Ameba IoT Demo Kit Application Note

Provide an example to setup a sensor/control application quickly. Sample code for both device code and Android app are provided. Sample code for both local control and remote control are provided.

Table of Contents

1 Introduction

This document introduces the implementation of Ameba WiGadget IoT Demo Kit feature. Smartphone and Ameba can transmit data to each other with Ameba WiGadget IoT Demo Kit feature. The Setup process is quite simple that users just need to connect the Ameba with Wi-Fi, then after pairing, the Smartphone can get the data from Ameba.

2 Associate Ameba WiGadget IoT Demo Kit to AP

The first time when power on, Ameba WiGadget IoT Demo Kit will wait for 20 seconds to connect Wi-Fi. After Wi-Fi connected, the SSID and password will be remembered by Ameba. So if the Wi-Fi disconnect in some cases, it will auto-reconnect using the same SSID and password.

3 Ameba WiGadget IoT Demo Kit Protocol

This sector will describe the protocol for communication between Ameba and smartphone APP.

There are two CONTROL TYPES for each device: just local control (CONTROL_TYPE = 0), multi-control: both local control and cloud control (CONTROL_TYPE = 1).

3.1 Architecture

There are three main functions provided by Ameba WiGadget IoT Demo Kit Feature.

The first is discovering. Discovering is based on the protocol of mDNS. Smartphone can find the device through the same AP. Smartphone with Ameba IoT demo interface also can get the text record sending by Ameba, and then the Smartphone can know the device's information: CONTROL_TYPE, PAIR_STATE, IP, PORT, MAC_ADDRESS.

The second is pairing. For the pairing, there will be two times handshakes (CONTROL_TYPE = 0) or three times handshakes (CONTROL_TYPE = 1) between smartphone and device which has Ameba WiGadget IoT Demo Kit interface.

The other is transmitting data. The device which CONTROL_TYPE = 0 will use TCP socket to transmit data with smartphone, and the device which CONTROL_TYPE = 1 will use both TCP socket and cloud thread to transmit data.

3.1.1 Discovery

3.1.1.1 Discovery

MDNS is a protocol that resolves host names to IP addresses or Service name to IP address with layer4 type with port in local network that do not include a local name server. It is a zero configuration service, using essentially the same programming interfaces, packet formats and operating semantics as the unicast Domain Name System (DNS). The mDNS protocol is published as RFC 6762, uses IP multicast User Datagram Protocol (UDP) packets with 224.0.0.251 and port 5353.

After IOT KIT connecting Wi-Fi and get IP address, the mDNS register process started. The service name for shtc1 is "ht_sensor.Ameba._tcp.local" with IP address and PORT. It also sends text records include the information of IP, PORT, MAC_ADDR, PAIR_STATE, SERVICE_NAME and CONTROL_TYPE.

The smartphone which also has mDNS module can receive the mDNS packets and discover the service that the IOT KIT registered and read the text record to judge device's control type and pair state.

