests with browser-based UI.



**Note:** Before you do this, do not forget to run HKWHub App and connect to HKIoTCloud.

**Get an Access Token and Refresh Token (HKIoTCloud mode only)** In case of HKIoTCloud mode, the client should get an access token from the HKIoTCloud to be able to call the REST APIs. HKIoTCloud supports two authorization modes: **password** and **authorization code**. For mode detailed information, please refer to the section of [OAuth2 Authorization API Specification](#).

With **password** grant mode, you can get an access token and a refresh token as shown below:

- curl -X POST -H “Authorization: Basic bjdlaGIUbktZakpkNHptTTpBTIJmQjl6OTR4dGN4RkdYcmQ1WEhYRWILZzQzVVk=” -d “grant\_type=password&username=yyy&password=xxx” <http://hkiotcloud.herokuapp.com/oauth/token>

Result:

```
{ "token_type": "bearer",
  "access_token": "15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8",
  "expires_in": 3600,
  "refresh_token": "1b470edc539681803de95c919bc3779acdf34e01" }
```

When you call the HKIoTCloud API calls, you should pass the value of the access token into the request header. Specifically, create an Authorization header and give it the value Bearer <access token>.

#### a. Init session

- curl -H “Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8” [http://hkiotcloud.herokuapp.com/api/v1/init\\_session](http://hkiotcloud.herokuapp.com/api/v1/init_session)

This returns the SessionToken. The returned SessionToken is used by all subsequent REST API request in the body.

```
{ "ResponseOf": "init_session", "SessionToken": "r:abciKaTbUgdpQFuvYtgMm0FRh" }
```

**b. Add alls speaker to session** After HKWHub app is launched, none of speakers is selected for playback. You need to add one or more speakers to play audio. To add all speakers to playback session, use `set_party_mode`. **Party Mode** is the mode where all speakers are playing the same audio together with synchronization. So, by `set_party_mode`, you can select all speakers to play.

- curl -H “Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8” “[http://hkiotcloud.herokuapp.com/api/v1/set\\_party\\_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh](http://hkiotcloud.herokuapp.com/api/v1/set_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh)”

```
{ "Result": "true", "ResponseOf": "set_party_mode" }
```

**c. Get the list of speakers available** To control speakers individually, you can get the list of speakers available by using `device_list` command.

- curl -H “Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8” “[http://hkiotcloud.herokuapp.com/api/v1/device\\_list?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh](http://hkiotcloud.herokuapp.com/api/v1/device_list?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh)”

```
{ "DeviceList": [
  {
    "IsPlaying": false,
    "MacAddress": "",
    "GroupName": "Garage",
    "Role": 21,
    "Version": "0.1.6.2",
    "Port": 44055,
    "Active": true,
    "GroupID": "4625984469",
    "ModelName": "Omni Adapt",
    "DeviceID": "4625984469168",
    "IPAddress": "10.0.1.6",
```

```

        "Volume":17,
        "DeviceName":"Adapt",
        "WifiSignalStrength":-62
    },
    {
        "IsPlaying":false,
        "MacAddress":"b0:38:29:11:19:54",
        "GroupName":"Living Room",
        "Role":21,
        "Version":"0.1.6.2",
        "Port":44055,
        "Active":true,
        "GroupID":"9246663882",
        "ModelName":"Omni 10",
        "DeviceID":"92466638829744",
        "IPAddress":"10.0.1.9",
        "Volume":17,
        "DeviceName":"Omni Left",
        "WifiSignalStrength":-67
    }
],
"ResponseOf":"device_list"
}

```

**d. Add a speaker to session** If you want to add a speaker to session, use `add_device_to_session``. It requires `DeviceID` parameter to identify a speaker to add. This command does not impact other speakers regardless of their status.

```

• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/add_device_to_session?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&DeviceID=

```

```

{"Result":"true","ResponseOf":"add_device_to_session"}

```

#### e. Get the media list

```

• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/media_list?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh"

```

Here, `SessionToken` should be the session token you got from `init_session`. You will get a list of media in JSON like below

```

{"MediaList": [
  {
    "PersistentID":"7387446959931482519",
    "Title":"I Will Run To You",
    "Artist":"Hillsong",
    "Duration":436,
    "AlbumTitle":"Simply Worship"
  },
  {
    "PersistentID":"5829171347867182746",
    "Title":"I'm Yours [ORIGINAL DEMO]",
    "Artist":"Jason Mraz",
    "Duration":257,
    "AlbumTitle":"Wordplay [SINGLE EP]"
  }
]}

```

**f. Play a media item listed in the HKWHub app** If you want to play a media item listed in the HKWHub app, use `play_hub_media` by specifying the media item with `PersistentID`. The `PersistentID` is available from the response of `media_list` command.

---

**Note:** Note that, before calling `play_hub_media`, at least one or more speakers must be selected (added to session) in advance. If not, then the playback will fail.

---

```
• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/play_hub_media?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&PersistentID=100"
```

```
{ "Result": "true", "ResponseOf": "play_hub_media" }
```

**f. Play a media item in the HKWHub by specifying a speaker list to play** You can play a media item in the HKWHub app by specifying the list of speakers.

```
• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/play_hub_media_selected_speakers?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&DeviceIDList=100,101"
```

The list of speakers are listed by the parameter `DeviceIDList` with delimiter “,”.

```
{ "Result": "true", "ResponseOf": "play_hub_media_selected_speakers" }
```

**g. Play a HTTP streaming media as party mode**

```
• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/play_web_media_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&MediaID=100"
```

```
{ "Result": "true", "ResponseOf": "play_web_media_party_mode" }
```

**h. Stop playing**

```
• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/stop_play?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh"
```

```
{ "Result": "true", "ResponseOf": "stop_play" }
```

**i. Set Volume**

```
• curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8"
  "http://hkiotcloud.herokuapp.com/api/v1/set_volume?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&Volume=30"
```

```
{ "Result": "true", "ResponseOf": "set_volume" }
```

---

**Note:** Please see the REST API specification for more information and examples.

---

## Playback Session Management

Since the HKWHub app should be able to handle REST HTTP requests from more than one clients at the same time, the HKWHub app manages the requests with session information associated with the priority when a new playback is initiated.

The following is the policy of the session management:



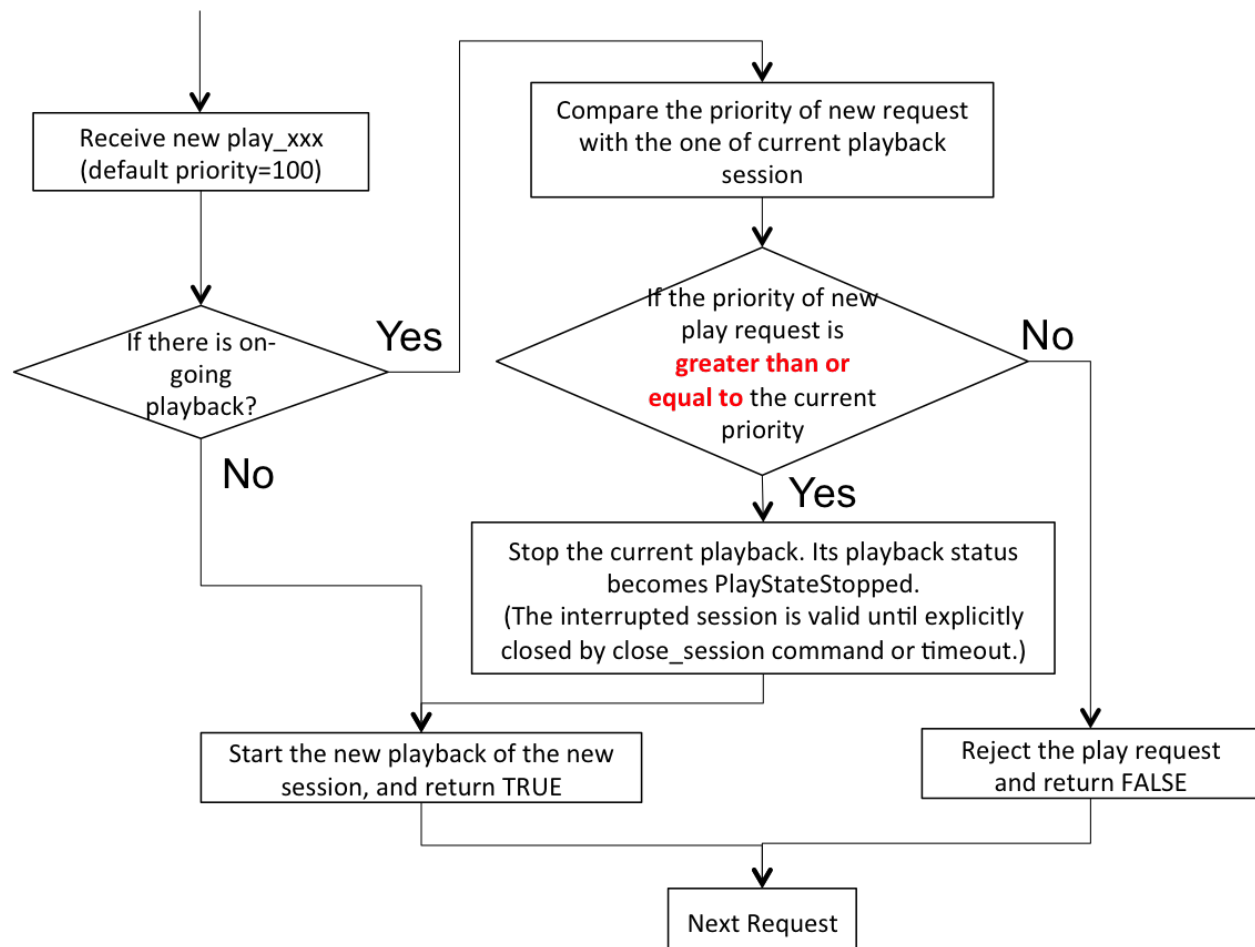
## Playback Session Creation

- When a client wants to start a playback, it sets the priority of the session (using `Priority=<priority value>` parameter).
- If `Priority` parameter is not specified, HKWHub app assumes it as default value, that is, 100.

## Priority of Session

- Each session is associated with a priority value which will be used to determine which request can override the current on-going playback session.
- **The priority value is specified as parameter (`Priority`) when the client calls `play_xxx`.**
  - If the command does not specify the `Priority` parameter, 100 is set as default value.
- **If the priority of a new playback request, such as `play_hub_media` or `play_web_media`, and so on, is greater than or equal to the current session's priority, then the current session is stopped.**
  - The playback status of the interrupted session becomes `PlayerStateStopped`. (see the related API in the next section)

The following diagrams show how HKWHub app handles incoming playback request based on the session priorities.



### Session Timeout

- A session becomes expired and invalid when about 60 minutes is passed since the last command was received.
  - Session timer is extended (renewed) once a playback is executed successfully.
  - All requests with expired session will be denied and “SessionNotFound” error returns.
- 

### REST API Specification (including PubNub JSON format)

This specification describes about the REST APIs to control HK Omni speakers and stream audio to the speakers via HKWHub app.

All the APIS are in REST API protocol.

---

**Note:** In this documentation, for HKIoTCloud mode, <server\_host> should be “hkiotcloud.herokuapp.com”. For Local server mode, <server\_host> should be the URL (IP address and port number) tat HKWHub app is showing.

---

---

**Note:** PubNub server mode does not use REST API. Instead, PubNub client needs to subscribe to the PubNub channel to get events from HKWHub, and use publish message to the PubNub channel to send request to HKWHub. The commands and parameters of each command are the same as REST API specification. However, PubNub message needs to include a couple of additional parameters in the JSON data to specify the **HKWHub UUID (HKWHubUUID)** that are talking to. The response message coming from the HKWHub app will include **ResponseOf** parameters to specify which request the resonse was for.

So, we will describe PubNub message specification along with REST API specification here.

---

---

**Note:** All the REST request should contain `Authorization` header that contains the access token, as described above.

---

### Session Management

**Start Session** This starts a new session. As a response, the client will receive a SessionToken. The SessionToken is required to be sent in any following requests. Note that the REST requests differs depending on the server mode.

