- QC Wireless SDK Instructions for Use
    - update note:
    - 1.introduce
        - 1.1The role of the SDK
    - 2. API description
    - 2.0 Access Conditions
    - 2.1 Permissions required by the SDK
    - 2.2 Access conditions
    - 2.3.1API
        - Scanning Devices
        - device connection:BleOperateManager.getInstance()
    - 2.3.2 Feature list:
        - Synchronize time, get the list of functions supported by the device
        - bracelet battery
        - Continuous heart rate, blood oxygen, blood pressure switch
        - Set watch sports goals
        - Find equipment
        - Ring photo control
        - Set the Ring to factory reset
        - firmware version number, hardware version number
    - 2.3.3 Data synchronization:
        - Synchronize steps, distance, kcal for the day
        - Synchronized step data details
        - Sync Sleep Data Details
        - Synchronize the details of new sleep data and return according to SetTimeRsp
        - Sync heart rate data
        - Synchronized blood pressure function
        - Synchronized blood oxygen function
        - Synchronized pressure function
    - 2.3.6 OTA upgrade function:
    - 2.3.7Manual measurement

- 2.3.9 Changes in Ring measurement data are proactively reported to the APP
- 2.3.10 APP opens exercise type

QC Wireless SDK Instructions for Use

> 1. Author：James
> 2. Shenzhen QC.wireless Technology Co., Ltd.
> 3. Version：1.0.0

update note:

> 1. （2021/07/06）scan, connect, measure commands
> 2. （2021/07/20）Add setting command
> 3. （2021/07/21）Increase step count, heart rate, sleep data sync
> 4. （2022/02/28）Add new sleep algorithm
> 5. (2023/03/10) Add message switch synchronization, body temperature, user information, watch manual measurement of heart rate, blood pressure results
> 6. (2023/03/16) Add bt connection and contact person
> 7. (2024/02/23) Add manual pressure, APP movement, pressure synchronization, pressure setting

1.introduce

1.1The role of the SDK

Provide partner companies with the Android Bluetooth SDK for use with Green Orange wireless devices that provide basic and advanced functionality for a major watch or other device. This document is intended to explain the usage context, functionality, etc. of the API. Intended Audience and Reading Recommendations The intended audience and reader recommendations in this article are shown in Annex 1.

Reader	Role
Software Architecture Engineer	Architecture Analysis and Technical Guidance
Android development engineer	Have a certain android development ability, understand Ble related development technology

2. API description

2.0 Access Conditions

> Android 5.0 or above, Bluetooth 4.0 or above.

2.1 Permissions required by the SDK

```
    //network permissions
    <uses-permission android:name="android.permission.INTERNET" />
    //Bluetooth related permissions
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />
    //Storage related permissions
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
    //Location permission
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />

    If it is android 12 or higher system
      <uses-permission
android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission
android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission
android:name="android.permission.BLUETOOTH_ADVERTISE" />
```

2.2 Access conditions

- Green Orange Wireless Wearables
- Green Orange Wireless SDK and documentation

2.3.1API

Scanning Devices

```
    //start scan
    BleScannerHelper.getInstance().scanDevice(final Context
context, UUID mUuid, final ScanWrapperCallback scanCallBack);
    //stop scan
    BleScannerHelper.getInstance().stopScan(Context context)
    //Specified device scan
    BleScannerHelper.getInstance().scanTheDevice(final Context
context, final String macAddress, final OnTheScanResult scanResult)
```

device connection:BleOperateManager.getInstance()

```
    //direct connection

BleOperateManager.getInstance().connectDirectly(smartWatch.deviceAd
dress)
    //scan connections

BleOperateManager.getInstance().connectWithScan(smartWatch.deviceAd
dress)
    //disconnect
    BleOperateManager.getInstance().unBindDevice()
    //reconnect
    BleOperateManager.getInstance().setNeedConnect(boolean
needConnect)
    //Called when bluetooth is turned off
    BleOperateManager.getInstance().setBluetoothTurnOff(false)
    BleOperateManager.getInstance().disconnect()
    //Turn on the system bluetooth monitor
    BleOperateManager.getInstance().setBluetoothTurnOff(true)
```

