ProtoBuf protocol Android Bluetooth SDK documentation

Access instructions

1. Please copy the **xxx.aar** file in the SDK to the lib in your peoject project.

2. In the build.gradle dependencies in your project app, add `implementation(name: 'blesdk-debug', ext: 'aar')` and add the following code to build.gradle android:

```
repositories {
        flatDir {
            dirs 'libs'
        }
    }
```

3. Start **BleService.class** in the Application and add `SuperBleSDK.addBleListener(this, new IDataReceiveHandler())` . For specific instructions, refer to the sample code **BleApplicaiton**

4. Register a broadcast receiver `MyReceiver.class` to inherit `BluetoothCallbackReceiver.class` so that the receiver can receive the data globally, can be uniformly distributed to the page that needs data, copy the `BaseActionUtils.class` and `BluetoothCallbackReceiver. Class` in the Demo to your own project, you can write `oncrete()` in your own application, the specific code is as follows:

```
IntentFilter intentFilter = BaseActionUtils.getIntentFilter();
MyReceiver receiver = new MyReceiver();
LocalBroadcastManager.getInstance(this).registerReceiver(receiver, intentFilter);
```

5. Copy the code in the `BleApplication.class` in the Demo `SuperBleSDK.addBleListener(this, new IDataReceiveHandler())` to include the implementation and send the broadcast to the code in your Application.

6. Initialize the SDKType: Call the code `SuperBleSDK.switchSDKTYpe(context, Constants.Bluetooth.Zeroner_protobuf_Sdk)` . Note the second parameter selection **Bluetooth.Zeroner_protobuf_Sdk**, then call `MyApplication.getInstance().getmService().setSDKType(context .getApplicationContext(), Constants.Bluetooth.Zeroner_Zg_Sdk)` .** Note that this method must be started in `BleService.class` to be executed. If the project is started, it will start to scan and add a delay.:

```
Handler handler = new Handler(Looper.getMainLooper());
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {

MyApplication.getInstance().getmService().setSDKType(context.getApplicationContext(),
Constants.Bluetooth.Zeroner_protobuf_Sdk);
            }
        },3000);
```

Scan and connection

1. It is recommended to use `BluetoothUtil.class` to operate. Scan the `BluetoothUtil.startScan()` method. Go back and forth through `LocalBroadcastManager.getInstance(context).registerReceiver(your receiver object, BaseActionUtils.getIntentFilter()). The callback method is explained as follows:

- `onPreConnect()` Initiate a connection callback
- `onScanResult(WristBand band)` Scan result callback, returning the scanned bracelet in this callback method
- `connectStatue(boolean isConnect)` Connection state callback, disconnect and connection when this method is successful
- `onBluetoothInit()` If the connection is successful, data can be exchanged with Bluetooth, and the method is called by the sending and receiving instructions. **Generally, the method successfully processes the identifier of the Bluetooth connection in order to send and receive instructions.**

2. Call `BluetoothUtil.stopScan()` to stop scanning and call `BluetoothUtil.connect(band)` to connect. **Note that the band object is the band in `onScanResult(WristBand band)`**

Send and receive instructions

1. Sending a command requires sending data after `onBluetoothInit()`
2. Commands with return values need to add `BackgroundThreadManager.getInstance().addWriteData(context,cmd)` at the end:

```
/*set time*/
byte[] bytes = SuperBleSDK.getSDKSendBluetoothCmdImpl(context).setTime();
BackgroundThreadManager.getInstance().addWriteData(context,bytes);
```

APP send command

> Generally, the command ring is returned to the model to resolve the same, the notification is successful, and the transmission succeeds or fails.

- **set time**

```
//Set time The real time of the current system
public byte[] setTime()
//unit(s)
byte[] setTime(long time)
```

- **Set heart rate alarm**

```
/**
 * Set heart rate alarm
 *
 * @param
 * @param enable
 * @param hrH      heart max // 50~200
 * @param hrL      heart min // 40~190
 * @param second   trigger alarm when couter reaches this value  start time
 * @param interval minutes,re-alarm interval (minute)
 */
byte[] setHeartAlarm(boolean enable, int hrH, int hrL, int second, int interval)
```

- **set user info**

```
/**
 * set user info
 *
 * @param height default 170
 * @param weight default 60
 * @param gender default male
 * @param age    default 20
 * @param walk   50-200 default 100
 * @param run    50-200 default 100
 */
byte[] setUserConf(int height, int weight, boolean gender, int age, int walk, int run)
```

- **set sport target**

```
/**
 * set sport target
 *
 * @param calorie  default = 400 kC
 * @param step     default = 10000 unit seconds
 * @param distance default = 10000 unit meters
 */
byte[] setGoalConf(int calorie, int step, int distance)
```

