MediaTek LinkIt™ Smartphone Notification Developer's Guide

Version:	1.0
Release date:	9 April 2015

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	9 April 2015	Initial release

Table of Contents

Lists of tables and figures

1 Introduction

The MediaTek SmartDevice Library provides functionality to scan, connect, disconnect, autoconnect and push notifications to Wearables and IoT devices. The library supports two modes of operation using serial port profile (SPP) and data transfer over generic attribute profile (GATT) (DOGP) Bluetooth-based data transfer protocols, which are mainly targeted to Bluetooth Smart (low energy) technology enabled devices.

The library also provides notification push; enabling the SmartDevice to listen for missed call, newly arrived message, and other notifications, then forward them to the connected wearable device. The MediaTek SmartDevice Library is a collection of Java class files and other resources stored in `wearable.jar`. The architecture of the library is shown in *Figure 1*.

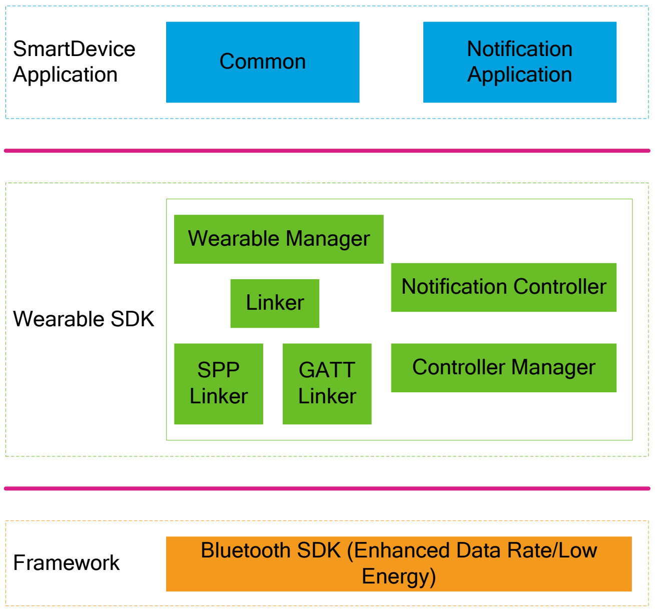

Figure 1 Architecture of the SmartDevice library

The SmartDevice app provides the main user interface (UI) and implementation of the `NotificationAppListActivity` class. The Wearable SDK is the `.jar` source code that contains classes, metadata and other resources for SmartDevice connectivity. `WearableManager` is the fundamental class that provides scanning for available wearable devices and is the connection interface to the main app. The library supports two work modes, which are part of Wearable SDK's linker codes: SPP linker and GATT linker. Notification Controller API accepts notifications and forwards them to the wearable device. Behind this operation is the Android Bluetooth SDK enhanced data rate (EDR)/low energy API as the main framework.

To design and implement an Android app using the SmartDevice library use of the following is recommended:

1) Eclipse IDE
   Eclipse IDE is recommended as a SmartDevice application development tool. By placing SmartDevice library into the Android application project's `libs` folder, you'll be able to utilize the contents of the library appropriately. The SmartDevice library is compatible with Android 4.3 SDK and above, as it supports Bluetooth Smart technology application development. Any lower versions of Android SDK will result in project build failure in Eclipse. For more detailed description on how to create an Android application in Eclipse refer to Android development with Eclipse tutorial.

2) Android SDK API package

   The SmartDevice library works on any Android device that has Android API level 14 or higher. If the Android application package (APK) with SmartDevice library is installed on a smartphone Android 5.0 (Kitkat) or above, it'll enable SPP and DOGP. The SmartDevice library can also run in Android 4.0 (Ice Cream Sandwich) (SDK level 14), but it supports the SPP protocol only.

The communication flow between the application and a wearable device is shown in *Figure 2*. The SmartDevice app is able to scan for and connect to the wearable device, once it's successfully connected it can send a notification to the wearable and receive commands or data from the wearable device.

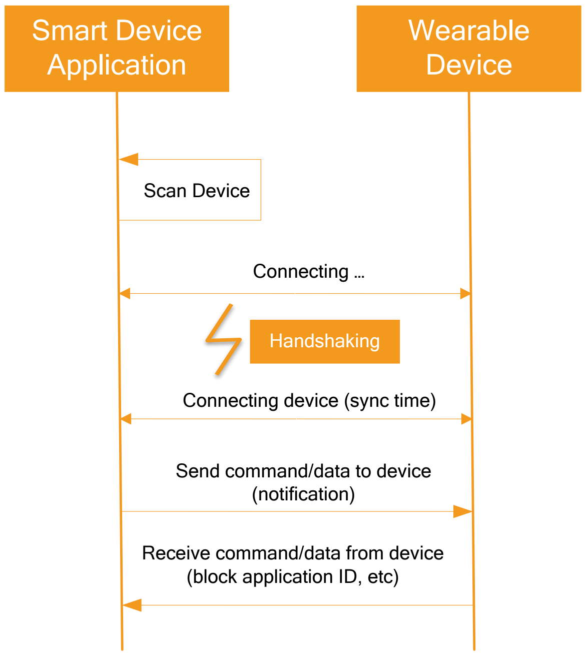
Smart Device Application
Wearable Device
Scan Device
Connecting …
Handshaking
Connecting device (sync time)
Send command/data to device (notification)
Receive command/data from device (block application ID, etc)