- API: GET /api/v1/init\_session
- **Response**
  - Returns a unique session token
  - The session token will be used for upcoming requests.
- **Example:**
  - Request:

```
curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" http://<server_host>
```

- Response:

```
{ "ResponseOf": "init_session", "SessionToken": "r:abciKaTbUgdpQFuvYtgMm0FRh" }
```

- **PubNub**

- Publish Message

```
{Command = "init_session"}
```

- Message from HKWHub (via Subscribe)

\* Note that the response of `init_session` will contain **HKWHubUUID** to identify the HKWHub the PubNub client is getting talking to. The subsequent Publish message should include this HKWHubUUID information as well as SessionToken.

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "init_session"
}
```

**Close Session** Close the session. The SessionToken information is removed from the session table.

- API: GET `/api/v1/close_session?SessionToken=<session token>`

- **Response**

- Returns true or false indicating success or failure

- **Example:**

- Request:

```
http://<server_host>/api/v1/close_session?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh
```

- Response:

```
{ "Result" : "true" }
```

- **PubNub**

- Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Command = "close_session"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "close_session",
  Result = true
}
```

### Device Management

**Get the device count** Returns the number of speakers available in the network.

- API: GET /api/v1/device\_count?SessionToken=<session token>
- **Response**
  - Returns the number of devices connected to the network
- **Example:**
  - Request:

```
http://<server_host>/api/v1/device_count?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh
```

- Response:

```
{"DeviceCount": "2"}
```

- **PubNub**
  - Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Command = "device_count"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "device_count",
  DeviceCount = 2
}
```

---

**Get the list of devices and their information** Returns the list of speakers and their information including several status information.

- API: GET /api/v1/device\_list?SessionToken=<session token>
- **Response**
  - Returns the list of devices with all the device information
- **Example:**
  - Request:

```
http://<server_host>/api/v1/device_list?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh
```

- Response:

```
{"DeviceList":
  [{"GroupName": "Bathroom",
    "Role": 21,
    "MacAddress": "b0:38:29:1b:36:1f",
    "WifiSignalStrength": -47,
```

```

        "Port":44055,
        "Active":true,
        "DeviceName":"Adapt1",
        "Version":"0.1.6.2",
        "ModelName":"Omni Adapt",
        "IPAddress":"192.168.1.40",
        "GroupID":"3431724438",
        "Volume":47,
        "IsPlaying":false,
        "DeviceID":"34317244381360"
    },
    { "GroupName": "Temp",
      "Role":21,
      "MacAddress":"b0:38:29:1b:9e:75",
      "WifiSignalStrength":-53,
      "Port":44055,
      "Active":true,
      "DeviceName":"Adapt",
      "Version":"0.1.6.2",
      "ModelName":"Omni Adapt",
      "IPAddress":"192.168.1.39",
      "GroupID":"1293219209",
      "Volume":47,
      "IsPlaying":false,
      "DeviceID":"129321920968880"
    }
  ]
}

```

#### • PubNub

##### – Publish message

```

{
  Command = "device_list",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000"
}

```

##### – Response message (from Subscribed)

```

{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "device_list",
  DeviceList =
    [ { "GroupName":"Bathroom",
      "Role":21,
      "MacAddress":"b0:38:29:1b:36:1f",
      "WifiSignalStrength":-47,
      "Port":44055,
      "Active":true,
      "DeviceName":"Adapt1",
      "Version":"0.1.6.2",
      "ModelName":"Omni Adapt",
      "IPAddress":"192.168.1.40",
      "GroupID":"3431724438",
      "Volume":47,
      "IsPlaying":false,

```

```
        "DeviceID": "34317244381360"
      },
      { "GroupName": "Temp",
        "Role": 21,
        "MacAddress": "b0:38:29:1b:9e:75",
        "WifiSignalStrength": -53,
        "Port": 44055,
        "Active": true,
        "DeviceName": "Adapt",
        "Version": "0.1.6.2",
        "ModelName": "Omni Adapt",
        "IPAddress": "192.168.1.39",
        "GroupID": "1293219209",
        "Volume": 47,
        "IsPlaying": false,
        "DeviceID": "129321920968880"
      }
    ]
  }
}
```

---

**Get the Device Information** Gets the device information of a particular device (speaker) identified by DeviceID.

- API: GET /api/v1/device\_info?SessionToken=<session token>&DeviceID=<device id>

- **Response**

- Returns the information of the device

- **Example:**

- Request:

```
http://<server_host>/api/v1/device_info?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&DeviceID=12
```

- Response:

```
{ "GroupName": "Temp",
  "Role": 21,
  "MacAddress": "b0:38:29:1b:9e:75",
  "WifiSignalStrength": -52,
  "Port": 44055,
  "Active": true,
  "DeviceName": "Adapt",
  "Version": "0.1.6.2",
  "ModelName": "Omni Adapt",
  "IPAddress": "192.168.1.39",
  "GroupID": "1293219209",
  "Volume": 47,
  "IsPlaying": true,
  "DeviceID": "129321920968880" }
```

- **PubNub**

- Publish message

```
{
  Command = "device_list",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000"
}
```

– Response message (from Subscribed)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "device_list",
  GroupName: "Temp",
  Role = 21,
  MacAddress = "b0:38:29:1b:9e:75",
  WifiSignalStrength = -52,
  Port = 44055,
  Active = true,
  DeviceName = "Adapt",
  Version = "0.1.6.2",
  ModelName = Omni Adapt",
  IPAddress = "192.168.1.39",
  GroupID = 1293219209",
  Volume = 47,
  IsPlaying = true,
  DeviceID = 129321920968880"
}
```

**Add a Device to Session** Add a speaker to playback session. Once a speaker is added, then the speaker will play the music. There is no impact of this call to other speakers.

- API: GET /api/v1/add\_device\_to\_session?SessionToken=<session token>&DeviceID=<device id>
- **Response**
  - Returns true or false
- **Example:**
  - Request:

```
http://<server_host>/api/v1/add_device_to_session?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&D
```

– Response:

```
{ "Result": "true" }
```

- **PubNub**

– Publish message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Command = "add_device_to_session",
  DeviceID = "129321920968880"
}
```

– Response:

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "add_device_to_session",
  Result = true
}
```

**Remove a Device from Session** Removes a speaker from playback session. Once a speaker is removed, then the speaker will not play the music. There is no impact of this call to other speakers.

- API: GET /api/v1/remove\_device\_from\_session?SessionToken=<session token>&DeviceID=<device id>

- **Response**

- Returns true or false

- **Example:**

- Request:

```
http://<server_host>/api/v1/remove_device_from_session?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Command = "remove_device_from_session",
  DeviceID = "129321920968880"
}
```

- Response:

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "remove_device_from_session",
  Result = true
}
```

**Set party mode** Adds all speakers to playback session. Once it is done, all speakers will play music.

- API: GET /api/v1/set\_party\_mode?SessionToken=<session token>

- **Response**

- Returns true or false

- **Example:**

- Request:

```
http://<server_host>/api/v1/set_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message



```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Command = "set_party_mode"
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  ResponseOf = "set_party_mode",
  Result = true
}
```

## Media Playback Management

**Get the list of media item in the Media List of the HKWHub app** Returns the list of media items added to the Media List of the app. User can add music items to the **Media List** of the app via **Setting** of the app.

**Note:** A music item downloaded from Apple Music is not supported. The music file from Apple music is DRM-enabled, and cannot be played with HKWirelessHD. Only music items purchased from iTunes Music or added from user's own library are supported.

To be added to the Media List, the music item must be located locally on the device. No streaming from iTunes or Apple Music are supported.

- API: GET /api/v1/media\_list?SessionToken=<session token>
- **Response**
  - Returns JSON of the list of store media in the HKWHub app.
- **Example:**
  - Request:

```
http://<server_host>/api/v1/media_list?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

– Response:

```
{ "MediaList": [
  { "PersistentID": "7387446959931482519",
    "Title": "I Will Run To You",
    "Artist": "Hillsong",
    "Duration": 436,
    "AlbumTitle": "Simply Worship"
  },
  { "PersistentID": "5829171347867182746",
    "Title": "I'm Yours [ORIGINAL DEMO]",
    "Artist": "Jason Mraz",
    "Duration": 257,
    "AlbumTitle": "Wordplay [SINGLE EP]"
  }
]}
```

- PubNub

- Publish Message

```
{Command = "media_list",
HKWHubUUID = "XXX-XXX-XXX-XXX",
SessionToken = "PubNub-1000"}
```

- Message from HKWHub (via Subscribe)

```
{
HKWHubUUID = "XXX-XXX-XXX-XXX",
MediaList = [
    { "PersistentID": "7387446959931482519",
      "Title": "I Will Run To You",
      "Artist": "Hillsong",
      "Duration": 436,
      "AlbumTitle": "Simply Worship"
    },
    { "PersistentID": "5829171347867182746",
      "Title": "I'm Yours [ORIGINAL DEMO]",
      "Artist": "Jason Mraz",
      "Duration": 257,
      "AlbumTitle": "Wordplay [SINGLE EP]"
    }
],
ResponseOf = "media_list"
}
```

---

**Play a song in the Media List of the HKWHub app** Plays a song in the Media List of the Hub app. Each music item is identified with MPMediaItem's PersistentID. It is a unique ID to identify a song in the iOS Music library.

---

**Note:** `play_hub_media` does not specify speakers to play. It just uses the current session setting. If there is no speaker in the current session, then the play fails.

---

- API: GET `/api/v1/play_hub_media?SessionToken=<session token>&PersistentID=<persistent id>`

- **Response**

- Play a song stored in the hub, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_hub_media?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&PersistentID=7387446959931482519
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{
HKWHubUUID = "XXX-XXX-XXX-XXX",
Command = "play_hub_media",
PersistentID = 7387446959931482519,
SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_hub_media",
  Result = true
}
```

**Play a song in the Media list as party mode** Plays a song in the Media List with all speakers available. So, regardless of current session setting, this command play a song to all speakers.

- API: GET /api/v1/play\_hub\_media\_party\_mode?SessionToken=<session token>&PersistentID=<persistent id>

- **Response**

- Play a song in the hub’s media list to all speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_hub_media_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{
  Command = "play_hub_media_party_mode",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  PersistentID = 7387446959931482519,
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_hub_media_party_mode",
  Result = true
}
```

**Play a song in the Media list with selected speakers** Plays a song in the Media List with selected speakers. The selected speakers are represented in DeviceIDList parameter as a list of DeviceID separated by “,”.

- API: GET /api/v1/play\_hub\_media\_selected\_speakers?SessionToken=<session token>&PersistentID=<persistent id>&DeviceIDList=<xxx,xxx,...>

- **Response**

- Play a song in the hub’s media list to selected speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_hub_media_selected_speakers?SessionToken=r:abciKaTbUgdpQFuv
```

- Response:

```
{"Result": "true"}
```

- **PubNub**

- Publish Message

```
{
  Command = "play_hub_media_selected_speakers",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  PersistentID = 7387446959931482519,
  SessionToken = "PubNub-1000",
  DeviceIDList = 34317244381360,129321920968880
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_hub_media_selected_speakers",
  Result = true
}
```

---

**Play a Song from Web Server** Plays a song from Web (http:) or rstp (rstp:) or mms (mms:) server. The URL of the song to play is specified by `MediaUrl` parameter.

---

**Note:** `play_web_media` does not specify speakers to play. It just uses the current session setting. If there is no speaker in the current session, then the play fails.

---

**Note:** `play_web_media` cannot be resumed. If it is paused by calling `pause`, then it just stops playing music, and cannot resume.

- API: GET `/api/v1/play_web_media?SessionToken=<session token>&MediaUrl=<URL of the song>`

- **Response**

- Play a song from HTTP server, and then return true or false.

- **Example:**

- Request:

```
http://<server_host_name>/api/v1/play_web_media?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&Media
```

- Response:

```
{"Result": "true"}
```

---

**Note:** This API call takes several hundreds millisecond to return the response.

---

- **PubNub**

- Publish Message

```
{
  Command = "play_web_media",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  MediaUrl = "http://seonman.github.io/music/hyolyn.mp3"
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_web_media",
  Result = true
}
```

---

**Play a Song from Web Server as party mode** Plays a song from Web server with all speakers. The URL of the song to play is specified by `MediaUrl` parameter.

---

**Note:** `play_web_media` cannot be resumed. If it is paused by calling `pause`, then it just stops playing music, and cannot resume.