- **set GPS**

```
/**
* GPS
*
* @param  altitude
* @param  latitude
* @param  longitude
*/
byte[] setGnssConf(int altitude, String latitude, String longitude)
```

- **set BP**

```
/**
 * BP
 *
 * @param src_sbp;
 * @param src_dbp
 * @param dst_sbp
 * @param dst_dbp
 * @param dif_sbp   // 0
 * @param dif_dbp   // 0
 */
 byte[] setBpCaliConf(int src_sbp,int src_dbp,int dst_sbp,int dst_dbp,int dif_sbp,int dif_dbp)
```

- **Do not disturb**

```
/**
 * Do not disturb
 *
 * @param
 * @param policy    true on  false off
 * @param startHour
 * @param endHour
 * @param startMin
 * @param endMin
 */
byte[] setMsgNotificationTime(boolean policy, int startHour, int endHour, int startMin, int
endMin)
```

- **set message notify**

```
/**
 * message notify
 *
 * @param
 * @param id
 * @param type   ADDED = 0; // incoming call and sms
 *               REMOVED = 1; // incoming call accepted or rejected
 *               UPDATED = 2; // incoming call turn to missed-call
 * @param accept
 * @param reject
 * @param mute
 * @param title
 * @param detail send content
 *               <p>
 *               B.for call
 *               {id = xxx , type = TYPE_CALL , status = ADDED , option = {accept = true , reject =
true , mute = true} , title = “Incoming Call”  , detail = “ ( 0755 ) 10010” }
 *               <p>
 *
 *               {id = xxx , type = TYPE_CALL , status = REMOVED}
 *               <p>
 *
 *               {id = xxx , type = TYPE_CALL , status = UPDATED}
 *               <p>
 *               C.
 *               for message
 *               {id = xxy , type = TYPE_SMS , status = ADDED}
 */
public byte[] setMsgNotificationNotifyByCall(int id, int type, boolean accept, boolean reject,
boolean mute, String title, String detail)
```

- **set phone notify**

```
/**
 * same as msg notify
 */
public byte[] setMsgNotificationNotifyByCall(int id, int type, boolean accept, boolean reject,
boolean mute, String title, String detail)
```

- **set weather**

```
/**
 * @param timeMills   units  Use the morning time of the day as the ID
 * @param weatherDesc // required for ADD
 *                    WeatherDesc {
 *                    Sunny = 0;
 *                    Cloudy = 1;
 *                    Overcast = 2;
 *                    LightRain = 3;
 *                    ModerateRain = 4;
 *                    HeavyRain = 5;
 *                    Rainstorm = 6;
 *                    Snow = 7;
 *                    Haze = 8;
 *                    SandStorm = 9;
 *                    CloudyTurnToFine = 10;
 *                    Thunderstorm = 11;
 *                    }
 * @param weatherType EACH_HOUR = 0;
 *                    EACH_DAY = 1;
 * @param degreeMax   max temp
 * @param degreeMin   min temp
 * @param pm2p5       pm2.5
 */
public byte[] setWeather(int timeMills, int weatherDesc, int weatherType, int degreeMax, int
degreeMin, int pm2p5)

/**
 *WeatherEvent Same as the above parameters. Pass 24 hours of weather. This setting will change
every hour.
 *
 */
byte[] setWeather(List<WeatherEvent> weatherEvents)
//clear weather cmd
byte[] clearWeather()
```

- **add alarm clock**

```
/**
 * add alarm clock
 * Do not exceed 5 alarm clocks
 * @param id      AlarmId
 * @param repeat repeat times
 * @param week   0000 0001   mon 0x01
 *               0000 0010   tue 0x20
 *               0000 0100    ...
 *               0000 1000
 *               0001 0000
 *               0010 0000
 *               0100 0000
 *               <p>
 *               sample set mon,tue,wed --  0000 0111
 * @param hour   set hour
 * @param min    set min
 * @param text   set text Do not exceed 12 length
 */
byte[] addAlarm(int id, boolean repeat, int week, int hour, int min, String text)
//Remove alarm clock based on ID
byte[] removeAlarm(int id)
//clear clock
byte[] clearAlarm()
```

- **Set sedentary reminder**

```
/**
 *
 *
 * @param repeat
 * @param week      as setAlarm()
 * @param startHour  // 0~23
 * @param endHour      // 0~23
 * @param duration  // 0~1440
 * @param threshold
 */
byte[] setSedentariness(boolean repeat, int hash, int week, int startHour, int endHour, int
duration, int threshold)
//clear
byte[] clearSedentariness()
```

