

CONFIDENTIAL B

The MediaTek logo consists of the word "MEDIATEK" in white, uppercase, sans-serif font, centered within an orange parallelogram shape that is wider at the top and bottom and tapers in the middle.

MEDIATEK

MediaTek IoT SmartDevice App Programming Guide

2017.04.19

Outline

- Overview
- Customization
- Wearable SDK

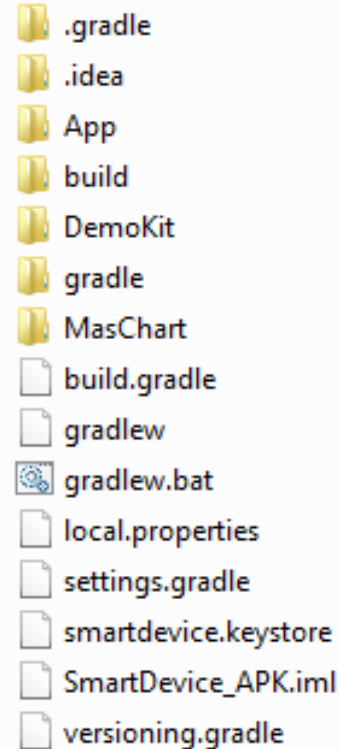
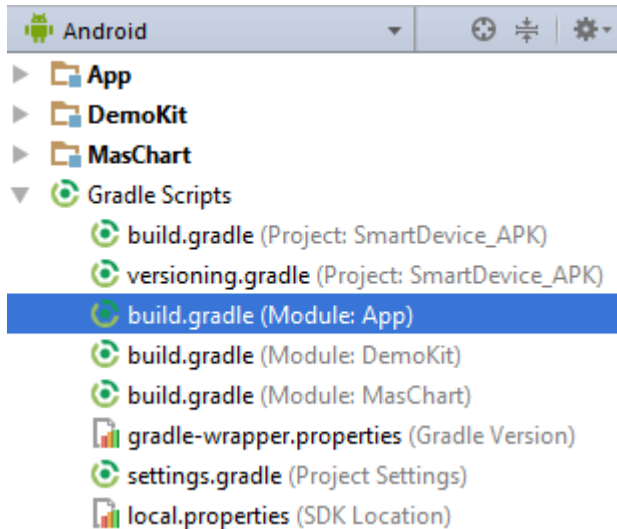
Overview

Overview (1/3)

- **MediaTek IoT SmartDevice** is an Android application project used for MediaTek IoT device (based on MT2523/MT2533 chip).
- It uses **BTNotify** (MTK BT Transport Protocol) to communicate with device.
- It is an Android Studio project from IoT SDK 4.3.

Overview (2/3)

■ Folder Structure



Overview (3/3)

■ Module Dependencies

SmartDevice APP Module

App

Wearable.jar

SmartDevice App Module: MainActivity/Notification /FOTA/AboutActivity, use BTNotify transport protocol to communicate with device by using wearable.jar SDK.