---

- API: GET `/api/v1/play_web_media_party_mode?SessionToken=<session token>&MediaUrl=<URL of the song>`

- **Response**

- Play a song from HTTP server to all speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_web_media_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "Result": "true" }
```

---

**Note:** This API call takes several hundreds millisecond to return the response.

---

- **PubNub**

- Publish Message

```
{
  Command = "play_web_media_party_mode",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
}
```

```
MediaUrl = "http://seonman.github.io/music/hyolyn.mp3"
SessionToken = "PubNub-1000"
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_web_media_party_mode",
  Result = true
}
```

---

**Play a Song from Web Server with selected speakers** Plays a song from Web server with selected speakers. The URL of the song to play is specified by `MediaUrl` parameter. The selected speakers are represented in `DeviceIDList` parameter as a list of `DeviceID` separated by “,”.

---

**Note:** `play_web_media` cannot be resumed. If it is paused by calling `pause`, then it just stops playing music, and cannot resume.

---

- API: GET `/api/v1/play_web_media_selected_speakers?SessionToken=<session Token>&MediaUrl=<URL of the song>&DeviceIDList=<xxx,xxx,...>`

- **Response**

- Play a song from HTTP server to selected speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_web_media_selected_speakers?SessionToken=r:abdiKaTbUgdpQFuv
```

- Response:

```
{ "Result": "true" }
```

---

**Note:** This API call takes several hundreds millisecond to return the response.

---

- **PubNub**

- Publish Message

```
{
  Command = "play_web_media_selected_speakers",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  MediaUrl = "http://seonman.github.io/music/hyolyn.mp3"
  SessionToken = "PubNub-1000",
  DeviceIDList = "34317244381360,129321920968880"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_web_media_selected_speakers",
  Result = true
}
```

**Play TTS (Text-to-Speech)** Plays a Text-to-Speech audio from VoiceRRS server. The Text to play is specified by Text parameter.

**Note:** In order to use APIs for playing TTS (Text-To-Speech), you need to set VoiceRRS Application key on the setting menu of HKWHub App. You can go to the [VoiceRRS](#) web site to get your application key.

**Note:** play\_tts does not specify speakers to play. It just uses the current session setting. If there is no speaker in the current session, then the play fails.

**Note:** play\_tts cannot be resumed. If it is paused by calling pause, then it just stops playing music, and cannot resume.

- API: GET /api/v1/play\_tts?SessionToken=<session token>&Text=<Text>
- **Response**
  - Play TTS audio, and then return true or false.
- **Example:**
  - Request:

```
http://<server_host_name>/api/v1/play_tts?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&Text="Hello World"
```

- Response:

```
{ "Result": "true" }
```

**Note:** This API call takes more than several hundreds millisecond to return the response, depending on the network condition.

- **PubNub**
  - Publish Message

```
{
  Command = "play_tts",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Text = "Hello World. How are you today?",
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_tts",
  Result = true
}
```

---

**Play TTS (Text-to-Speech) as party mode** Plays a Text-to-Speech audio from VoiceRRS server with all speakers. The Text to play is specified by `Text` parameter.

- API: GET `/api/v1/play_tts_party_mode?SessionToken=<session token>&Text=<Text>`
- **Response**
  - Play TTS audio to all speakers, and then return true or false.
- **Example:**
  - Request:

```
http://<server_host>/api/v1/play_tts_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&Text=
```

- Response:

```
{ "Result": "true" }
```

---

**Note:** This API call takes several hundreds millisecond to return the response.

---

- **PubNub**
  - Publish Message

```
{
  Command = "play_tts_party_mode",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Text = "Hello World. How are you today?"
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_tts_party_mode",
  Result = true
}
```

---

**Play a Song from Web Server with selected speakers** Plays a Text-to-Speech audio from VoiceRRS server with selected speakers. The Text to play is specified by `Text` parameter. The selected speakers are represented in `DeviceIDList` parameter as a list of `DeviceID` separated by “,”.

- API: GET `/api/v1/play_tts_selected_speakers?SessionToken=<Session Token>&Text=<Text>&DeviceIDList=<xxx,xxx,...>`



- **Response**

- Play TTS from VoiceRSS server to selected speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/play_tts_selected_speakers?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{"Result": "true"}
```

---

**Note:** This API call takes several hundreds millisecond to return the response.

---

- **PubNub**

- Publish Message

```
{
  Command = "play_tts_selected_speakers",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Text = "Hello World. How are you today?",
  SessionToken = "PubNub-1000",
  DeviceIDList = "34317244381360,129321920968880"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "play_tts_selected_speakers",
  Result = true
}
```

---

**Pause the Current Playback** Pauses the current playback. The client can resume the playback by `resume_hub_media`.

- API: GET `/api/v1/pause_play?SessionToken=<session token>`

- **Response**

- Pause the current playback, and then return true or false.
- It can resume the current playback by calling `resume_hub_media` if and only if the playback is playing hub media. `play_web_media` cannot be resumed once it is paused or stopped.

- **Example:**

- Request:

```
http://<server_host>/api/v1/pause_play?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{"Result": "true"}
```

- **PubNub**

– Publish Message

```
{  
  Command = "pause_play",  
  HKWHubUUID = "XXX-XXX-XXX-XXX",  
  SessionToken = "PubNub-1000"  
}
```

– Message from HKWHub (via Subscribe)

```
{  
  HKWHubUUID = "XXX-XXX-XXX-XXX",  
  SessionToken = "PubNub-1000",  
  ResponseOf = "pause_play",  
  Result = true  
}
```

---

## Resume the Current Playback with Hub Media

- API: GET /api/v1/resume\_hub\_media?SessionToken=<session token>&PersistentID=<persistent id>

- **Response**

- Resume the current playback with Hub Media, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/resume_hub_media?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&Persiste
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{  
  HKWHubUUID = "XXX-XXX-XXX-XXX",  
  Command = "resume_hub_media",  
  PersistentID = 7387446959931482519,  
  SessionToken = "PubNub-1000"  
}
```

- Message from HKWHub (via Subscribe)

```
{  
  HKWHubUUID = "XXX-XXX-XXX-XXX",  
  SessionToken = "PubNub-1000",  
  ResponseOf = "resume_hub_media",  
  Result = true  
}
```

---

**Resume the Current Playback with Hub Media as Party Mode**

- API: GET /api/v1/resume\_hub\_media\_party\_mode?SessionToken=<session token>&PersistentID=<persistent id>

- **Response**

- Resume the current playback with Hub Media with all speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/resume_hub_media_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Command = "resume_hub_media_party_mode",
  PersistentID = 7387446959931482519,
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "resume_hub_media_party_mode",
  Result = true
}
```

**Resume the Current Playback with Hub Media with selected speakers**

- API: GET /api/v1/resume\_hub\_media\_selected\_speakers?SessionToken=<session token>&PersistentID=<persistent id>&DeviceIDList=<xxx,xxx,...>

- **Response**

- Resume the current playback with Hub Media with selected speakers, and then return true or false.

- **Example:**

- Request:

```
http://<server_host>/api/v1/resume_hub_media_selected_speakers?SessionToken=r:abciKaTbUgdpQF
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Command = "resume_hub_media_selected_speakers",
  PersistentID = 7387446959931482519,
  SessionToken = "PubNub-1000",
  DeviceIDList = "34317244381360,129321920968880"
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "resume_hub_media_selected_speakers",
  Result = true
}
```

---

## Stop the Current Playback

- API: GET /api/v1/stop\_play?SessionToken=<session token>
- **Response**
  - Stop the current playback with Hub Media, and then return true or false.
  - If the playback has stopped, then it cannot resume.
- Example:
  - Request:

```
http://<server_host>/api/v1/stop_play?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

– Response:

```
{"Result": "true"}
```

- **PubNub**

- Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Command = "stop_play",
  SessionToken = "PubNub-1000",
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "stop_play",
  Result = true
}
```

**Get the Playback Status (Current Playback State and Elapsed Time)**

- API: GET /api/v1/playback\_status?SessionToken=<session token>
- **Response**
  - It returns the current state of the playback and also return the elapsed time (in second) of the playback.
  - If it is not playing, then the elapsed time is (-1)
  - **The following is the value of each playback state:**
    - \* PlayerStatePlaying : Now playing audio
    - \* PlayerStatePaused : Playing is paused. It can resume.
    - \* PlayerStateStopped : Playing is stopped. It cannot resume.
  - Note that if the playback has stopped, then it cannot resume.
  - Developers need to check the playback status during the playback to handle any possible exceptional cases like interruption or errors. We recommend to call this API every second.

• **Example:**

- Request:

```
http://<server_host>/api/v1/playback_status?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "PlaybackState": "PlayerStatePlaying",  
  "TimeElapsed": "15" }
```

• **PubNub**

- PubNub mode does not support playback\_status command, because clients subscribing the channel will automatically receive the playback\_status event from the HKWHub app when an event is available.
- Event from HKWHub app

```
{  
  HKWHubUUID = "XXX-XXX-XXX-XXX",  
  SessionToken = "PubNub-1000",  
  Envset = PlaybackTimeChanged,  
  PlaybackTime = 15  
}
```

**Check if the Hub is playing audio**

- API: GET /api/v1/is\_playing?SessionToken=<session token>
- **Response**
  - Returns true (playing) or false (not playing)

• **Example:**

- Request:

```
http://<server_host>/api/v1/is_playing?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "IsPlaying": "true" }
```

- **PubNub**

- Publish Message

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Command = "is_playing",
  SessionToken = "PubNub-1000",
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  ResponseOf = "is_playing",
  Result = true
}
```

## Volume Control

### Get Volume for all Devices

- API: GET /api/v1/get\_volume?SessionToken=<session token>

- **Response**

- Returns the average volume of all devices.
  - The range of volume is 0 (muted) to 50 (max)

- **Example:**

- Request:

```
http://<server_host>/api/v1/get_volume?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F
```

- Response:

```
{ "Volume": "10" }
```

- **PubNub**

- Publish Message

```
{
  Command = "get_volume",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000"
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  ResponseOf = "get_volume",
  SessionToken = "PubNub-1000",
  Volume = 10
}
```

**Get Volume for a particular device**

- API: GET /api/v1/get\_volume\_device?SessionToken=<session token>&DeviceID=<device id>

- **Response**

- Returns the volume of a particular device
- The range of volume is 0 (muted) to 50 (max)

- **Example:**

- Request:

```
http://<server_host>/api/v1/get_volume_device?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&DeviceID=1234567
```

- Response:

```
{ "Volume": "10" }
```

- **PubNub**

- Publish Message

```
{
  Command = "get_volume_device",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  DeviceID=1234567
}
```

- Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  ResponseOf = "get_volume_device",
  SessionToken = "PubNub-1000",
  Volume = 10
}
```

**Set Volume for all devices**

- API: GET /api/v1/set\_volume?SessionToken=<session token>&Volume=<volume>

- **Response**

- Returns true or false

- **Example:**

- Request:

```
http://<server_host>/api/v1/set_volume?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&Volume=10
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

– Publish Message

```
{
  Command = "set_volume",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  Volume = 10
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  ResponseOf = "set_volume",
  SessionToken = "PubNub-1000",
  Result = true
}
```

---

### Set Volume for a particular device

- API: GET /api/v1/set\_volume\_device?SessionToken=<session token>&DeviceID=<device id>&Volume=<volume>

- **Response**

- Returns true or false

- **Example:**

- Request:

```
http://<server_host>/api/v1/set_volume_device?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F&DeviceID=1234567
```

- Response:

```
{ "Result": "true" }
```

- **PubNub**

- Publish Message

```
{
  Command = "set_volume_device",
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  SessionToken = "PubNub-1000",
  DeviceID = 1234567,
  Volume = 10
}
```

– Message from HKWHub (via Subscribe)

```
{
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  ResponseOf = "set_volume_device",
  SessionToken = "PubNub-1000",
  Result = true
}
```

---



**Device Status Change Event (only available for PubNub mode)**

- **PubNub**

- Whenever a speaker status change occurs, the HKWHub sends DeviceStateUpdated event to subscribers.