2.3.2 Feature list:

Synchronize time, get the list of functions supported by the device

```
    //set time
    CommandHandle.getInstance().executeReqCmd(SetTimeReq(0),
ICommandResponse<SetTimeRsp>() {})
    //Callback Description
       public class SetTimeRsp extends BaseRspCmd {
    //support body temperature
    public boolean mSupportTemperature;
    //watch face
    public boolean mSupportPlate;
    //Support the menstrual cycle
    public boolean mSupportMenstruation;
    //Support custom watch faces
    public boolean mSupportCustomWallpaper;
    //Support blood oxygen
    public boolean mSupportBloodOxygen;
    //blood pressure support
    public boolean mSupportBloodPressure;
    //Support fatigue
    public boolean mSupportFeature;
    //Support one-key detection
    public boolean mSupportOneKeyCheck;
    //weather support
    public boolean mSupportWeather;
    //Support for new sleep protocol
    public boolean mNewSleepProtocol;
    //Supports up to 6 or 3 dials
    public int mMaxWatchFace;
}
```

bracelet battery

```
       CommandHandle.getInstance().executeReqCmd(new
SimpleKeyReq(Constants.CMD_GET_DEVICE_ELECTRICITY_VALUE), new
ICommandResponse<BatteryRsp>() {

            @Override
            public void onDataResponse(BatteryRsp resultEntity) {
                if (resultEntity.getStatus() ==
BaseRspCmd.RESULT_OK) {
                    //Get battery successfully
                }
            }

        });

    //Callback Description
    public class BatteryRsp extends BaseRspCmd {
    //battery【0-100】
    private int batteryValue;
}
```

Continuous heart rate, blood oxygen, blood pressure switch

```
     //Read continuous heart rate settings
      CommandHandle.getInstance()
          .executeReqCmd(HeartRateSettingReq.getReadInstance(),
              ICommandResponse<HeartRateSettingRsp> {
              //it.isEnable         switch
              //it.heartInterval    heart rate measurement interval

              })
      //Read Continuous SpO2 settings
      CommandHandle.getInstance()
          .executeReqCmd(BloodOxygenSettingReq.getReadInstance(),
              ICommandResponse<BloodOxygenSettingRsp> {

              })
      //Read continuous blood pressure settings
      CommandHandle.getInstance()
          .executeReqCmd(BpSettingReq.getReadInstance(),
              ICommandResponse<BpSettingRsp> {

              })
       //Read pressure setting
        CommandHandle.getInstance()
```

```
            .executeReqCmd(PressureSettingReq.getReadInstance(),
                ICommandResponse<PressureSettingRsp>() {
                  switch:it.isEnable
                })

        //Write continuous heart rate switch isEnable: true on,
false: off  hrInterval：10，15，20，30，60

         CommandHandle.getInstance().executeReqCmd(
           HeartRateSettingReq.getWriteInstance(true,hrInterval),
           ICommandResponse<HeartRateSettingRsp> {

           })
        //Write continuous blood oxygen switch isEnable: true on,
false: off
        CommandHandle.getInstance().executeReqCmd(
           BloodOxygenSettingReq.getWriteInstance(boolean isEnable),
           ICommandResponse<BloodOxygenSettingRsp> {

           })
        //write blood pressure switch

CommandHandle.getInstance().executeReqCmd(BpSettingReq.getWriteInst
ance(boolean isEnable, StartEndTimeEntity startEndTimeEntity, int
multiple),  ICommandResponse<BpSettingRsp> {

      })
      BpSettingRsp,Parameter Description
      isEnable: true on false off
      StartEndTimeEntity The parameter description is the same as
above
      multiple  default 60

      //Write pressure setting switch,switch isEnable: true on,
false: off
      CommandHandle.getInstance().executeReqCmd(
          PressureSettingReq.getWriteInstance(isEnable),
          ICommandResponse<PressureSettingRsp> {

          })
```

Set watch sports goals

```
        CommandHandle.getInstance().executeReqCmd(
                    TargetSettingReq.getWriteInstance(
                      final int step, final int calorie, final int
distance, final int sportMinute, final int sleepMinute
                    ), null
                )

   Parameter Description
   step: step target
   calorie: Calorie target target, unit card, write kcal to a*1000
   distance: Distance to target, in meters
   sportMinute: Exercise duration target, in minutes
   sleepMinute: Sleep duration target, in minutes
```

Find equipment

```
CommandHandle.getInstance().executeReqCmd(FindDeviceReq(), null)
```

