HKWirelessHD API Specification

Release 1.0

Harman International

June 22, 2016

Contents

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.

CHAPTER 1

Overview

This documentation is organized into a few different sections as follows:

Introduction This section explains about the HKWirelessHD and its REST API Specification.

Architecture Overview This section explains about the overall context diagram, use cases and architecture of HK-WirelessHD APIs. It explains how your apps or services can connect Omni speakers using HKWirelessHD REST APIs.

REST API Specification This section describe on REST APIs.

OAuth2 API Specification This section describe on OAuth2 REST APIs.

Getting start with HKWHub App This section explains how to use HKWHub App (iOS) to simulate Hub Speaker to control other HK Omni speakers.

Playback Session Management This section explains about Playback Session Management.

CHAPTER 2

Contents

2.1 Introduction

2.1.1 Introduction

The Harman Kardon WirelessHD (or HKWirelessHD) SDK is provided for 3rd party developers to connect Harman/Kardon Omni and JBL Link Series audio devices. The intent of this SDK is to provide the APIs, tools and libraries needed for developers to build, test and deploy their wireless audio applications on your platform.

Currently, three kinds of APIs are provided by Harman Developer Program team.

- iOS APIs and SDK
- Android APIs and SDK
- Web (REST) APIs

This document describes only about the REST APIs, and for iOS and Android SDK, you can refer to Harman developer website.

2.2 HKWirelessHD Architecture Overview

2.2.1 HKWirelessHD Overview

Connecting Omni speakers to your services or devices

HKWirelessHD API is designed for Harman Kardon and JBL wireless speakers to be connected with third-party services or devices in standard base web protocols. This section describes the overall use cases and architecture of HKWirelessHD API.

The following figures show the brief diagram of context diagram explaining how the Harman Kardon and JBL speakers can be connected with other devices or services.

Users (End Customer and Developer)

There are two types of users as below, and HKWirelessHD API and SDK is intended to support developers to create Harman Kardon and JBL compatible applications or services for end user.

- **End user**

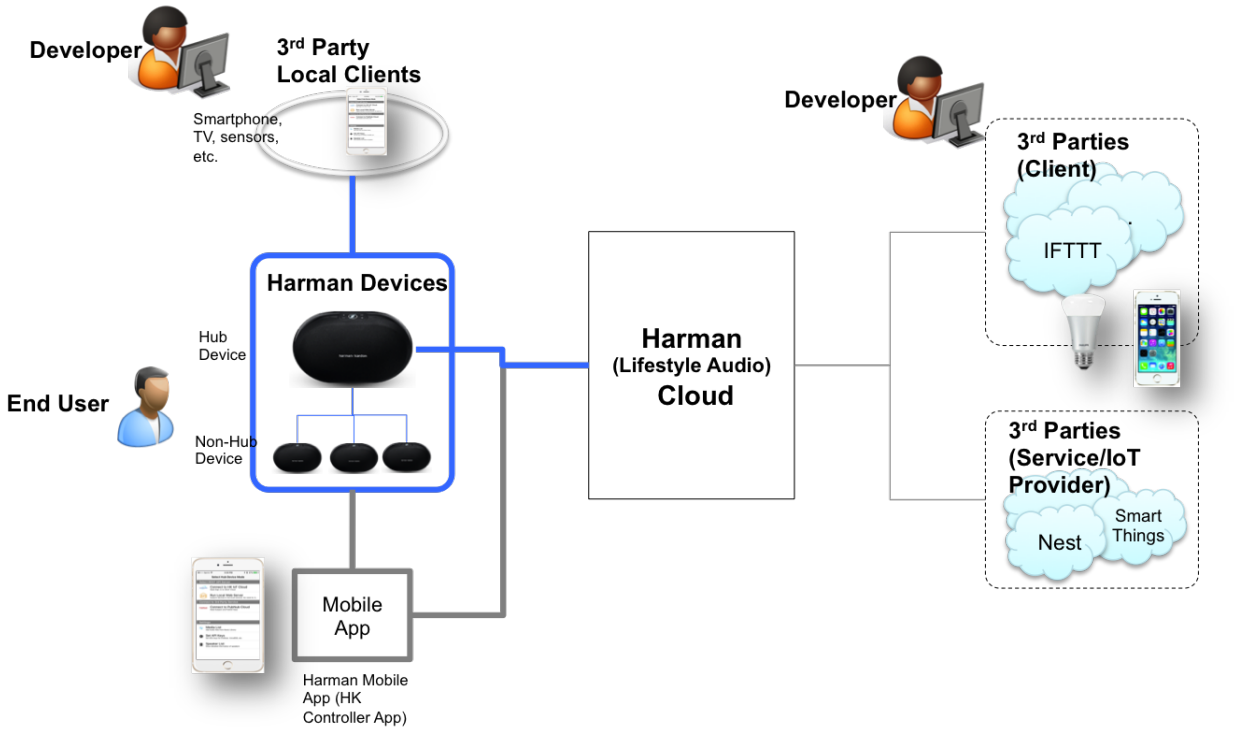


Fig. 2.1: Context Diagram

  - Install Harman Kardon/JBL speakers and use to play sound
  - Use Harman speakers to work with 3rd party devices or services to perform IoT functions
  - Optionally, user can use voice commands to control 3rd party devices or to do voice search (this feature is not supported by the API yet.)

- **Developer**
  - **3rd party developers create Harman Kardon/JBL applications or services for end user based on Harman APIs c**
    - e.g. Harman Cloud REST APIs
    - e.g. REST APIs provided by Harman device directly

Harman Devices

Harman Kardon Omni and JBL Link wireless speakers. All Harman speakers are enabled with both common functions (described below) and Hub-specific functions. User is supposed to choose one speaker as Hub speakers. The Hub speaker will connect to the Harman Cloud, and also run local web server to process REST requests from third party devices in the local network.

(a) Common functions Every Harman devices support the following functions:

- Play WirelessHD multi-room audio
- Support 3rd party streaming services
- Voice command (optional)

(b) Hub-specific function Any Harman device will be enabled with Hub-specific functions, but only one Harman device should be selected as Hub device. (In this document, we call the Hub function-enabled devices as "Hub Device". And the device with no hub-function enabled as "Non-Hub Device".)

Hub device needs to perform the following additional functions:

- Communicate with the Harman Cloud to handle multi-room audio requests.
- Handle requests from local devices or services

Mobile App

The mobile app is used for initial setup and configuration. In some cases, mobile app can be used to provide user interface for 3rd party authorization, and so on.

It communicates with Harman Cloud to get and set the configuration information from/to the cloud.

Harman Cloud Servers (HKIoTCloud)

Harman Cloud, or HKIoTCloud, communicates with Harman users (via mobile app or web page), Harman devices, 3rd party clients (services or devices), 3rd party providers. Harman Cloud allows the users or services in the Internet (outside of local network) to access HK/JBL speakers located in the local home network.

3rd Party client service or device

3rd party client service or device connect to Harman Cloud and send REST API request to control Harman devices.

3rd Party service providers

Harman Cloud can connect to 3rd party service providers to control 3rd party devices or to receive services on behalf of end customer. In most case, user will use voice commands or button presses to initiate a command to control 3rd party devices via 3rd party service provider.

For example, user will say a voice command to speaker (Harman device), then the speaker send the voice to or get it interpreted by 3rd party voice services. The result will be sent to Harman Cloud. Then, Harman Cloud connects Nest server or SmartThings servers to control user's Nest or SmartThings devices through the 3rd party servers.

3rd Party Local clients

3rd party client devices or application which is connected in the local network can also send requests to Harman device to control audio playback or other device control using voice command.

