Application Note : Development Manual for Telink BLE Mesh Android SDK

AN-17071702-E1

Ver 1.0.0

2017/7/18

Brief:

This document is the guide for Telink BLE Mesh Android SDK development.

TELINK SEMICONDUCTOR

Published by
Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,
Zhangjiang Hi-Tech Park, Shanghai, China

Legal Disclaimer

Telink Semiconductor reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein or in any other disclosure relating to any product.

Telink Semiconductor does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others

The products shown herein are not designed for use in medical, life-saving, or life-sustaining applications. Customers using or selling Telink Semiconductor products not expressly indicated for use in such applications do so entirely at their own risk and agree to fully indemnify Telink Semiconductor for any damages arising or resulting from such use or sale.

Information:

For further information on the technology, product and business term, please contact Telink Semiconductor Company (www.telink-semi.com).

For sales or technical support, please send email to the address of:

telinkcnsales@telink-semi.com

telinkcnsupport@telink-semi.com

Revision History

Version	Major Changes	Date	Author
1.0.0	Initial release	2017/7	KeChangWei, Cynthia

Table of contents

1 Brief Introduction

TelinkLightSDK is Android SDK solution based on Telink BLE Mesh Light protocol. It mainly implements light device scanning, adding specific device into Mesh network, automatic reconnection, as well as light control function.

2 Project Configuration

1) Create a project and import “libTelinkLight.aar”.
2) Define Service components, and extend from “LightService”.
3) Define Application, and extend from “TelinkApplication”.
4) Configure “AndroidManifest.xml”.

```
<?xml version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.telink.bluetooth.light">

<!--蓝牙权限 Bluetooth permission -->
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-feature
android:name="android.hardware.bluetooth_le"
android:required="true"/>

<application
android:name=".TelinkLightApplication"
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme">
<service
android:name=".TelinkLightService"
android:enabled="true"/>
</manifest>
```

3 Development Guide

1) Set manufacturer info:

"Manufacture" serves to set manufacturer information, including manufacturer name and description, as well as factory default name, password, LTK (Long Term Key) and VendorId of device. Default manufacturer is "Telink".

Sample:

```
Manufacture.Builder builder = new Manufacture.Builder();
builder.setFactoryName(factoryName);
builder.setFactoryPassword(factoryPwd);
Manufacture manufacture = builder.build();
Manufacture.setManufacture(manufacture);
```

2) Scan and add light node:

Invoke the "startScan" method of "lightService" to implement light device scanning, and invoke the "updateMesh" of "lightService" to add the specified device into new Mesh network.

User needs to listen for related events including LeScanEvent, DeviceEvent and MeshEvent.

Sample:

a) Event listening

```
this.mApp = SmartLightApplication.getInstance();
this.mApp.addEventListener(LeScanEvent.LE_SCAN, this.mListener);
this.mApp.addEventListener(LeScanEvent.LE_SCAN_TIMEOUT, this.mListener);
this.mApp.addEventListener(DeviceEvent.STATUS_CHANGED, this.mListener);
this.mApp.addEventListener(MeshEvent.ERROR, this.mListener);
```

b) Scanning

```
private void startScan() {
    LeScanParameters params = LeScanParameters.create();
params.setMeshName(Manufacture.getDefault().getFactoryName());
params.setOutOfMeshName("out_of_mesh");
params.setTimeoutSeconds(5);
params.setScanMode(true);
SmartLightService.getInstance().startScan(params);
}
```

c) Light adding

```
private void startUpdate(DeviceInfo deviceInfo) {
    LeUpdateParameters params = LeUpdateParameters.create();
params.setOldMeshName(Manufacture.getDefault().getFactoryName());
params.setOldPassword(Manufacture.getDefault().getFactoryPassword());
params.setNewMeshName(this.mApp.getCurrentMesh().getMeshName());
params.setNewPassword(this.mApp.getCurrentMesh().getPassword());
params.setLtk(this.mApp.getCurrentMesh().getLtk());
params.setUpdateDeviceList(deviceInfo);
SmartLightService.getInstance().idleMode(true);
SmartLightService.getInstance().updateMesh(params);
}
```

3) Automatic reconnection:

Invoke the “autoConnect” method of “LigthService” to scan current network and login certain light device. Generally during automatic reconnection, network auto refresh mechanism should be enabled by invoking the “autoRefreshNotify” method. To disable network auto refresh, the “disableAutoRefreshNotify” method should be invoked correspondingly.