Ring photo control

```
 //The bracelet enters the camera interface

CommandHandle.getInstance().executeReqCmd(CameraReq(CameraReq.ACTIO
N_INTO_CAMARA_UI), null)
 //The wristband is controlled by the bright screen on the camera
interface. The APP will send the bright screen command to keep the
watch bright. It is recommended to send it every 2 seconds.
  CommandHandle.getInstance().executeReqCmd(

CameraReq(CameraReq.ACTION_KEEP_SCREEN_ON),
                            null
                        )
 //Bracelet click to take a photo event monitoring

BleOperateManager.getInstance().addNotifyListener(Constants.CMD_TAK
ING_PICTURE,new ICommandResponse<CameraNotifyRsp>(){

            @Override
            public void onDataResponse(CameraNotifyRsp
resultEntity) {


            }
        });
    resultEntity.getAction()
    Parameter Description
    //The watch exits the camera interface
    CameraNotifyRsp.ACTION_FINISH
    //The watch clicked on the photo event
    CameraNotifyRsp.ACTION_TAKE_PHOTO


CommandHandle.getInstance().executeReqCmd(CameraReq(CameraReq.ACTIO
N_FINISH), null)
```

Set the Ring to factory reset

```
CommandHandle.getInstance().executeReqCmd(RestoreKeyReq(Constants.C
MD_RE_STORE),null)
```

- message push

```
 //The watch message push switch is fully turned on, and the APP
should be actively opened
  CommandHandle.getInstance().executeReqCmd(
              SetANCSReq(), null
          )
//Send message push to watch
  MessPushUtil.pushMsg(type,message:String)

PushMsgUintReq parameter description
   type:
       0x00: Call reminder 0x01: SMS reminder 0x02: QQ reminder
0x03: WeChat reminder,
       0x04: incoming call to answer or hang up 0x05: Facebook
message reminder 0x06: WhatsApp message reminder
       0x07: Twitter message reminder 0x08: Skype message reminder
0x09: Line message reminder 0x0a: Linkedln
       0x0b: Instagram 0x0c: TIM message 0x0d: Snapchat
       0x0e: others other types of notifications
```

firmware version number, hardware version number

```
      //hardware information

CommandHandle.getInstance().execReadCmd(CommandHandle.getInstance()
.getReadHwRequest());
       //firmware information

CommandHandle.getInstance().execReadCmd(CommandHandle.getInstance()
.getReadFmRequest());

       Receiving implements this QCBluetoothCallbackCloneReceiver
refer to demo MyBluetoothReceiver
       Judging UUID in the callback onCharacteristicRead

         override fun onCharacteristicRead(uuid: String?, data:
ByteArray?) {
         if (uuid != null && data != null) {
             val version = String(data, Charsets.UTF_8)
             when(uuid){
                 Constants.CHAR_FIRMWARE_REVISION.toString() -> {
                     //Firmware version number version
                 }
                 Constants.CHAR_HW_REVISION.toString() -> {
                     //hardware version number version number
version
                 }
             }
         }
     }
```

2.3.3 Data synchronization:

Synchronize steps, distance, kcal for the day

```
    CommandHandle.getInstance().executeReqCmd(
           SimpleKeyReq(Constants.CMD_GET_STEP_TODAY),
           ICommandResponse<TodaySportDataRsp> {})

TodaySportDataRsp parameter description
    // days ago
    private int daysAgo;
     // date: year
    private int year;
     // date: month
    private int month;
     // date: day
    private int day;
     // total steps
    private int totalSteps;
    // running steps/aerobic steps
    private int runningSteps;
     // calorie value
    private int calorie;
      // walking distance
    private int walkDistance;
     // Movement time, in seconds
    private int sportDuration;
    // sleep time in seconds
    private int sleepDuration;
```

Synchronized step data details

```
    dayOffset 0: Today 1: Yesterday 2: The day before yesterday,
supports synchronization for up to 7 days
    CommandHandle.getInstance().executeReqCmd(
            ReadDetailSportDataReq(dayOffset, 0, 95),
            ICommandResponse<ReadDetailSportDataRsp> {

            })
    BleStepDetails parameter description
     //year
     private int year;
     //moon
     private int month;
     //day
     private int day;
     //15 minutes a point, the total number of points in a day is
96 points, [0, 95], used to calculate the details of the number of
steps per hour
     private int timeIndex=0;
     // calorie unit card
     private int calorie=0;
     //Step count
     private int walkSteps=0;
      //distance in meters
     private int distance=0;
      // keep for now
     private int runSteps=0;
```