- **Type of Reason**

- \* NetworkUnavailable : the network became unavailable
- \* DeviceAvailable : a device became available
- \* DeviceUnavailable : a device became unavailable
- \* DeviceError : some error occurred to a particular speaker
- \* SpeakerInfoUpdated: some of speaker information has been updated
- \* WiFiSignalChanged: wifi signal has changed

- Example:

```
{
  Event = DeviceStateUpdated,
  HKWHubUUID = "XXX-XXX-XXX-XXX",
  Reason = SpeakerInfoUpdated,
  SessionToken = "PubNub-1000"
}
```

---

## OAuth2 Authorization API Specification

### Introduction

In order to access the HKIoTCloud REST APIs to control Omni speakers, your HKIoTCloud-enabled product needs to obtain a HKIoTCloud access token that grants access to the APIs on behalf of the product's user.

---

**Note:** Please refer [OAuth 2.0 Getting Started in Web-API Security by Matthias Biehl](#) for your more understanding on OAuth2.

---

The workflow for obtaining and using an access token is as follows:

1. The user visits your product registration website and enters information about their specific instance of your product.
2. Your website creates a HKIoTCloud consent request using the user-supplied registration information and forwards the user to the HKIoTCloud website.
3. The user logs in to HKIoTCloud.
4. The user authorizes their instance of your product to be used with HKIoTCloud on their behalf.
5. HKIoTCloud returns an access token to your product registration website.
6. Your product registration website securely transfers the access token to the user's specific instance of your product.
7. The user's specific instance of your product uses the access token to make HKIoTCloud API calls.



The consent request is constructed as follows:

- **Redirect the user to HKIoTCloud at <https://hkiotcloud.herokuapp.com/oauth/token> with the following URL-encoded query parameters:**
  - `client_id`: The client ID of your application. This information can be found on the HKIoTCloud website.
  - `response_type`: code for authorization code grant.
  - `redirect_uri`: Specifies the return URI that you added to your app's profile when signing up.

#### Sample Request:

Send as GET request.

```
https://hkiotcloud.herokuapp.com/oauth/authorize?response_type=code&client_id=n7HhiTnKYjJd4zmM&redirect_uri=
```

#### HKIoTCloud Returns a Response to Your Registration Website

After the user is authenticated, the user is redirected to the URI that you provided in the `redirect_uri` parameter of the request.

The response includes an authorization code.

#### Sample Authorizatio Code Grant Response:

```
https://your.app.com/oauthCallbackHKIoTCloud?code=0b368d49809048dd7424d6f7fd869a98f2372859
```

Next, your service leverages the returned authorization code to ask for an access token:

- Send a **POST** request to <https://hkiotcloud.herokuapp.com/oauth/token> with the following parameters:

#### HTTP Header Parameters:

- `Content-Type`: `application/x-www-form-urlencoded`

#### HTTP Body Parameters:

- `grant_type`: `authorization_code`
- `code`: The authorization code that was returned in the response.
- `client_id`: Your application's client ID. This information can be found on the HKIoTCloud website.
- `client_secret`: The application's client secret. This information can be found on the HKIoTCloud website.
- `redirect_uri`: The return URI that you added to your app's profile when signing up.

#### Sample Request:

```
POST /oauth/token HTTP/1.1
Host: hkiotcloud.herokuapp.com
Content-Type: application/x-www-form-urlencoded
Cache-Control: no-cache

grant_type=authorization_code&code=2b3711911f4f2263e785eeda386046ccc8da6aee&client_id=n7HhiTnKYjJd4zmM&redirect_uri=
```

#### Sample Response:

```
{
  "access_token": "902da699ed1d5d511bd750366889f3260c2015b4",
  "expires_in": 3600,
  "refresh_token": "5defcb0a9a49ac9b2403b8c78600638238d81011",
}
```

```
"token_type": "bearer"
}
```

Transfer the access and refresh tokens to the user's product.

---

**Note:** Currently, a refresh token is valid for one year, while an access token is valid only an hour and an authorization code is valid only a minute.

---

### Using the Access Token to Make HKIoTCloud API Calls

When you call the HKIoTCloud API calls, pass the value of the access token into the request header. Specifically, create an `Authorization` header and give it the value `Bearer <access token>`.

#### Sample Request using curl:

- `curl -X GET -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" http://hkiotcloud.herokuapp.com/api/v1/init_session`

### Getting a New Access Token with Refresh Token

The access token is valid for one hour. When the access token expires or is about to expire, you can exchange the refresh token for new access and refresh tokens.

- Send a POST request to `https://hkiotcloud.herokuapp.com/oauth/token` with the following parameters:

#### HTTP Header Parameters:

- `Content-Type: application/x-www-form-urlencoded`

#### HTTP Body Parameters:

- `grant_type: refresh_token`
- `refresh_token`: The refresh token returned with the last request for a new access token.
- `client_id`: Your application's client ID. This information can be found on the HKIoTCloud website.
- `client_secret`: The application's client secret. This information can be found on the HKIoTCloud website.

#### Sample Request:

```
POST /oauth/token HTTP/1.1
Host: hkiotcloud.herokuapp.com
Content-Type: application/x-www-form-urlencoded
Cache-Control: no-cache

grant_type=refresh_token&refresh_token=5defcb0a9a49ac9b2403b8c78600638238d81011&client_id=n7HhiTnKYj
```

#### Sample Response:

```
HTTP/1.1 200 OK

{
  "access_token": "90da03bdceb15cf75d99ff99715ce87b29602651",
  "expires_in": 3600,
  "refresh_token": "6a762dfce9146dbf149f881c5aa15fc6cfd1fd0",
```

```

    "token_type": "bearer"
}

```

## 5.7 Troubleshooting (iOS)

### 5.7.1 Troubleshooting (iOS)

#### Handling of link error “undefined symbols for architecture armv7”

HKWirelessHD SDK uses C++ codes, so the linker should include std c++ library. To prevent this kind of link error, your project setting should include “-lstdc++” in Targets > Build Settings > Linking > Other Linker Flags field.

#### Unspecified linking parameter is added in link command

If you encounter unspecified linking parameter such as library names, etc., there is possibility that Xcode is using cached build parameters that were used before. In this case, just delete “Xcode/DerivedData” folder in your ~/Library/Developer folder. That is,

```

$ cd ~/Library/Developer/Xcode/DerivedData/
$ rm -rf [your-project-name]*

```

#### Playback stops when the app turns to background mode

When an app playing music using HKWirelessHDSdk may stop playing when the app becomes background. It is because iOS stops all on-going network communication when the app is backgrounded. There are several exceptional cases that iOS allows even in background mode. Please refer to [iOS Developer Library](#) for more information on background execution.

To prevent our HKWirelessHD apps from stopping in background mode, we can do a trick based on iOS Audio background mode. For our sample apps not to be stopped during the background mode, we use [MMPDeepSleepPreventer](#). The idea of MMPDeepSleepPreventer is to play zero length of silent audio every 5 seconds by enabling iOS Audio background mode.

To enable iOS Audio background mode, you need to enable it in Project setting.

- Go to Project > Targets > Capabilities > Background Modes
- Turn on the option of **Audio and AirPlay**

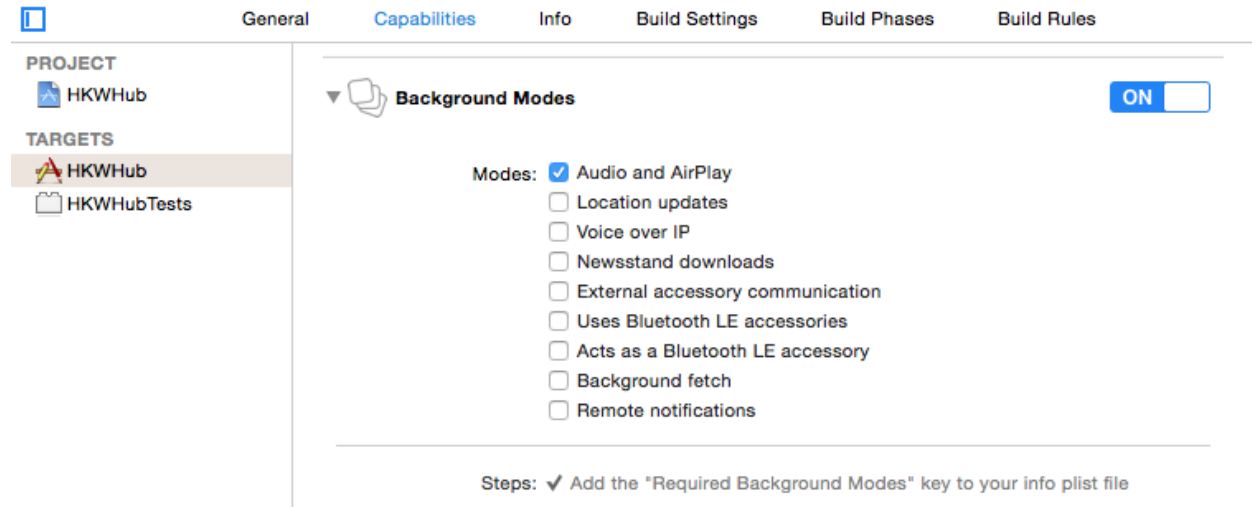
Then, just initialize and start DeepSleepPreventer in AppDelegate.application:didFinishLaunchingWithOptions() as follows:

```

func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject:
    // prevent from turning into background
    sleepPreventer = MMPDeepSleepPreventer()
    sleepPreventer.startPreventSleep()

```

You can stop the sleep preventer when the app becomes foregrounded (applicationWillEnterForeground()), and start it again just before the app becomes backgrounded (applicationWillResignActive()).



## Problem with iTunes Match and Apple Music

Currently, HKWirelessHD SDK does not support Cloud-based streaming from iTunes Match or Apple Music. To play an audio file on Omni speakers, the audio file should be available on the device in advance.

And, audio file from Apple Music is DRM-enabled. So, it is not supported by HKWirelessHD SDK either. Only the audio files that you purchased or uploaded to iTunes Match by matching can be played on Omni speakers after they are downloaded on the phone.

## MPMediaPicker not showing selection of items

With iOS8.4, MPMediaPicker does not show the items selected by user. This symptom appears only with iOS8.4. With earlier iOS version, such as iOS8.3 or before, the selected items turn grey color. But in iOS8.4, the picker does not show any change even the an item is selected. It seems a bug.

So, you need to be careful when you select items from MPMediaPicker. If you click on an item multiple times, the same item will appear on the Playlist the same multiple time as well.

## Creating a simple HTTP server for music streaming

For testing `PlayStreamingMedia()` in HKWirelessHD iOS SDK or `play_web_media` command in REST API, you just need to run a simple HTTP server on your local PC or Mac. The following is a quick example for setup a HTTP Server for music streaming.

- Put your mp3 or wav files on a folder, e.g. **music**
- Install python on your PC or Mac (Mac has already python installed.)
- **Run the followings:**
  - `$ python -m SimpleHTTPServer`
  - Then, you will get some logs like this: Serving on 0.0.0.0 port 8000 ...
- Find the IP address of your PC or Mac. Let's say it is 172.20.10.3.
- Now you can access the music file just like: <http://172.20.10.3:8000/music/sample.mp3>

## Compiling and Running HKWPlayer App

HKWPlayer app contains Apple Watch extension app inside. So, you can run the Watch app version of HKWPlayer app on your Apple Watch within the HKWPlayer app. A watch app is a kind of **Extension** app, and so we need to define a **App Group** so the main app and extension (watch) app can communicate with each other.

**App Groups** is defined by following the Bundle ID. The Bundle ID of HKWPlayer app is currently set as “com.harman.dev.hkwplayer”. So, App Group id should be set as “group.com.harman.dev.hkwplayer”, by adding “group” at the beginning. Currently, this bundle ID is used by Harman Developer Community team. So, you have to change this bundle ID for your use.

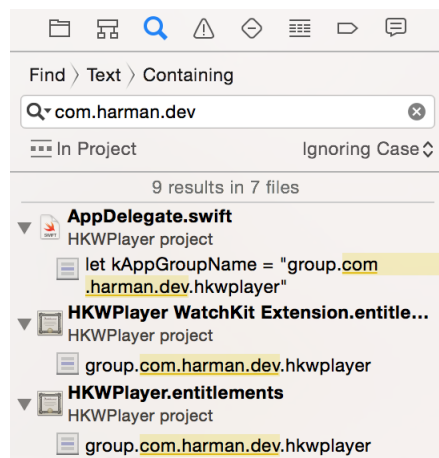
The following is how to change the bundle ID of the HKWPlayer app. First, change the bundle ID defined in App Groups. As example, let’s say your bundle id is “com.myproject”.