- **add calendar**

```
/**
 *
 * @param id
 * @param timeMillis s
 * @param text
 * @return
 */
byte[] addCalendar(int id, int timeMillis, String text)

//clear
public byte[] removeCalendar(int hash, int second)
```

- **set language**

```java
/**
 * set language
 *
 * @param context
 * @param language English = 0;
 *                 Chinese = 1;
 *                 Italian = 2;
 *                 Japanese = 3;
 *                 France = 4;
 *                 German = 5;
 *                 Portuguese = 6;
 *                 Spanish = 7;
 *                 Russian = 8;
 *                 Korean = 9;
 *                 Arabic = 10;
 *                 Vietnam = 11;
 *                 Polish = 12;
 */
void setLanguage(Context context, int language)
```

- **set distance,temp,time,date unit**

```
/**
 * set distance unit
 *
 * @param distance_unit  false[default]: metric unit, true 1: imperial units
 */
byte[] setDistenceUnit(boolean distance_unit)


/**
 * set temp unit
 *
 * @param temperature_unit  false[default]: Celsius, true: Fahrenheit
 */


byte[] setTemperatureUnit(boolean temperature_unit)


/**
 * set hour unit
 *
 * @param hour_format  false[default]: 24hour, true1: 12hour
 */
byte[] setHourFormat(boolean hour_format)


/**
 * set date unit
 *
 * @param date_format  0[default]: month/day, 1: day/month
 */
byte[] setDateFormat(boolean date_format)
```

- **set auto Heart**

```
/**
 * set auto Heart
 *
 * @param autoHeartrate true on  false off
 */
byte[] setAutoHeartrate(boolean autoHeartrate)
```

- **set auto sport**

```
/**
 * set auto sport
 *
 * @param auto_sport true on  false off
 */
byte[] setAutoSport(boolean auto_sport)
```

- **set gesture**

```
/**
 * @param lcdGsswitch  0 disable lcd switching by gesture, 1[default]：enable lcd switching by
gesture
 * @param startHour   0[default]~23:  the start hour of switching lcd by gesture
 * @param endHour     0[default]~23:  the end hour of switching lcd by gesture
 */
byte[] setLcdGsTime(boolean lcdGsswitch, int startHour, int endHour)
```

- **set motor vibrate**

```
/**
 * motor vibrate type
 *
 * @param mode
 * @param round
 * @param type  ALARM_CLOCK = 0;
 *              INCOMING_CALL = 1;
 *              SMS = 2;
 *              SEDENTARINESS = 3;
 *              CHARGING = 4;
 *              CALENDAR = 5;
 *              DISTANCE_ALARM=6;
 *              HEARTRATE_ALARM=7;
 *              OTHERS = 8;
 */
byte[] setMototConf(int mode, int round, int type)

/**
 * motor vibrate
 *
 * @param mode  max 15
 * @param round
 */
byte[] setMotorVibrate(int mode, int round)
```

- **set smart shot**

```
/**
 * set smart shot
 *
 * @param mode RT_MODE_BACK_NORMAL = 0; //exit
 *             RT_MODE_ENTER_CAMERA = 1; enter
 */
byte[] setSmartShotData(int mode)
```

> ## APP receiver command
>
> The return value is returned as a json string, which can be parsed according to Google Gson or Alibaba fastJson

- **get device info**

```
byte[] getHardwareFeatures()

//retrurn model:
public class ProtoBufHardwareInfo {

    private String version;
    private String model;//name
    private String mac;
    private String deviceTime;
    private String factory;
    private String fota;
    private int fotaType;
}
```

- **get Battery**

```
//et Battery
byte[] getBattery() send this command the isBattery = true .such as model

//return model
public class ProtoBufRealTimeData {
    /**
     is battery cmd
     */
    private boolean isBattery;
    private int  level;//battery
    private boolean charging;//charge

    /**
     * is Hearth cmd
     */
    private boolean isHearth;
    private float calorie;
    private int distance;
    private int steps;

    /**
     * is time cmd
     */
    private boolean isTime;
    private int seconds;


    /**
     * is key 0 normal 1 camera For smart shot
     */
    private boolean isKey;
    private int keyMode;
}
```

- **get health data**

```
//get health data
byte[] getRealHealthData()

//as battery
```

- **get device support function**

```
/**
 * get device support function
 */
byte[] getDataInfo()

//retrun model:
public class ProtoBufSupportInfo {

    private boolean  support_health;
    private boolean  support_gnss;
    private boolean  support_ecg;
    private boolean  support_ppg;
    private boolean  support_rri;
}
```