Sync Sleep Data Details

```
  //deviceAddress Device mac address
  //dayOffset 0: Today 1: Yesterday 2: The day before yesterday,
supports synchronization for up to 7 days
  //ISleepCallback callback

SleepAnalyzerUtils.getInstance().syncSleepReturnSleepDisplay(device
Address,dayOffset, ISleepCallback {
                    //callback：SleepDisplay
                   })

   SleepDisplay parameter description
     // total sleep time
     private int totalSleepDuration;
     // total time of deep sleep
     private int deepDuration;
     // total time of light sleep
     private int shallowDuration;
     // Go to sleep timestamp in seconds
     private int sleepTime;
     // wake up timestamp in seconds
     private int wakeTime;
     // A set of sleep data
     private List<SleepDataBean> list;
     private String address;

     SleepDataBean parameter description
     sleepStart start timestamp of a sleep in seconds
     sleepEnd The end timestamp of a sleep in seconds
     type sleep type 2: light sleep 1: deep sleep 3: awake
```

Synchronize the details of new sleep data and return according to SetTimeRsp

```
    //offset 0 today 1 yesterday
    public void syncSleepList(int offset, final
ILargeDataSleepResponse response)
    SleepNewProtoResp   parameter description
    //sleep start time
    private int st;
    //sleep end time
    private int et;
    //sleep collection
    private List<DetailBean> list;
    DetailBean parameter description
   // duration of a sleep type
     private int d;
     //type of sleep 2: light sleep 3: deep sleep 5: awake
     private int t;
```

Sync heart rate data

```
 nowTime current time zone * 3600 + unix second value of current
time
     Sync yesterday: nowTime-(24*3600)*1,
     Sync the day before yesterday: nowTime-(24*3600)*2
     Data can be synchronized for up to three days
     val time = (getTimeZone() * 3600).toInt()
     val nowTime = date.unixTimestamp + time

  CommandHandle.getInstance().executeReqCmd(
             ReadHeartRateReq(nowTime),
             ICommandResponse<ReadHeartRateRsp> {

             })

  ReadHeartRateRsp parameter description
   //nothing yet
   private int size = 0;
    //nothing yet
   private int index = 0;
    // unix second value of heart rate data
   private int mUtcTime;
   //The heart rate data array is one point every 5 minutes, the
data subscript *5 is equal to the number of minutes of the day
   private byte[] mHeartRateArray;
   private boolean endFlag = false;
```

Synchronized blood pressure function

```
//Synchronized automatic blood pressure, measured once an hour
   CommandHandle.getInstance()

.executeReqCmd(SimpleKeyReq(Constants.CMD_BP_TIMING_MONITOR_DATA),
ICommandResponse<BpDataRsp> {}
    BpDataEntity parameter description
    //year
    private int year;
    //moon
    private int mouth;
    //day
    private int day;
    private int timeDelay;
    private ArrayList<BpValue> bpValues;

    BpValue parameter description
    //The minute of the day, usually the whole hour
    int timeMinute;
    //measured heart rate value
    int value;

    Get the blood pressure value calculated from the measured value
    //The heart rate value returned by the hr callback, age is the
age of the user
     val sbp= CalcBloodPressureByHeart.cal_sbp(hr, age) (systolic
blood pressure)
     //sbp heart rate calculated value
    val dbp=CalcBloodPressureByHeart.cal_dbp(sbp) (diastolic
pressure)

    //Confirm blood pressure synchronization, call after receiving
the callback, the watch will delete the records that have been
synchronized

CommandHandle.getInstance().executeReqCmd(BpReadConformReq(true),nu
ll)

      //synchronize manual blood pressure
       CommandHandle.getInstance()
            .executeReqCmd(ReadPressureReq(0),
ICommandResponse<ReadBlePressureRsp> {}

    ReadBlePressureRsp.getValueList() parameter description
    BlePressure parameter description
    //time seconds value
    public long time;
```

```
    public long time;
    //(Diastolic pressure)

    public int dbp;
    //(systolic blood pressure)
    public int sbp;
```

Synchronized blood oxygen function

```
 LargeDataHandler.getInstance().syncBloodOxygenWithCallback(new
IBloodOxygenCallback() {
            @Override
            public void readBloodOxygen(List<BloodOxygenEntity>
data) {

            }
        });

     BloodOxygenEntity parameter description
         //data date
         private String dateStr;
          //Data minimum value array, one data per hour, a total of
24
         private List<Integer> minArray;
         //Data maximum value array, one data per hour, a total of
24
         private List<Integer> maxArray;
         //Data value at 0:00 on a certain day
         private long unix_time;
```