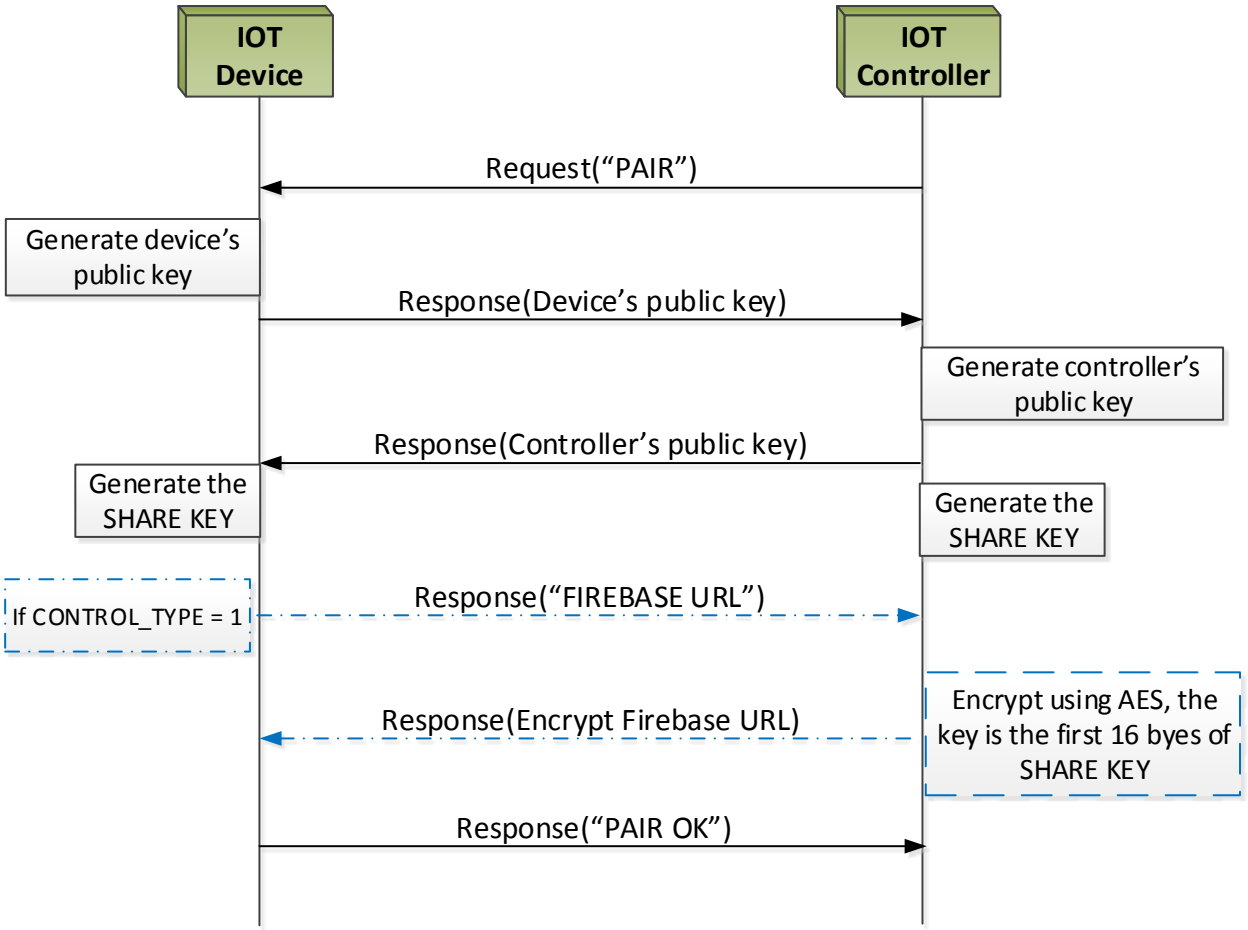
With the information getting from mDNS, smartphone can connect with the IOT KIT device and start to pair with it.

3.1.1.2 Device's information

- IP IP address of Ameba
- PORT PORT of Ameba
- MAC_ADDR MAC_ADDR of Ameba
- PAIR_STATE Value: 0 or 1 (Default value is 0, after pairing will be changed to 1. After reset of remove the device on APP will initial the value to 0. If just reboot, the value will not be changed.)
- SERVICE_NAME Now with the extension of SHTC1, SERVICE_NAME is ht_sensor.
- CONTROL_TYPE Value: 0 or 1 (0: Just supporting the local control; 1: Supporting both local control and cloud control.)

3.1.2 Pair

3.1.2.1 Flow chart of pair



3.1.2.2 Generate key

The method to generate key is implementing with curve 25519. The device and smartphone both generate 32 bytes random as their private key. And generate their public key using curve 25519. Then after exchanging their public key during pairing, the device and smartphone can generate the same shared key using their own private key and other's public key.

After that, all messages will encrypt by AES which key is the first 16 bytes of shared key.

3.1.3 Transmitting data

It is supported two types of devices for IOT KIT. One is just support local link, its CONTROL_TYPE is 0. The other is support both local link and cloud link, its CONTROL_TYPE is 1.

3.1.3.1 Local link

Local link is based on the TCP protocol. Each connection will complete one time receive and transmit. Once the device receiving an encrypted “request” data, the device will response the encrypt data. The method to encrypt data is AES. The KEY is the first 16 bytes of shared key which generated while pairing.

3.1.3.2 Cloud link

Cloud link now is based on the service of Firebase (the database of Google Nest). It’s using the https protocol. To use it, the controller must transmit a Firebase URL to device while pairing. Then the device will update data to Firebase.

