



# ChipletRing APP SDK manual

## Android

Document Version number:	1.3.5	Document No. :	
Document secret level:		Department:	Chiplet Research and Development Department
Product Name:	BCL_603M	Attribution Items:	
Writer:	Zheng Yuhu	Date written:	2023.10.19

**Wuxi Yongxin technology Co., Ltd.**

## nullRevision Record:

Version Number	Revision person	Date of revision	Revised Description
1.0.0	Zheng Yuhu	2023.10.19	Modifying Navigation Directory
1.0.1	Zheng Yuhu	2023.10.26	Modify the description of integration and initialization SDK, modify part of the demo code
1.0.2	Zheng Yuhu	2023.10.31	Complete code, add FAQ
1.0.3	Zheng Yuhu	2023.11.23	Updated the Q&A, modified the parameter description for reading data
1.0.4	Zheng Yuhu	2023.11.29	Added Q&A as a summary for follow-up simple questions
1.1.1	Zheng Yuhu	2023.12.26	1. Added screening method for Bluetooth scanning 2. Measure heart rate, measure blood oxygen increase upload data
1.1.2	Zheng Yuhu	2024.01.02	Added waveform graph data analysis
1.1.3	Zheng Yuhu	2024.04.19	Increase heart rate, blood oxygen, PPG raw data analysis
1.3	Zheng Yuhu	2024.05.20	Added a detailed description of the collection cycle
1.3.1	Zheng Yuhu	2024.06.12	A detailed description of the number of steps has been added
1.3.2	Zheng Yuhu	2024.07.09	Added sleep logic flowchart
1.3.3	Zheng Yuhu	2024.07.17	Added interface to blood pressure algorithm
1.3.4	Zheng Yuhu	2024.08.15	Modify the description of the synchronization time and read time
1.3.5	Zheng Yuhu	2024.11.04	Add the interface to set and read Bluetooth name, real-time blood pressure, and add functions related to long connection

## Catalogue

<b>ChipletRing APP SDK manual .....</b>	<b>1</b>
<b>I、 Document Brief .....</b>	<b>4</b>
1、 Document purpose .....	4
2、 Scope of application .....	4
3、 Function Description .....	4
<b>II、 Quick Start Overview .....</b>	<b>5</b>
1、 Preconditions .....	5
2、 Use process .....	5
3、 Flowchart .....	6
<b>III、 Integration with the ChipletRing APP SDK .....</b>	<b>7</b>
1、 Integrate the ChipletRing APP SDK .....	7
2、 Initialize the ChipletRing APP SDK .....	8
3、 Use the ChipletRing APP SDK .....	9
<b>IV、 Other .....</b>	<b>29</b>
1、 Filter Related .....	29
2、 Problems that may be encountered .....	31
3、 Hardware algorithm logic or firmware related .....	33
4、 Q&A .....	34

# I、Document Brief

## 1、 Document purpose

In order to facilitate the secondary development of the communication between the Android APP and the ring, the communication protocol is specially encapsulated to achieve simplicity and clarity, so that developers do not need to pay attention to the communication layer with the ring, and focus on the development of business logic interaction level.

## 2、 Scope of application

This SDK is based on Android native development, and is ultimately provided as a jar package library. It can be used in the Android environment.

## 3、 Function Description

Function module	Instructions	Relevant documentation
Bluetooth basic module	<ul style="list-style-type: none"> <li>● Switch operation of Bluetooth</li> <li>● Bluetooth search link operation</li> <li>● Bluetooth data write listening operation</li> </ul>	
Communication protocol module	<ul style="list-style-type: none"> <li>● Time management</li> <li>● Version number management</li> <li>● Battery Management</li> <li>● Heart rate measurement</li> <li>● Blood oxygen measurement</li> <li>● Temperature measurement</li> <li>● Step management</li> <li>● History management</li> <li>● System Settings</li> <li>● Log management</li> </ul>	

## II、 Quick Start Overview

### 1、 Preconditions

- The operating system is Android and the operating system version is  $\geq 5.0$
- Must support Bluetooth 5.0
- The jar package can be invoked using the language

### 2、 Use process

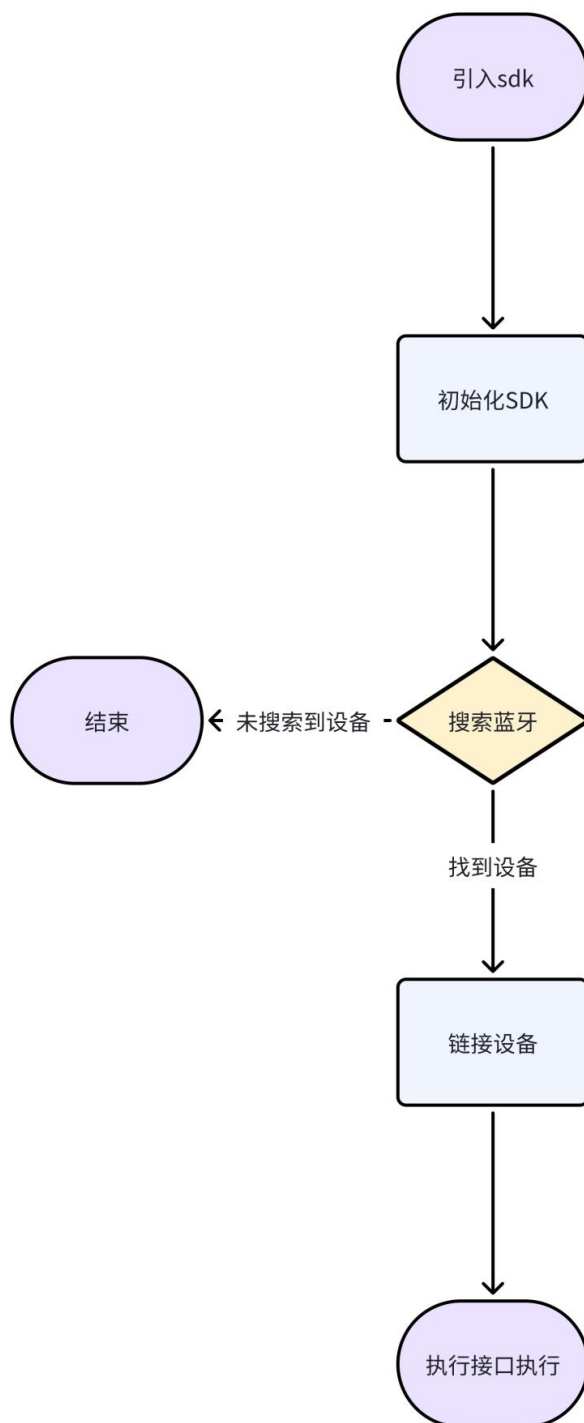
Follow the following process to use the SDK:

Step 1: Integrate the SDK

Step 2: Initialize the SDK

Step 3: Use the SDK

### 3、 Flowchart



### III、 Integration with the ChipletRing APP SDK

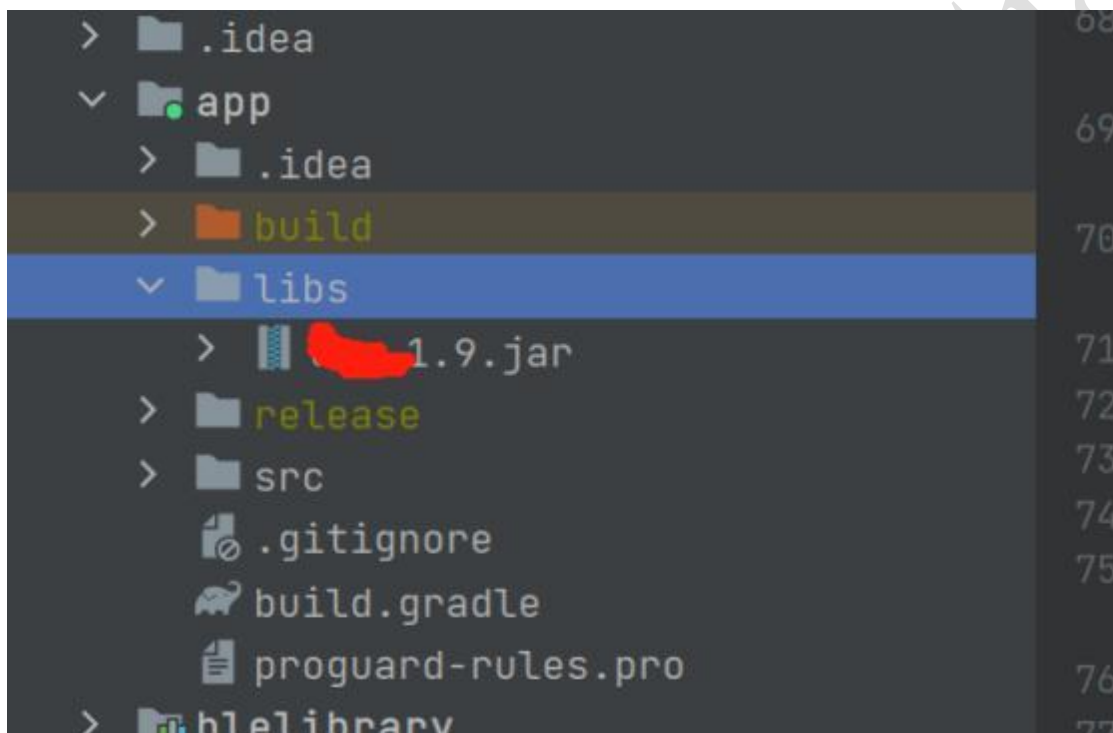
#### 1、 Integrate the ChipletRing APP SDK

##### 1.1 Integration method

##### 1.1.1 Get the jar package file of ChipletRing APP SDK

##### 1.1.2 Place the jar package in the libs directory

##### 1.1.3 Right click and set to Add as Library(add as class library)



##### 1.1.4 Configure the required permissions, such as storage and other permissions can be configured by yourself, involving dynamic permissions, need to do related processing

Add the following code to the Manifest.xml

```
1. <uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
2. <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
3. <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
4. <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
5. <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

## 2、 Initialize the ChippetRing APP SDK

### 2.1.1 Initialize in the onCreate method of the Application

```
1. LmAPI.init(this);
2. LmAPI.setDebug(true);
```

### 2.1.2 In the BaseActivity class, enable listening, which is used to listen for the Bluetooth connection status and the data fed back by the ring's basic instructions

```
1. LmAPI.addWLSCmdListener(this, this);
2. // Monitor the status of the connection between the Bluetooth device and the
   APP
3. IntentFilter intentFilter = new IntentFilter();
4. intentFilter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
5. intentFilter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
6. intentFilter.addAction(BluetoothDevice.ACTION_ACL_CONNECTED);
7. registerReceiver(broadcastReceiver, intentFilter);
8.
9. // Ask for permission to disable Bluetooth before using it
10. if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
11.     if (!checkPermissions(new String[]{Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.BLUETOOTH_SCAN, Manifest.permission.BLUETOOTH_CONNECT, Manifest.permission.BLUETOOTH_ADVERTISE})) {
12.         new XPopup.Builder(this).asConfirm(getRsString(R.string.hint), getString(R.string.location_auth),
13.             new OnConfirmListener() {
14.                 @Override
15.                 public void onConfirm() {
16.                     requestPermission(new String[]{Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.BLUETOOTH_SCAN, Manifest.permission.BLUETOOTH_CONNECT, Manifest.permission.BLUETOOTH_ADVERTISE}, 100);
17.                 }
18.             }).show();
19.         return;
20.     }
21.
22. } else {
23.     if (!checkPermissions(new String[]{Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_FINE_LOCATION})) {
24.         new XPopup.Builder(this).asConfirm(getRsString(R.string.hint), getString
```



```

(R.string.localtion_auth),
25.     new OnConfirmListener() {
26.         @Override
27.         public void onConfirm() {
28.             requestPermission(new String[]{Manifest.permission.ACCESS_COARSE
                _LOCATION, Manifest.permission.A CCESS_FINE_LOCATION}, 100);
29. }
30.     }).show();
31.     return;
32. }
33. }

```

### 3、 Use the ChipletRing APP SDK

#### 3.1 Bluetooth Operation (BLEUtils)

Bleutils is a public class that uses Bluetooth to search, connect, disconnect, and is fed back by the IResponseListener interface.

##### 3.1.1 Search for devices

Interface: Enable the Bluetooth search function to search for nearby Bluetooth devices.

Interface declaration:

```

1. BLEUtils.startLeScan(Context context, BluetoothAdapter.LeScanCallback leScan
    Callback);

```

Parameter description: context: context leScanCallback: callback for Bluetooth search

Return value (void onLeScan(BluetoothDevice device, intrssi, byte[] bytes))

The return value of this interface is described as follows:

```

1. private BluetoothAdapter.LeScanCallback leScanCallback = new BluetoothAdapter
    .LeScanCallback() {
2.
3.     @Override
4.     public void onLeScan(BluetoothDevice device, int rssi, byte[] bytes) {
5.         // Process the searched device
6.     }
7. };

```

Precautions: 1. Ensure that the Bluetooth device has power

2. If you want to screen the Bluetooth device (manufacturer ID == 0xFF01), please refer to four, other, screening related
3. The demo contains an example of filtering persistent connections

### 3.1.2 Stop the search

Interface function: Disable Bluetooth search function.

Interface declaration:

```
1. BLEUtils.stopLeScan(Context context, BluetoothAdapter.LeScanCallback leScanCallback);
```

Note: When calling this interface, ensure that it is connected to the ring

Parameter Description: context: context leScanCallback: callback of Bluetooth search

Returned value: None

### 3.1.3 Connected device

Interface function: Initiate connection to Bluetooth device.

Interface declaration:

```
1. BLEUtils.connectLockByBLE(Context context, BluetoothDevice bluetoothDevice);
```

Note: 1. When calling this interface, ensure that it is connected to the ring

2. When scanning the connection, it will automatically determine whether it is a persistent connection, and when you reconnect after a long connection, you need to manually enter the BOOL value to confirm whether it is reconnected (there is an example in the demo)

Parameter description: context: context bluetoothDevice: Bluetooth device

Returned value:

```
1. @Override
2. public void lmBleConnecting(int code) {
3. // Connecting now
4.
5. }
6.
7. @Override
8. public void lmBleConnectionSucceeded(int code) {
9. // The connection succeeded
```

```

10. }
11.
12. @Override
13. public void lmBleConnectionFailed(int code) {
14. // Connection failed
15. }

```

### 3.1.4 Disconnect Bluetooth

Interface function: Disconnect the device.

Interface declaration:

```

1. BLEUtils.disconnectBLE(Context context);

```

Note: When calling this interface, ensure that it is connected to the ring

Parameter description: context: context

Returned value: None

## 3.2 Instruction function (LmAPI)

This class is a public class that uses the ring function. The ring function can be called directly through this class. The data feedback is fed back by the IResponseListener interface except for measurement and history

### 3.2.1 Synchronizing time

Interface function: synchronize time, call this interface, will get the phone's current time synchronization to the ring, keep time synchronization.

Interface declaration:

```

1. LmAPI.SYNC_TIME();

```

Note: The synchronization time and the read time share a return value. When calling this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value (void syncTime(byte datum, byte[] time))

Parameter name	Type	Sample Values	Instructions
----------------	------	---------------	--------------

datum	byte	0 or 1	0 indicates that the synchronization is successful 1 represents the read time
time	byte[]	null	Sync time does not return byte[]

### 3.2.2 Read time

Interface function: Read time, call this interface, will get the ring current time.

Interface declaration:

Note: The synchronization time and the read time share a return value. When calling this interface, ensure that it is connected to the ring

Parameter Description: None

```
1. LmAPI.READ_TIME();
```

Return value (void syncTime(byte datum, byte[] time))

Parameter name	Type	Sample Values	Instructions
datum	byte	0 or 1	0 indicates that the synchronization is successful 1 represents the read time
time	byte[]	[48, -23, -1, 83, -111, 1, 0, 0, 8] = 1723691166000	If the read time is successful, it needs to be converted to a timestamp

### 3.2.3 Version information

Interface function: Version information, get the ring version information.

Interface declaration:

```
1. LmAPI.GET_VERSION((byte) 0x00); // 0x00 gets the software version, 0x01 gets the hardware version
```

Note: When invoking this interface, ensure that it is connected to the ring

Parameter Description: type: 0x00 gets the software version, 0x01 gets the hardware version

Return value (void VERSION(byte type, String version))

Parameter name	Type	Sample Values	Instructions
type	byte	0 or 1	<ul style="list-style-type: none"> <li>● 0 for software</li> <li>● 1 for hardware</li> </ul>
version	String	1.0.0.1	<ul style="list-style-type: none"> <li>● Version number</li> </ul>

### 3.2.4 Battery level

Interface function: Obtain battery power and battery status.

Interface declaration:

```
1. LmAPI.GET_BATTERY((byte) 0x00); //0x00 gets the charge and 0x01 gets the charging status
```

Note: When calling this interface, ensure that it is connected to the ring

Parameter Description: type: 0x00 to obtain the power, 0x01 to obtain the charging state

Return value (void battery(byte status, byte datum))

Parameter name	Type	Sample Values	Instructions
status	byte	0 or 1	<ul style="list-style-type: none"> <li>● 0 represents battery power</li> <li>● 1 is the state of charge</li> </ul>
datum	byte		<ul style="list-style-type: none"> <li>● Power</li> </ul>

### 3.2.5 Reading steps

Function This interface is used to obtain the total number of steps for the day.

Interface declaration:

```
1. LmAPI.STEP_COUNTING ()
```

Note: When invoking this interface, ensure that it is connected to rings

Parameter Description: None

Return value (void stepCount(byte[] bytes, byte subCmd))

Parameter name	Type	Sample Values	Instructions
bytes	byte[]	3303	<ul style="list-style-type: none"> <li>● Step count 819</li> </ul>
subCmd	byte	0, 1	<ul style="list-style-type: none"> <li>● 0 steps</li> <li>● 1 Clear Steps successfully</li> </ul>

### 3.2.6 Number of cleared steps

Interface Function: Clear steps.

Interface declaration:

```
1. LmAPI.CLEAR_COUNTING ()
```

Note: When invoking this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value: Please refer to the previous article

### 3.2.7 factory data reset.

Interface function: Restore factory Settings

Interface declaration:

```
1. LmAPI.RESET ()
```

Note: When calling this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value: None, if the reset method is used, it is considered successful

### 3.2.8 Collection period setting

Interface function: collection period setting

Interface declaration:

```
1. LmAPI.SET_COLLECTION (collection) // Collection period, in seconds
```

Note: When calling this interface, ensure that it is connected to the ring

Parameter Description: collection: Collection interval, in seconds

Return value: (void collection(byte[] bytes, byte subCmd))

The name of the parameter	type	Example values	illustrate
bytes	byte[]	b0040000	Acquisition interval in seconds as: 1200s
subCmd	byte	0 or 1	0 success 1 failed

### 3.2.9 Collection cycle read

Interface function: collection period read

Interface declaration:

```
1. LmAPI.GET_COLLECTION () // Collection period, in seconds
```

Note: When invoking this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value: (void collectionResult(int time))

Parameter name	Type	Sample Values	Instructions
time	int	60	● Collection interval, in seconds

### 3.2.10 Measuring heart rate

Interface function: Measure heart rate.

Interface declaration:

```
1. LmAPI.GET_HEART_ROT (byte waveForm,IHeartListener iHeartListener)
```

Note: When calling this interface, ensure that the ring was connected

Parameter Description:

waveForm: Configured or not waveform 0 No upload 1 Upload

iHeartListener: This interface was used to listen to the measurement data

Return value:

```
1. LmAPI.GET_HEART_ROTATE(new IHeartListener() {
2.     @Override
3.     public void progress(int progress) {
4.         setMessage(" Heart rate being measured..." + String.format
5.         ("%02d%", progress));
6.     }
7.
8.     @Override
9.     public void resultData(int heart, int heartRota, int yaLi, int temp) {
10.         if (colorFragment != null) {
11.             colorFragment.heartAndRota(heart, heartRota, yaLi, temp);
12.         }
13.     }
14.     @Override
15.     public void waveformData(byte seq, byte number, String waveData) {
16.         // Heart rate returns to the waveform data analysis: waveData
17.     }
18.
19.     @Override
20.     public void error(int value) {
21.         switch (value) {
22.             case 0:
23.                 dismissProgressDialog();
24.                 ToastUtils.show(" not worn ");
25.                 break;
26.             case 2:
27.                 dismissProgressDialog();
28.                 ToastUtils.show(" No collection allowed on charge ");
29.                 break;
30.             case 4:
31.                 dismissProgressDialog();
32.                 ToastUtils.show(" busy, not executing ");
33.                 break;
34.             default:
35.                 break;
36.         }
37.     }
}
```



```
38.  
39.  @Override  
40.  public void success() {  
41.      dismissProgressDialog();  
42.  }  
43. });  
44.
```

### 3.2.11 Measuring blood oxygen

Interface function: Measure blood oxygen.

Interface declaration:

```
1. LmAPI.GET_HEART_Q2 (byte waveForm,IQ2Listener iq2Listener)
```

Note: When calling this interface, ensure that it was connected to the ring

Parameter Description:

waveForm: Configured or not waveform 0 No upload 1 Upload

IQ2Listener: This interface was used to listen to the measurement data

Return value:

```
1. LmAPI.GET_HEART_Q2(new IQ2Listener() {  
2.  @Override  
3.  public void progress(int progress) {  
4.      setMessage(" Blood oxygen being  
        measured..." + String.format("%02d%", progress));  
5.  }  
6.  
7.  @Override  
8.  public void resultData(int heart, int q2, int temp) {  
9.      if (colorFragment != null) {  
10.         colorFragment.updateData(heart, q2, temp);  
11.     }  
12. }  
13.  
14. @Override  
15. public void waveformData (byte seq, byte number, String waveData) {  
16.     // Analysis of blood oxygen return waveform data: waveData  
17. }  
18. @Override  
19. public void error(int value) {  
20.     switch (value) {
```

```
21.     case 0:
22.         dismissProgressDialog();
23. ToastUtils.show(" not worn ");
24.         break;
25.     case 2:
26.         dismissProgressDialog();
27. ToastUtils.show(" No collection allowed on charge ");
28.         break;
29.     case 4:
30.         dismissProgressDialog();
31. ToastUtils.show(" busy, not executing ");
32.         break;
33.     default:
34.         break;
35. }
36. }
37.
38. @Override
39. public void success() {
40.     dismissProgressDialog();
41. }
42. });
```

### 3.2.12 Take the temperature

Interface function: Measure temperature.

Interface declaration:

```
1. LmAPI.GET_HEART_Q2 (IQ2Listener iQ2Listener)
```

Note: When invoking this interface, ensure that it is connected to the ring

Parameter Description: IQ2Listener: This interface is to listen to the measurement data

Return value: Ibid. Temperature is also returned when blood oxygen is measured

### 3.2.13 History Management

Interface function: Read history.

Interface declaration:

```
1. LmAPI.READ_HISTORY (int type,IHistoryListener iHistoryListener)
```

Note: When invoking this interface, ensure that it is connected to the ring

Parameter Description: type: 1, get all history;0: get the history that has not been uploaded

Return value:

```
1. LmAPI.READ_HISTORY(type, new IHistoryListener() {
2.     @Override
3.     public void error(int code) {
4.         handler.removeMessages(0x99);
5.         dismissProgressDialog();
6.         switch (code) {
7.             case 0:
8.                 ToastUtils.show(" measuring, please try again later ");
9.                 break;
10.            case 1:
11.                ToastUtils. Show (" historical record is uploaded, please try again later ");
12.                break;
13.            case 2:
14.                ToastUtils. Show (" deleting history records, please try again later ");
15.                break;
16.            default:
17.                break;
18.        }
19.    }
20.
21.    @Override
22.    public void success() {
23.        // synchronization is complete
24.    }
25.
26.    @Override
27.    public void progress(double progress, com.lm.sdk.mode.HistoryDataBean dataBean) {
28.        // Work with historical data
29.    }
30. });
```

### 3.2.14 Clear historical data

This interface is used to clear historical data.

Interface declaration:

```
1. LmAPI.CLEAN_HISTORY ()
```

Note: To invoke this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value: None

### 3.2.15 Blood pressure test algorithm

This interface is used to clear historical data.

Interface declaration:

```
1. LmAPI.GET_BPwaveData()
```

Note: Ring firmware must be supported, otherwise it cannot be used. To invoke this interface, ensure that it is connected to the ring

Parameter Description: None

Returned value: (byte seq,byte number,String waveDate)

Parameter name	type	Example value	Instructions
seq	byte	0	● Number 0
number	byte	10	● There are 10 pieces of data
waveDate	String	green/:14289393 ir/:10108995 cur_green/:4704 cur_ir/:4704	● None

### 3.2.16 Real-time PPG blood pressure measurement

Interface functions: Real-time measurement of blood pressure values and raw waveforms at 500Hz

Interface declarations:

```
1. LmAPI.GET_REAL_TIME_BP (byte time,byte isWave,byte  
isProgress,IRealTimePPGBpListener iRealTimePPGBpListener)
```

Precautions: The ring firmware must support it, otherwise it cannot be used. Call this API,

It must be connected to the ring

Parameter description:

time: Acquisition time, byte type, default 30s

isWave: Whether to upload a waveform. 0: Not uploaded, 1: Uploaded

isProgress: Whether or not to upload progress. 0: Not uploaded, 1: Uploaded

```

1. LmAPI.GET_REAL_TIME_BP((byte) 0x30, (byte) 1, (byte) 1, new IRealTimePPGBpL
   istener() {
2.             @Override
3.             public void progress(int progress) {
4.                 //progress
5.             }
6.
7.             @Override
8.             public void bpResult(byte type) {
9.                 //[0]:Diastolic blood pressure
10.                //[1]:Systolic blood pressure
11.            }
12.
13.            @Override
14.            public void resultData(String bpData) {
15.                //bpData(Contains infrared values)
16.            }
17.        });
  
```

### 3.2.17 Real-time PPG blood pressure stop collection

Interface functions: Stop the acquisition

Interface declarations:

```

1. STOP_REAL_TIME_BP()
  
```

Parameter description: None

Callback:

```

1. @Override
2. public void stopRealTimeBP(byte isSend) {
3.     if(isSend == (byte)0x01){
4.         Logger.show("TAG", "Stop Acquisition Sent");
5.     }
6. }
  
```

### 3.2.18 Set the Bluetooth name

Interface functions: Set the Bluetooth name

Interface declarations:

### 1. Set\_BlueTooth\_Name(String name)

Parameter description:

Name:Bluetooth name, no more than 12 bytes, can be Chinese, English, numbers, that is, 4 Chinese characters or 12 English

Note: After setting the Bluetooth name, the broadcast will not change immediately, and you need to wait for a while

Callback:

```
1. @Override
1.     public void setBlueToolName(byte data) {
2.         if(data == (byte)0x00){
3.             Logger.show("TAG","The setup failed");
4.         }else if(data == (byte)0x01){
5.             Logger.show("TAG","The setup success");
6.         }
7.     }
```

### 3.2.19 Get the Bluetooth name

Interface functions: Set the Bluetooth name

Interface declarations:

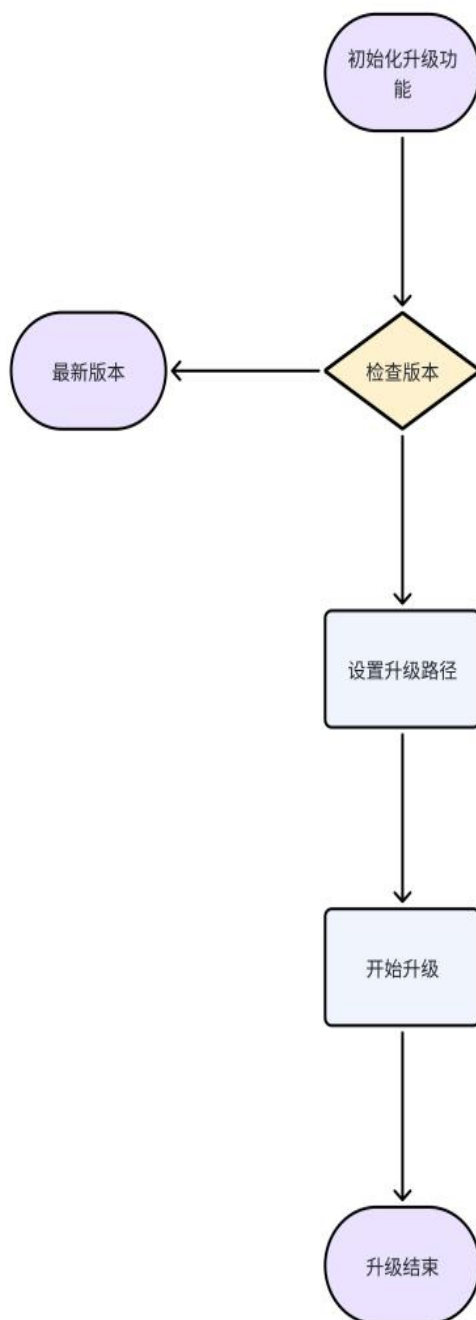
#### 1. Get\_BlueTooth\_Name()

Parameter description: None

Callback:

```
1. @Override
2.     public void readBlueToolName(byte len, String name) {
3.         Logger.show("TAG","Bluetooth name length: " + len + " Bluetooth name:
    " + name);
4.     }
```

### 3.3 Firmware Upgrade (OTA)



#### 3.3.1 Check version

Interface function: Check whether the firmware version is up to date.

Interface declaration:

```
1. OtaApi.checkVersion(String version, VersionCallback versionCallback);
```

Note: To invoke this interface, ensure that it is connected to the ring

Parameter Description: version: version number of the current ring

versionCallback: indicates the latest version information callback

Return value:

```
1. OtaApi.checkVersion(version, new VersionCallback() {  
2.     @Override  
3.     public void success(String newVersion) {  
4.         //newVersion: indicates the latest version number of the cloud  
5.         if (!StringUtils.isEmpty(newVersion)){  
6.             // New version available  
7.         }else{  
8.             // This is the latest version  
9.         }  
10.    }  
11.  
12.    @Override  
13.    public void error() {  
14.        // Failed to get latest version  
15.    }  
16. });
```

### 3.3.2 Start leveling up

Interface Function: Perform a firmware upgrade.

Interface declaration:

```
1. void startUpdate(BluetoothDevice bluetoothDevice, int rssi, LmOTACallback otaC  
   allback)
```

Note: To call this interface, ensure that it is connected to the ring

Parameter Description: bluetoothDevice: The device to be upgraded at present

rssi: device signal value

otaCallback: upgrade callback

Return value:

```
1. OtaApi.startUpdate(App.getInstance().getDeviceBean().getDevice(), App.getIn  
   stance().getDeviceBean().getRssi(), new LmOTA Callback() {  
2.     @Override  
3.     public void onDeviceStateChange(int i) {
```



```
4. // Device status callback
5. }
6.
7. @Override
8. public void onProgress(int i, int i1) {
9. // Upgrade progress
10. }
11.
12. @Override
13. public void onComplete() {
14. // Upgrade complete
15. }
16. });
```

## 3.4 Database related (DataApi)

### 3.4.1 Query history

Function This interface is used to query historical data for a specified time

Interface declaration:

```
1. // Query historical data
2. List<HistoryDataBean> queryHistoryData(long dayBeginTime, long dayEndTime, String mac)
3. // Query historical data in positive chronological order
4. List<HistoryDataBean> queryHistoryDataOrderByTimeAsc(long dayBeginTime, long dayEndTime, String mac)
5. // Query historical data in reverse chronological order of steps
6. List<HistoryDataBean> queryHistoryDataOrderByStepCountDesc(long dayBeginTime, long dayEndTime, String mac)
```

Note: The corresponding permissions need to be enabled

Parameter Description: dayBeginTime: Start time stamp, unit: second

dayEndTime: end time stamp, in seconds

mac: indicates the MAC address of the device

Return value:

```
1. public class HistoryDataBean{
2.
3. @Id
4. private Long id;
```

```

5.
6.   private String mac;
7.   // Total packet number 4 bytes
8.   private long totalNumber;
9.   // Current number of packets 4 bytes
10.  private long indexNumber;
11. // Current record time 4 bytes
12.  private long time;
13. // Total 2 bytes of steps today
14.  private int stepCount;
15. // Heart rate 1 byte
16.  private int heartRate;
17. // Blood oxygen 1 byte
18.  private int bloodOxygen;
19. // heart rate variability 1 byte
20.  private int heartRateVariability;
21. // Mental Stress Index 1 byte
22.  private int stressIndex;
23. // Temperature 2 bytes
24.  private int temperature;
25. // Motion intensity 1 byte
26.  private int exerciseIntensity;
27. / * *
28. * Chronotype 1 byte
29. * 0: invalid
30. * 1: Awake
31. * 2: Light sleep
32. * 3: Deep sleep
33. * 4: Eye movement period
34. * /
35.  private int sleepType;
36. // Reserve 2 bytes
37.  private int reserve;
38. // The RR interval is 1 byte
39.  private int rrCount;
40. //RR array data 1 byte
41.  private byte[] rrBytes;
42. }

```

### 3.4.2 Clear historical data

This interface is used to clear all historical data.

Interface declaration:

```
1. void deleteHistoryData();
```

Note: Permissions need to be enabled

Parameter Description: None

Returned value: None

### 3.5 Logical algorithm correlation (LogicalApi)

#### 3.5.1 Calculate distance, calories

Interface function: Calculate distance and calories according to steps.

Interface declaration:

```
1. DistanceCaloriesBean calculateDistance(int stepCount,double height,double weight);
```

Note: need to enable the corresponding permissions

Parameter Description: stepCount: indicates the number of steps

height: The unit of height (cm)

weight: A unit of body weight in kg

Return value:

```
1. public class DistanceCaloriesBean {  
2. // Distance, in meters  
3. private double distance;  
4. // calories, unit calories  
5. private double kcal;  
6. }
```

#### 3.5.2 Calculate sleep data

Interface function: Calculate the specified time sleep data.

Interface declaration:

```
1. SleepBean calculateSleep(String date, String mac, int type);
```

Note: need to enable the corresponding permissions

Parameter Description: date: Date in the format YYYY-MM-DD HH:mm:ss

mac: Bluetooth MAC for the device

type: 0 indicates daily data, 1 indicates weekly data, and 2 indicates monthly data

Return value:

```
1. public class SleepBean {
2. // Sleep sporadic hours
3. int hours = 0;
4. // sporadic minutes of sleep
5. int minutes = 0;
6. // Total sleep hours
7. int allHours = 0;
8. // Total sleep minutes
9. int allMinutes = 0;
10. // deep sleep time
11. long highTime = 0;
12. // light sleep time = 0
13. long lowTime = 0;
14. // eye movement time = 0
15. long ydTime = 0;
16. // awake time = 0
17. long qxTime = 0;
18. // sleep time stamp
19. long startTime = 0;
20. // Wake time stamp
21. long endTime = 0;
22. // sleep data
23. private List<HistoryDataBean> historyDataBeanList;
24. }
```

Data in database

```
1. public class HistoryDataBean{
2.
3. @Id
4. private Long id;
5. private String mac;
6. // Total packet number 4 bytes
7. private long totalNumber;
8. // Current number of packets 4 bytes
9. private long indexNumber;
10. // Current record time 4 bytes
11. private long time;
12. // Total 2 bytes of steps today
13. private int stepCount;
```

```
14. // Heart rate 1 byte
15.  private int heartRate;
16. // Blood oxygen 1 byte
17.  private int bloodOxygen;
18. // heart rate variability 1 byte
19.  private int heartRateVariability;
20. // Mental Stress Index 1 byte
21.  private int stressIndex;
22. // Temperature 2 bytes
23.  private int temperature;
24. // Motion intensity 1 byte
25.  private int exerciseIntensity;
26. / * *
27. * Chronotype 1 byte
28. * 0: invalid
29. * 1: Awake
30. * 2: Light sleep
31. * 3: Deep sleep
32. * 4. Eye movement period
33. * /
34.  private int sleepType;
35. // Reserve 2 bytes
36.  private int reserve;
37. // The RR interval is 1 byte
38.  private int rrCount;
39. // RR array data 1 byte
40.  private byte[] rrBytes;
41. }
```

## IV、 Other

Note: Check the ring status before using the ring API

### 1、 Filter Related

Blood Oxygen Ring devices broadcast under the name XXXXXX.XXX is any character and is broadcast at 500ms intervals. In this example, XXX is BCL603.

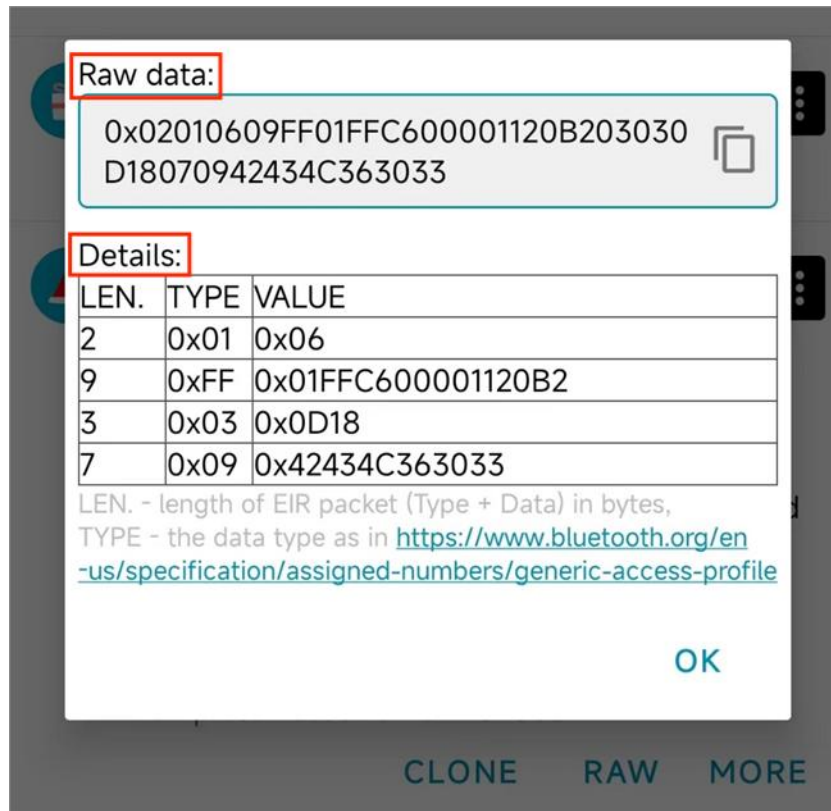
The length of the broadcast packet must be 31 bytes. If it is less than 31 bytes, the rest is filled with zeros. This part of the data is invalid

## 1.1 Broadcast data unit

A broadcast packet contains several broadcast data units, also known as AD structures.

**Broadcast Data unit = Length value Length + AD type + AD Data**

The Length value length takes only one byte and is located in the first byte of the broadcast data unit.



### 1.1.1 Raw data:

0x means the string is a hexadecimal string. Two hexadecimal numbers represent one byte. Because the maximum size of a two-character hexadecimal string is FF, that is 255, and the value range of byte type in Java is -128 to 127, which can just represent a size of 255. So two hexadecimal strings represent one byte.

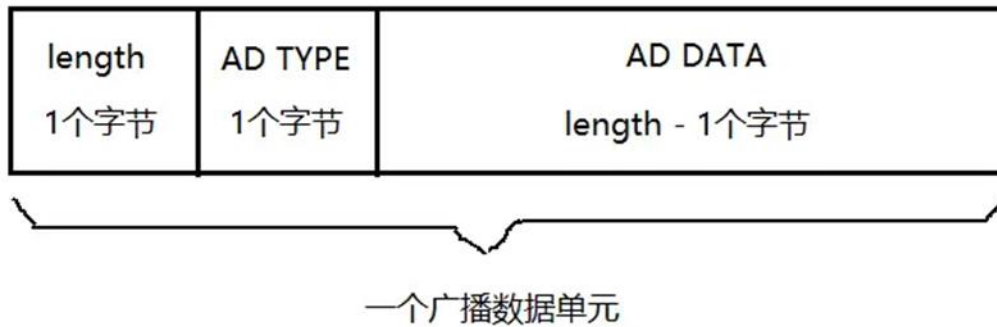
Continue looking at the contents of the message and begin reading the first broadcast data unit. Read the first byte :0x02, converted to decimal is 2, that is, the following 2 bytes are the data content of this broadcast data unit. When the data content exceeds these 2 bytes, it indicates a new broadcast data unit.

For the second broadcast data unit, the value of the first byte is 0x09, which is 9 when converted to decimal, indicating that the next 9 bytes are the second broadcast data unit.

And the third broadcast data unit, the value of the first byte is 0x03, converted to decimal is 3, indicating that the next 3 bytes are the third broadcast data unit.

And so on.

In the data section of the broadcast data unit, the first byte represents the data type (AD type), determining what data the data section represents. (i.e. the second byte of the broadcast data unit is the AD type)



### 1.1.2 Details:

- (1) Type = 0x01 indicates that the device LE is physically connected.
- (2), Type = 0xFF indicates manufacturer data. The first two bytes indicate the manufacturer ID, that is, the manufacturer ID is 0xFF01. The last byte is the manufacturer's data, which is defined by the user
- (3) Type = 0x03 indicates a full 16bit UUID. Its value is 0x0D18.
- (4), Type = 0x09 indicates the full name of the device, for example, 0x42434C363033 to byte[] and then the string is "BCL603"

## 1.2 Application

Note: The data transfer mode is small-end mode