Use Cases Illustration

Hub Speaker

To enable your Omni speakers to be controlled by other services or devices either in the Internet or in local network, one of your speakers should become a Hub Speaker. A Hub speaker is just a regular HK Omni speaker, but is configured as a Hub. A Hub device can control other HK Omni speakers when it receives a REST requests from other clients. Hub speaker needs to be connected all the time to the Harman Cloud (or HKIoTCloud) which actually receives HTTP

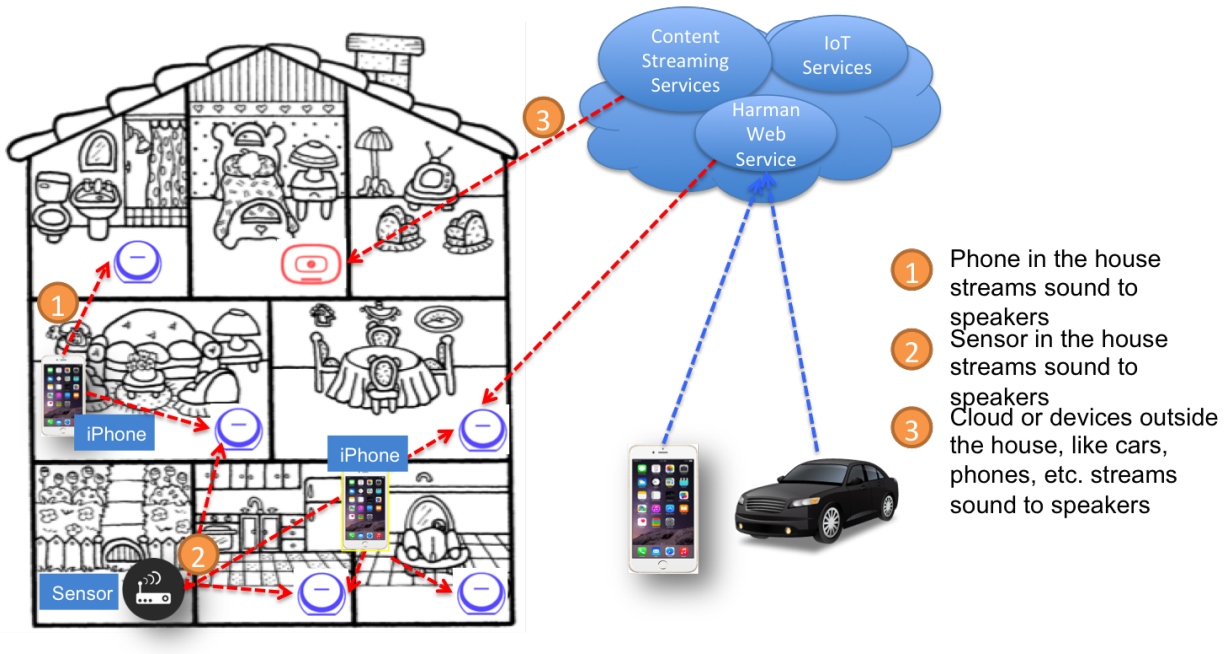


Fig. 2.2: Use Case: Speakers as Internet of Things

request from client in the Internet. Harman Cloud forwards the command to the Hub speaker to control other speakers as well as Hub speaker.

In addition, to handle HTTP requests from devices in local network, not through HKIoTCloud, Hub Speaker runs a web server inside.

HKWHub App

HKWHub app is an iOS app that simulates Hub speaker with HKWirelessHD SDK. It acts like a Hub Speaker that handles REST requests to control speakers. Like Hub Speaker, HKWHub App is always connected to HKIoTCloud and also runs a web server inside that handles HTTP requests of REST API.

HKWHub App has following 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.

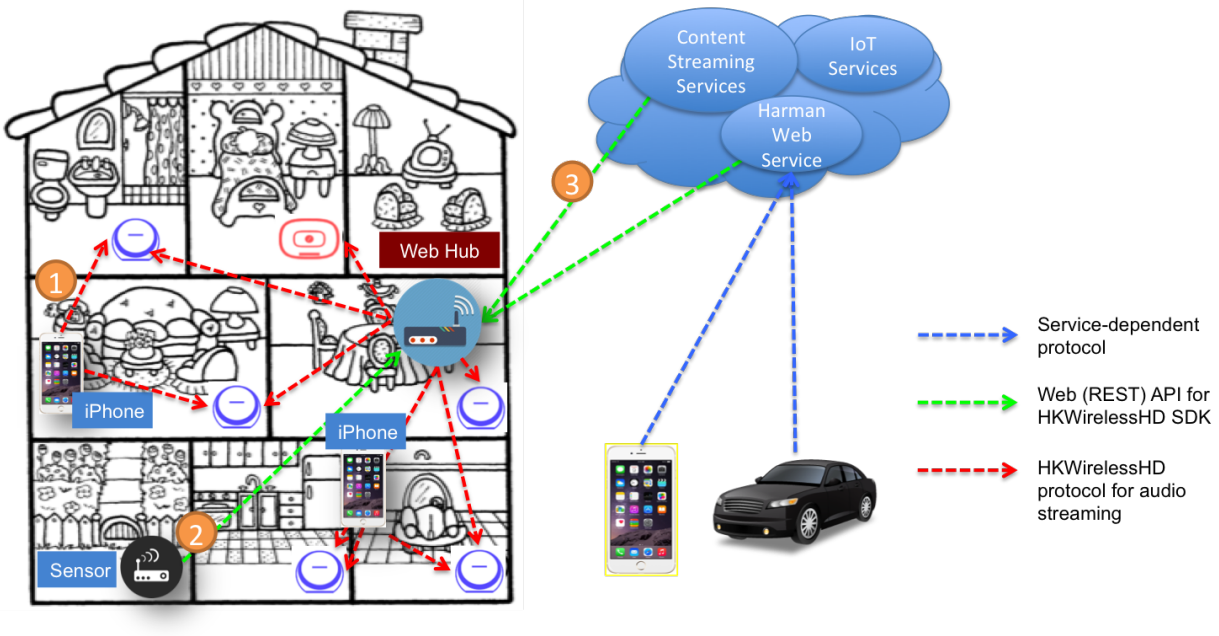


Fig. 2.3: Hub Speaker (Hub App) to connect IoT devices and services to speakers in your home

Overall Architecture Hub Device (or Hub App)

Hub Device or 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.

Hub Speaker (or HKWHub App) supports the following two modes:

- **Cloud mode (HKIoTCloud)**
  - Hub Speaker or 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**
  - Hub Speaker or HKWHub app runs a web server internally, and handles the REST requests for speaker control and playback from devices, sensors or applications in the same local network.
  - Hub Speaker or 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.

The following figure explains how HKWHub app handles the different modes.

