[Mediatek SmartDevice Library]

User Guideline

Version: 1.1
Release date: 2017-01-04

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Author	Description
1.0	2016-10-20		Initial Version (Only Contains DOGP & FOTA)
1.1	2017-01-03		Add MT2511 Health

Table of Contents

1 Overview

Mediatek SmartDevice Library provides interfaces for scanning and connecting function for Mediatek wearable device. It supports the following profiles:

- **Apple Notification Center Service(ANCS)**

  ANCS is built-in iOS, which supports forwarding missed call information, un-read message and other related info to connected wearable device.

- **DOGP**
  - **FOTA**
  - **MT2511 Health**

2 MTKBleManager

MTKBleManager object are used to discovery and connect to remote wearable device (represented by CBPeripheral object), including scanning, discovering, and connecting to advertising peripherals.

Before you call MTKBleManager methods, the state of the central manager object must be powered on, as indicated by the CBCentralManagerStatePoweredon constant. This state indicates that the central device (your iPhone, for instance) supports Bluetooth low energy and that the Bluetooth is on and available to use.

Inheritance

NSObject

Conforms To

NSObject

Import Statement

import MTKBleManager.h

iOS Deployment Target

iOS 7.0 and later

2.1 Initializing a MTKBleManager

+ (id) sharedInstance;

Initializes the MTKBleManager object.

Return value

Returns an initialized MTKBleManager object.

2.2 Scanning or Stopping Scans of Peripherals

- (void)startScanning;

Scans for peripherals that are advertising services.

Discussion

MTKBleManager returns all discovered peripherals regardless of their support services. When the MTKBleManager object discovers a peripheral, it calls the BleDiscoveryDelegate methods of its delegate object.

- (void)stopScanning;

Asks the MTKBleManager to stop scanning for peripherals.

-(int)getScanningState;

Returns the Bluetooth central manager's current scanning state

2.3 Establishing Connections with Peripherals

- (void)connectPeripheral: (CBPeripheral *)peripheral;

Establishes a local connection to a peripheral.

Parameters

peripheral	The peripheral to which the central is attempting to connect.

Discussion

If a local connection to a peripheral is successfully established, the MTKBleManager calls the BleConnectDlegate: connectDidRefresh:(int)connectionState deviceName:(CBPeripheral*)peripheral; methods of its delegate object.

connectionState indicates connection result, CONNECTION_STATE_CONNECTED constant means connect successfully, while CONNECTION_STATE_DISCONNECTED constant indicates fail.

2.4 Register and Unregister Delegate

- (void)registerDiscoveryDelgegate: (id<BleDiscoveryDelegate>)discoveryDelegate;
- (void)unRegisterDiscoveryDelgegate: (id<BleDiscoveryDelegate>)discoveryDelegate;

Register and unregister the delegate for scanning result .

Discussion

If APP cares about the scanning result, it should call registerDiscoveryDelgegate. After this, MTKBleManager call BleDiscoveryDelegate: discoveryDidRefresh: (CBPeripheral *)peripheral; if any search result feedback from CBCentralManager.

If APP does not care about the scanning any longer, it should call unRegisterDiscoveryDelgegate. After this, MTKBleManager will not notify related info any more.

Parameters

discoveryDelegate	The delegate including scanning result and Bluetooth Power states

- (void)registerConnectDelgegate: (id<BleConnectDlegate>)connectDelegate;
-(void)unRegisterConnectDelgegate: (id<BleConnectDlegate>)connectDelegate;

Register and unregister the delegate for connecting states.

Discussion

APP calls registerConnectDelgegate before calling connectPeripheral: method to receive connected or disconnected event coming from MTKBleManager. If AP does not care the connect states any longer, unRegisterConnectDelgegate should be called.

Parameters

connectDelegate	The delegate which indicates connect states to AP.

- (void)registerBluetoothStateChangeDelegate:
- (void)unRegisterBluetoothStateChangeDelegate:

Register and unregister Bluetooth state change delegate

Discussion

AP calls registerBluetoothStateChangeDelegate to monitor changes to Bluetooth central manager's state.

Parameters

bluetoothStateChangeDelegate	The delegate which indicates Bluetooth central manager's state change to AP.