DemoKit

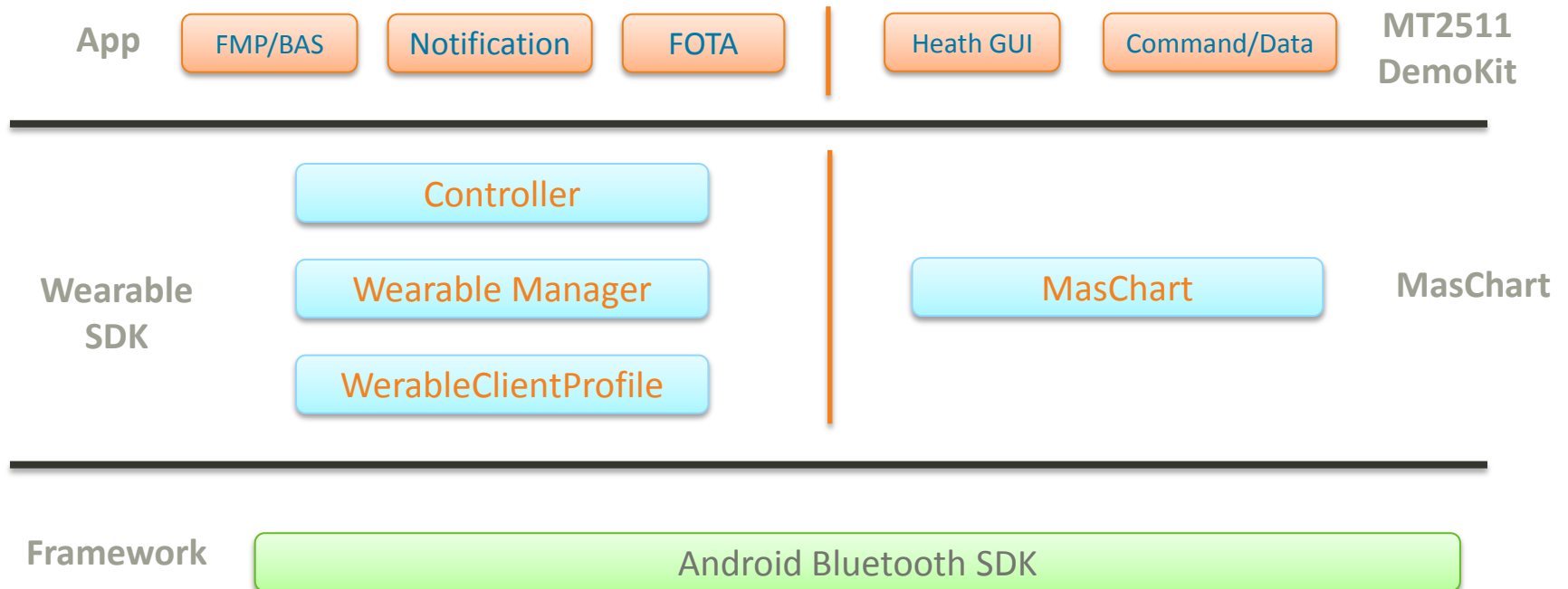
MT2511 Health GUI
Health Command/Data/Parser

MasChart

MTK Android Chart Library
for SmartDevice Health

Architecture

SmartDevice APK



Introduction

- “Customization” will introduce some methods about how to customize your APP.
- “Wearable SDK” will introduce BTNotify transport protocol, wearable SDK API, how to implement your own controller.
- To study implementation detail of Health features, please refer to [*<MT2511_Health_Module_Programming_Guide.pdf>*](#).

Customization

How to Build (1/2)

■ IDE

- The [Android Studio \(AS\)](#) is recommended as IDE.
- Our AS version is 2.2.3, use embedded JDK.
- [AS Download Link](#)
- For more AS introduction, please refer to the [Google AS Training](#).

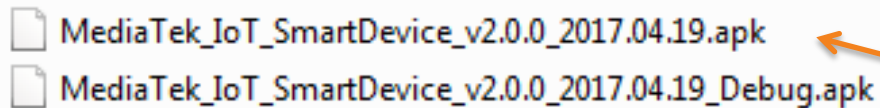
■ First Build

- 1. Tap “Open an existing AS project” in AS welcome UI.
- 2. Select “SmartDevice App folder” (unzip from *SourceCode.rar*).
- 3. Gradle View -> App/Tasks/Build -> assemble.
- For first build, it may take a long time due to Gradle sync, download dependencies and build release/debug APK/aar for all modules.