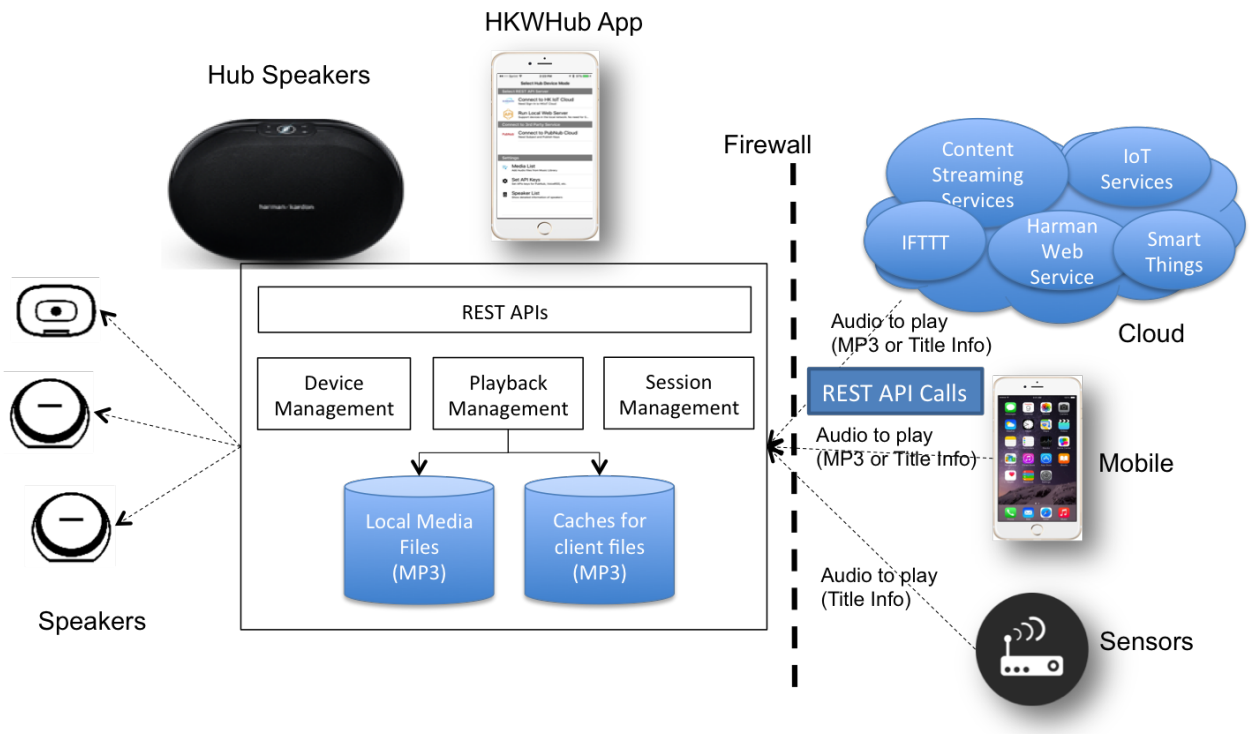


Fig. 2.4: Hub Speaker Architecture

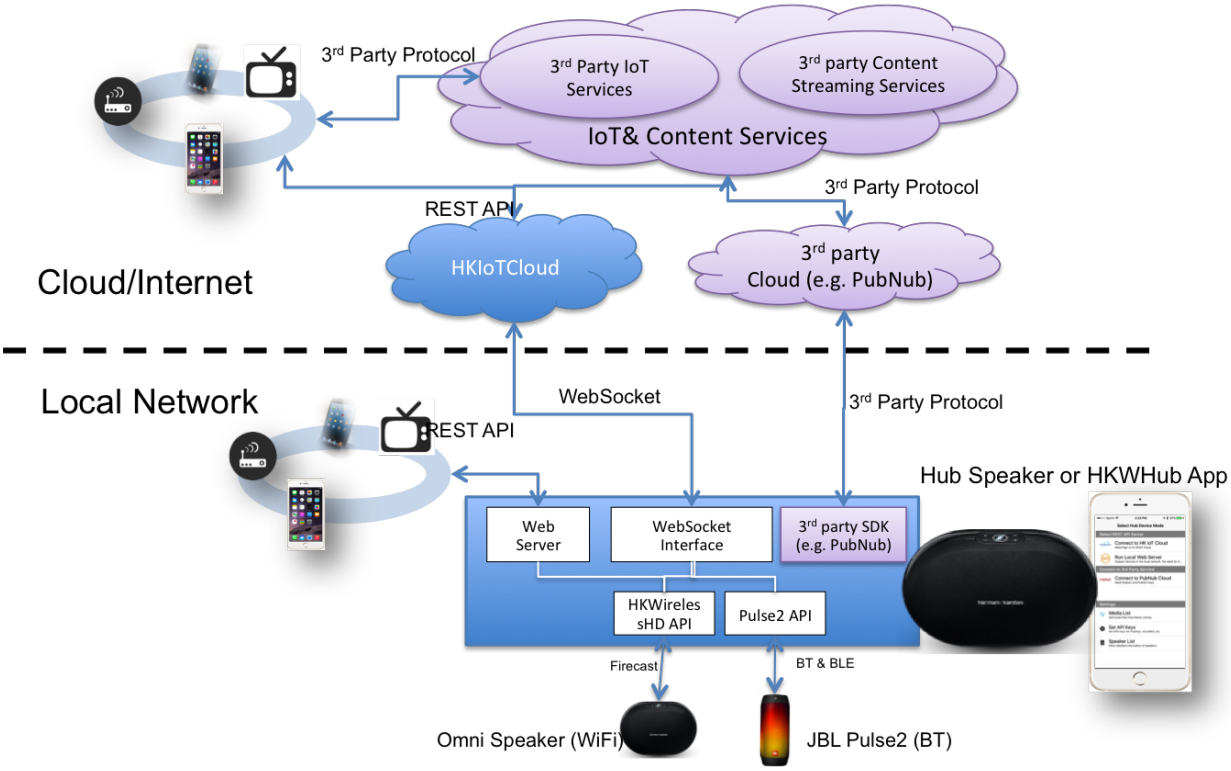


Fig. 2.5: Integrating with third party devices and services

HKWirelessHD API Architecture

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 a mobile device like iPhone or Android phone, but Harman Kardon or JBL speakerx can stream an audio to other speakers. Destination devices are Harman Kardon Omni speakers (Omni Adapt, Omni 10, Omni 20, Omni Bar, etc.) and JBL Link speakers. You can find more information on HK Omni speakers at http://www.harmankardon.com/content?ContentID=omni-v2.

As mentioned earlier, a speaker configured as Hub speaker can stream audio to other speakers when it receives commands from HKIoTCloud or devices in the local network.

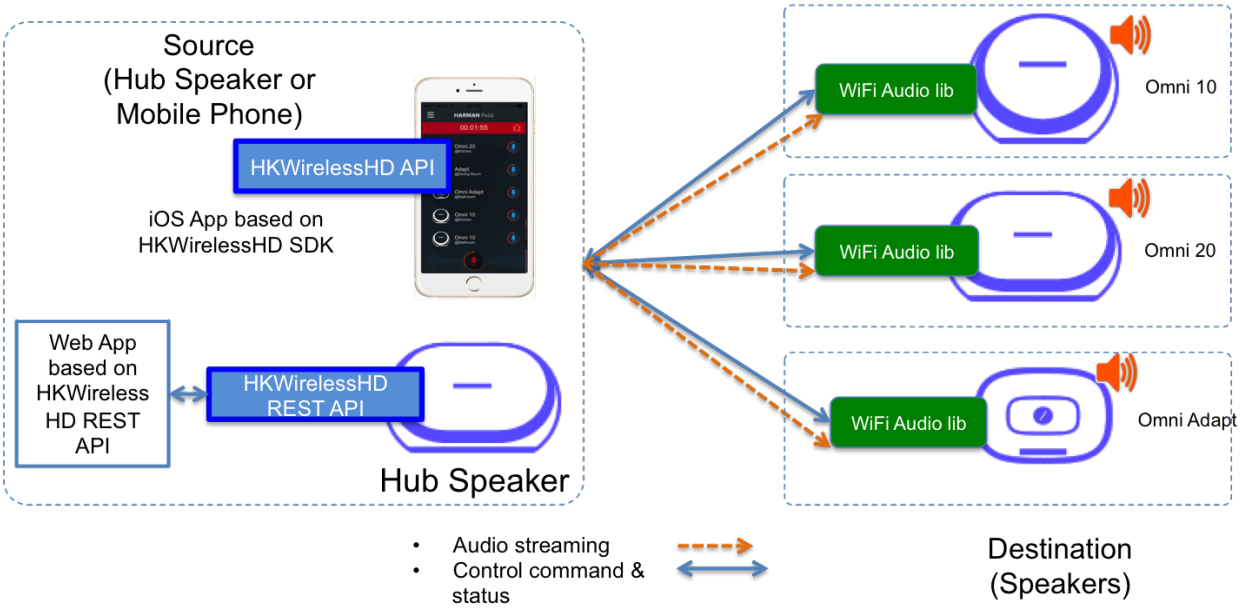


Fig. 2.6: Hub Speaker controlling other speakers

Use of HKWirelessHD API to stream audio to Omni Speakers

To send audio stream to destination devices, a client (either Mobile app or Web app) should use HKWirelessHD API. HKWirelessHD SDK provides APIs for Web app, iOS and Android App, and this documentation most describes about REST APIs for Web apps.

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 speaker to the current running playback session by long-pressing the Home button on the control panel. Please refer to Omni 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 sour**
    - 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 efficient 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 a latency 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 for speakers 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 APIs for source-side manages the latest device status information.

Visibility of Speakers

Any speakers in a network are visible to source devices (mobile devices or Hub speaker) if a source device is successfully initialized 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.

For example, as described in the figure below, let's assume that Speaker-A and Speaker-B are being used by Source A (Hub Speaker), and Speaker-D and Speaker-E are being used by Source B(Mobile App). Once Source A and Source B are successfully initialized, 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() to return a boolean value indicating if the speaker is being playing audio or not, regardless that which source audio stream comes from.

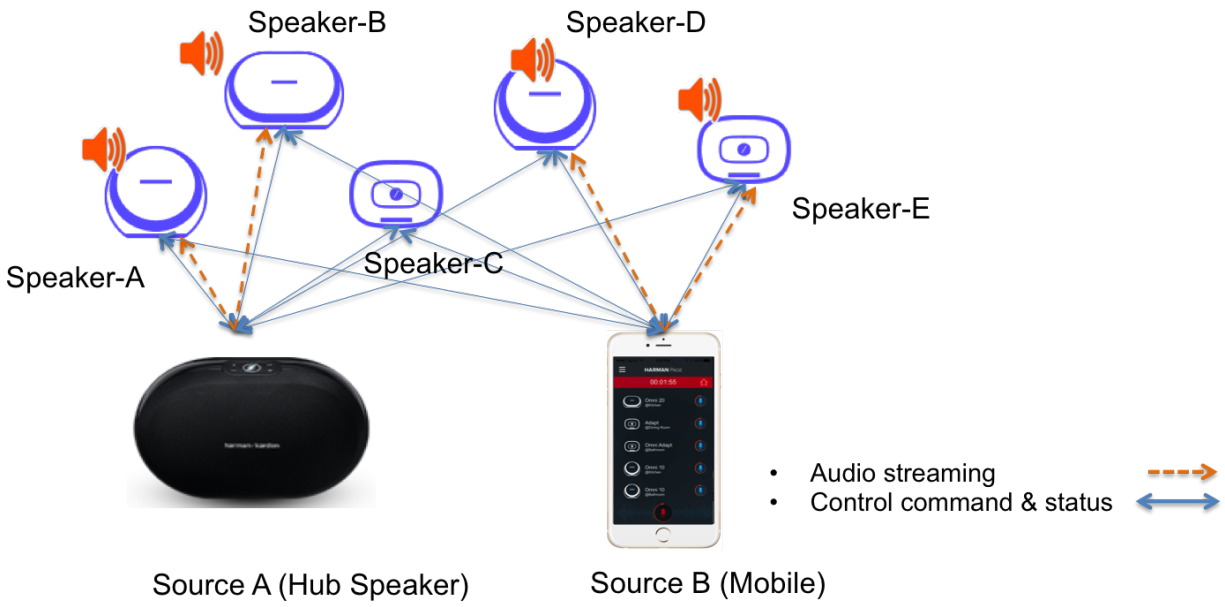


Fig. 2.7: Visibility of speakers

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 a corresponding APIs provided by the specification.