2.5 Forget paired device

- (void)forgetPeripheral

Call this method to notify MTKBleManager to clear the saved device.

Discussion

If APP needs to forget the already paired/connected wearable device, forgetPeripheral should be called to notify MTKBleManager to clear the saved device.

2.6 Constants

2.6.1 Scanning States

const static int SCANNING_STATE_ON = 1;
const static int SCANNING_STATE_OFF = 0;

Indicate the scanning states to AP.

2.6.2 Connection States

const static int CONNECTION_STATE_CONNECTED = 2;
const static int CONNECTION_STATE_CONNECTING = 1;
const static int CONNECTION_STATE_DISCONNECTING = 3;
const static int CONNECTION_STATE_DISCONNECTED = 0;

Indicates the connection states with remote wearable device.

2.7 Protocols to AP

2.7.1 BleDiscoveryDelegate

The BleDiscoveryDelegate protocol defins the method which allows the delegate to get the scanning result.

- (void) discoveryDidRefresh: (CBPeripheral *)peripheral;

Invoked when the MTKBleManager discovers a peripheral while scanning.

Parameters

peripheral	The discovered peripheral.

2.7.2 BleConnectDlegate

The connectDidRefresh protocol defines the methods which allow the delegate to get the connection states with remote wearable device.

- (void) connectDidRefresh:(int)connectionState deviceName:(CBPeripheral*)peripheral;

Invoked when connection is successfully or fail created with a remote wearable device.

Parameters

connectionState	The parameter which indicates the connect result.
peripheral	The peripheral that has been connected to the system

- (void) disconnectDidRefresh: (int)connectionState devicename: (CBPeripheral *)peripheral;

Invoked when an existing connection with a wearable device is torn down

Parameters

connectionState	The parameter which indicate the disconnect result.
peripheral	The peripheral that has been disconnected to the system

2.7.3 BleScanningStateChangeDelegate

The BleScanningStateChangeDelegate protocol defines the method which allow the delegate to get the current scanning states(Scanning/not Scanning).

- (void) scanStateChange:(int)state;

Invoked when MTKBleManager starts scanning or stop scanning.

Parameters

state	The parameter which the scanning states.

2.7.4 BluetoothAdapterStateChangeDelegate

The BluetoothAdapterStateChangeDelegate protocol defines the method which allows the delegate to get the Bluetooth central manager's states.

-(void)onBluetoothStateChange:(int)state;

Invoked when the Bluetooth central manager's state is updated.

Parameters

state	The latest state of Bluetooth central manager that updated. See `CBCentralManagerState` in iOS Development Document.

3 DOGP

DOGP (Data transfer Over GATT Profile) means app can send/receive data over BLE link. App only needs to inherit the class Controller, do some init action and override methods, it's easy to send and receive data.

Inheritance
NSObject

Conforms To
NSObject

Import Statement
#import Controller.h

iOS Deployment Target
iOS 7.0 and later

3.1 Initialization

- (id) init: (NSString *)tag cmdtype: (int)cmdtype;
Initialize the self-defined controller

Parameters

tag	A unique string to distinguish with other controllers.
cmdtype	A number between 0 and 9. For normal data, it should be set to 9

Discussion
App must call this method in its self-defined controller's initial function.

3.2 Set Tags

- (void)setReceiversTags: (NSArray *)tagStrArray;
Set tags to the DOGP controller.

Parameter

tagStrArray	A string array which contains the tags. Controller will add

3.3 Sending Data

- (void)send: (NSString *)cmd data: (NSData *)dataBuffer response: (BOOL)re progress: (BOOL)pr priority: (int)priority;

Sending data to remote device.

Parameters

cmd	The header string of the command.
dataBuffer	Real data which is to be sent
response	Boolean value which indicates need response from receiver or not
progress	Boolean value which indicates apps need sending progress or not
priority	Sending data priory. It should be PRIORITY_NORMAL, PRIORITY_LOW or PRIORITY_HIGH. Default value is PRIORITY_NORMAL.

Discussion

App calls this method to send data to DOGP receiver.

3.4 Receiving Data

-(void)onReceive:(NSData*)data;

Getting data send by remote wearable device.

Parameter

data	data sent by remote wearable device.

Discussion

Self-defined controller must override this method to receive data sent by remote wearable device.