Synchronized pressure function

```
   //offset  0-6
   //Synchronization pressure 7 days 0 only synchronizes today 1
yesterday 2 synchronizes the day before yesterday .... supports up
to 7 days
  CommandHandle.getInstance()
            .executeReqCmd(PressureReq(offset),
                ICommandResponse<PressureRsp> {

                })

    //PressureRsp Parameter Description
    //Pressure data, the return value divided by 10 is the value
displayed by the APP, and one data is generated every half hour.
    private byte[] pressureArray;
     //Stress test interval
    private int range=30;
    //Pressure data date
    private DateUtil today;
```

2.3.6 OTA upgrade function:

```
   //dfu upgrade instance
     val fuHandle= DfuHandle.getInstance()
      //initialize callback
     dfuHandle.initCallback()
       //DFU file verification, path firmware file path
      if (dfuHandle.checkFile(path)) {
          dfuHandle.start(dfuOpResult)
     }
     //dfuOpResult callback description
       private val dfuOpResult: DfuHandle.IOpResult = object :
DfuHandle.IOpResult {
         override fun onActionResult(type: Int, errCode: Int) {
             if (errCode == DfuHandle.RSP_OK) {
                 when (type) {
                     1 -> dfuHandle.init()
                     2 -> dfuHandle.sendPacket()
                     3 -> dfuHandle.check()
                     4 -> {
                         //The upgrade is successful, wait for the
device to restart
                         dfuHandle.endAndRelease()
                     }
                 }
             } else {
                  //Upgrade exception or failure
             }
         }

         override fun onProgress(percent: Int) {
             // file upgrade progress
         }
     }
```

2.3.7Manual measurement

```
 //StartHeartRateRsp parameter description
                private byte type; type 1: heart rate 2: blood
pressure 3: blood oxygen
                private byte errCode; measurement error code 0:
normal 1: measurement failed 2: measurement failed
                private byte value; measurement value: heart rate
or blood oxygen
                private byte sbp; blood pressure sbp
```

```
                private byte dbp; blood pressure dbp

 // manual heart rate
 BleOperateManager.getInstance().manualModeHeart(new
ICommandResponse<StartHeartRateRsp>() {
          @Override
          public void onDataResponse(StartHeartRateRsp
resultEntity) {

          }
      });
 //manual blood pressure
   BleOperateManager.getInstance().manualModeBP(new
ICommandResponse<StartHeartRateRsp>() {
            @Override
            public void onDataResponse(StartHeartRateRsp
resultEntity) {

            }
        });
  //manual blood oxygen
  BleOperateManager.getInstance().manualModeSpO2(new
ICommandResponse<StartHeartRateRsp>() {
            @Override
            public void onDataResponse(StartHeartRateRsp
resultEntity) {

            }
        });
  //manual pressure
  BleOperateManager.getInstance().manualModePressure(new
ICommandResponse<StartHeartRateRsp>() {
            @Override
            public void onDataResponse(StartHeartRateRsp
resultEntity) {

            }
        });
```

2.3.9 Changes in Ring measurement data are proactively reported to the APP

```
 //Add listener

BleOperateManager.getInstance().addOutDeviceListener(ListenerKey.He
```

```
art,myDeviceNotifyListener)

ListenerKey Parameter Description
 public class ListenerKey {
    public static int Heart=1;          Heart
    public static int BloodPressure=2;  Blood Pressure
    public static int BloodOxygen=3;    Blood Oxygen
    public static int Temperature=5;    Temperature
    public static int SportRecord=7;    Sport Record
    public static int All=7;            All
}

Monitoring instructions
    inner class MyDeviceNotifyListener : DeviceNotifyListener() {
        override fun onDataResponse(resultEntity: DeviceNotifyRsp?)
{
            if (resultEntity!!.status == BaseRspCmd.RESULT_OK) {

BleOperateManager.getInstance().removeOthersListener()
                when (resultEntity.dataType) {
                    1 -> {
                        //Watch heart rate test changes
                    }
                    2 -> {
                        //Watch blood pressure test changes
                    }
                    3 -> {
                        //Watch blood oxygen test changes
                    }
                    4 -> {
                        //Changes in watch step counting details
                    }
                    5 -> {
                        //Watch body temperature changes on the day
                    }
                    7 -> {
                        //Generate new exercise records
                    }
                    0x12 -> {
                        //7312 00005200025100003c0000000066
                        AwLog.i(Author.HeZhiYuan,
ByteUtil.bytesToString(resultEntity.loadData))

                        val step = BLEDataFormatUtils.bytes2Int(
                            byteArrayOf(
                                resultEntity.loadData[1],
                                resultEntity.loadData[2],
                                resultEntity.loadData[3]
```

```
                            )
                        )

                        val calorie = BLEDataFormatUtils.bytes2Int(
                            byteArrayOf(
                                resultEntity.loadData[4],
                                resultEntity.loadData[5],
                                resultEntity.loadData[6]
                            )
                        )
                        val distance =
BLEDataFormatUtils.bytes2Int(
                            byteArrayOf(
                                resultEntity.loadData[7],
                                resultEntity.loadData[8],
                                resultEntity.loadData[9]
                            )
                        )
                        deviceNotification(step, distance, calorie)
                    }
                }
            }
        }
    }

    //Remove the listener. It must be removed after registration,
otherwise multiple callbacks will appear.

BleOperateManager.getInstance().removeNotifyListener(ListenerKey.He
art)
    //Remove all listeners

BleOperateManager.getInstance().removeNotifyListener(ListenerKey.Al
l)
```