- **sync history data **

```
1.First sync index table
2.Historical data synchronization based on startSeq and endSeq in the index table.
3. The synchronization type can get the supported types through getDataInfo() to synchronize the
data. Otherwise, the new send command will not reply and the APP will die.

/**
 *
 * get index table
 *
 * @param type  Reference document
 *        HEALTH_DATA = 0;
 *        GNSS_DATA = 1;
 *        ECG_DATA = 2;
 */
byte[] itHisData(int type)

/**
 * get history data
 *
 * @param
 * @param type  Reference document
          HEALTH_DATA = 0;
 *        GNSS_DATA = 1;
 *        ECG_DATA = 2;
 * @param startSeq
 * @param endSeq
 */
byte[] startHisData(int type, int startSeq, int endSeq)


//index table is used to record synchronization information
public class ProtoBuf_index_80 extends DataSupport {

    private long uid;
    private String data_from;
    private int start_idx;start seq
    private int end_idx;end seq
    private int year;
    private int month;
    private int day;
    private int hour;
    private int min;
    private int second;
    private int time;//timestamp
    private int indexType;//type
}

Since the synchronization history data is not a piece of data, the amount of data is large, the
SDK only provides commands, and the sample code has a demo data synchronization, as follows:
//sync history data
ProtoBufSync.getInstance().syncData();

//return model
public class ProtoBuf_80_data {

    private long uid;
    /**
     * data
     */
    private int year;
    private int month;
    private int day;
    private int hour;
    private int minute;
    private int second;

    /**
     * timestamp(s)
     */
    private int time;
    /**
     * index table seq
     */
    private int seq;
    /**
     * device info
     */
    private String data_from;
    /**
     * sleepData
     */
    private String sleepData;
    private boolean charge;
    private boolean shutdown;
    /**
     * health
     */
    private int type;
    private int state;
    private float calorie;
    private int step;
    private float distance;
    /**
     * heart rate
     */
    private int min_bpm;
    private int max_bpm;
    private int avg_bpm;
    /**
     * hrv
     */
    private float SDNN;
    private float RMSSD;
    private float PNN50;
    private float MEAN;
    private float fatigue;


    /**
     * bp
     *
     */
    private int sbp;
    private int dbp;
    private int bpTime;
}
```

other command

- **AGPS update**

```
Only GPS-enabled devices support AGPS upgrades, which are not supported by other devices.
/**
 * download init
 *
 * @param fuType
 * @param fileSize
 * @param fileCyc
 * @param fileName
 * @param fileOffset
 */
byte[] setFileDescUpdate(boolean isDesc)
/**
 * download data
 *
 * @param fuType
 * @param
 * @param
 * @param buf
 */
byte[] setFileDataUpdate(int fuType, int fileCyc, int crc32AtOffset, ByteString buf)
byte[] setFileDataExit(int fuType)

The SDK provides instructions for AGPS upgrades. For specific functions, see the demo in the
sample code, as follows:
// ProtoBufUpdate.Type.TYPE_GPS FONT
ProtoBufUpdate.getInstance().startUpdate(ProtoBufUpdate.Type.TYPE_GPS);
```

- **Firmware upgrade**
  - Refer to the sample code `ProtoBufFirmwareUpdateActivity.class`, which is the entire implementation of the firmware upgrade. You can check the effect of the firmware upgrade in the sample code app. Add your own project to the sample code **no.nordicsemi.android** All the files below.

Receive data

- As in the **access instructions step 4**, you can register a global broadcast receiver to inherit the `BluetoothCallbackReceiver` to globally send and receive broadcasts. The specific classes can be sent using **EventBus**, the implementation class is as follows:

```
/*
 *
 * @param ble_sdk_type  Refers to the type of SDK is generally specified
Constants.Bluetooth.Zeroner_protobuf_Sdk
 * @param dataType OPT Reference document
 * @param data The data parsed by the SDK is parsed by JSON. The SDK uses fastJson. About the
data of the boolean type, Gson may appear to be always false. In this case, it is recommended to
use fastJson.
 */
void onDataArrived(Context context, int ble_sdk_type, int dataType, String data)
```

- Global broadcasts are generally used to parse real-time data in synchronous historical data, etc. Local broadcasts can also be used for a single page. Copy the code in the Demo `BleReceiverHelper.class` to the project as follows:

```
BleReceiverHelper.registerBleReceiver(context,new MyDataReceive());

private class MyDataReceive extends BluetoothCallbackReceiver{
    @Override
    public void onDataArrived(Context context, int ble_sdk_type, int dataType, String data) {
        super.onDataArrived(context, ble_sdk_type, dataType, data);
    }
}
```