How to Build (2/2)

■ APP Module









- App: SmartDevice APP module
 - SmartDevice_APK\App\build\outputs\apk



MediaTek_IoT_SmartDevice_v2.0.0_2017.04.19.apk
MediaTek_IoT_SmartDevice_v2.0.0_2017.04.19_Debug.apk

Release Version

- Select Build Variants

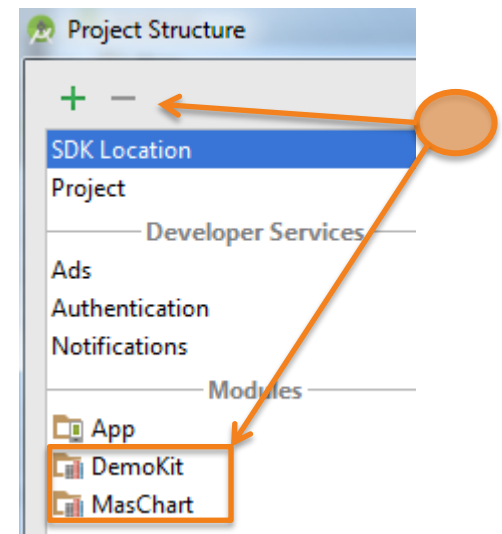
Build Variants			 
Module		Build Variant	
 App		debug	
 DemoKit		debug	
 MasChart		debug	

Non Health Production

■ Non Health Production

- If your production hasn't MT2511 chip, you should not need "health" modules.
- Slim for pure Bluetooth (No health chip) Production.
 - 1. AS Menu -> File -> Project Structure.
 - 2. Select "DemoKit", "MasChart", then click "—" to remove them.
 - 3. Remove "DemoKit" dependency from "APP build.gradle".
 - 4. Remove MT2511Controller and MainActivity "2511" related code.
 - 5. Build "App".

```
dependencies {  
    ... compile.project(':DemoKit')  
    ... compile.files('libs/wearable.jar')  
    ... compile 'com.android.support:appcompat-v7:25.2.0'  
}
```



Device Compatibility

■ Android Device Compatibility

- Our APPs must install in Android Phone with Bluetooth.

```
<uses-feature android:name="android.hardware.bluetooth" android:required="true" />
```

- SmartDevice App minSdkVersion is 16 (Jelly Bean 4.1).
- SmartDevice GATT Feature required SDK 18+ & Support BLE.
- Please refer to [Android Doc](#) to study more.

Customize APP (1/6)

■ ApplicationID

- Please be sure to modify ApplicationID in [App/Build.gradle](#) for your APP.

```
defaultConfig {  
    ... applicationId "com.mtk.btnotification"  
    ... minSdkVersion 16  
    ... targetSdkVersion 21  
    ...  
}
```

■ App Launcher Icon

- Please be sure to replace App launcher icon ([App/src/main/res/drawable-*/ic_launcher.png](#)) for your APP.



Customize APP (2/6)

■ APK Label

- Please be sure to modify APK label “*app_name*” string in *App\src\main\res\values-*\strings.xml*.

```
.<!-- App Label -->  
.<string name="app_name">MediaTek IoT SmartDevice</string>
```

■ Signature

- Please be sure to modify signature for APK release.
- Replace our *smartdevice.keystore* with your *keystore* and *config signing config* in APP signing folder.
- Please refer to *signing.gradle* and Android APP Sign Training.

```
android {  
    signingConfigs {  
        config {  
            keyAlias 'mtksmartdevice'  
            keyPassword 'mtk123456'  
            storeFile file('../smartdevice.keystore')  
            storePassword 'mtk123456'  
        }  
    }  
}
```

Customize APP (3/6)

■ Version

- Please modify APP version in [versioning.gradle](#).

```
ext {  
    ... VersionCode = {200;}  
    ... VersionName = {"2.0.0"};  
  
    ... buildDate = {new Date().format("yyyy.MM.dd", TimeZone.getTimeZone("UTC"))};  
}
```

■ APK Name

- Please modify APK name in [App/Build.gradle](#).

```
android.applicationVariants.all { variant ->  
    ... variant.outputs.each { output ->  
        ... def file = output.outputFile  
        ... if (variant.buildType.name.equals('release')) {  
            ... output.outputFile = new File(file.parent,  
                ... "MediaTek_IoT_SmartDevice_v" + versionName + "_" + buildDate() + ".apk")  
        } else {  
            ... output.outputFile = new File(file.parent,  
                ... "MediaTek_IoT_SmartDevice_v" + versionName + "_" + buildDate() + "_Debug.apk")  
        }  
    }  
}
```


Customize APP (4/6)

■ Customize GUI

- SmartDevice GUI only is used for demo, not your APP's final GUI solution.
- Please refer to Android training to customize your APP GUI.
- We only provide three strings: en (English), zh (Chinese Taiwan/Hongkong), zh-rCN (Simplified Chinese for China).
- If your production/APK will be used in other languages, please add related translation string (refer to link).