Figure 2. Communication flow between SmartDevice app and a wearable device

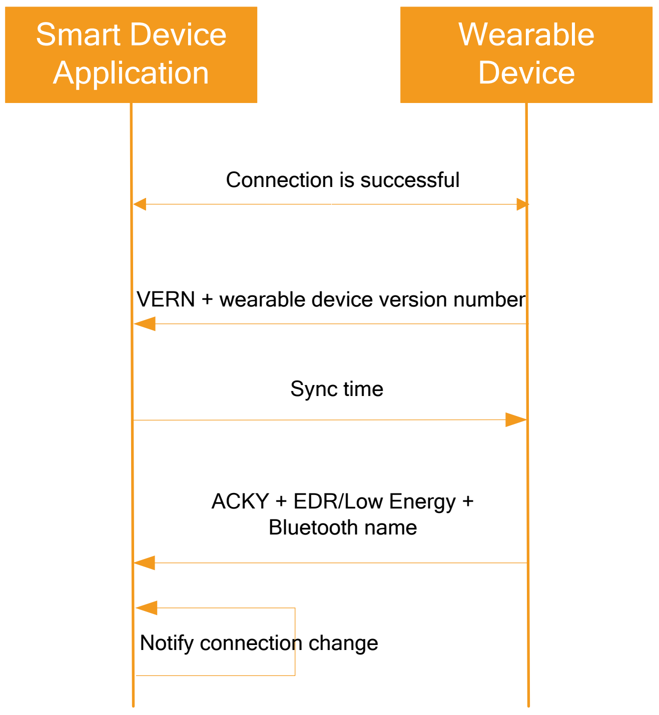

Figure 3. Handshaking flow between SmartDevice app and a wearable device

As shown in Figure 3, once the app is connected to the wearable device there is a handshake process established either by the SPP or DOGP protocol. The process is as follows:

1) The app receives virtual environment resource navigator VERN command and wearable version number from the wearable device.
2) The app sends the SYNC command and smartphone's current time and time zone to the wearable device.
3) The wearable device replies to the app with the ACKY command and its Bluetooth 4.0 address and name. Finally, the APK notifies any connection change and updates UI.

2 SmartDevice Library

The section provides detailed description of the library's function prototypes.

2.1 Initializing WearableManager

```
+ (void) init(Context appContext, String key);
```

Initializes the `WearableManager` object, application calls `WearableManager`'s `getInstance` static method to get the object.

Parameter	Description
`appContext`	The application context, application calls Android API `getApplicationContext` to get it.
`key`	An input string that must match with watch Shake Hands Key, default value of which is "we had".

2.1.1 Description

`WearableManager` is the primary class in the SmartDevice library (`wearable.jar`). The `WearableManager` class implements scanning, connecting/disconnecting, switching mode, and registering `WearableListener` (callback interface to notify `WearableManager` state to app).

`WearableManager` should be initialized in the main thread Handshaking key is required as an input to initialize SmartDevice application. Default key is "we had". If the input key is "my.com" in SmartDevice APK code, handshake key must be set to "my.com" in the wearable device source code. Otherwise, the handshake flow will fail.

Example code: BTNotificationApplication onCreate method.

2.2 WearableListener

```
+ (void) onConnectChange(int oldState, int newState);
```

Notify connection change of SPP/DOGP from an old state to a new state.

```
+ (void) onDeviceChange(BluetoothDevice device);
```

Notify the selected remote device that the application is ready to connect.

```
+ (void) onDeviceScan(BluetoothDevice device);
```

Notify of the newly discovered device after a Bluetooth scan is complete.

```
+ (void) onModeSwitch(int newMode);
```

Notify a change of work mode.

2.2.1 Description

The interface class `WearableListener` provides methods to listen to the connection state, device and work modes (SPP or DOGP). The application calls `WearableManager` `registerWearableListener` to register and `unregisterWearableListener` to deregister the listener.

In addition, `WearableManager` provides `getRemoteDevice`, `getConnectState`, `getWorkingMode` methods to get current device or connection state information. The application should call the `isAvailable` method to make sure that the current Bluetooth connection (SPP/DOGP) is available for further processing ready after data are sent.