```
@Override
public void onLeScan(BluetoothDevice device, int rssi, byte[] bytes) {
    if (device == null || TextUtils.isEmpty(device.getName())) {
        return;
    }
    Log.d(tag: "TAG", msg: "=== " + device.getName());
    Log.d(tag: "TAG", msg: "=== " + device.getAddress());
    Log.d(tag: "TAG", msg: "onLeScan bytes = " + Arrays.toString(bytes));
}
```

[illegible]

Note: byte data needs to be converted to hexadecimal

If you know the format and meaning of the data, set the filtering condition as the manufacturer ID == "FF01" according to the rules

Or verify "01FF" directly to the received data

## 2、 Problems that may be encountered

Materials with a simple demo, you can first view the simple demo using SDK logic, and then their own development

## 2.1 Version related

Gradle version. It can be modified in gradle-wrapper.properties

1. `distributionBase=GRADLE_USER_HOME`
2. `distributionPath=wrapper/dists`
3. `distributionUrl=https\://services.gradle.org/distributions/gradle-8.0-all.zip`
4. `zipStoreBase=GRADLE_USER_HOME`
5. `zipStorePath=wrapper/dists`

You can download the required version at the following website, put zip in the path of wrapper/dists (in the random code folder, clean up the original), and sync it again

[Gradle Distributions](#)

## 2.2 Gradle 4.0 or higher causes Xpopup to be unusable

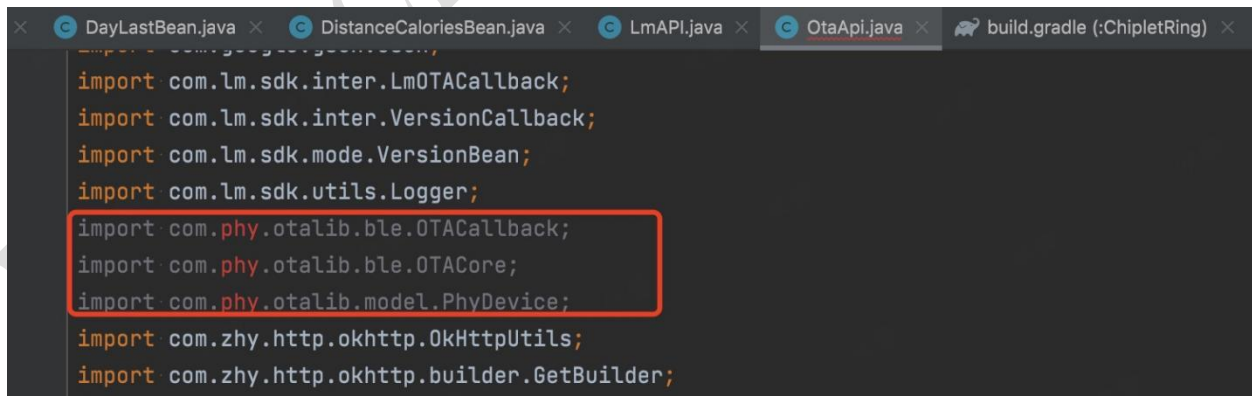
Xpopup is a third party popup frame, replacing it with a regular popup can solve the problem

## 2.3 No interface required

ringSDK1.0.2 supports scanning, connection and other functions in service without interface

## 2.4 OTA class reference not found

The following may occur:



```
import com.lm.sdk.inter.LmOTACallback;
import com.lm.sdk.inter.VersionCallback;
import com.lm.sdk.mode.VersionBean;
import com.lm.sdk.utils.Logger;
import com.phy.otalib.ble.OTACallback;
import com.phy.otalib.ble.OTACore;
import com.phy.otalib.model.PhyDevice;
import com.zhy.http.okhttp.OkHttpUtils;
import com.zhy.http.okhttp.builder.GetBuilder;
```

Official other dependent libraries, if you can not find, you can contact the developers of Yong core, to the original class

The original jar package has been placed under the OTA class folder of the SDK



### 3、 Hardware algorithm logic or firmware related

#### 3.1 Ring related

Q: How often do rings store data

A: Five minutes

Q: Does the OTA upgrade erase the data

A: Yes

Q: The data in the ring can be stored for a few days

A: Seven days, after which it is automatically overwritten

Q: Is restoring factory Settings only restoring rings

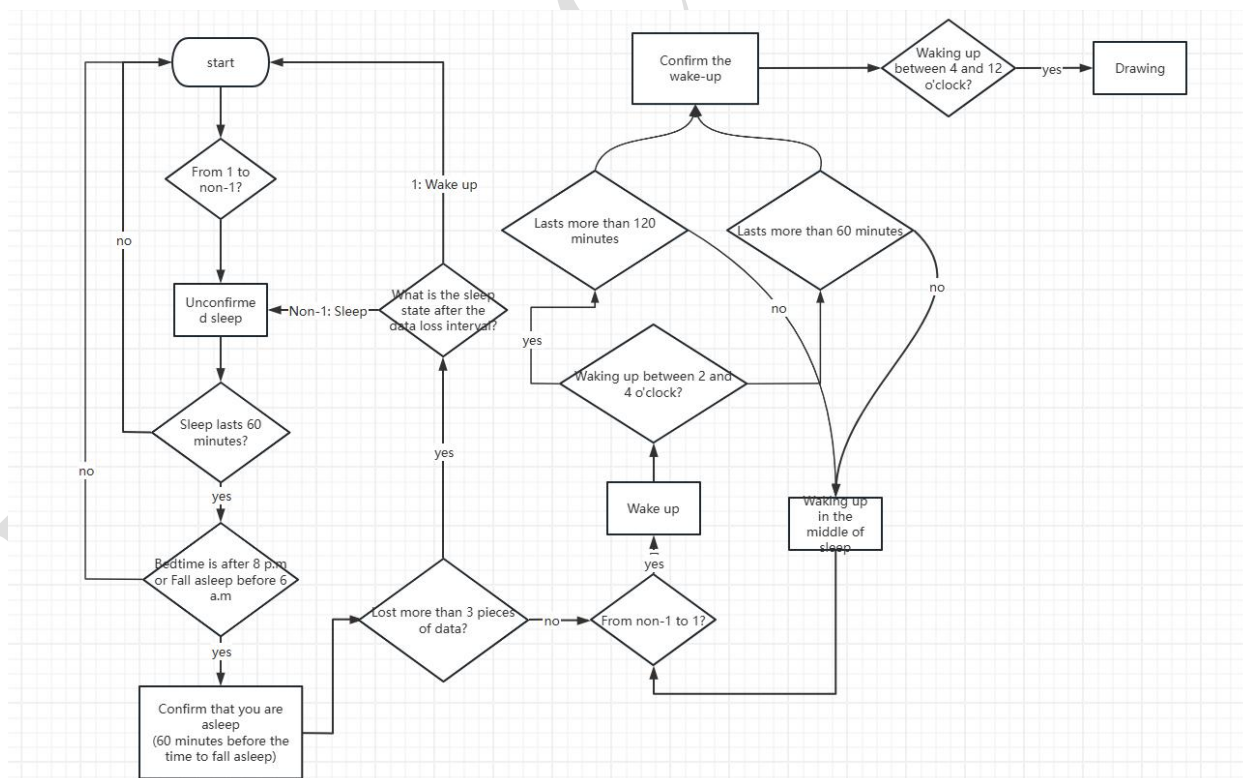
A: Yes, only for the ring hardware

#### 3.2 Algorithmic correlation

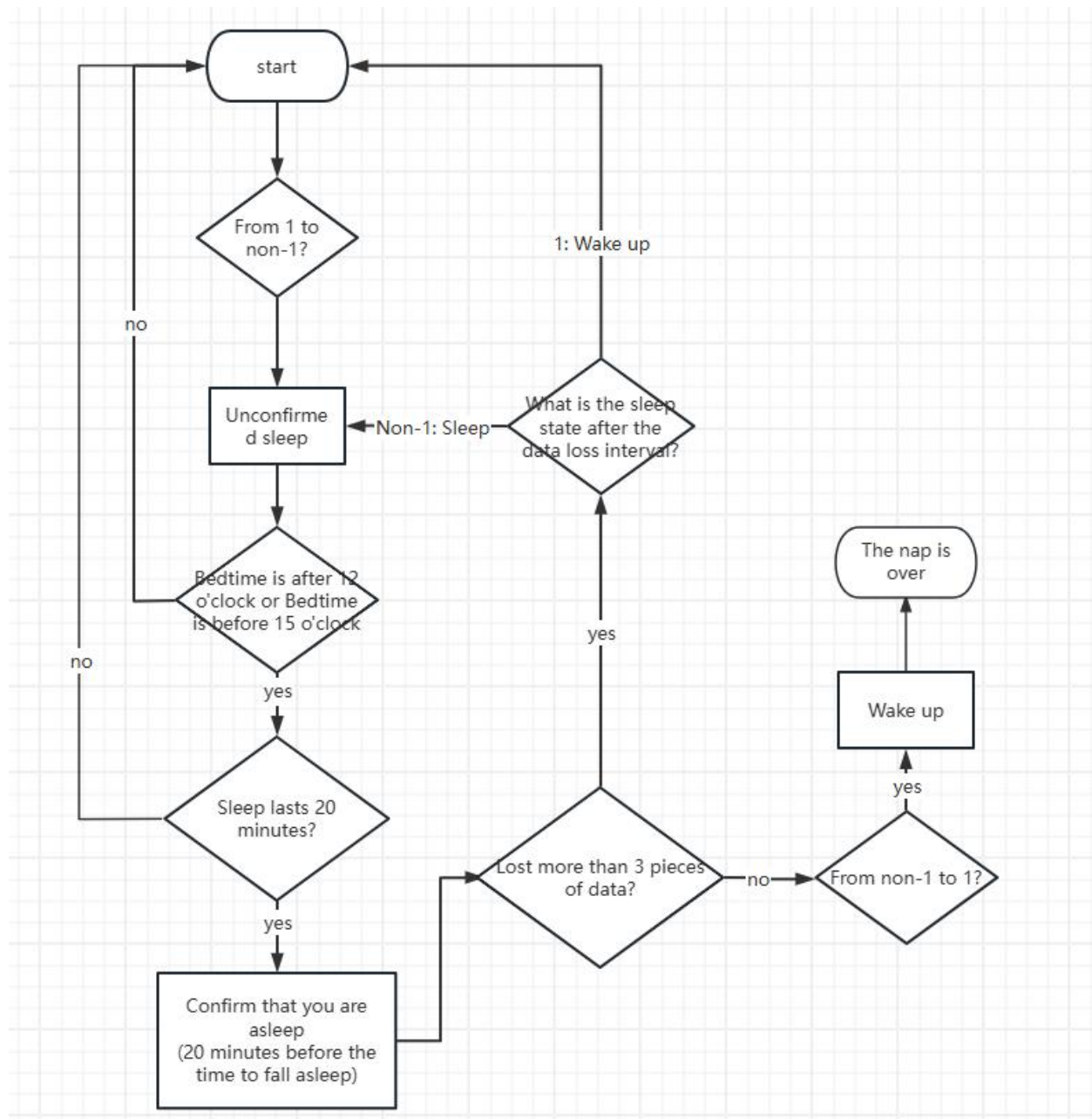
Q: Why does the heart rate/oxygen measurement with the SDK always display a timeout

A: The heart rate oximetry cannot be performed while the ring is charging

#### 3.3 Sleep logic diagrams



### 3.4 Nap logic diagram



### 4、 Q&A

Q: When I read history records, do I report that each tag has been synchronized, or do I upload the whole tag?

A: One mark at a time

Q: What are the limitations of the collection cycle setting

A: The unit of the collection period is second. The normal value is at least 60s. 0 indicates that the collection is closed.