3.5 Getting Sending Progress

-(void)onProgress: (float)sentPercent;

Sending data progress.

Parameter

sentPercent	sending progress which is between 0.00~1.00

Discussion

Self-defined controller must override this method to get sending progress. It's effective if apps set the parameter *progress* to YES while calling - (void)send: (NSString *)cmd data: (NSData *)dataBuffer response: (BOOL)re progress: (BOOL)pr priority: (int)priority; method.

4 FOTA

4.1 Overview

It's used to send firmware data to wearable device to do update action, including following steps:

- Get current version from wearable device via BT(BLE)
- Check new version exist or not on server
- If exist, download the newest version
- Begin to transfer data via BT(BLE)
- Wearable device start to upgrade according the data
- Feedback the result to smart phone

4.1.1 FotaOperator

The operator used to send data to wearable device, receive data from wearable device, parse the received data, and while the data received, notify the changing to UX.

4.1.2 FotaVersion

While send the get version command to wearable device, the wearable device should transfer back the version, module, platform, domain information and so on. All the information will be built into the FotaVersion.

4.1.3 FotaDelegate

A callback which should be called by FotaOperator while data received from wearable device, contains received version, connection state change, firmware data transfer progress, and so on.

4.2 FotaOperator

4.2.1 +(id)sharedInstance

Get the FotaOperator shared instance from this static public API which can do the operations.

E.g :

```
FotaOperatpr* operator = [FotaOperator sharedInstance];
```

4.2.2 –(void)registerFotaDelegate: (id<FotaDelegate>)delegate;

Register a FOTA delegate to FotaOperator, while the status or data received from wearable device, then all the delegates which registered in the FotaOperator will be notified. If not register, any change will not be notified out.

e.g:

```
[operator registerFotaDelegate:delegate];
```

4.2.3 – (void)unregisterFotaDelegate: (id<FotaDelegate>)delegate;

Unregister the FotaDelegate from FotaOperator, and then the change will not be notified to it.

e.g

```
[operator unregisterFotaDelegate:delegate];
```

4.2.4 – (void)sendFotaTypeCheckCommand;

Send the FOTA type check command to wearable device to get the FOTA type, which maybe RedBend fota and Separate Bin fota.

The onFotaTypeReceived will be called once received the type from wearable device.

e.g

```
[operator sendFotaTypeCheckCommand];
```

4.2.5 – (BOOL)sendFotaVersionGetCommand: (int)whichType;

Send version get command to wearable device

The onVersionReceived will be called once received version from wearable device.

If the whichType is not match RedBend FOTA or Separate Bin FOTA, return NO;
Others return YES;

e.g:

```
[operator sendFotaVersionGetCommand];
```

4.2.6 – (BOOL)sendFotaFirmwareData: (int)whichType dataFromFile: (NSData*)data;

Once download the firmware data finished, and then can begin to transfer the data to wearable device. data should be built from the downloaded firmware file.

While the data is transferring, the onProgress will be called, if any problem happens, or transfer finished, the onStatusReceived will be called.

If whichType is not match RedBend FOTA or Separate Bin FOTA, return NO;
If data is nil or data length is 0, return NO;
Others return YES;

e.g :

```
[operator sendFotaFirmwareData:data];
```

4.2.7 -(void)cancelCurrentSending;

Cancel the current transferring, delete the transfer session, and free the memories.
If called, the onProgress or onStatusReceived will not receive any more.

4.3 FotaVersion

```
-NSString*    version;
-NSString*    releaseNote;
-NSString*    module;
-NSString*    platform;
-NSString*    deviceId;
-NSString*    brand;
-NSString*    domain;
-NSString*    downloadKey;
-NSString*    pinCode;
-BOOL         isLowBattery;
```

The wearable device’s information.

4.4 FotaDelegate

4.4.1 – (void)onFotaTypeReceived: (int)fotaType;

While received FOTA type from wearable device, this callback will be called.

fotaType :

```
REDBEND_FOTA_UPDATE             = 0;
SEPARATE_BIN_FOTA_UPDATE        = 1;
ROCK_FOTA_UPDATE                = 4;
FBIN_FOTA_UPDATE                = 5;
```

The UX maybe need be updated according to this callback.