3.1.3.3 Data type

This example supplies two types of data. One is for the customers who has the extension board with shtc1(a temperature and humility sensor), the other is using the pseudo data. The flag to open using pseudo data is in wigadget.h.

3.1.4 Compile guide

To enable IOT KIT in Ameba, please make sure the macro as follow are configured correctly.

To enable this application: configure CONFIG_ EXAMPLE_WIGADGET to 1 and undefined all functions that don’t need in platform_opt.h.

```
/* platform_opts.h */
#define CONFIG_EXAMPLE_WIGADGET        1
#define CONFIG_OTA_UPDATE      0
#define CONFIG_ENABLE_WPS      0
#define CONFIG_INCLUDE_SIMPLE_CONFIG 0
```

For network setting: configure LWIP_IGMP and LWIP_DNS to 1 in lwipopt.h and opt.h.

```
/* lwipopts.h */
#define LWIP_IGMP                       1

/* opt.h */
```

```
#define LWIP_IGMP                       1
#define LWIP_DNS                        1
```

Choose a control type: define the CONTROL_TYPE in wigadget.c.

```
/* wigadget.c */
#define CONTROL_TYPE                1//or 0
```

Choose a data type: define the PSEUDO_DATA in wigadget.h.

```
/* wigadget.h */
#define PSEUDO_DATA                 1//or 0
```

To ensure the application has enough heap size: set more than 115 kbytes heap size in FreeRTOSConfig.h

```
/* FreeRTOSConfig.h */
#define configTOTAL_HEAP_SIZE                   ( ( size_t ) ( 115 * 1024 ) )
```

3.1.4.1 Pairing for Device CONTROL_TYPE is 0

```
WIFI initialized
init_thread(47), Available heap 0x14340
========>Start to register mDNS service

<========Registering mDNS service OK!!

<<<<<<CONTROL_TYPE = 0  Need 2 times handshake>>>>>>

==========>PAIR_STATE = 0  Start to pair

===>Start the first handshake

<===First handshake OK!!

=====>Start the second handshake

<=====Second handshake OK!!

<==========Pairing OK!!

-----><START LOCAL LINKING
Sending data : {
        "TEM":  "1.01",
        "HUM":  "2.02"
}

<-----LOCAL LINK OK!!

-----><START LOCAL LINKING
Sending data : {
        "TEM":  "2.02",
        "HUM":  "4.04"
}
```

After Pairing, the device's PAIR_STATE will change from 0 to 1 and store to flash. The key is also stored in flash. If you reboot the device, it will also in paired state.

```
WIFI initialized
init_thread(47), Available heap 0x14340
========>Start to register mDNS service


<========Registering mDNS service OK!! PAIR_STATE = 1


----->START LOCAL LINKING
Sending data : {
        "TEM":  "1.01",
        "HUM":  "2.02"
}
```

3.1.4.2 Pairing for Device CONTROL_TYPE is 1

After Pairing, the device's PAIR_STATE will change from 0 to 1 and store to flash. The key and the Firebase URL are also stored in flash. If you reboot the device, it will also in paired state.

```
WIFI initialized
init_thread(47), Available heap 0x14340
========>Start to register mDNS service


<========Registering mDNS service OK!! PAIR_STATE = 1


----->START LOCAL LINKING
Sending data : {
        "TEM":  "1.01",
        "HUM":  "2.02"
}

=====>START CLOUD LINKING

CLOUD-LINK--Sending data :
{
        "TEM":  "1.01",
        "HUM":  "2.02"
}
```

Notice: The Firebase App ID inputted on the APP is "your-app-id"in the URL https://<your-app-id>.firebaseio.com/

3.1.4.3 *Reset*

I. Resetting the device through removing the device on APP.

II. Resetting the device through connecting the GPIO to GND(erase the PAIR_STATE, KEY and FIREBASE URL in flash).

4 APP

The smartphone application, WiGadget, implements three main functions provided by Ameba. The first is discovering the Ameba devices at a local network environment. The second is handshaking with Ameba. The third function is getting data from Ameba on local or from Firebase on cloud.