After login, if network auto refresh mechanism is enabled, device status in current network (i.e. “OnlineStatus”) will be returned. Listen for the “NotificationEvent#ONLINE_STATUS” event and handle device status.

Sample:

```
private void autoConnect() {
SmartLightService service = SmartLightService.getInstance();

if (service.getMode() != LightAdapter.MODE_AUTO_CONNECT_MESH) {
        Mesh mesh = this.mApp.getCurrentMesh();
LeAutoConnectParameters params = LeAutoConnectParameters.create();
params.setMeshName(mesh.getMeshName());
params.setPassword(mesh.getPassword());
params.autoEnableNotification(true);
service.autoConnect(params);
}
```

```
    LeRefreshNotifyParameters refreshNotifyParams =
LeRefreshNotifyParameters.create();
refreshNotifyParams.setRefreshRepeatCount(2);
refreshNotifyParams.setRefreshInterval(1000);
service.autoRefreshNotify(refreshNotifyParams);
}
```

4) Light device control:

User can control status of single device or device group in the Mesh network, e.g. switch on/off device, adjust color, and etc.

The "LightService" supplies two methods to send a control command, including "sendCommand" and "sendCommandNoResponse". It's recommended to adopt the second method.

Sample:

```
SmartLightService.Instance().sendCommandNoResponse((byte) 0xD0, 0xFFFF,
        new byte[]{0x01, 0x00, 0x00});
```

5) OTA/MeshOTA

OTA for single light: Invoke the "LightService#startScan" method to implement scanning. After device is discovered, establish connection via "connect", and invoke the "getFirmwareVersion" to obtain version information of current firmware. Finally invoke "startOta" to implement firmware upgrade.

In new OTA procedure, MeshOTA related function is added, i.e. all online devices with old firmware in the Mesh network can batch upgrade firmware via advertising upgrade packet data by device with newer firmware.

Before OTA is started, firmware version information of all devices in current Mesh network will be obtained.

```
byte opcode = (byte) 0xC7;
int address = 0xFFFF;
byte[] params = new byte[]{0x20, 0x00};
TelinkLightService.Instance().sendCommandNoResponse(opcode, address,
        params);
```

In the “NotificationEvent”, first firmware version information of device will be available via analysis.

To ensure currently connected device can handle OTA operation correctly, OTA status information of the directly connected device will be obtained before upgrade is started.

```
byte opcode = (byte) 0xC7;
int address = 0x0000;
byte[] params = new byte[]{0x20, 0x05};
TelinkLightService.Instance().sendCommandNoResponse(opcode, address,
        params);
```

In the “NotificationEvent”, data will also be analyzed.

If no device has newer firmware version in the device list, it will first establish connection with any device, and start OTA for single light. After the single-light OTA is finished, MeshOTA start command will be sent. It’s only needed to listen for MeshOTA progress information.

6) Encryption/Decryption

Data encryption and decryption interface (com.telink.crypto. AES) has already been assembled in the SDK. The two methods below correspond to encryption and decryption, respectively.

```
public static byte[] encrypt(byte[] key, byte[] nonce, byte[] plaintext) {

    if (!AES.Security)
        return plaintext;

    return encryptCmd(plaintext, nonce, key);
}

public static byte[] decrypt(byte[] key, byte[] nonce, byte[] plaintext) {

    if (!AES.Security)
        return plaintext;

    return decryptCmd(plaintext, nonce, key);
}
```