Customize APP (5/6)

- Compatible with your device
 - Your IoT device (based on MT2523/MT2533) could be only connect/work with your APK, not MTK demo APK or other manufacturer APK.
 - **Method 1: Add your Handshake Flow.**
 - After APK BTNotify connect and handshake successfully, your IoT device must receive an especial data from your APK, otherwise your IoT device should timeout and disconnect.

Customize APP (6/6)

■ Compatible with your device

– Method 2: Modify BTNotify SPP/GATT UUID.

– Your production is only connectable with your APK due to BT UUID.

– Customize APK BTNotify UUID: “App\src\main\res\xml\wearable_config.xml”

```
<!-- Device.BTNotify.SPP.UUID.-->
<string name="spp_uuid">0000FF01-0000-1000-8000-00805F9B34FF</string>
<!-- Device.BTNotify.GATT(DOGP).UUID.-->
<string name="dogp_uuid">000018A0-0000-1000-8000-00805F9B34FB</string>
<string name="dogp_read_uuid">00002AA0-0000-1000-8000-00805F9B34FB</string>
<string name="dogp_write_uuid">00002AA1-0000-1000-8000-00805F9B34FB</string>
```

– Device BTNotify SPP UUID: Such as

“SDK\project\mt2523_hdk\apps\fota_download_manager\src\bt_common.c”

```
#define BT_SPP_STANDARD_UUID 0x00, 0x00, 0xFF, 0x01, 0x00, 0x00, 0x10, 0x00, 0x80, 0x00, 0x00, 0x80, 0x5F, 0x9B, 0x34, 0xFF
```

Device BTNotify DOGP UUID: Such as

“SDK\middleware\MTK\bt_notify\src\dogp\ble_dogp_service.c”

```
#define DOGP_SERVICE_UUID 0x18A0
#define DOGP_READ_CHAR_UUID 0x2AA0
#define DOGP_WRITE_CHAR_UUID 0x2AA1
```