2.3.10 APP opens exercise type

```
 // status 1 Start movement 2 Pause 3 Continue 4 End 6 Movement
start timestamp
     //Sport type 4 Walking 5 Jumping rope 7 Running 8 Hiking 9
Cycling 10 Other sports 20 Hiking 21 Badminton
     22 Yoga 23 Aerobics 24 Spinning 25 Kayaking 26 Elliptical
machine 27 Rowing machine 28 Table tennis 29 Tennis
     30 Golf 31 Basketball 32 Football 33 Volleyball 34 Rock
climbing 35 Dance 36 Roller skating 60 Outdoor hiking
     CommandHandle.getInstance().executeReqCmd(
```

```
                PhoneSportReq.getSportStatus(

                    1, sportType
                ), gpsResponse
            )

    private var gpsResponse: ICommandResponse<AppSportRsp> =
        ICommandResponse<AppSportRsp> { resultEntity ->
            AwLog.i(Author.HeZhiYuan, resultEntity)
            if (resultEntity != null) {
                    when (resultEntity.gpsStatus) {
                        6 -> {
                            //Exercise start time (Unit second)
                        }

                        2 -> {
                           //Exercise pause
                        }

                        3 -> {
                          // //Exercise continues
                        }

                        4 -> {
                           //Exercise end
                        }
                }
            }
        }


    //Report data during exercise
    //Add motion data reporting and monitoring
     BleOperateManager.getInstance().addSportDeviceListener(0x78,
myDeviceSportNotifyListener)
    //Remove sports data reporting monitoring

BleOperateManager.getInstance().removeSportDeviceListener(0x78)
    //Listening to inner classes
    inner class MyDeviceNotifyListener :
DeviceSportNotifyListener() {
        override fun onDataResponse(resultEntity: DeviceNotifyRsp?)
{
            super.onDataResponse(resultEntity)

                   if (resultEntity!!.status ==
BaseRspCmd.RESULT_OK) {
                //Movement duration, unit seconds
                val sportTime = bytes2Int(
```

```
                val sportTime = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[2],
                        resultEntity.loadData[3]
                    )
                )
                //Exercise real-time heart rate
                val heart = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[4]
                    )
                )

                //The number of steps generated during exercise
will only have data when the exercise type is 4, 7, or 8, otherwise
it will be 0
                val step = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[5],
                        resultEntity.loadData[6],
                        resultEntity.loadData[7]
                    )
                )
                //The distance generated during exercise will only
have data when the exercise type is 4, 7, or 8, and the others will
be 0.
                val distance = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[8],
                        resultEntity.loadData[9],
                        resultEntity.loadData[10]
                    )
                )
                //Calories generated during exercise
                val calorie = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[11],
                        resultEntity.loadData[12],
                        resultEntity.loadData[13]
                    )

                )

                  //Error status during exercise
                val status = bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[1]
                    )
                )
```

```
                )
                val sportType = BLEDataFormatUtils.bytes2Int(
                    byteArrayOf(
                        resultEntity.loadData[0]
                    )
                )
                if (status == 0x03) {
                    //Not wearing
                }
            }
        }
        }


    /**
     * Convert the byte array to int type, with the high byte of
the array first
     *
     * @param data
     * @return
     */
    public static int bytes2Int(byte[] data) {
        int length = data.length;
        int res = 0;
        for (int i = 0; i < length; i++) {
            res |= (data[i] & 0xFF) << (8 * (length - 1 - i));
        }
        return res;
    }
```