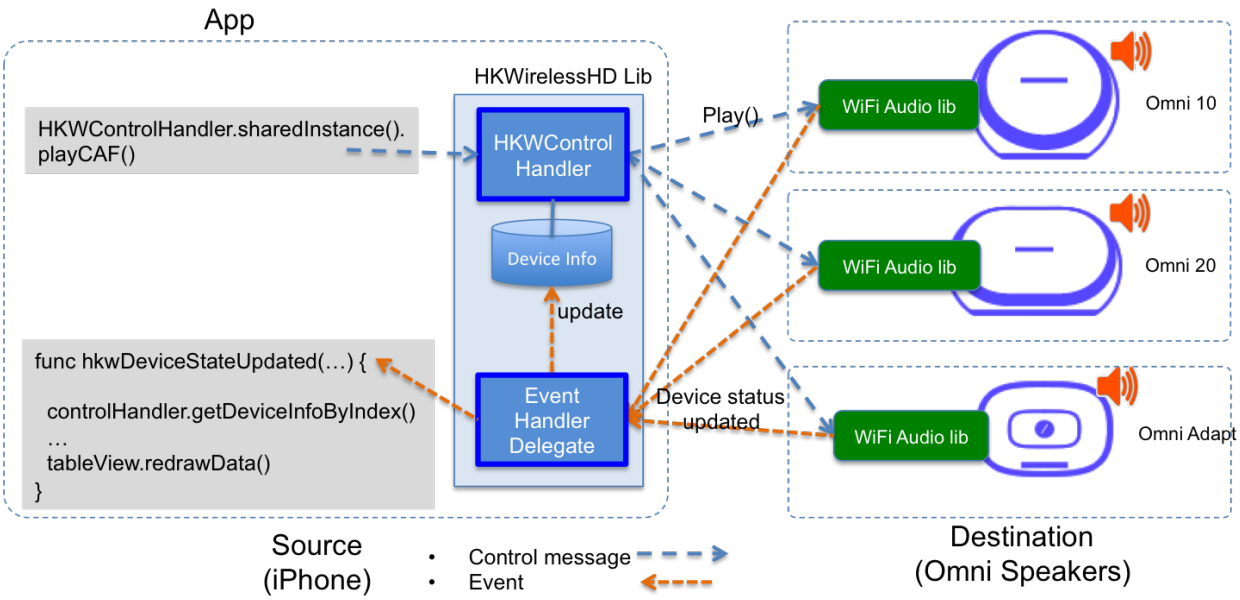


Fig. 2.8: Control Handler of Hub Speaker

Handling events from speakers

On the other hand, receiving an event from speakers is different. Because REST API is basically one-directional communication initiated by a client, it is hard for speaker as a server to report an event to a client when necessary. So, the client of Web app need to call corresponding APIs for checking events regularly, in a way of polling.

2.3 REST API Specification

2.3.1 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 Hub Speaker or HKWHub app.

All the APIS are in REST API protocol.

Note: In this documentation, for HKIoTCloud mode, <server_host> should be "hkiotcloud.azurewebsites.net". For Local server mode, <server_host> should be the URL (IP address and port number) that Hub speaker or HKWHub app is showing.

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>/api/v1/init_session
```

  - Response:

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

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"}
```

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"}
```

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"
                }]
        }
```

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=129321920968880
```

  - 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"}
```

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&DeviceID=129321920968880
```

  - Response:

```
{"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:abciKaTbUgdpQFuvYtgMm0FRh&DeviceID=129321920968880
```

  - Response:

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

Set party mode

Addes 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"}
```

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]"}
        ]}
```

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"}
```

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&PersistentID=7387446959931482519
```

  - Response:

```
{"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:abciKaTbUgdpQFuvYtgMm0F&PersistentID=7387446959931482519&
                    DeviceIDList=34317244381360,129321920968880
```

  - Response:

```
{"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&MediaUrl=http://seonman.github.io/music/hyolyn.mp3
```

  - Response:

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

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

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&MediaUrl=http://seonman.github.io/music/hyolyn.mp3
```

- Response:

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

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

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:abciKaTbUgdpQFuvYtgMm0F&MediaUrl=http://seonman.github.io/music/hyolyn.mp3&
                DeviceIDList=34317244381360,129321920968880
```

  - Response:

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

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

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
```

  - Response:

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

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

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.

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&Text="Hello World. How are you today?"&
          DeviceIDList=34317244381360,129321920968880
```

  - Response:

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

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

Pause the Current Playback

Pauses the current playback. The client can resue 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"}
```

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&PersistentID=7387446959931482519
```

  - Response:

```
{"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:abciKaTbUgdpQFuvYtgMm0F&PersistentID=7387446959931482519
```

  - Response:

```
{"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:abciKaTbUgdpQFuvYtgMm0F&PersistentID=7387446959931482519&
                DeviceIDList=34317244381360,129321920968880
```

  - Response:

```
{"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"}
```

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 recommedn to call this API every second.
- **Example:**
  - Request:

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

  - Response:

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

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"}
```

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"}
```

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"}
```

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&Volume=10
```

  - Response:

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

2.4 OAuth2 API Specification

2.4.1 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 grans 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 specifi instance of your product.
7. The user's speific instance of your product uses the access token to make HKIoTCloud API calls.

Types of Authorization

HKIoTCloud supports two types of authorization:

- Authorization Code Grant - Send a client ID and a client secret to get an access token and a refresh token.
- Password Grant - Send username and password along with client ID and client secret to get an access token and refresh token

Note: If you are able to implement server-side scripting, then using authorization code grant is recommended. If you are not able to implement server-side scripting, then using password grant is your choice.

Note: You must generate a new access token every hour, that is, expiration is set to 3,600 seconds. You can use refresh token in conjunction with your client ID and client secret to obtain a new access token without your user having to re-authenticate.

Using the Password Grant Type

To obtain an access token (and a refresh token) with password grant, you should **POST** to `/oauth/token`. You should include your client ID and client secret in the `Authorization` header by combining them with a colon ":" and then encoding in Base64. That is, `Base64(client_id:client_secret)`. And also, you should include `grant_type: password`, username and passworkd in the request body.

Sample Request:

```
POST /oauth/token HTTP/1.1
Host: hkiotcloud.azurewebsites.net
Authorization: Basic RkZjUE9iS2h4OThvNXhtMzpjcENZQ1BrUjA4NXFSR3hFempDMUlGeEoxQWRhZFQ=
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w
```

Here, `RkZjUE9iS2h4OThvNXhtMzpjcENZQ1BrUjA4NXFSR3hFempDMUlGeEoxQWRhZFQ=` is the result of Base64 encoding of clientId:clientSecret.

Sample Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
   "access_token":"62b8c11cfa0840b506230cb8af747230052775e1",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"7a7687b6b32247573b366d5bf2eeb707ba0a1b4d"
 }
```

Creating a Consent Request

By creating a consent request, your user will be redirected to the HKIoTCloud website where they can enter their HKIoTCloud credentials in order to authorize their devices of your product to be used with the HKIoTCloud service.