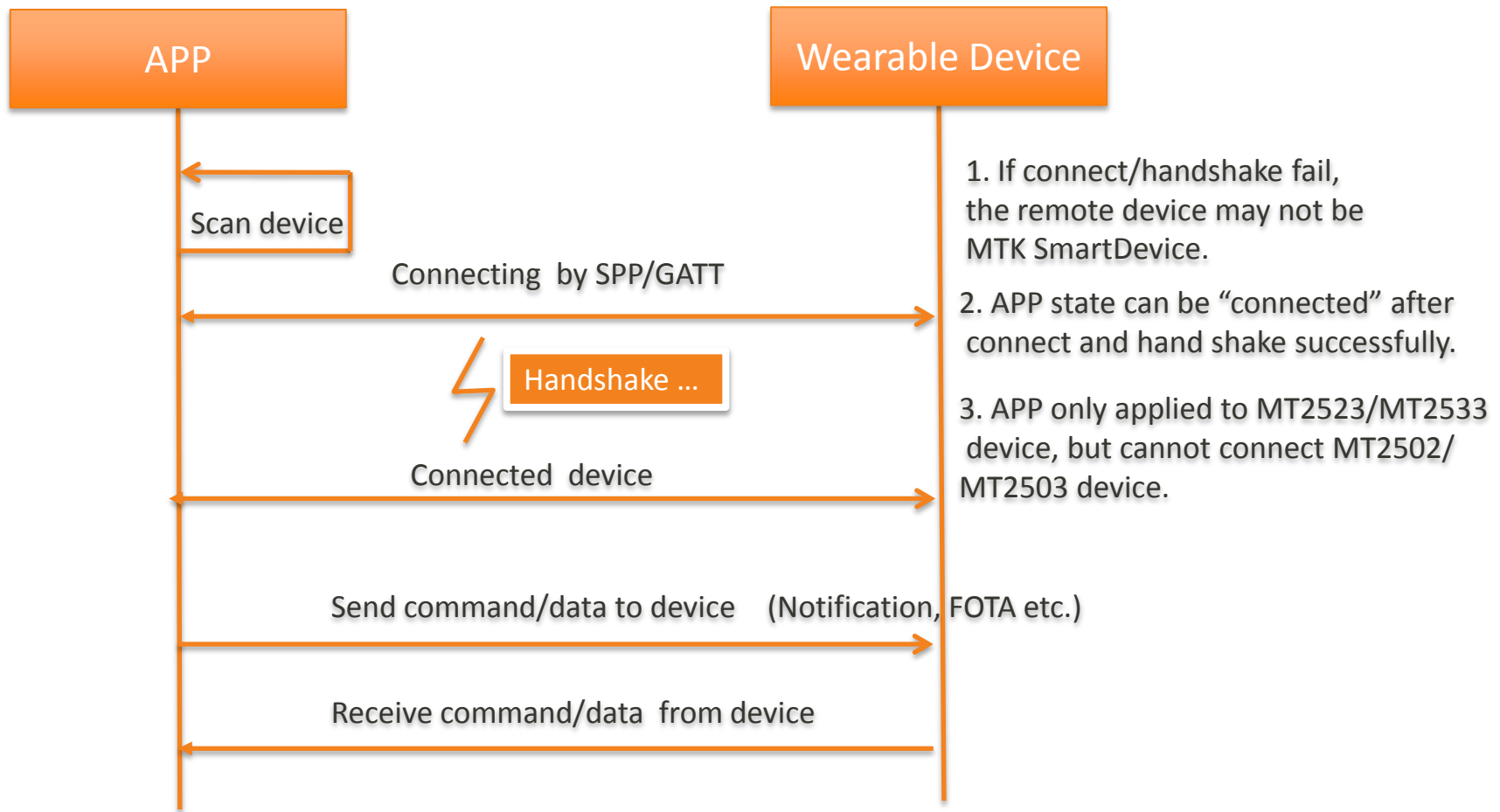
– Note: APK and Device BTNotify UUID must be **modified simultaneously**, otherwise APK may connect fail.

Wearable SDK

BTNotify (1/2)

- BTNotify
 - BTNotify: MTK BT Transport Protocol
 - Hand Shake: APK must hand shake to confirm that the remote device is MTK Smart Device (based on BTNotify) after connect BT device successfully.
 - **Wearable SDK** (wearable.jar): BTNotify Implement, provide API.
 - Two Mode
 - SPP (Based on BT SPP Profile)
 - GATT (i.e. DOGP mode, Based on MTK defined BLE GATT Profile - DOGP - Data over GATT Profile)

BTNotify (2/2)



How to Use (1/2)

■ IDE

- The [Android Studio \(AS\)](#) is recommended as SmartDevice application development tool.

■ Add Wearable SDK

- Put [wearable.jar](#) into libs folder under android module, then call library API.
- Must confirm your module [build.gradle](#) as follows:

```
dependencies {  
    ... compile fileTree(dir: 'libs', include: ['*.jar'])  
}
```
- For API reference, please refer to [JavaDoc](#) in release package.
- For sample code, please refer to [App](#) module code, such as MainActivity, DeviceScanActivity, FmpGattClient.

How to Use (2/2)

■ APP SDK Level

- SmartDevice.apk BTNotify(SPP) required [minSdkVersion 14](#).
- SmartDevice.apk BTNotify(GATT) required [minSdkVersion 18](#).
- Notification feature required [minSdkVersion 16](#).