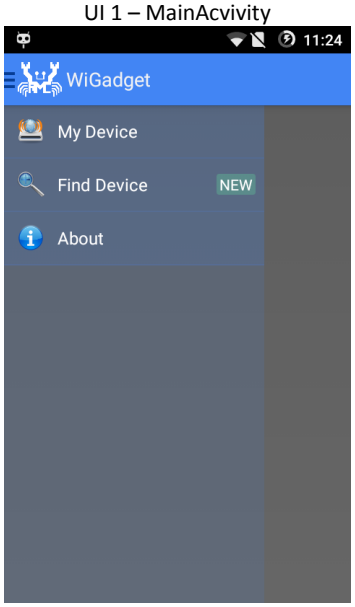
4.1.1 Android

The Android app provided in this release is targeted to Android phones with min. API level 16, which requires Android 4.1 (Jelly Bean) or above to function correctly. WiGadget demonstrates a typical procedure of using a mobile phone to control smart devices with common experiences include device discovery, device pairing and data gathering.

4.1.2 The UI & Flow

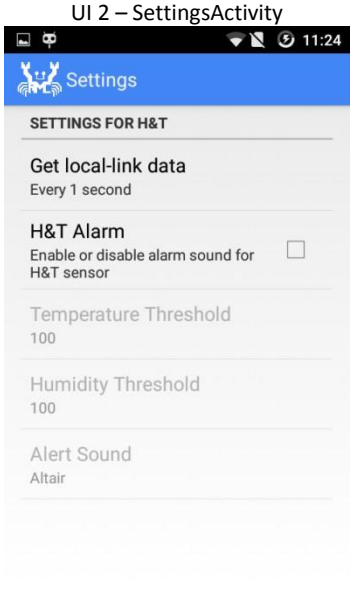
The UI screenshots and descriptions are list as below, please reference to the application flowchart attached at the end of the document to find out the internal interactions between these UIs.

UI 1 – MainAcvivity



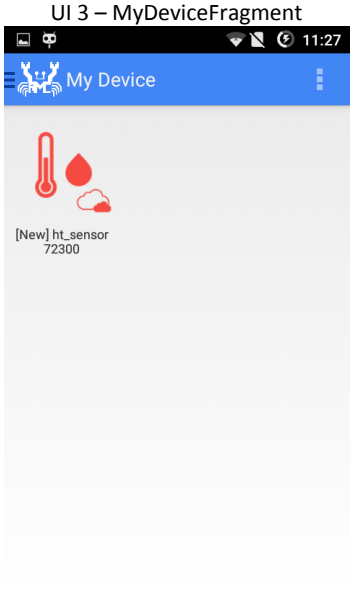
App launched and mDNS discover started, “NEW” text label appears if new Ameba device is discovered.

UI 2 – SettingsActivity



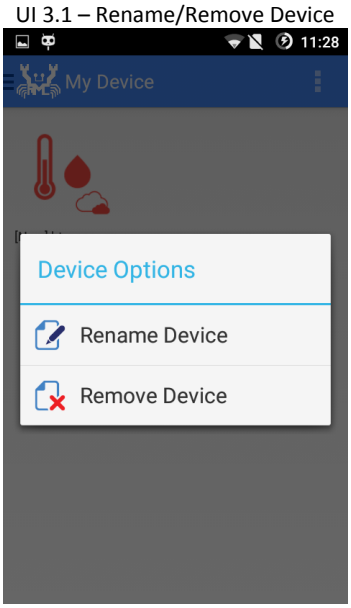
Presents customized settings for the app components, settings are stored in the internal storage as shared profile.

UI 3 – MyDeviceFragment



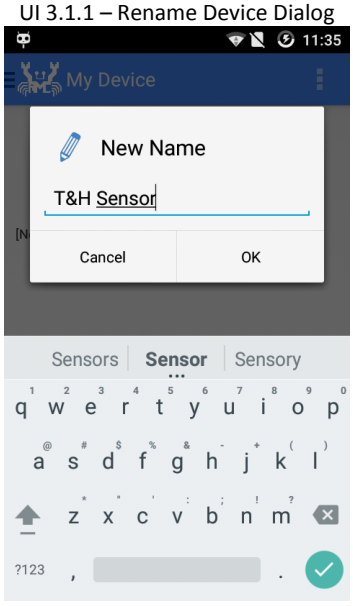
Displays exist and newly paired devices in a customized grid-view adapter.