- **Go to Targets > HKWPlayer > Capabilities > App Groups**
  - Change the App Groups with “group.com.myproject”
- **Go to Targets > HKWPlayer WatchKit Extension > Capabilities > App Groups**
  - Change the App Groups with “group.com.myproject”

Now Change other parts of the codes that uses the bundle ID.

- **To go Search menu in project navigator, and type “com.harman.dev”**
  - You will see all the texts that contains the string.
- Click each item on the list, and then replace the stream with your own bundle ID, that is, “com.myproject”

The following is the screen capture of the list of the search.



Please follow the instruction [Configuring App Groups](#) in iOS Developer Library for more information.

## 5.8 Version History (iOS)

### 5.8.1 Version History (iOS)

V1.2 - Jul. 8 , 2015

- HKWirelessHDSDK supports two versions of SDK: normal version and lightweight version

- HKWirelessHDS SDK (normal version) - support for playback of web streaming audio
- HKWirelessHDS SDKlw (lightweight version) : lightweight version : no support for web streaming audio
- Added mute(), unmute(), isMuted()
- **New sample apps**
  - HKWPlayer app: a sample player app with full usage of HKWirelessHD APIs. Apple Watch support.
  - HKWHub app : Web hub app for handling REST API request
  - HKWSimple app: a very simple player app

### V1.1 - Apr. 21, 2015

- **Use delegate patterns for receiving events from HKWControlHandler.**
  - Added HKWDeviceEventHandlerDelegate.h and HKWPlayerEventHandlerDelegate.h

### V1.0 - Jan, 2015

- Initial version
- 

## 5.9 SDK Overview (Android)

### 5.9.1 Overview of HKWirelessHD SDK (Android)

The Harman Kardon WirelessHD (HKWirelessHD) SDK is provided for 3rd party developers to communicate with Harman/Kardon Omni Series audio/video devices. The intent of this SDK is to provide the tools and libraries necessary to build, test and deploy the latest audio applications on the Android platform.

#### What's Included

- HKWirelessHD library and jar package
- License Agreement. Located within the root directory of the zip file.
- **This Document. Located in the HKWirelessHDS SDK.zip (HKWirelessHDS SDK/doc folder). There are some documents available:**
  - Getting Started (Android)
  - Programming Guide (Android)
  - API documentation (Android)
- **Sample Application source code Located within the Sample Apps page:**
  - WirelessSDK Demo App



## Requirements

The HKWirelessHD SDK requires Android 4.1(API 16) minimum for Android devices. The SDK supports both 32bit and 64bit architecture.

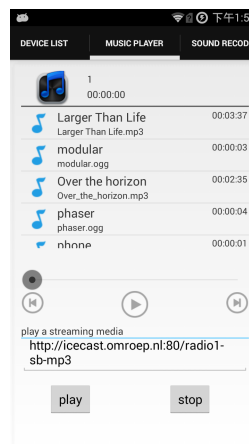
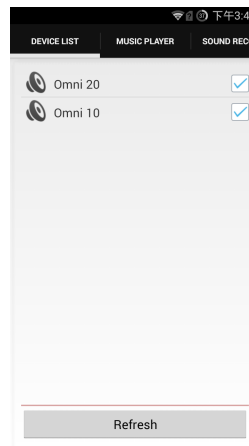
## Demo Applications

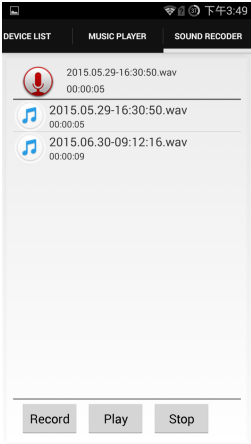
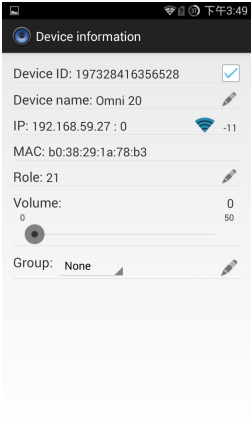
### WirelessSDK Demo (a music player and voice recorder for getting started with HKWirelessHDSDK)

HKWSimple app is a simple music player that was created to explain how to create an app with HKWirelessHD SDK. This app is very simple, but contains key features of HKWirelessHD SDK, such as, manage speakers, control audio playback and volume, play local media files, play http stream media and so on.

The app can record user's voice and broadcast it to a set of selected speakers in the network. User can select speakers individually or select rooms.

Just get started with the WirelessSDK Demo app to quickly build your own HKWirelessHD app!





## 5.10 Getting Started Guide (Android)

### 5.10.1 Getting Started Guide (Android)

The Harman/Kardon WirelessHD SDK is provided for Android 3rd party developers to communicate with Harman/Kardon Omni Series audio/video devices. The intent of this SDK is to provide the tools and libraries necessary to build, test and deploy the latest audio applications on the Android platform.

#### Requirements

The HKWirelessHD SDK requires Android 4.1(API 16) minimum for Android devices. The SDK supports both 32bit and 64bit architecture.

#### Creating a Sample Application

##### 1. Add Jar package and library in your project

Add the libHKWirelessHD.jar package and libHKWirelessHD.so library in your libs folder.

- The step in Android Studio:

1. Copy the HKWirelessHD.jar to app/libs folder and copy the libHKWirelessHD.so to app/src/main/jniLibs/armeabi folder. If the folders don't exist, please create them.

The directory hierarchy as following:

2. Add the jar package as library. Select the menu: File->Project Structure->app(under Modules)->Dependencies, push +then File dependencyselect HKWirelessHD.jar. As following:

- The step in Eclipse:

1. copy the HKWirelessHD.jar to libs folder and copy the libHKWirelessHD.so to libs/armeabi folder. If the folders don't exist, please create them.

The directory hierarchy as following:

2. Add the jar package as library. Select the menu: Project property->Java Build Path->Libraries select "Add External JARs"choose the HKWirelessHD.jar.Then add the jar package as external JARs.

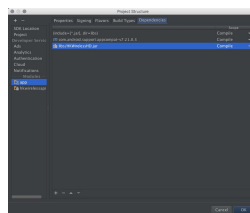
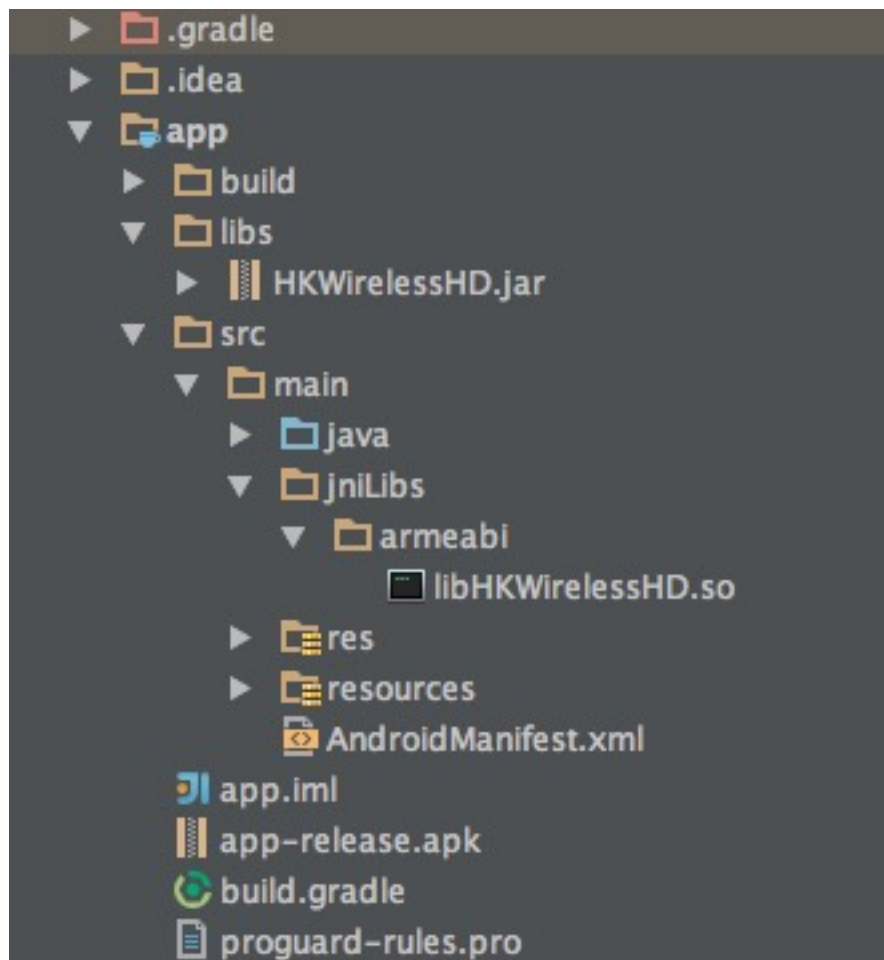
##### 2. Add Permission in AndroidManifest.xml file

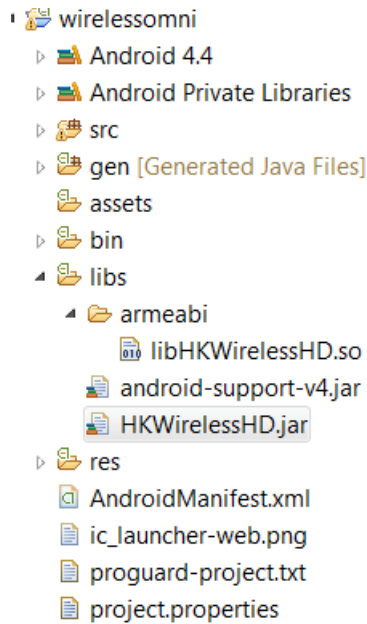
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
```

##### 3. Import package

Add the package header to your code:

```
import com.harman.hkwirelessapi.*
```





#### 4. Create HKWirelessHD Control Handler and initialize the Wireless Audio

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

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

// Initialize the HKWControlHandler and start wireless audio
AudioCodecHandler hAudioControl = new AudioCodecHandler();

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

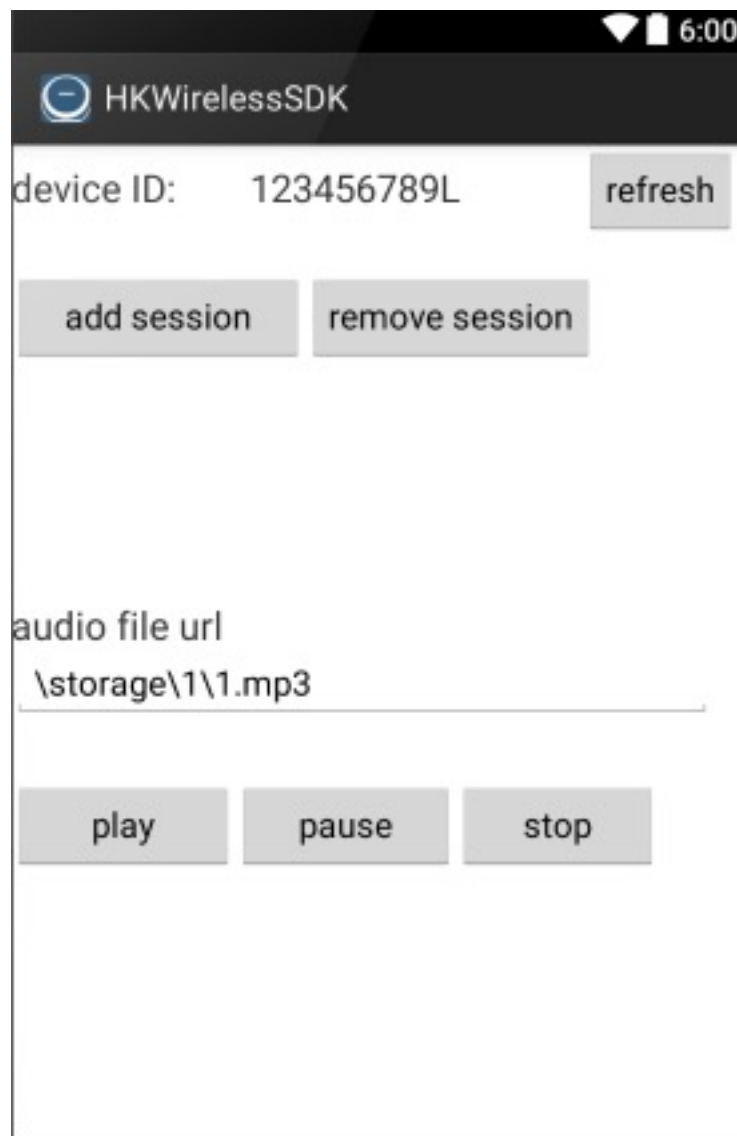
    // Initialize the HKWControlHandler and start wireless audio
    hControlHandler.initializeHKWirelessController("...");
}
```