The consent request is constructed as follows:

- **Redirect the user to HKIoTCloud at https://hkiotcloud.azurewebsites.net/oauth/token with the following URL-encoded qu**
  - `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.azurewebsites.net/oauth/authorize?response_type=code&
          client_id=n7HhiTnKYjJd4zmM&redirect_uri=https://your.app.com/oauthCallbackHKIoTCloud
```

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 Authorizatino 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.azurewebsites.net/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.azurewebsites.net
Content-Type: application/x-www-form-urlencoded
Cache-Control: no-cache

grant_type=authorization_code&code=2b3711911f4f2263e785eeda386046ccc8da6aee&
    client_id=n7HhiTnKYjJd4zmM&client_secret=ANRfB9z94xtcxFGXrd5XHXEiKg43UY
        &redirect_uri=https://hkvoicecloud.azurewebsites.net/oauthCallbackHKIoTCloud
```

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.azurewebsites.net/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.azurewebsites.net/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.herukuapp.com
Content-Type: application/x-www-form-urlencoded
Cache-Control: no-cache

grant_type=refresh_token&refresh_token=5defcb0a9a49ac9b2403b8c78600638238d81011&
client_id=n7HhiTnKYjJd4zmM&client_secret=ANRfB9z94xtcxFGXrd5XHXEiKg43UY
```

Sample Response:

```
HTTP/1.1 200 OK

{
        "access_token": "90da03bdceb15cf75d99ff99715ce87b29602651",
        "expires_in": 3600,
        "refresh_token": "6a762dfce9146dbf149f881c5aa15fc6cfdf1fd0",
        "token_type": "bearer"
}
```

2.5 Getting start with HKWHub App

2.5.1 Quick Getting Started Guide to HKWHub App

Quick Getting Started Guide to HKWHub App

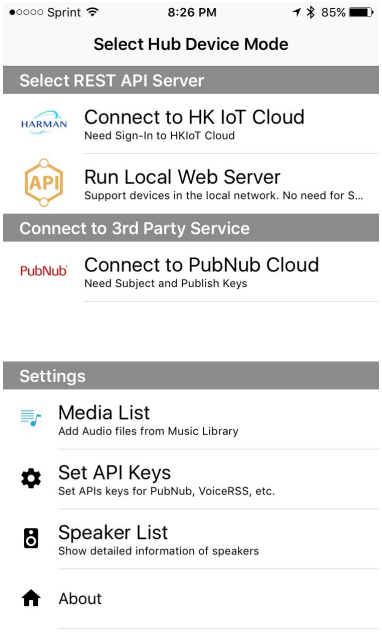
Overview of HKWHubApp

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** (amonng 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.

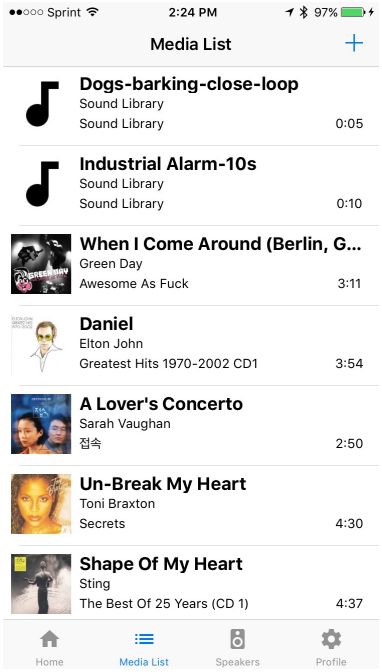
The **Settings** menu has four sub menus:

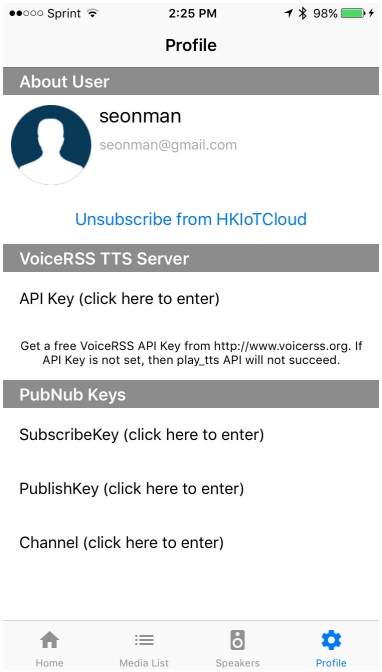
- **Media List**
  - User can maintain the list of audio files for audio playback.
  - User can add audio from iOS Media Library.

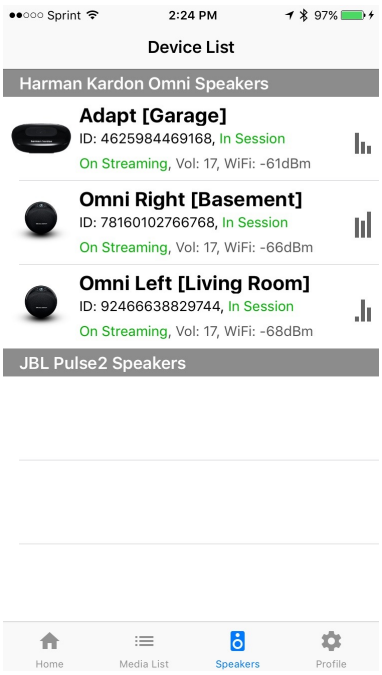
> **Note:** Note that only the media file available offline and not from Apple Musica can be added. The music file that came from Apple Music cannot be added by DRM issue.

- **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.


Sprint
2:24 PM
97%
Media List
Dogs-barking-close-loop
Sound Library
Sound Library
0:05
Industrial Alarm-10s
Sound Library
Sound Library
0:10
When I Come Around (Berlin, G...
Green Day
Awesome As Fuck
3:11
Daniel
Elton John
Greatest Hits 1970-2002 CD1
3:54
A Lover's Concerto
Sarah Vaughan
접속
2:50
Un-Break My Heart
Toni Braxton
Secrets
4:30
Shape Of My Heart
Sting
The Best Of 25 Years (CD 1)
4:37
Home
Media List
Speakers
Profile


Sprint
2:25 PM
98%
Profile
About User
seonman
seonman@gmail.com
Unsubscribe from HKIoTCloud
VoiceRSS TTS Server
API Key (click here to enter)
Get a free VoiceRSS API Key from http://www.voicerss.org. If API Key is not set, then play_tts API will not succeed.
PubNub Keys
SubscribeKey (click here to enter)
PublishKey (click here to enter)
Channel (click here to enter)
Home
Media List
Speakers
Profile



HKIoTCloud Mode

Connecting to HKIoTCloud In HKIoTCloud demo, 3rd party clients can connect to HKIoTCloud (http://hkiotcloud.azurewebsites.net) and send REST requests to control speakers and play audio. In order to use HKIoTCloud mode, user needs to sign up to the cloud with username, emaill 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.

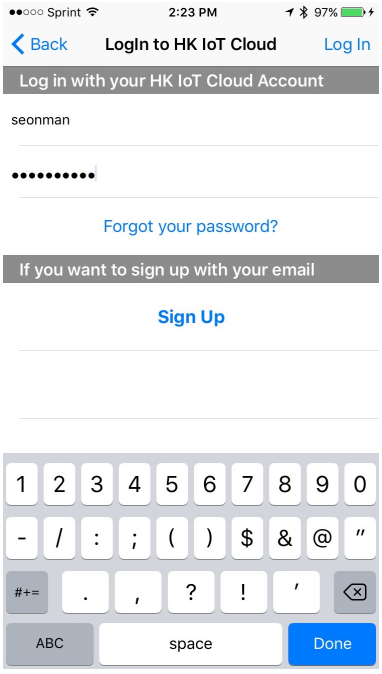
Note: For a client to connect to HKIoTCloud, the same username and password are required.

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

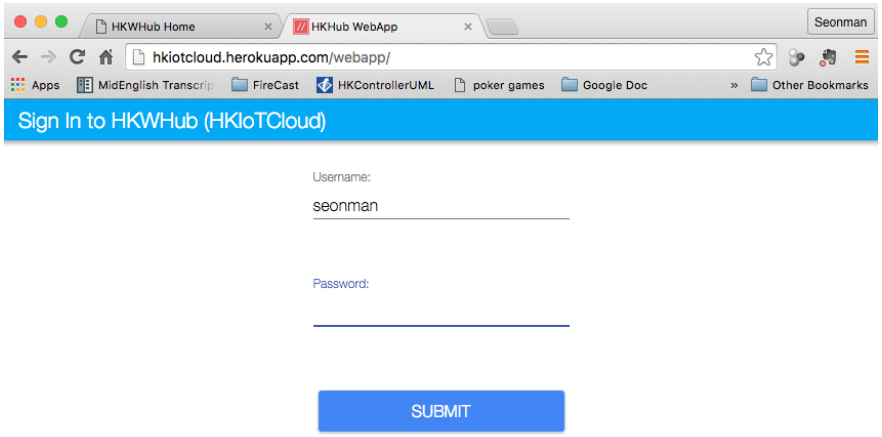
Once user authentication is done successfully, the Web app will switch the screen to the Playlist screen.

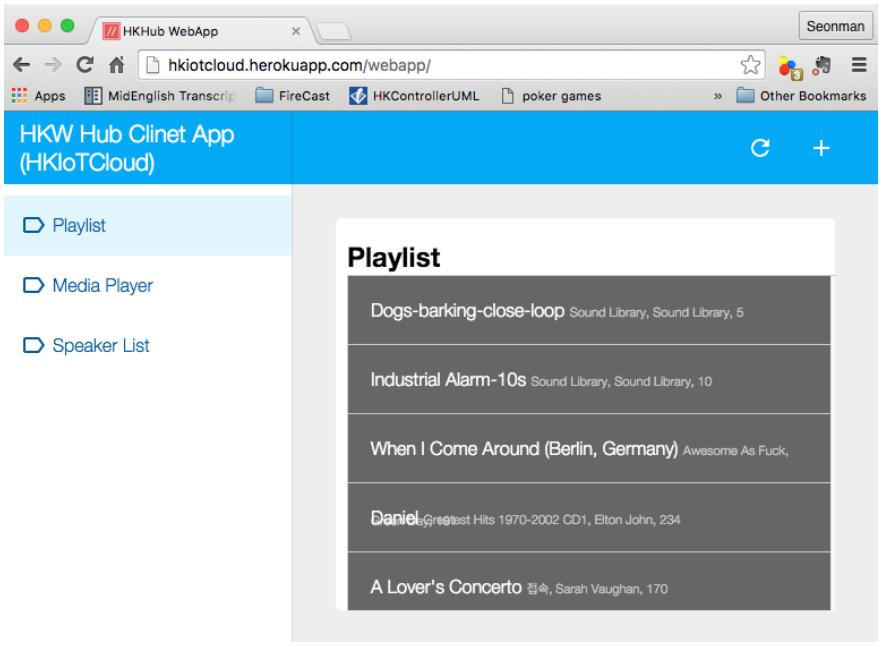
Now, you can click one of the titles in the list, and see how the web app is playing the title, showing the information of the title, volume, and playback time, and so on.

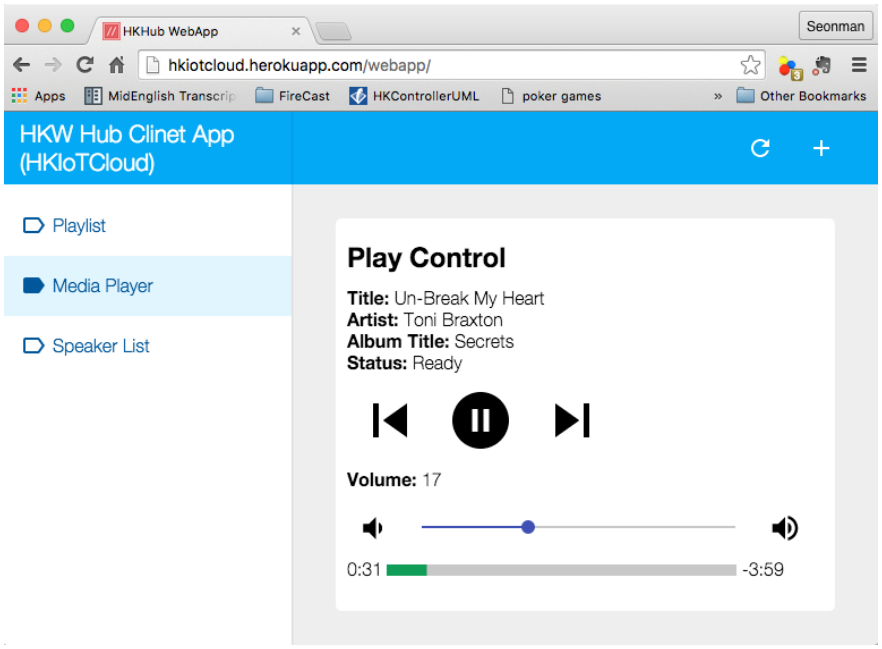
If you click **Speaker List** menu on the left, you can see more detailed information of speakers like below, and can control speakers, like remove a speaker from the current playback session or add a speaker to playback.

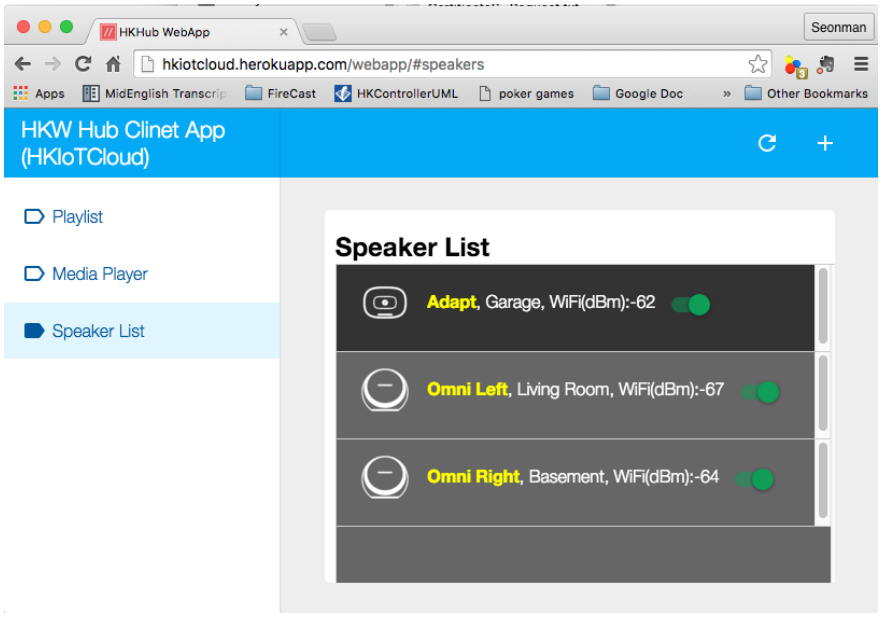

Sprint 2:23 PM 97%
Back
LogIn to HK IoT Cloud
Log In
Log in with your HK IoT Cloud Account
seonman
Forgot your password?
If you want to sign up with your email
Sign Up
ABC
space
Done


Sprint 9:32 PM 100%
HK Cloud Connection
Disconnect
Cloud: https://hkiotcloud.herokuapp.com
Status: Connected (Token verified)
Speakers: HK Omni (3), JBL Pulse2 (0)
[9:32:07] Status: Connected to the server
[9:32:07] < hello
[9:32:07] > hello & received ack
[9:32:07] Status: Connected (Token verified)
Home
Media List
Speakers
Profile


HKWHub Home
HKHub WebApp
Seonman
hkiotcloud.herokuapp.com/webapp/
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Google Doc
Other Bookmarks
Sign In to HKWHub (HKIoTCloud)
Username:
seonman
Password:
SUBMIT


HKHub WebApp
Seonman
hkiotcloud.herokuapp.com/webapp/
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Other Bookmarks
HKW Hub Clinet App
(HKIoTCloud)
Playlist
Media Player
Speaker List
Playlist
Dogs-barking-close-loop Sound Library, Sound Library, 5
Industrial Alarm-10s Sound Library, Sound Library, 10
When I Come Around (Berlin, Germany) Awesome As Fuck,
Daniel Greatest Hits 1970-2002 CD1, Elton John, 234
A Lover's Concerto 정숙, Sarah Vaughan, 170


HKHub WebApp
Seonman
hkiotcloud.herokuapp.com/webapp/
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Other Bookmarks
HKW Hub Clinet App
(HKIoTCloud)
Playlist
Media Player
Speaker List
Play Control
Title: Un-Break My Heart
Artist: Toni Braxton
Album Title: Secrets
Status: Ready
Volume: 17