UI 3.1 – Rename/Remove Device



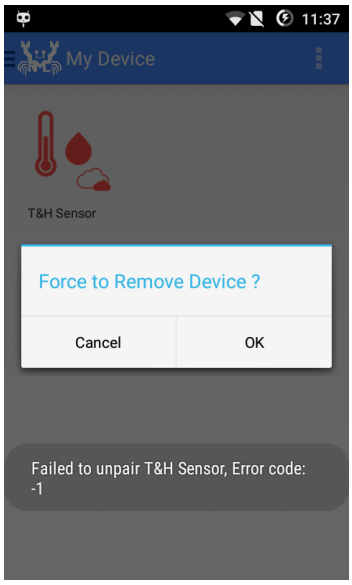
Long press (press & hold) on device icon to call out popup dialog to rename or remove saved device.

UI 3.1.1 – Rename Device Dialog



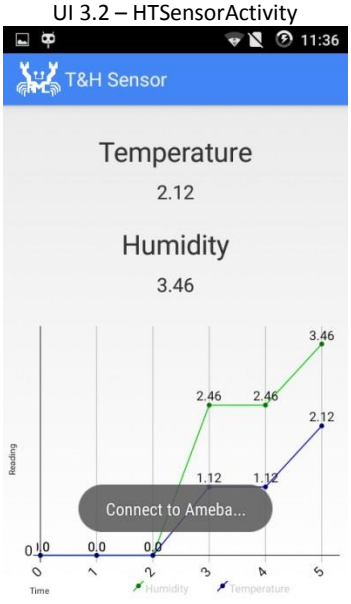
Rename device dialog, the length of the new name is not limited, but only two lines of characters will be displayed.

UI 3.1.2 – Force to Remove Device



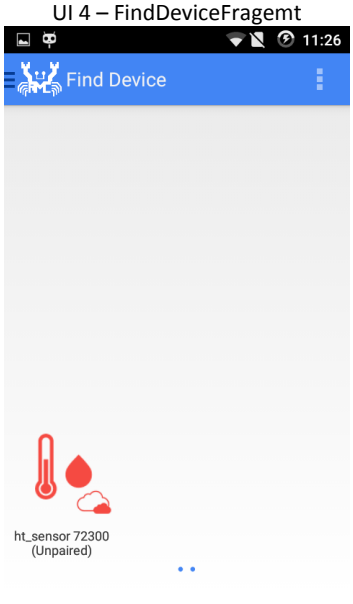
If the unpairing process is not finished successfully, a popup dialog will prompt user to remove device mandatorily.

UI 3.2 – HTSensorActivity



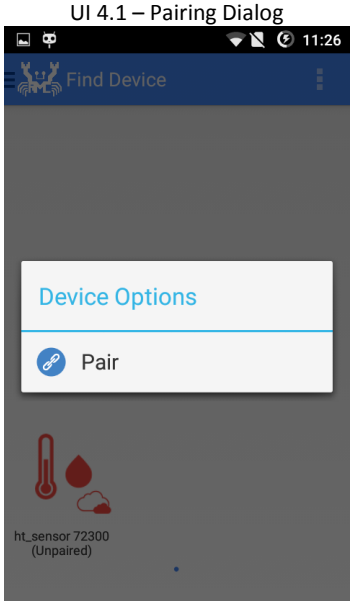
Get Temperature & Humidity Sensor (SHTC1) readings form cloud/local link and present them on the graph

UI 4 – FindDeviceFragemt



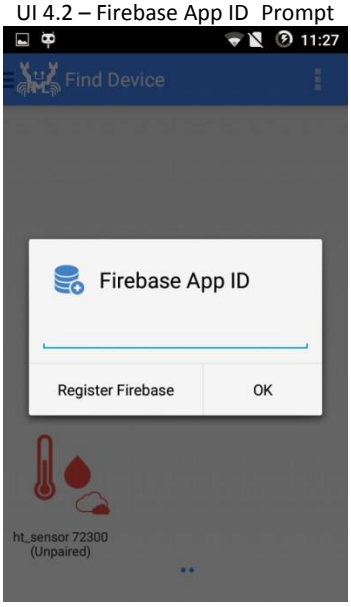
Displays newly discovered devices in a customized grid-view adapter.