#### 5. Create application interface

The app is a simple example. Make an activity with some buttons to control the omni device. The interface of app as following:

#### 6. Discovery of the available speakers

To discover and update the status of speakers, you need to refresh the status regularly. The SDK provides a pair of convenient APIs to refresh device status. You can make a button to check the status of devices in the network. For this application, there is only one device in the network.



```
//refresh device button
(this.findViewById(R.id.refresh_btn)).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // start to refresh devices
        hControlHandler.refreshDeviceInfoOnce();
    }
});
```

## 7. Implement callbacks

All the updates from the speaker side are reported to the phone via callbacks. So, if your app needs the latest information of the speakers in certain cases, you should use corresponding callbacks accordingly. We can get the device ID from `onDeviceStateUpdated()`. For this application, it is supposed that device ID “123456789” is found in the network.

```
hControlHandler.registerHKWirelessControllerListener(new HKWirelessListener() {
    @Override
    public void onPlayEnded() {
        Log.d(LOG_TAG, "PlayEnded");
    }

    @Override
    public void onPlaybackStateChanged(int playState) {
        Log.d(LOG_TAG, "PlaybackState : " + playState);
    }

    @Override
    public void onPlaybackTimeChanged(int timeElapsed) {
        Log.d(LOG_TAG, "TimeElapsed : " + timeElapsed);
    }

    @Override
    public void onVolumeLevelChanged(long deviceId, int deviceVolume, int avgVolume) {
        Log.d(LOG_TAG, "DeviceId: " + deviceId + "Volume:" + deviceVolume);
    }

    @Override
    public void onDeviceStateUpdated(long deviceId, int reason) {
        Log.d(LOG_TAG, "DeviceStateUpdated: " + deviceId);
    }

    @Override
    public void onErrorOccurred(int errorCode, String errorMsg) {
        Log.d(LOG_TAG, "Error: " + errorMsg);
    }
});
```

## 8. Find info of speakers and groups

Get DeviceInfo with deviceId You know the deviceId of a speaker from `onDeviceStateUpdated()` when new device and group found, then you can get the information of the device using DeviceInfo object.

```
@Override
public void onDeviceStateUpdated(long deviceId, int reason) {
```

```

        // get the number of available speakers
        DeviceObj deviceInfo = hControlHandler.findDeviceFromList(deviceId)
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);
}

```

### And GroupInfo

```

@Override
public void onDeviceStateUpdated(long deviceId, int reason) {
    //Get the number of groups available in the network
    int groupCount = hControlHandler.getGroupCount();
    Log.d(LOG_TAG, "group cnt :" + groupCount);
    for(int i = 0; i < groupCount; i++){
        // get the each group
        GroupObj group = wireless.getDeviceGroupByIndex(i);
        Log.d(LOG_TAG, group.groupId + " group groupName :" + group.groupName);
        Log.d(LOG_TAG, group.groupId + " group device cnt :" + group.deviceList.length);

        // get the speakers of this group
        for(int j = 0; j < group.deviceList.length; j++){
            DeviceObj obj1 = wireless.getDeviceInfoFromTable(i, j);
            Log.d(LOG_TAG, obj1.deviceId + " obj1 :" + obj1.deviceName);
            Log.d(LOG_TAG, group.groupId + " group deviceId :" + group.deviceList[j].deviceId);
        }
    }
}

```

## 9. Add/remove a speaker to/from a playback session

If want to play audio through a speaker, the speaker must be added to the playback session through `addDeviceToSession()`. And the speaker can be removed from the playback session through `removeDeviceFromSession()`.

We use the device ID that you get from `onDeviceStateUpdated()`.

```

        //add the speaker to the current playback session from the select list
        (this.findViewById(R.id.add_btn)).setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
    long deviceId = 123456789L;
    hControlHandler.addDeviceToSession(deviceId)
}
});

```



```
//remove a speaker from the current playback session from the unselect list
(this.findViewById(R.id.remove_btn)).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        long deviceId = 123456789L;
        hControlHandler.removeDeviceFromSession(deviceId)
    }
});
```

## 10. Play an audio file

If one or more speakers are added to the session, then you can start to play a song. Currently, use `playCAF()` to play mp3, wav, flac, sac, m4a and ogg file, and `playWAV` only for WAV file, and `playStreamingMedia()` for http web server.

```
//Play a audio file from the play list
(this.findViewById(R.id.play_btn)).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        //play a song
        String url = textUrl.getText().toString();
        String songTitle = textName.getText().toString();
        hAudioControl.playCAF(url, songTitle, false)
    }
});

//pause
(this.findViewById(R.id.pause_btn)).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        hAudioControl.stop();
    }
});

//stop
(this.findViewById(R.id.stop_btn)).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        hAudioControl.stop();
    }
});
```

## 11. Volume Control

You can set device volume through `setVolumeDevice()`. This application shows how to control the device volume through phone volume button. The volume level ranges from 0 (mute) to 50 (max).

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    //get device from your select list
    long deviceId = 123456789L;
    DeviceObj deviceInfo = hControlHandler.findDeviceFromList(deviceId)
    int volume = DeviceInfo.volume;
```

```
switch (keyCode) {
    case KeyEvent.KEYCODE_VOLUME_UP:
        volume+=5;
        hAudioControl.setVolumeDevice(deviceId, volume);
    }
    return true;

    case KeyEvent.KEYCODE_VOLUME_DOWN:
        volume -= 5;
        hAudioControl.setVolumeDevice(deviceId, volume);
    }
    return true;
}
return super.onKeyDown(keyCode, event);
}
```

## 5.11 Programming Guide (Android)

### 5.11.1 Programming Guide (Android)

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

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

#### Key Features of HKWirelessHD

- **Wirelessly stream audio to HK Omni Speakers** Omni speakers are attached to a WiFi network. You can manage and control speakers in the same WiFi network to play audio wirelessly.
- **Support multi-room audio streaming** You can play audio to 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, you can send different channels of a single audio stream to different speakers. For example, to play 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.

#### SDK information

##### Overview

There are two kinds 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 a one-to-many way. That is, there is one single source device streaming an audio file, and multiple destination devices receiving 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.)

### Use of HKWirelessHD API to stream audio to Omni Speakers

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

### Communication channels between source and destinations

As shown in the figure above, there are two kind of communications between the 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 does not belong 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 device availability and change of its attributes, whether or not it is playing music, the Wi-Fi signal strength, volume change, etc.

### Asynchronous Communication

The communication between the source device and the destination speakers is done in an asynchronous way. Asynchronous behavior is expected because all the commands and status updates are executed in 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 hundred 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 milliseconds later.

### 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 of each attribute in the device are described in DeviceObj.java.

### 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.

## **Controlling Speakers and Handling the Events from Speakers**

### **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`.

### **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

### **Programming Guide**

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

All APIs can be accessed through the object pointer of `HKWirelessHandler` and `AudioCodecHandler`. All 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`.

### **Initialization**

Controlling HK Omni speakers is done by calling APIs provided by `HKWControlHandler`. So, the first thing to do to use `HKWirelessHD` APIs is to acquire the 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. In the event you have not received a license key code yet you should use the key code in the sample app.

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

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

**Note** `InitializeHKWirelessController()` 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** 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.

### 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.

#### 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 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** 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	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
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
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
volume	Int	the volume level value (0 to 50)	Variable	Yes
active	boolean	indicates if added to the current playback session	Variable	Yes
wifiSignal-Strength	Int	Wi-Fi strength in dBm scale, -100 (low) to 0 (high)	Variable	No
role	Int	the role definition (stereo or 5.1 channel)	Variable	Yes
version	String	the firmware version number as String	Fixed (when firmware update)	No
balance	Int	the balance value in stereo mode. -6 to 6, 0 is neutral	Variable	No
isPlaying	boolean	indicates whether the speaker is playing or not	Variable	No
channelType	Int	the channel type: 1 is stereo.	Variable	No
isMaster	boolean	indicates if it is the master in stereo or group mode	Variable	No

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);
```

### Getting a speaker (device) information

HKWControlHandler maintains the list of speakers internally. 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.

#### Get the speaker information from the table

You can retrieve 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.

### Get speaker information from the group list

You can retrieve 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 speaker information in two section ListView, the groupIndex can correspond to the section number, and deviceIndex can correspond to the row number. Get 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);
```

### 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.

### 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;
```

### 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.

### 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** A speaker can be removed from the current on-going playback session anytime.

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

### Play a song

**Play audio 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. The “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** `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 device.

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();
```

### Play a streaming media

```
void playStreamingMedia(String url)
```

Plays a streaming media from web server. Because this API takes a little while to get the result of play because of all networking stuffs, the API is a block call. So, it will not return until the caller successfully or fail.

`String playStreamingMedia` - a string that specifies the URL of the streaming media source. It starts with a protocol name, such as `http://`. Currently, only `http` is supported. The supported file format is mp3, wav, flac, sac, m4a or ogg.

### Playback controls

**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.

```
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.

**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.

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

**Volume Control** You can set volume 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 milliseconds) 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.

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

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

**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);
```

### Get volume of all speakers

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

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

### Get volume of a particular speaker

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

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

**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.

### 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.

**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.

**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);
```

### 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. Similarly, 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`.

**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.

**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.

**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.

**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.

**onPlaybackStateChanged** This function is invoked when playback state is changed during the playback. The call-back delivers the playState value as parameter.

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

**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 seconds) 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);
```

### 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) {  
}  
}
```

## 5.12 API Documentation (Android)

### 5.12.1 API Documentation (Android)

#### Initialization

##### initializeHKWirelessController()

Initializes and starts HKWirelessHD controller. This API requires a license key as string. If the input license key fails in key validation, then the API returns -1. If it is successful, return 0.

The license key will be delivered to you once you register on developer.harman.com. Until it is delivered, you may use the license key included in the sample apps.

#### Note

This is a blocking call, and it will not return until HKWireless controller initialized. If the phone is not connected to a Wi-Fi network, or any other app in the same phone is using the HKWireless controller, it waits until the other app releases the controller.

If you need non-blocking behavior, you should call this API asynchronously using other thread.

#### Signature:

```
int initializeHKWirelessController(String key)
```

#### Parameters:

String key - the license key

#### Returns:

int - success (0 or HKW\_INIT\_SUCCESS) or failure (-1 or HKW\_INIT\_FAILURE\_LICENSE\_INVALID)

##### isInitialized()

Checks if HKWirelessHD controller is initialised.

#### Signature:

```
boolean isInitialized()
```

#### Returns:

boolean - True if HKWirelessController has been initialized, false not initialized.

## Refreshing Speaker Information

### refreshDeviceInfoOnce()

Refresh the devices one time. The device information will be refreshed and updated to DeviceInfo objects. If there is any update the onDeviceStateUpdated function will be called. you can find which device has been updated and what was the reason.

**Signature:**

```
void refreshDeviceInfoOnce()
```

**Returns:**

```
void
```

### startRefreshDeviceInfo()

Start to refresh DeviceInfo every two seconds, until stopRefreshDeviceInfo() is called.

**Signature:**

```
void startRefreshDeviceInfo()
```

**Returns:**

```
void
```

### stopRefreshDeviceInfo()

Stop refreshing DeviceInfo, which was started by startRefreshDeviceInfo().

**Signature:**

```
void stopRefreshDeviceInfo()
```

**Returns:**

```
void
```

## Playback Control

### playCAF()

Play an audio file in local storage. If it is successful, onPlaybackStateChanged() function will be called and return the status value of HKPlayerState.EPlayerState\_Play.