4.4.2 – (void)onVersionReceived: (FotaVersion*)version;

While received the FOTA version from wearable device, this callback will be called. The version is an instance of FotaVersion, which including wearable device version, module, platform, deviceId, domain, pinCode, brand, and so on.

Then need transfer these data to download server to check the new version exist or not.

If get version failed, the version should be nil.

4.4.3 – (void)onStatusReceived: (int)status;

While transferring the firmware data, if any problem happens, the callback will be called.

While the transfer finished, the transfer result (transfer success or fail) will be transferred through this callback.

While the transfer succeeds, and then the wearable device will trigger the update program, if trigger failed, the callback will be called.

Or any other problem happens, this callback will be called.

status :

FOTA_UPDATE_VIA_BT_TRANSFER_SUCCESS = 2;
FOTA_UPDATE_VIA_BT_UPDATE_SUCCESS = 3;
FOTA_UPDATE_VIA_BT_COMMON_ERROR = -1;
FOTA_UPDATE_VIA_BT_WRITE_FILE_FAILED = -2;
FOTA_UPDATE_VIA_BT_DISK_FULL = -3;
FOTA_UPDATE_VIA_BT_TRANSFER_FAILED = -4;
FOTA_UPDATE_VIA_BT_TRIGGER_FAILED = -5;
FOTA_UPDATE_VIA_BT_UPDATE_FAILED = -6;
FOTA_UPDATE_VIA_BT_TRIGGER_FAILED_CAUSE_LOW_BATTERY = -7;

4.4.4 – (void)onConnectionStateChange: (int)newState;

If the DOGP connection state changed, this callback will be called, UX maybe need to be updated according to the newState.

newState :

CONNECTION_STATE_DISCONNECTED = 0
CONNECTION_STATE_DISCONNECTING = 3;
CONNECTION_STATE_CONNECTING = 1;

CONNECTION_STATE_CONNECTED = 2;

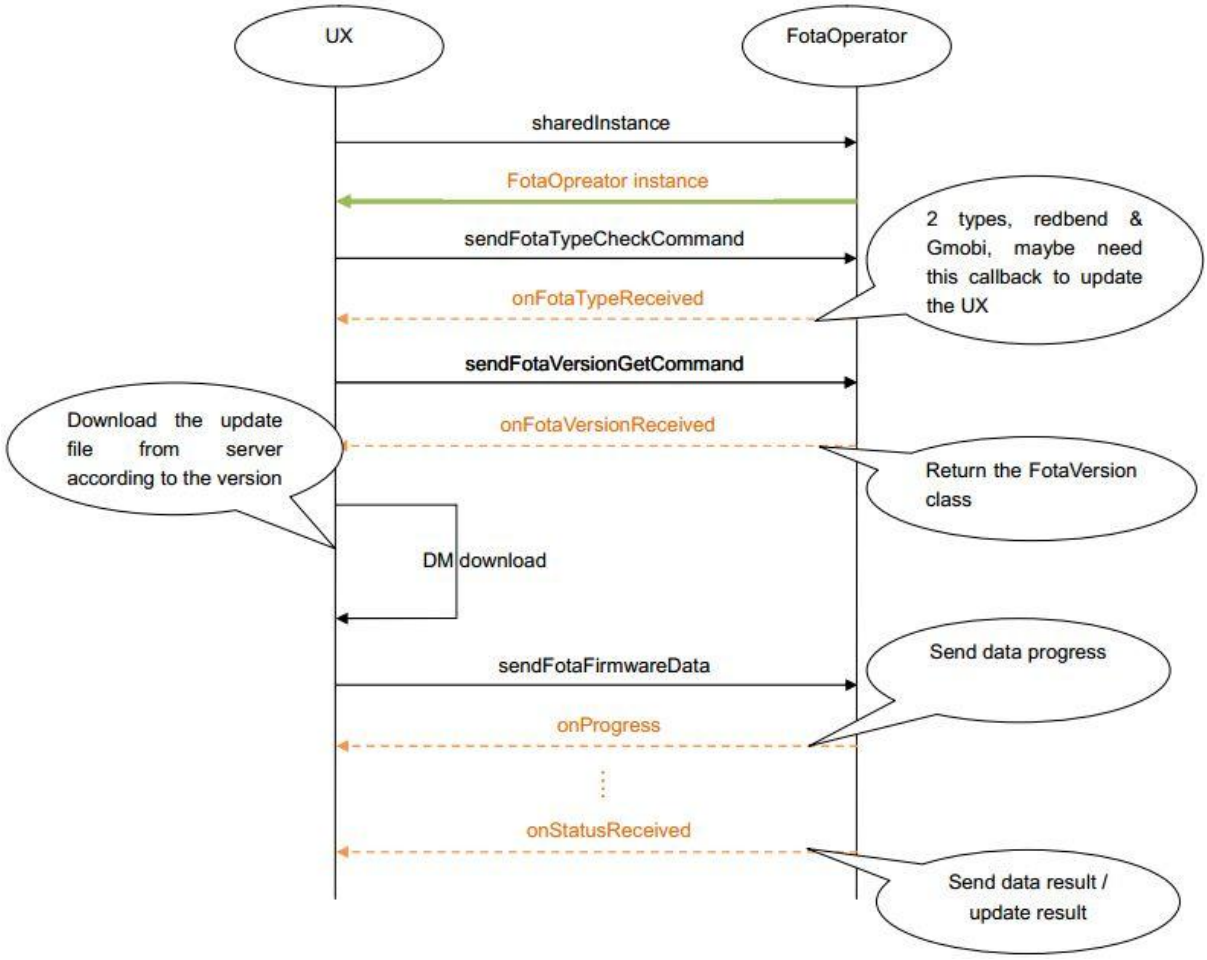
4.4.5 – (void)onPorgress: (int)progress;