0:31
-3:59

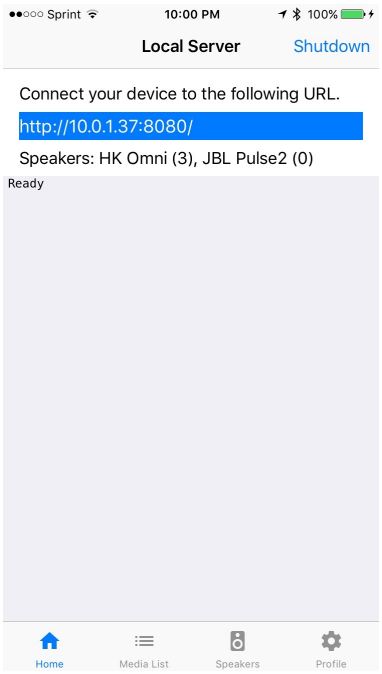

HKHub WebApp
Seonman
hkiotcloud.herokuapp.com/webapp/#speakers
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Google Doc
Other Bookmarks
HKW Hub Clinet App
(HKIoTCloud)
Playlist
Media Player
Speaker List
Speaker List
Adapt, Garage, WiFi(dBm):-62
Omni Left, Living Room, WiFi(dBm):-67
Omni Right, Basement, WiFi(dBm):-64

Local Server Mode

Running Local Server Loca 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 direclty 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.

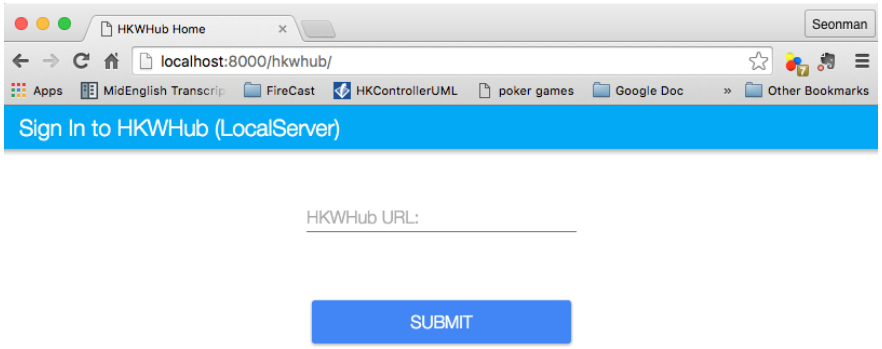
```
$ cd WebHubWebApp
$ python -m SimpleHTTPServer
```

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

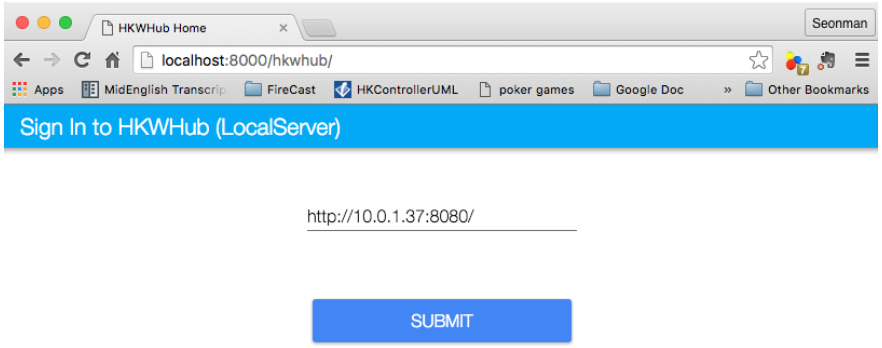
Next, launch your web browser (Chrome, Safari, ...) and go to http://localhost:8000/hkwhub/

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

At the fist 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.

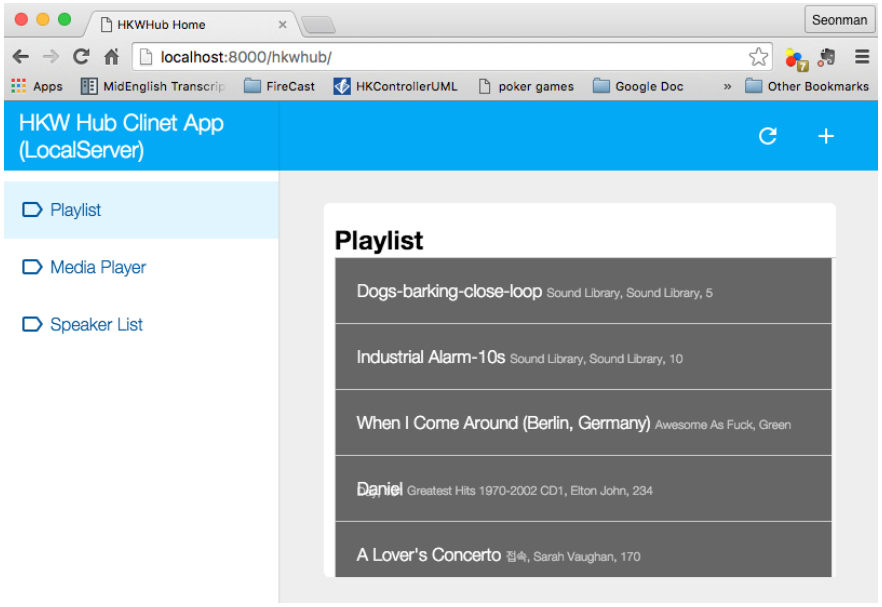
The UI of the Web app is exactly the same as HKIoTCloud web app. So, we skip to explain the rest parts of the app.

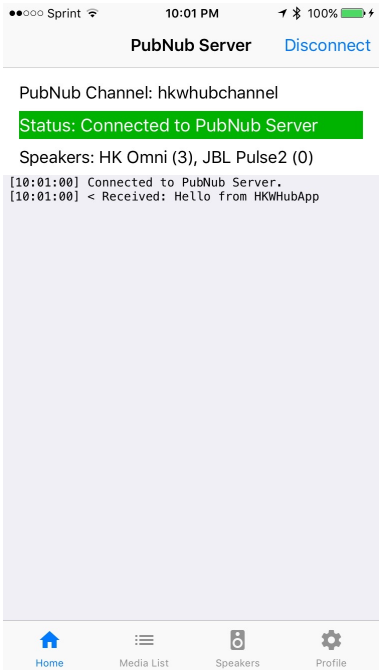
PubNub Server Mode

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 saysing 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.


HKWHub Home
Seonman
localhost:8000/hkwhub/
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Google Doc
Other Bookmarks
HKW Hub Clinet App
(LocalServer)
Playlist
Media Player
Speaker List
Playlist
Dogs-barking-close-loop Sound Library, Sound Library, 5
Industrial Alarm-10s Sound Library, Sound Library, 10
When I Come Around (Berlin, Germany) Awesome As Fuck, Green
Daniel Greatest Hits 1970-2002 CD1, Elton John, 234
A Lover's Concerto 접속, Sarah Vaughan, 170


Sprint
10:01 PM
100%
PubNub Server
Disconnect
PubNub Channel: hkwhubchannel
Status: Connected to PubNub Server
Speakers: HK Omni (3), JBL Pulse2 (0)
[10:01:00] Connected to PubNub Server.
[10:01:00] < Received: Hello from HKWHubApp
Home
Media List
Speakers
Profile

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 subsribers.

In this reason, the messages of request and response for speaker control are a littke 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 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.

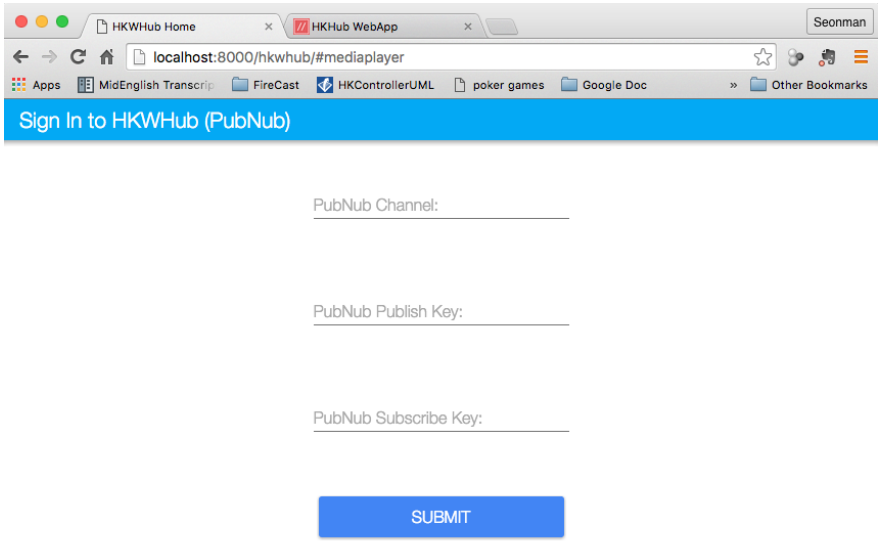
At the fist screen looking like below. Note that it looks different from the screen from Local Server mode, which requires only URL of the web server.

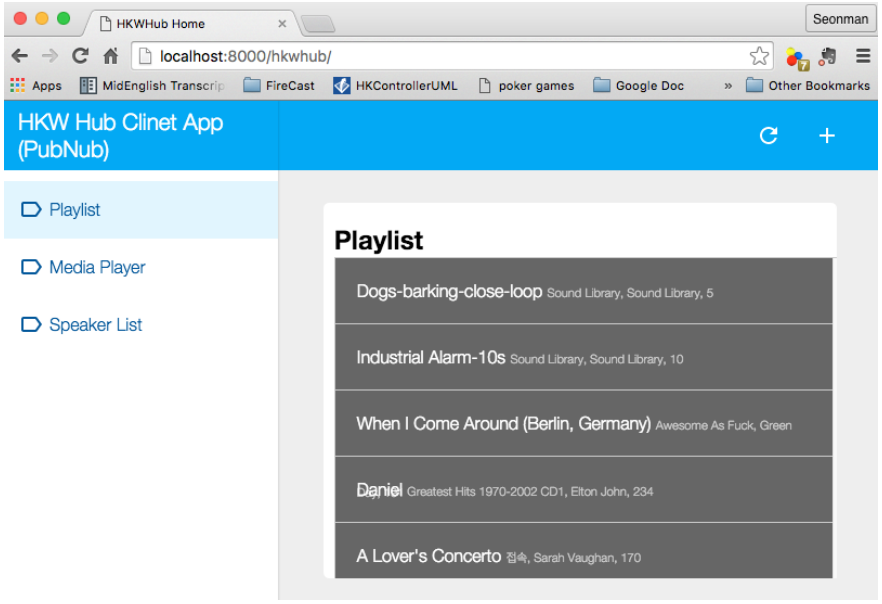
Enter the same PubNub publish key, subscribe key, and channel name that you used for HKWHub app, and click **Submit**, as below.

https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop/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.


HKWHub Home
HKHub WebApp
Seonman
localhost:8000/hkwhub/#mediaplayer
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Google Doc
Other Bookmarks
Sign In to HKWHub (PubNub)
PubNub Channel:
PubNub Publish Key:
PubNub Subscribe Key:
SUBMIT

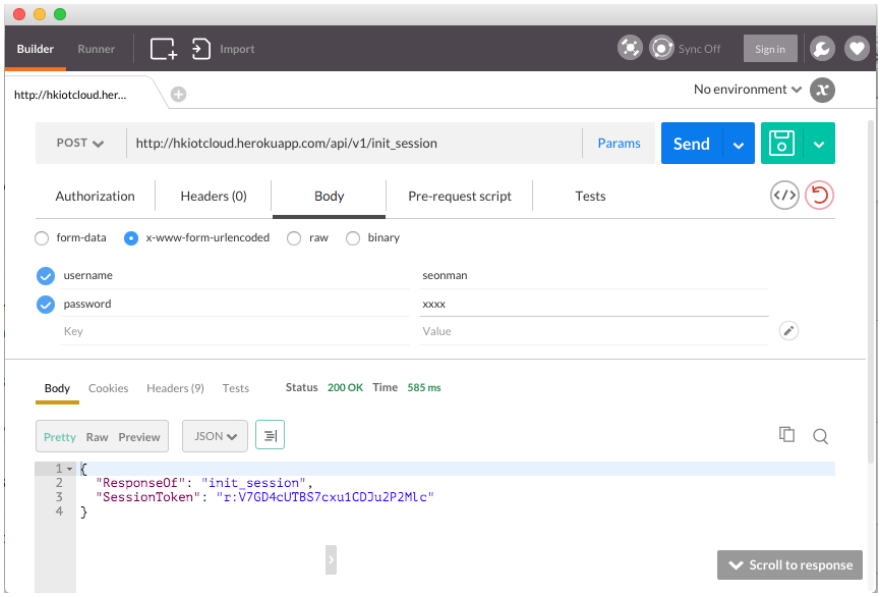

HKWHub Home
Seonman
localhost:8000/hkwhub/
Apps
MidEnglish Transcrip
FireCast
HKControllerUML
poker games
Google Doc
Other Bookmarks
HKW Hub Clinet App (PubNub)
Playlist
Media Player
Speaker List
Playlist
Dogs-barking-close-loop Sound Library, Sound Library, 5
Industrial Alarm-10s Sound Library, Sound Library, 10
When I Come Around (Berlin, Germany) Awesome As Fuck, Green
Daniel Greatest Hits 1970-2002 CD1, Elton John, 234
A Lover's Concerto 접속, Sarah Vaughan, 170

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/fhbjgbiflinjbdggehcddcbnc 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 more, 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 bjdIaGlUbktZakpkNHptTTpBTlJmQjl6OTR4dGN4RkdYcmQ1WEhYRWlLZzQzVVk=" -d "grant_type=password&username=yyy&password=xxx" http://hkiotcloud.azurewebsites.net/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.azurewebsites.net/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.azurewebsites.net/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.azurewebsites.net/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.azurewebsites.net/api/v1/add_device_to_session?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&DeviceI


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


e. Get the media list

- curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" "http://hkiotcloud.azurewebsites.net/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.azurewebsites.net/api/v1/play_hub_media?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&PersistentID=1