■ Connection Mode

- When APP call Wearable SDK API, you don't need to consider whether the connection mode is SPP or GATT.

Wearable SDK (1/2)

- MediaTek Wearable SDK Features:
 - Connection
 - Controller
 - Notification Push
 - FOTA
 - Add Customized BLE Server/Client
 - Customized Wearable Parameter

Wearable SDK (2/2)

- The communication between Smart Phone (SP) and wearable, should includes 3 steps: **Scan**, **Connection**, **Data Transfer**.
- *WearableManager*
 - Used to init/scan/connect/disconnect wearable device and notify state/device change.
- *Controller*
 - Inherit this class and override send() & onReceive() API to implement send/receive command/data.

Connection (1/3)

- Init and register WearableListener
 - API: *init(true, connext, null, R.xml.wearable_config)*
 - Sample Code: *Wearable.java*
 - API: *registerWearableListener, unregisterWearableListener*
 - *WearableListener*

```
public interface WearableListener {  
    ... void onConnectChange(int oldState, int newState);  
    ... void onDeviceChange(BluetoothDevice device);  
    ... void onDeviceScan(BluetoothDevice device);  
    ... void onModeSwitch(int newMode);  
}
```

Connection (2/3)

■ Switch Mode

- API: *WearableManager switch (boolean enable)*
- Default Mode: GATT.
- APK will keep the last mode when SP or APK reboot.
- WearableManager will callback *onModeSwitch(mode)* to notify mode change.
- Sample Code: *MainActivity.java*

■ Scan BT Device

- API: *WearableManager scan (boolean enable)*
- Register *WearableListener* and implement *onDeviceScan()*.
- Sample Code: *DeviceScanActivity.java*

Connection (3/3)

■ Connect & Disconnect

- API: *WearableManager setRemoteDevice, connect, disconnect*
- WearableManager will callback *onDeviceChange(BluetoothDevice)* after call *setRemoteDevice* to set a remote device.
- Call *connect* method to establish the connection with wearable device.
- WearableManager will callback *onConnectChange(state)* to notify connection state.
- *Disconnect* method could disconnect SPP/GATT connection.
If the connection is disconnected due to call *Disconnect* method, auto-reconnect will be disable for this “disconnection” state.
- *getConnectState* method could return current connection state.
- *isAvailable* will return true after connect and shake hand successfully.
- Sample Code: *CustomPreference.java*

Controller (1/5)

■ Controller: Data Transfer

- Inherit this class and override *send* & *onReceive* API to implement send/receive command/data.
- *onReceive(byte[] dataBuffer)*
 - Need implement, receive and decode the command and data.
- *onConnectionStateChange(int state)*
 - Need implement, receive latest connection state .
- *Controller(String tag, int cmdType)*
 - The controller tag must be unique.
 - Must call *setReceiverTags* API with your *Receiver* string for wearable SDK to notify upper *controller* to receive your data.
 - The cmd Type must be *CMD_9* (EXCD).
- Sample Code: *DemoController.java* *MT2511Controller.java*

Controller (2/5)

■ Controller

- Must use *WearableManager addController* and *removeController* to add/remove your controller.
- Sample Code: *MainService.java*

```
.WearableManager manager = WearableManager.getInstance();  
.manager.addController(NotificationController.getInstance(sContext));  
.manager.addController(EpoDownloadController.getInstance());  
.///MT2511.Feature  
.if (FeatureConfig.isAPKSupport2511()) {  
.    .manager.addController(MT2511Controller.getInstance());  
.}  
.manager.addController(DemoController.getInstance());
```

Controller (3/4)

■ Extensible Command

- Data Format (Device -> SP APK)



- Data Format (SP APK -> Device)



cmd: must be CMD_9 (EXCD)

Sender/Receiver: APP ID, such as demo_sender, demo_receiver.

Action/Err: Action or Error code. Default value is 1. Recommend keep default value.

data_type: data type. Default value is 0. Recommend keep default value.