Example code: `MainActivity onConnectChange` method; `DeviceScanActivty onDeviceScan` method; `CustomPreference onBindView` method;

2.3 Scan Device

```
+ (void) scanDevice(boolean enable);
```

Start or stop scanning the BT device.

Parameter	Description
enable	If enable is true, `WearableManager` will scan device, otherwise it will stop scanning.

2.3.1 Description

The `scanDevice` API will call `BluetoothAdapter startDiscovery` and `cancelDiscovery` in SPP mode. If current work mode is DOGP, it will call `startLeScan` and `stopLeScan`.
The `WearableManager` will callback `onDeviceScan(BluetoothDevice)` when the SmartDevice APP finds a Bluetooth device.

Example code: `DeviceScanActivty scanDevice` method;

2.4 Connect/Disconnect

```
+ (void) setRemoteDevice(BluetoothDevice device);
+ (void) connect();
+ (void) disconnect();
```

`WearableManager` connect/disconnect API.

2.4.1 Description

Call `setRemoteDevice` to select a Bluetooth device from the scanned device user interface (UI) before connecting, this will trigger `onDeviceChange(BluetoothDevice)` callback to update APP UI.
Call connect or disconnect interfaces directly depending on SPP or DOGP working modes. They will trigger `onConnectChange(int oldState, int newState)` callback to notify APP connection state.

Example code: `CustomPreference`;

2.5 Switch Mode

```
+ (void) switchMode();
```

Call this method to switch between work modes (SPP or DOGP).

2.5.1 Description

The API will close the previous connection, switch `WearableManager`'s work mode, and trigger an `onModeSwitch(int newMode)` callback. If switching the mode is successful, the parameter "newMode" is set to either **MODE_SPP** or **MODE_DOGP**, otherwise it will be equal to -1.

The switch mode will fail and callback mode will be set to -1 in the following scenarios:

1) The `WearableManager` connection state is connecting, connected or disconnecting.
2) The `WearableManager` is sending data to a remote smart device.
3) The smartphone SDK version is lower than Android 4.3 (API 18) or Bluetooth Smart (low energy) feature is disabled.

2.6 Notification Controller

A Notification Controller inherited from a `Controller` class forwards smartphone notifications to a wearable device. `getInstance` static method provides a unique instance of **`Notification Controller`** class, and `WearableManager` `addController/removeController` methods use that instance to add or remove the`NotificationController`.

The `NotificationController` class provides the following interfaces

2.6.1 sendNotfications

```
+(void) sendNotfications(String appId, CharSequence package, CharSequence
tickerText, long when, String[] textList);
```

Forward a notification to a remote wearable device.

Parameter	**Description**
`appId`	Unique ID to distinguish different applications
`packageName`	Indicate the application package name where this notification comes from
`tickerText`	The notification ticker text
`when`	Time when smartphone received notification
`textList`	The notification body list

2.6.2 sendSmsMessage

```
+ (void) sendSmsMessage(String msgBody, String address);
```

Pack an SMS message and send to a wearable device once a new message arrives on a Smartphone.

Parameter	**Description**
`msgbody`	the SMS text content
`address`	the SMS sender address

2.6.3 sendCallMessage

```
+ (void) sendCallMessage(String phoneNum, String sender, String content, int
count);
```

Pack missed-call-count info and send to a wearable device.

Parameter	Description
phoneNum	Sender address of the missed call
sender	Sender name of the missed call
content	The missed-call notification message body
count	The number of missed calls

2.6.4 sendReadMissedCallData

```
+ (void) sendReadMissedCallData();
```

Update read status on a remote wearable device after reading all the missed calls (count = 0) in smartphone.

2.6.5 sendLowBatteryMessage

```
+ (void) sendLowBatteryMessage(String title, String content, String appId, String
value);
```

Pack low battery message and send to a wearable device.

Parameter	Description
title	The low battery message title
content	The low battery message body
appID	Unique id to distinguish different applications
value	Current battery capacity, such as %1-100%

2.6.6 NotificationEventListener

```
+ (void) notifyBlockListChanged(String appId);
```

Callback when user marks any notification message as blocked in a wearable side. The notification about blocked application (appId) will not be sent to the wearable.

User needs to implement `NotificationEventListener` and call `NotificationContorller setListener` method to receive response from the wearable device.

2.7 Constants

2.7.1 Work Mode

```
public static final int MODE_SPP = 0;
public static final int MODE_DOGP = 1;
```

Indicate current work mode.

2.7.2 Connection States

```
public static final int STATE_NONE = 0;
public static final int STATE_LISTEN = 1;
public static final int STATE_CONNECT_FAIL = 2;
public static final int STATE_CONNECT_LOST = 3;
public static final int STATE_CONNECTED = 4;
public static final int STATE_CONNECTING = 5;
public static final int STATE_DISCONNECTING = 6;
```

Indicate the connection states of a remote wearable device.