While transferring the firmware data through DOGP, the progress will be update according to this callback. The UX maybe need to be updated according to progress.(NEED TO BE FINISH)
progress : 0 ~ 100

4.4.6 -(void)onReadyToSend;

While the BLE connection is connected, and the DOGP handshake done with remote wearable device, this delegate will be notified. AP can start to transfer the firmware data to remote device.

4.5 FOTA Operation Sample Sequence



5 MT2511 Health

5.1 Overview

MT2511 is a bio-AFE support PPG (photoplethysmography) / ECG (electrocardiography) signal. It could measure heart rate (HR) and blood pressure (BP). If your device with MT2511 sensor and corresponding software, you could receive MT2511 HR/BP data by DOGP.

5.1.1 DOGP CMD

MT2511 HR/BP module uses DOGP command 9 (see 3.1 错误!未找到引用源。) and "`health_sender/health_receiver`" as DOGP command parameter.

5.1.2 Hand Shake

IPA must send "Health Hand Shake" command to wearable device to query health feature.
If IPA support HR feature, it should send "`health_sender health_receiver 0 0 12 hr_handshake`" by DOGP API. Then, IPA will receive "`health_sender health_receiver 0 12 hr_handshake`" when remote wearable device support HR measure feature.
Likewise, "`bp_handshake`" and "`calibration`" will be used as BP measure and BP calibration Hand Shake command.

5.2 HR Measure

IPA will receive 64 bytes (16 integers) raw data (Little Endian) by DOGP API after wearable device enabled MT2511 sensor and sent HR measure data. The HR data format is as follows:

Data Format	Comments
[0] magic	Ignore
[1] sensor type (HR=22)	Sensor type must be 22, otherwise this data isn't HR measure data.
[2] sequence	The sequence number should be increased from 1.
[3] BPM	Beats per minute
[4] status	Bit 0~7: Confidence Level of HR signal (= status & 0x000000FF). Range: -1 ~ 3, only 2/3 indicates that HR signal is stable and good. Bit 8~31: sensor debug info, ignore.
[5] timestamp	Ignore
[6-15] reserved	Ignore

5.3 BP Measure

IPA must send firstly personal information to wearable device.
The personal information uses TLV (Tag, Length and Value) format, and sent by DOGP API.
The supported tag is as follows:

Tag Name	Tag
User ID	1
Height	2
Weight	3
Gender	4
Age	5
Arm Length	6
BP Mode	7
BP Calibration Parameter	8

All tag length only is 1 byte, range 1~255.
User ID is a string with maximum limit of 20.
Other data should is integer (4 bytes). Gender male is 1, female is 2.
Arm length could be set 0.
The BP mode has three optional values: 0 (general mode), 127 (personal mode) and 1023 (calibration mode).