```
{"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.azurewebsites.net/api/v1/play_hub_media_selected_speakers?SessionToken=r:abciKaTbUgdpQFuvYtgMm0F

The list of speakers are listed by the parameter `DeviceIDList` with delimitor ",".

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

g. Play a HTTP streaming media as party mode

- curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" "http://hkiotcloud.azurewebsites.net/api/v1/play_web_media_party_mode?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh&M

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

h. Stop playing

- curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" "http://hkiotcloud.azurewebsites.net/api/v1/stop_play?SessionToken=r:abciKaTbUgdpQFuvYtgMm0FRh"

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

i. Set Volume

- curl -H "Authorization: Bearer 15c0507f3a550d7a31f7af5dc45e4dd9fd9f4bc8" "http://hkiotcloud.azurewebsites.net/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.

2.6 Playback Session Management

2.6.1 Playback Session Management

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

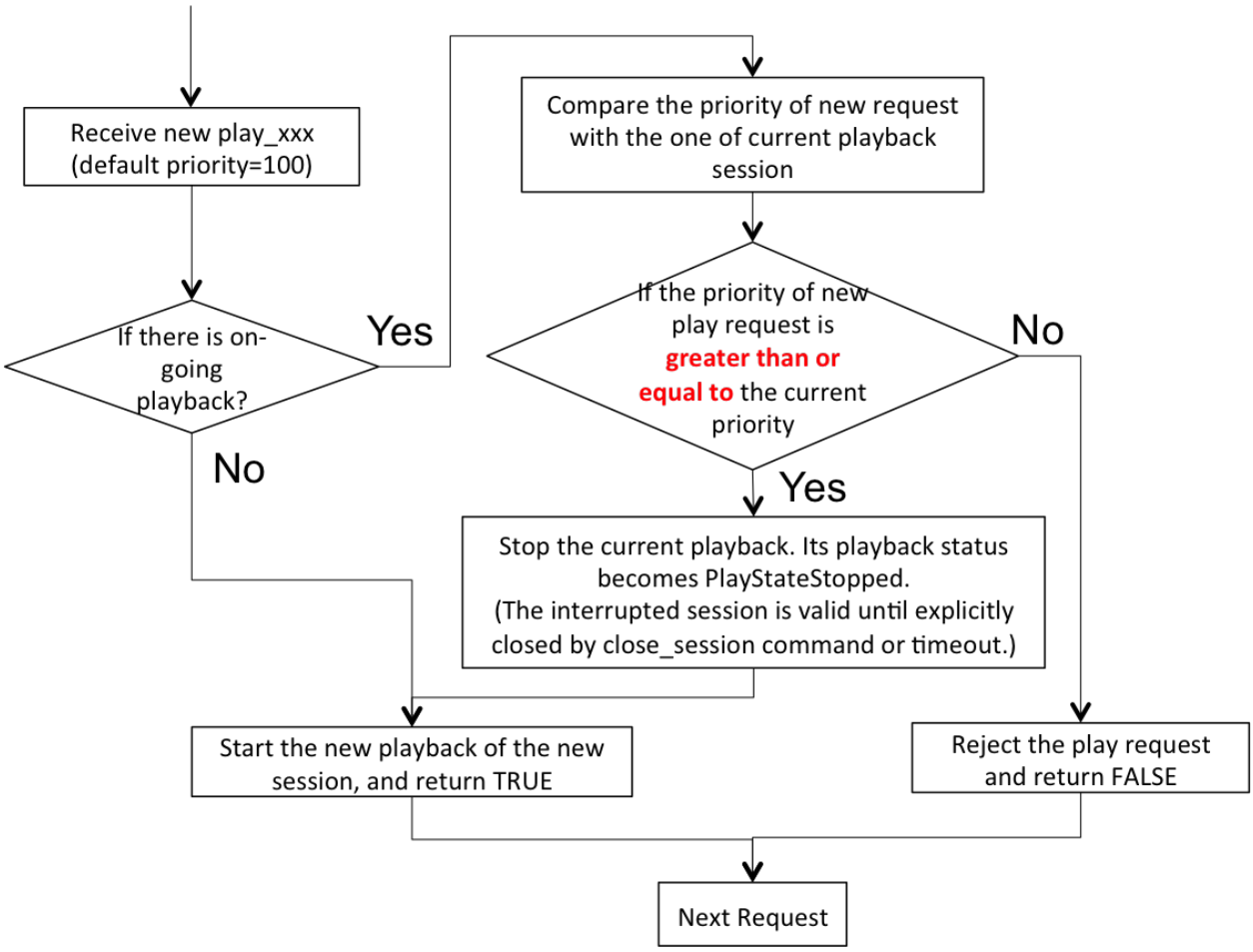
- When a client wants to start a playback, it sets the priority of the session (using `Priority=<priority value>` parameter).
- If Priority parameter is not specified, HKWHub app assumes it as default value, that is, 100.

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**
  - 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.

CHAPTER 3

Search

- search