Currently, it supports formats of mp3, wav, flac, sac, m4a and ogg, and the sample rate must be 44100 and above. playCAF is used to play mp3, wav, flac, sac, m4a or ogg file, and playWAV is only for WAV file. **Signature:**

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

**Parameters:**

String url - Path of audio file

String songName - the song name is used internally to save the temporary PCM file generated from the original audio file.

`boolean resumeFlag` - If the playback should resume from the paused point . ResumeFlag should be false if start the song from the beginning, and if the playback was stopped, then the playback will start from the beginning even with resumeFlag true.

Returns:

`boolean` - Success or failure

### **playCAFFromCertainTime()**

Play an audio file from a certain time specified by `startTime`. For example, you can start a song from the point of 10 seconds of the song. `onPlaybackStateChanged()` function will be called and return the status value of `HKPlayerState.EPlayerState_Play`.

**Signature:**

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

**Parameters:**

`String url` - Path of audio file.

`String songName` - the song name is used internally to save the temporary PCM file generated from the original audio file. `int startTime` - time in seconds that specifies the start time.

**Returns:**

`boolean` - Success or failure

### **playWAV()**

Play a WAV file. `onPlaybackStateChanged` function will be called and return the status of `HKPlayerState.EPlayerState_Play`.

**Signature:**

```
boolean playWAV(String url)
```

**Parameters:**

`String url` - Path of the wav file

**Returns:**

`boolean` - Success or failure

### **playStreamingMedia()**

Plays a streaming media from web server. The API is a synchronous call. It will not return until the caller succeeds or fails. You can find the error message from the function of `onErrorOccurred()`

**Signature:**

```
void playStreamingMedia(String streamingMediaUrl)
```

**Parameters:**

`String streamingMediaUrl` - The URL of the streaming media source. It starts with a protocol name, such as `http://`. Currently, only `http` is supported.

**Returns:**

void

### **pause()**

Pause current playing. `onPlaybackStateChanged()` function will be called and return the status of `HKPlayerState.EPlayerState_Pause`. Once the playback is paused, it can be resumed by `playCAF()` with `resumeFlag` to be true and the same `songName`. the playback is stopped by `pause`, if it was played by `playStreamingMedia()`.

#### **Signature:**

```
void pause()
```

#### **Returns:**

void

### **stop()**

Stop playing. `onPlaybackStateChanged()` function will be called and return the status of `HKPlayerState.EPlayerState_Stop`.

#### **Signature:**

```
void stop()
```

#### **Returns:**

void

### **isPlaying()**

Check if the player is playing or not.

#### **Signature:**

```
boolean isPlaying()
```

#### **Returns:**

boolean - boolean value indicating if the player is playing or not.

### **getPlayerState()**

Get current state of playback.

#### **Signature:**

```
HKPlayerState getPlayerState()
```

#### **Returns:**

HKPlayerState - Current player state.



## Volume Control

### setVolumeAll()

Set a volume level to all speakers.

The range of volume level is 0 to the `maximumVolumeLevel` (currently, 50) by `getMaximumVolumeLevel()`.

It is asynchronous call. The volume will be set within a few milliseconds. The `onVolumeLevelChanged()` function defined in `HKWirelessListener` will be called when volume of specified speaker has been changed.

#### Signature:

```
void setVolumeAll(int volume)
```

#### Parameters:

`int volume` - the volume level

#### Returns:

`void`

### setVolumeDevice()

Set a volume level to a speaker specified by device ID. The range of volume level is 0 to the `maximumVolumeLevel` (currently, 50) by `getMaximumVolumeLevel()`. It is asynchronous call. The effect of the API call will occur after a few milliseconds. The `onVolumeLevelChanged()` function will be called when volume of specified speaker has been changed.

#### Signature:

```
void setVolumeDevice(long deviceId, int volume)
```

#### Parameters:

`long deviceId` - Device ID of the speaker

`int volume` - Volume level

#### Returns:

`void`

### getVolume()

Get the average volume level for all devices.

#### Signature:

```
int getVolume()
```

#### Returns:

`int` - the average volume level of all speakers

### **getDeviceVolume()**

Get the volume level of the specified speaker.

#### **Signature:**

```
int getDeviceVolume(long deviceId)
```

#### **Parameters:**

long deviceId - Device Id of the speaker.

#### **Returns:**

int - Volume level

### **getMaximumVolumeLevel()**

Return the maximum volume level that the system provides. Currently, it is 50.

#### **Signature:**

```
int getMaximumVolumeLevel()
```

#### **Returns:**

int - the maximum volume level

## **Device (Speaker) Management**

### **addDeviceToSession()**

Add a speaker to the current playback session. The added speaker will start playing. This can be called during playing.

#### **Signature:**

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

#### **Parameters:**

long deviceId - Device Id of the speaker.

#### **Returns:**

boolean - Whether the addition is successful or not.

### **removeDeviceFromSession()**

Remove a speaker from the current playback session. The removed speaker will stop playing. This can be called during playing.

#### **Signature:**

```
boolean removeDeviceFromSession(long deviceid)
```

#### **Parameters:**

long deviceId - Device Id of the speaker.

#### **Returns:**

boolean - Whether the removal is successful or not.

**getDeviceCount()**

Get the number of speakers in the HKWirelessHD network.

**Signature:**

```
int getDeviceCount ()
```

**Returns:**

int - the number of speakers.

**getGroupCount()**

Get the number of groups.

**Signature:**

```
int getGroupCount ()
```

**Returns:**

int - the number of groups

**getDeviceCountInGroupIndex()**

Get the number of speakers in the group specified by the index.

**Signature:**

```
int getDeviceCountInGroupIndex(int groupIndex)
```

**Parameters:**

int groupIndex - the index of the group looking for. It starts from 0 to (GroupCount-1).

**Returns:**

int - the number of speakers

**getDeviceInfoFromTable()**

Return the DeviceInfo of specified device in specified group.

**Signature:**

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

**Parameters:**

int groupIndex - group index in the group list.

int deviceIndex - The index of the speaker in the group.

**Returns:**

DeviceObj - the device information

### **getDeviceInfoByIndex()**

Return the DeviceInfo of specified device from the global table from 0 to (deviceCount - 1).

#### **Signature:**

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

#### **Parameters:**

int deviceIndex - The index of the device from the table with all devices.

#### **Returns:**

DeviceObj - the device information

### **getDeviceGroupByDeviceId()**

Return the DeviceGroup the speaker belongs to.

#### **Signature:**

```
GroupObj findDeviceGroupWithDeviceId(long deviceId)
```

#### **Parameters:**

long deviceId - the device ID

#### **Returns:**

GroupObj - the group information

### **getDeviceInfoById()**

Find the a DeviceInfo by device Id. It is used to retrieve the DeviceInfo of specified device.

#### **Signature:**

```
DeviceObj getDeviceInfoById(long deviceId)
```

#### **Parameters:**

long deviceId - the device ID.

#### **Returns:**

DeviceObj - the device information

### **isDeviceAvailable()**

Check whether the specified speaker is available or not.

#### **Signature:**

```
boolean isDeviceAvailable(long deviceId)
```

#### **Parameters:**

long deviceId - the device ID.

#### **Returns:**

boolean - the speaker is available or not.

**isDeviceActive()**

Check whether the speaker is active (added to the current playback session) or not.

**Signature:**

```
boolean isDeviceActive(long deviceId)
```

**Parameters:**

long deviceId - The ID of the speaker

**Returns:**

boolean - if the speaker is active or not.

**removeDeviceFromGroup()**

Remove the specified speaker from its group.

**Signature:**

```
void removeDeviceFromGroup(long groupId, long deviceId)
```

**Parameters:**

long groupId - group ID

long deviceId - device ID

**Returns:**

void

**getDeviceGroupByIndex()**

Get the group information.

**Signature:**

```
GroupObj getDeviceGroupByIndex(int groupIndex)
```

**Parameters:**

int groupIndex - group index

**Returns:**

GroupObj - the group information

**getDeviceGroupById()**

Get the group information by group ID.

**Signature:**

```
GroupObj getDeviceGroupById(long groupId)
```

**Parameters:**

long groupId - the ID of the group

**Returns:**

GroupObj - the group information

### **getDeviceGroupNameByIndex()**

Get the name of the DeviceGroup by index.

#### **Signature:**

```
String getDeviceGroupNameByIndex(int groupIndex)
```

#### **Parameters:**

int groupIndex - the index of the group in the group table.

#### **Returns:**

String - the string of group name

### **getDeviceGroupIdByIndex()**

Gets the ID of the DeviceGroup by index.

#### **Signature:**

```
long getDeviceGroupIdByIndex(int groupIndex)
```

#### **Parameters:**

int groupIndex - the index of the group in the table

#### **Returns:**

long - the group id

### **setDeviceName()**

Set device name.

Note

If you set a device name while it is playing, the speaker will stop playing and return error.

#### **Signature:**

```
void setDeviceName(long deviceId, String deviceName)
```

#### **Parameters:**

long deviceId - The ID of the device

String deviceName - The name of the device to set

#### **Returns:**

void

**setDeviceGroupName()**

Set group name.

Note

If you set a group name while the a playback is running, the speaker stops playing and return error.

**Signature:**

```
void setDeviceGroupName(long deviceId, String groupName)
```

**Parameters:**

long deviceId - The device ID within a group

String groupName - the group name

**Returns:**

void

**getActiveDeviceCount()**

Get the number of active speakers (the speakers that are added to the current playback session.)

**Signature:**

```
int getActiveDeviceCount()
```

**Returns:**

int - the number of active devices

**getActiveGroupCount()**

Get the number of active groups. An active group is the one that all the speakers in the group are active.

**Signature:**

```
int getActiveGroupCount()
```

**Returns:**

int - the number of active groups

**refreshDeviceWiFiSignal()**

Refresh the devices Wifi Signal strength value. This is asynchronous call, and the result of refreshing will come a few milliseconds later. The new WiFi signal strength value will be reported by `onDeviceStateUpdated()`. You should get the wifi signal value by getting `wifiSignalStrength` attribute of `DeviceInfo` object.

**Signature:**

```
void refreshDeviceWiFiSignal(long deviceId)
```

**Parameters:**

long deviceId - the device ID

**Returns:**

void

### getWifiSignalStrengthType()

Get Wifi signal strength type by signal value.

#### Signature:

```
HKWifiSingalStrength getWifiSignalStrengthType(int wifiSignal)
```

#### Parameters:

int wifiSignal - the wifi signal value

#### Returns:

HKWifiSingalStrength - Wifi signal strength type

### APIs for events handling

#### onDeviceStateUpdated()

The function is called when devices status is changed, including active or inactive, model name, group name, wifi strength, etc.

Note if the volume level changed, VolumeLevelChanged function would be called instead.

This function is essential to retrieve and update speaker information in timely manner. If you want to show the available speakers and its latest information, you could get from the function in timely manner.

#### Signature:

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

#### Parameters:

long deviceId - the device ID

int reason - the reason code

#### Returns:

void

#### onErrorOccurred()

The function is invoked when an error occurs. The function returns the error code and error message.

#### Signature:

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

#### Parameters:

int errorCode - error code

String errorMesg - error description

#### Returns:

void



### **onPlayEnded**

This function is invoked when the current playback has ended.

**Signature:**

```
void onPlayEnded()
```

**Returns:**

```
void
```

### **onVolumeLevelChanged()**

This function is invoked when volume level is changed.

The function delivers the device ID device volume level, and average volume level.

**Signature:**

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

**Parameters:**

long deviceId - the device ID

int deviceVolume - the volume level

int avgVolume - the average volume level

**Returns:**

```
void
```

### **onPlaybackStateChanged()**

This function is invoked when playback state is changed. The function delivers the playState.

**Signature:**

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

**Parameters:**

int playState - The player state

**Returns:**

```
void
```

### **onPlaybackTimeChanged()**

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

**Signature:**

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

**Parameters:**

int timeElapsed - the time (in second) passed since the beginning of the playback.