For more detail, please refer to [JavaDoc Controller](#) class, and [App\src\main\java\com\mtk\main\DemoController](#).

Controller (4/5)

■ DemoController

```
class DemoController extends Controller {

    private static final String DEMO_SENDER = "demo_sender";
    private static final String DEMO_RECEIVER = "demo_receiver";

    private DemoController() {
        // must add CMD_9 i.e. EXCD
        super("DemoController", CMD_9);
        HashSet<String> receivers = new HashSet<String>();
        // must set your receiver string
        receivers.add(DEMO_RECEIVER);
        super.setReceiverTags(receivers);
    }

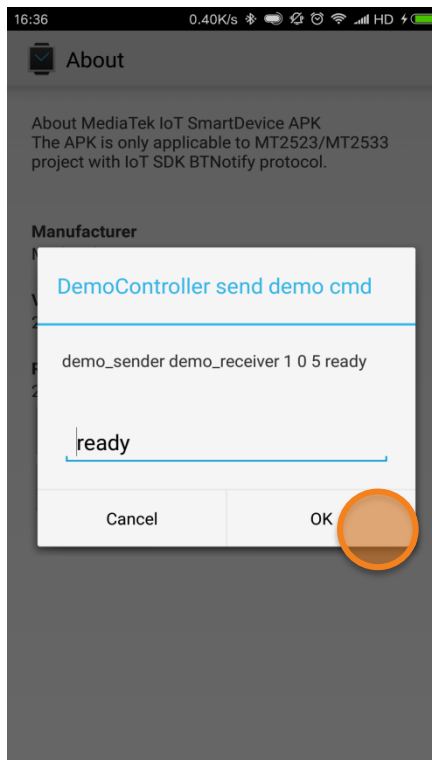
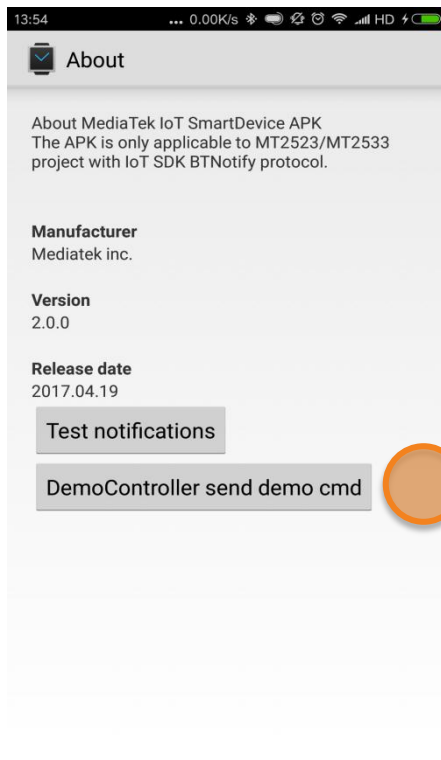
    public void sendDemoCmd(String data) {
        if (!WearableManager.getInstance().isAvailable() || data.isEmpty()) {
            Log.w(TAG, "sendDemoCmd return");
            return;
        }
        try {
            super.send(DEMO_SENDER, DEMO_RECEIVER, 1, data.getBytes(), false, PRIORITY_NORMAL);
        } catch (Exception e) {
            Log.e(TAG, "sendDemoCmd " + e.getMessage());
        }
    }

    @Override
    protected void onConnectionStateChange(int state) {
        Log.d(TAG, "onConnectionStateChange " + state);
    }

    @Override
    public void onReceive(byte[] dataBuffer) {
        Log.d(TAG, "onReceive " + new String(dataBuffer));
    }
}
```

Controller (5/5)

■ DemoController



Call *DemoController* *sendDemoCmd(String)* method.

SmartDevice APK send a demo command by using DemoController.

Notifications Push (1/2)

■ *NotificationController*

- Subclass of *Controller*, use this class to send notification.