UI 4.1 – Pairing Dialog



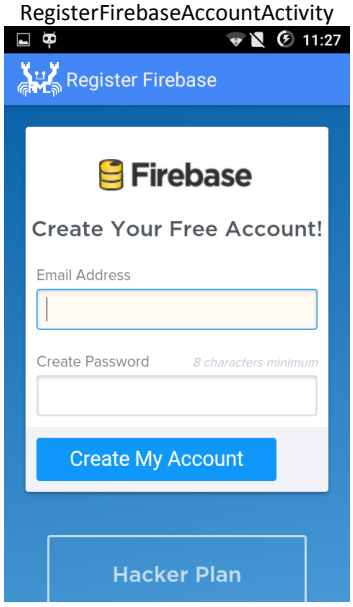
Click on (an unpaired) device to start pairing process, click on (an paired) device to start sharing process – *Device sharing will be supported in future release

UI 4.2 – Firebase App ID Prompt



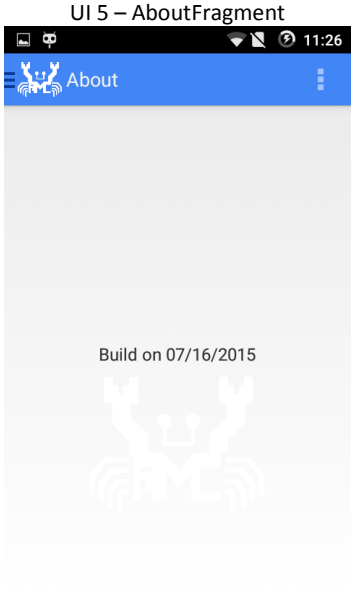
For cloud controlled devices, an availed firebase app id is required during pairing process, a cloud data base will be setup under the given app id
The Firebase App ID is “your-app-id“in the URL https://<your-app-id>.firebaseio.com/

UI 4.2.1 – RegisterFirebaseAccountActivity



If users don’t have a firebase account, the web-view will bring them to the firebase registration page, they can create a free firebase account and/or login to create an app id

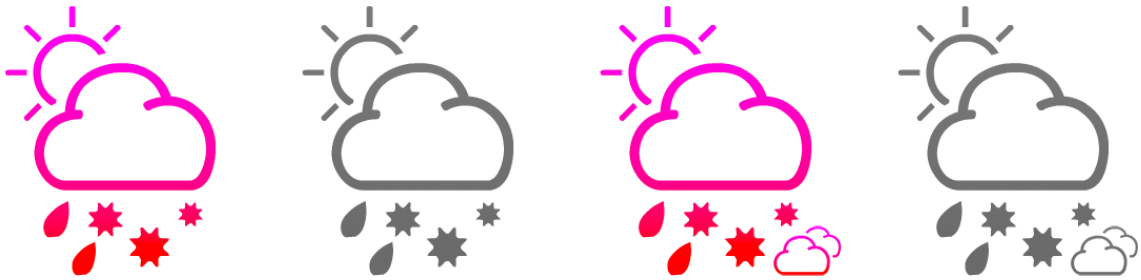
UI 5 – AboutFragment



Nothing here, just displays build information on text-view

4.1.3 The Device Icons

There are four types of icons used the in app to indicate the app-device link availability: local-controlled-online, local-controlled-offline, cloud-controlled-online, cloud-controlled-offline. Take H&T Sensor as an example:



local-controlled-online | local-controlled-offline | cloud-controlled-online | cloud-controlled-offline

Note: *A cloud-controlled device will be considered “online” as long as the firebase app id is availed -- even ameba side may be offline.*