**Returns:**

void

## 5.13 Version History (Android)

### 5.13.1 Version History

#### V1.0 - Jul, 2015

- Initial version

#### V1.1 - Aug, 2015

- Modify some document errors

#### V1.2 - Oct, 2015

- Add API playStreamingMedia()

#### V1.3 - Nov, 2015

- Add API document
  - Modify the getting-started document
- 

## 5.14 Pulse2 SDK Documentation

### 5.14.1 JBL Pulse2 iOS Framework Developer's Guide

**Version** 1.0

#### Prerequisites

The following steps should be done by Developer before start working with SDK:

- Add the Pulse2SDK framework to Linked Frameworks and Libraries
- Import API header file "HMNPulse2API.h"
- Import Apple ExternalAccessory.framework to Xcode project
- Add protocol string "com.jbl.connect" in the UISupportedExternalAccessoryProtocols section on the Info.plist

iPhone should establish BT connection with Pulse 2 Device in iPhone-> Settings -> Bluetooth.

## 5.14.2 API Description

### Device General API methods

Application should be first establish BT connection with Device using **connectToMasterDevice** function (HMNDeviceGeneral.h).

The notification `EVENT_DEVICE_CONNECTED` (HMNCommonDefs.h) will be received if connection established successfully. Other API functions can be used after this step.

The notification `EVENT_DEVICE_CONNECTED` includes dictionary with the following keys (defined in HMNDeviceGeneral.h):

- `KEY_IAP_CONNECTION_ID`;
- `KEY_IAP_MANUFACTURER`;
- `KEY_IAP_NAME`;
- `KEY_IAP_SERIAL_NUMBER`;

The notification `EVENT_DEVICE_DISCONNECTED` (HMNCommonDefs.h) will be received if connection with device lost.

### Device Info API methods

```
/**
 * @discussion Change device name.
 * @param deviceName An new device name. Nonnull.
 * @return none.
 */
• (void) setDeviceName :(NSString * _Nonnull)deviceName;

/**
 * @discussion Request information from Device like name, battery, active channel.
 * @return none.
 */
• (void) requestDeviceInfo;
```

Application will receive notification “`EVENT_DEVICE_INFO`” with Dictionary included device info. The following keys can present in dictionary (HMNDeviceInfo.h):

- `KEY_DEVICE_INFO_DEVICE_INDEX`;
- `KEY_DEVICE_INFO_DEVICE_NAME`;
- `KEY_DEVICE_INFO_PRODUCT_ID`;
- `KEY_DEVICE_INFO_MODEL_ID`;
- `KEY_DEVICE_INFO_BATTERY_IS_CHARGING`;
- `KEY_DEVICE_INFO_BATTERY_VALUE`;
- `KEY_DEVICE_INFO_LINKED_DEVICE_COUNT`;
- `KEY_DEVICE_INFO_ACTIVE_CHANNEL_VALUE`;

- KEY\_DEVICE\_INFO\_AUDIO\_SOURCE\_VALUE;
- KEY\_DEVICE\_INFO\_MAC\_ADDRESS\_VALUE;

/\*\*

\* @discussion Request information from Device for the specified token

\* @param token. Value from enumeration HMNToken

\* @param index. Device index: 0 - master, 1 - slave.

\* @return none.

\*/

- (void) **requestDeviceInfoToken** :(HMNToken)token forDeviceIndex:(UInt8)index;

## DFU API methods

/\*\*

\* @discussion Request version from Device.

\* @return none.

\*/

- (void) **requestVersion**;

Application will receive notification “EVENT\_VERSION” with Dictionary included version info. The following keys can present in dictionary (HMNDFU.h):

KEY\_SW\_VERSION;

KEY\_HW\_VERSION;

## Led Control API methods

/\*\*

\* @discussion Setup Background Color to Master Device

\* @param color - UIColor object

\* @param propagateToSlaveDevice - Need to propagate background color to Slave device

\* @return none.

\*/

- (void) **setBackgroundColor**:(UIColor \* \_Nonnull)color propagateToSlaveDevice:(BOOL)applyToSlave;

/\*\*

\* @discussion Draw 11(columns)x9(rows) color image bitmap on the Master Device

\* @param imageMatrix - Array of 99 UIColor\* objects, The color of each pixel is represented color by UIColor.

\* @return none.

\*/

- (void) **setColorImage**:(NSArray \* \_Nonnull)imageMatrix;

```

/**
 * @discussion Display Char on master and slave device
 * @param charAsciiCode - Char ascii code - in hex; Supported ascii symbols: 0..9 A..Z ? ! $ + - = % * / # " & ' ( ) , .
: ; < > ' { | } ~
 * @param charColor - Color of char;
 * @param backgroundColor - Background color;
 * @param applyToSlave - if need to propagate to slave device
 * @return none.
 */
    • (void) setLedChar:(UInt8)charAsciiCode charColor:(UIColor * _Nonnull)charColor background-
      Color:(UIColor * _Nonnull)backgroundColor applyToSlaveDevice:(BOOL)applyToSlave;
/**
 * @discussion Propagate Current Led Pattern on master to slave device
 * @return none.
 */
    • (void) propagateCurrentLedPattern;
/**
 * @discussion Query device led brightness.
 * It will reply with EVENT_BRIGHTNESS notification
 * @return none.
 */
    • (void) requestLedBrightness;

```

Application will receive notification “EVENT\_BRIGHTNESS” with Dictionary included brightness info. The following keys can present in dictionary (HMNLedControl.h):

```

    • KEY_LED_BRIGHTNESS
/**
 * @discussion Set Display Led Brightness on master device
 * @param brightness - 0..255;
 * @return none.
 */
    • (void) setLedBrightness:(UInt8)brightness;
/**
 * @discussion Set Pattern on master device
 * @param pattern - id of pattern
 * @param patternData - array of 99 unsigned int values wrapped to NSNumbers for Canvas and Firefly:
 * 0 - do not draw point
 * non-0 - draw point on canvas

```

\* - nil for other pattern types

\* @return none.

\*/

- (void) **setLedPattern**:(HMNPattern)pattern withData:(NSArray \* \_Nullable)patternData;

/\*\*

\* @discussion Query device led pattern information.

\* It will reply with EVENT\_PATTERN\_INFO notification.

\* @return none.

\*/

- (void) **requestLedPatternInfo**;

Application will receive notification “EVENT\_PATTERN\_INFO” with Dictionary included pattern info. The following keys can present in dictionary (HMNLedPattern.h):

KEY\_LED\_PATTERN\_ID its id of the pattern enumerated in the HMNCommonDefs.h

/// Device Patterns

```
typedef NS_ENUM(NSInteger, HMNPattern) {
```

```
HMNPattern_Firework = 0,
```

```
HMNPattern_Traffic,
```

```
HMNPattern_Star,
```

```
HMNPattern_Wave,
```

```
HMNPattern_FireFly,
```

```
HMNPattern_Rain,
```

```
HMNPattern_Fire,
```

```
HMNPattern_Canvas,
```

```
HMNPattern_Hourglass,
```

```
HMNPattern_Ripple
```

```
};
```

### Sensor Control API methods

/\*\*

\* @discussion Request microphone sound level from Device.

\* It will reply with EVENT\_SOUND notification

\* @return none.

\*/

- (void) **requestMicrophoneSoundLevel**;

Application will receive notification “EVENT\_SOUND” with Dictionary included pattern info. The following key can present in dictionary (HMNSensorControl.h):

```
/// The Value has NSNumber value
```

```
KEY_MICROPHONE_LEVEL;
```

```
/**
```

```
* @discussion Request for capturing color by Color Picker from the Device
```

```
* It will reply with EVENT_SENSOR_CAPTURE_COLOR notification
```

```
* @return none.
```

```
*/
```

- (void) **requestColorFromColorPicker;**

When device color sensor capture the color it send notification EVENT\_SENSOR\_CAPTURE\_COLOR.

Application will receive notification “EVENT\_SENSOR\_CAPTURE\_COLOR” with Dictionary included color info. The following keys can present in dictionary (HMNSensorControl.h):

```
KEY_LED_COLOR_R;
```

```
KEY_LED_COLOR_G;
```

```
KEY_LED_COLOR_B;
```

## Notes

The device can accept one command at second. If application required to send several commands it should send them with delay for 1 second for each command.

### 5.14.3 Sample Usage

#### Device Connection Example

```
(void) viewDidLoad {
    [super viewDidLoad];
    [self subscribeForNotification];
    [HMNDeviceGeneral connectToMasterDevice];
}

(void) subscribeForNotification {
    [[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(deviceConnected)
    name:EVENT_DEVICE_CONNECTED
    object:nil];
}
```

### Color Sensor handler example

```
(void) sensorCaptureColorReceived:(NSNotification *)notification {
    NSDictionary *devInfoDict = [notification userInfo];
    NSInteger R = [devInfoDict[KEY_LED_COLOR_R] integerValue];
    NSInteger G = [devInfoDict[KEY_LED_COLOR_G] integerValue];
    NSInteger B = [devInfoDict[KEY_LED_COLOR_B] integerValue];
    NSLog(@"RGB color: %ld %ld %ld ", (long)R, (long)G, (long)B);
}
```

### Device pattern handler example

```
(void) patternReceived:(NSNotification *)notification {
    NSDictionary *devInfoDict = [notification userInfo];
    HMNPattern pattern = [devInfoDict[KEY_LED_PATTERN_ID] integerValue];
    NSLog(@"Pattern ID = %ld ", (long)pattern);
}
```

### Version handler example

```
(void) versionReceived:(NSNotification *)notification {
    NSDictionary *devInfoDict = [notification userInfo];
    NSString *swVersion = devInfoDict[KEY_SW_VERSION];
    NSString *hwVersion = devInfoDict[KEY_HW_VERSION];
    NSLog(@"swVersion %@, hwVersion %@", swVersion, hwVersion);
    return;
}
```

### Brightness handler example

```
(void) brightnessReceived:(NSNotification *)notification {
    NSDictionary *devInfoDict = [notification userInfo];
    NSUInteger brightness = [devInfoDict[KEY_LED_BRIGHTNESS] unsignedIntegerValue];
    NSLog(@"brightness %lu", (unsigned long)brightness);
    return;
}
```



### Sound handler example

```
(void) soundEventReceived:(NSNotification *)notification {
    NSDictionary *devInfoDict = [notification userInfo];
    NSUInteger microphoneLevel = [devInfoDict[KEY_MICROPHONE_LEVEL] integerValue];
    NSLog(@"microphone level %lu", (unsigned long)microphoneLevel);
    return;
}
```

### Show Letter example

```
(void) showAChar { Byte charCode = 65; // A
    UIColor *charColor = [UIColor redColor];
    UIColor *backgroundColor =[UIColor blueColor];
    [HMNLedControl setLedChar:charCode charColor:charColor backgroundColor:backgroundColor apply-
    ToSlaveDevice:NO];
}
```

### Show Hourglass example

```
(void) showHourglassPattern {
    Byte imageMatrix[] = {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0
    };
    NSMutableArray *array = [NSMutableArray arrayWithCapacity:LED_COUNT];
    for (int i=0; i<LED_COUNT; i++) {
        [array addObject:[NSNumber numberWithInt:imageMatrix[i]]];
    }
    [HMNLedControl setLedPattern:HMNPattern_FireFly withData:array];
}
```

### Show color image example

```
(void)setDigit1ColorImage {
    Byte imageMatrix[] = {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0
    };
    NSMutableArray *colors = [NSMutableArray arrayWithCapacity:LED_COUNT];
    UIColor *redColor = [UIColor redColor];
    UIColor *blueColor = [UIColor blueColor];
    for (int i=0; i<LED_COUNT; i++) { UIColor *color = (imageMatrix[i] == 0)? redColor:
        blueColor;
        [colors addObject:color];
    }
    [HMNLedControl setColorImage:colors];
}
```

---

## 5.15 Demos and 3rd party integrations

### 5.15.1 Demos and 3rd Party Integrations

#### Cordova Integration

**HKAudio - A Harman Kardon Cordova Plugin** HKAudio - A Harman Kardon Cordova plugin for Harman Kardon SDK

#### SmartThings Integration

**Harman Developer SmartThings Integration** You will find information about how to hook up your HKWirelessHD SDK into SmartThings, and the basics of creating SmartApps for HKWirelessHD devices on the SmartThings platform.

**Harman IoT Demo** Demonstrates how Omni speakers can play an important role in IoT environment, integrated with other 3rd party services like SmartThings, ITFFF, and so on.

---

**Search**

---

- search