- Send normal app notification API:

sendNotifications(String appld, CharSequence packageName, CharSequence tickerText, long when, String[] textList)

sendNotifications(NotificationData notificationData)

- Send message type notification API:

sendSmsMessage(String msgbody, String address)

- Send missed call type notification API:

sendCallMessage(String phoneNum, String sender, String content, int count)

- Send low battery type notification API:

sendLowBatteryMessage(String title, String content, String appld, String value)

Notifications Push (2/2)

■ *Notification Listen*

- To listen common application notification (Android SDK < 18), please refer to *NotificationReceiver*.
- To listen common application notification (Android SDK 18+), please refer to *NotificationReceiver18*.
- To listen Phone Call info and miss call state, please refer to *CallService*.
- To listen Phone battery status, please refer to *SystemNotificationService*.
- To listen New received SMS, please refer to *SmsService*.

FOTA (1/3)

■ *FotaOperator*

- Send command to remote device
- Send Firmware data to remote device
- Notify the result from remote device

■ *IFotaOperatorCallback*

- Receive data from remote device
- Receive status change

■ *FotaVersion*

- The version of the remote device
- Including SW version, module, brand, dev_id....

FOTA (2/3)

■ *FotaOperator*

- *registerFotaCallback*

- Register a *IFotaOperatorCallback* to monitor the state change and data receiving.

- *unregisterFotaCallback*

- Unregister the FotaCallback, then no any information will be received.

- *sendFotaTypeCheckCommand*

- Send type check command to remote device.
 - *IFotaOperatorCallback#onFotaTypeReceived* will be received.

FOTA (3/3)

■ *FotaOperator*

- *sendFotaVersionGetCommand*
 - Send command to get remote device version.
 - *IFotaOperatorCallback#onFotaVersionReceived* will be received.
- *sendFotaCustomerInfoGetCommand*
 - Send command to get remote device customized information.
 - *IFotaOperatorCallback#onCustomerInfoReceived* will be received.
- *sendFotaFirmwareData*
 - Send firmware file to remote device according to the file path or file URI.
 - While sending the file, *IFotaOperatorCallback#onProgress* will notify the sending progress.

Add customized BLE Servers

■ Flow

- 1. Implement one or more *LeServers*
 - The *LeServer* interface include 3 APIs.
 - *List<BluetoothGattService> getHardCodeProfileServices*
 - You can prepare your GATT services in this method. The services should be completed, include necessary characteristics and descriptors.
 - *BluetoothGattServerCallback getGattServerCallback*
 - LeServer should implement a BluetoothGattServerCallback, when GATT server events coming, the methods in the callback will be called
 - *setBluetoothGattServer(BluetoothGattServer server)*
 - If your LeServer need use BluetoothGattServer, you may need save this instance.
- 2. Call *LeServerManager.addLeServers()* method to register your *LeServers*
- Sample Code: *FmpGattServer.java*

Add customized BLE Clients

■ Flow

- 1. Extends one or more *WearableClientProfile*
 - This class extends *BluetoothGattCallback*, so you can override some callback method.
 - If you want to receive the callback about a characteristic or descriptor, you should use *addUuids()* to mark the UUIDs of the char/desc.
 - If you want to receive the callback of readRssi, you should use *enableRssi* to mark it.
- 2. After you prepared your *WearableClientProfile*, call *WearableClientProfileManager.registerWearableClientProfile()* to register it.
- 3. Suggest calling *GattRequestManager.getInstance().readCharacteristic*, *writeCharacteristic*, *readDescriptor*, *writeDescriptor* instead of calling these functions in BluetoothGatt.
- Sample Code: *FmpGattClient.java*

Customized Wearable Parameter

■ WearableConfig

- User could configure wearable SDK parameter by modify *wearable_config.xml*.
- The *wearable_config.xml* will be used with *WearableManager init* method when the APP process start.
- “App\src\main\res\xml\wearable_config.xml”
- For more parameter info, please refer to comment in the *wearable_config.xml*.

Contact US

(If you have any questions, comments, or suggestions, please contact us by MTK ACS, or send mail to SmartDevice_App@mediatek.com)



everyday genius