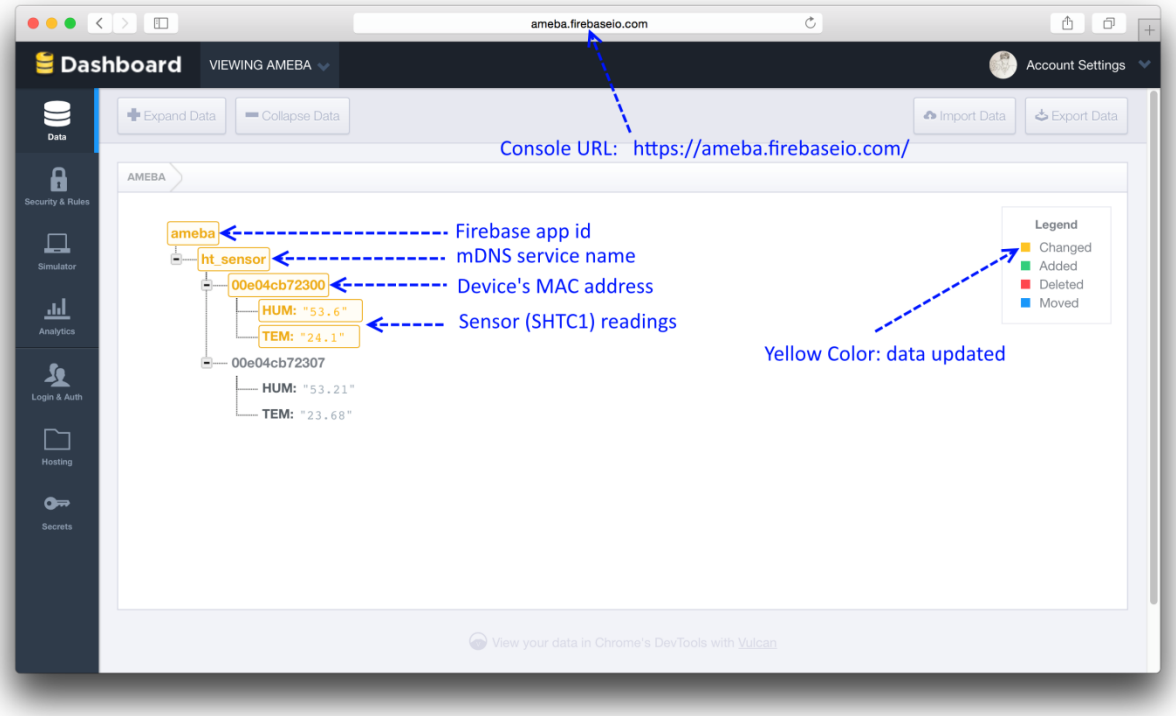
4.1.4 Cloud Link & Local Link

From a user perspective, the link type for WiGadget connect to Ameba is implicit. Users do not need to decide which connection method should be applied when WiGadget get data from Ameba. A simple description applies to all conditions: WiGadget always tries to use local link connection if Ameba is locally reachable.

Link / Control	Local Link Available	Local Link Not Available
Locally Controlled Device (controlType=0)	Local Link	No Connection
Cloud Controlled Device (controlType=1)	Local Link	Cloud Link

For cloud controlled devices, as soon as successfully paired with WiGadget, Ameba will start to push data to Firebase cloud. User can see the instantaneous updating of data readings from Firebase console by login to https://<your-app-id>.firebaseio.com/.

An illustration below shows the console of a Firebase app named "ameba" during WiGadget cloud-link testing:



4.1.5 The Library, Source & Issues

The library “commons-codec-1.6-repack.jar” and source code package “javax.jmdns.*” used in the app development are **not** as the same as given on the official repository.

The official build of Apache Commons Codec 1.6 library is known has methods naming conflicts with Android’s API, which can cause a function name resolving failure when compile project using Android Studio.

The official build of JmDNS library (current version is v3.4.1) is buggy and it can’t resolve new TXT Record correctly after Ameba changes its pairing state. After Ameba switches its pair state, it does detect the change of TXT Record but fails to resolve the new pair state out from the TXT Record cache.

Developers should be aware above issues if they want to use those libraries built form the official repositories.

4.1.6 Pairing Process & Error Code

The pairing process with Ameba involves several TCP-handshakes, pairing time and result may be various under different networking environment. A Toast Message may show Error Code on UI if the paring process ended up unsuccessfully, the meaning of error codes are listed at below:

Error Code	Meaning	Error Code	Meaning
-1	Pairing failed to start	-6	Java unknown host exception
-2	Server no response	-7	Java interrupted exception
-3	“PAIR” command rejected by server	-8	Java IO exception
-4	Public key rejected by server	-9	Firebase app id cannot be verified
-5	Firebase app URL rejected by server	-10	Any other unknown error

Note: *All error codes are specified in the “Constants” class, please refer to the source code to get more detailed information.*

Error reason:

-9: The Firebase app id is invalid or the network you connect cannot access firebase. Also checking the Firebase app id(if the URL of Firebase is: https://xxx.firebaseio.com, just inputting the “xxx”) .

Appendix: WiGadget Flowchart