For example, BP measure user name is `Jack`, `29` years old, `173cm`, `68kg`, `male`, and arm length is `50cm.` The personal information data should sent in the following format.

Packet length	47			2F
Sensor type	3001			B9 0B 00 00
	TAG	Length	Value	HEX Data
User ID	1	4	Jack	01 04 4A 61 63 6B
Height	2	4	173	02 04 AD 00 00 00
Weight	3	4	68	03 04 44 00 00 00
Gender	4	4	1	04 04 01 00 00 00
Age	5	4	29	05 04 1D 00 00 00
Arm Length	6	4	50	06 04 32 00 00 00
BP Mode	7	4	0 (general mode)	07 04 00 00 00 00

So, IPA will send personal information raw data: `2F B9 0B 00 00 01 04 4A 61 63 6B 02 04 AD 00 00 00 03 04 44 00 00 00 04 04 01 00 00 00 05 04 1D 00 00 00 06 04 32 00 00 00 07 04 00 00 00 00`.

IPA will receive 64 bytes (16 integers) raw data (Little Endian) by DOGP API after wearable device enabled MT2511 sensor and sent BP measure data. The BP data format is as follows:

Data Format	Comments
[0] magic	Ignore
[1] sensor type (BP=24)	Sensor type must be 24, otherwise this data isn't BP measure data.
[2] sequence	Ignore
[3] SBP	Systolic BP
[4] DBP	Diastolic BP
[5] BPM	Beats per minute
[6] status	It Should be 0.
[7] timestamp	Ignore
[8-15] reserved	Ignore

5.4 BP Calibration

The entire BP calibration process is divided into five operations.

5.4.1 Step 1: Send Personal Info

At first, IPA should sent personal information, as described in section 5.35.3, but need to set the BP mode as `1023` (calibration mode);

5.4.2 Step 2: Receive Calibration Parameter

After wearable device start to BP measure, IPA should receive calibration parameter (only 6 integers) by DOGP API, as shown in the following table.

Data Format	Comments
[0] magic	Ignore
[1] sensor type (8001)	Sensor type must be 8001.
[2] sequence	The sequence number should be increased from 1.
[3] feature type (3)	Feature type must be 3.
[4] size (6)	The size should be 6.
[5] timestamp	Ignore
[6~11] parameter	Calibration parameter (6 integers)
[12-N] reserved	Ignore

5.4.3 Step 3: Input Golden Sample

IPA should input two BP golden samples (sbp1/dbp1 and sbp2/dbp2) measured by professional blood pressure equipment, such as OMRON BP monitor.

5.4.4 Step 4: Send Calibration Data

IPA should send calibration data to wearable device. For example,

Packet length	79			4F
Sensor type	3001			B9 0B 00 00
	TAG	Length	Value	HEX Data
BP Calibration Parameter	8	72	“Detailed calibration data”	08 48 “Detailed calibration data”

The detailed calibration data:

BP sbp1	101	65 00 00 00
BP dbp1	81	51 00 00 00
BP sbp2	102	66 00 00 00
BP dbp2	82	52 00 00 00
BP sbp3	reserve	00 00 00 00
BP dbp3	reserve	00 00 00 00
BP Calibration para1	102	66 00 00 00
BP Calibration para2	71	47 00 00 00
BP Calibration para3	50	32 00 00 00
BP Calibration para4	223	DF 00 00 00
BP Calibration para5	274	12 01 00 00
BP Calibration para6	0	00 00 00 00
BP Calibration para7	reserve	00 00 00 00
BP Calibration para8	reserve	00 00 00 00
BP Calibration para9	reserve	00 00 00 00
BP Calibration para10	reserve	00 00 00 00
BP Calibration para11	reserve	00 00 00 00
BP Calibration para12	reserve	00 00 00 00

So, IPA will send calibration raw data: `4F B9 0B 00 00 08 48 65 00 00 00 00 51 00 00 00 66 00 00 00` and so on.

5.4.5 Step 5: enter personal mode

In order to achieve a more accurate BP measurement, not only personal information and corresponding calibration data should be sent to wearable device by DOGP, but also the BP mode must set to `127` (personal mode) to notify wearable device load those data.