- 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
        - Turn your wrist to brighten the screen
        - Do not disturb mode (command does not take effect if operating on the watch)
        - Time format, switch between public and English
        - weather
        - Continuous heart rate, blood oxygen, blood pressure switch
        - Set watch sports goals
        - Set the screen time of the watch, and customize the data type displayed by the watch face
        - Find equipment
        - Find phone features
        - Bracelet photo control
        - music control
        - Set the watch to factory reset
        - Sedentary reminder
        - Drink water reminder
        - Alarm data
        - 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
- 2.3.4 watch face
  - Read the list of watch faces in the watch
  - APP writes the dial to the watch
  - APP delete watch face
- 2.3.5 custom watch face
- 2.3.6 OTA upgrade function:
- 2.3.7Manual measurement
- 2.3.8 One-click measurement

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

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;
}
```

Turn your wrist to brighten the screen

```
    //Read the wrist roll setting in the watch
   CommandHandle.getInstance()
            .executeReqCmd(PalmScreenReq.getReadInstance(),
                ICommandResponse<PalmScreenRsp> {

                })

     //Write wrist roll settings  isEnable:true on, isLeft：true
left hand
      CommandHandle.getInstance()
            .executeReqCmd(PalmScreenReq.getWriteInstance(boolean
isEnable, boolean isLeft),
                ICommandResponse<PalmScreenRsp> {

                })

     PalmScreenReq Parameter Description
     /**
     * Whether the function is enabled
     */
    private boolean isEnable;
    /**
     * Whether to wear on the left hand
     */
    private boolean isLeft;
    /**
     * whether to turn the wrist
     */
    private boolean needPalm;
```

Do not disturb mode (command does not take effect if operating on the watch)

```
   //Read the watch into Do Not Disturb mode
        CommandHandle.getInstance()
            .executeReqCmd(DndReq.getReadInstance(),
                ICommandResponse<DndRsp> {

                })
    //Write Do Not Disturb

CommandHandle.getInstance().executeReqCmd(DndReq.getWriteInstance(b
oolean isEnable,StartEndTimeEntity dndEntity,
      ICommandResponse<DndRsp> {

                        })

    StartEndTimeEntity Parameter Description
    //start hour
    private int startHour;
    //start minute
    private int startMinute;
    //end hour
    private int endHour;
    //end minute
    private int endMinute;

    DndRsp  Parameter Description
    //function switch
    private boolean isEnable;
    //Please refer to StartEndTimeEntity
    private StartEndTimeEntity dndEntity;
```

Time format, switch between public and English

```
  //Read watch parameters
  CommandHandle.getInstance()
          .executeReqCmd(TimeFormatReq.getReadInstance(),
ICommandResponse<TimeFormatRsp> {})

  //write to watch  is24:true 24 false:12  metric:0 Metric
1：Imperial

CommandHandle.getInstance().executeReqCmd(TimeFormatReq.getWriteIns
tance(
          boolean is24, byte metric
      ),
          ICommandResponse<TimeFormatRsp> {

          })

  TimeFormatRsp Parameter Description
  //is24: true：24 hours  false:12hours
  private boolean is24;
   //isMetric: true：Metric  false:Imperial
  private boolean isMetric;
```

weather

```
  //set weather units

CommandHandle.getInstance().executeReqCmd(DegreeSwitchReq.getWriteI
nstance(
        final boolean enable, final boolean isCelsius
    ),
         ICommandResponse<DegreeSwitchRsp> {

         }
    )


    //read weather units
    CommandHandle.getInstance().executeReqCmd(DegreeSwitchReq
getReadInstance(),   ICommandResponse<DegreeSwitchRsp> {})

 Parameter Description
 enable：weather function true on，fase off
 isCelsius：true Celsius fase Fahrenheit
```

```
 //Set the weather information, support the weather in the next 5
days

CommandHandle.getInstance().executeReqCmd(WeatherForecastReq.getWri
teInstance(
                  WeatherForecastBuilder builder),
ICommandResponse<WeatherForecastRsp>() { deviceResp ->
                  if (deviceResp.status == BaseRspCmd.RESULT_OK) {

                  }
             })

 WeatherForecastBuilder Parameter Description
     //index Serial number 0-4 0 is today 1 is tomorrow ... support
up to 5 days
     private int index;
     //The unixtime of the weather is the second value at 0 o'clock
on a certain day
     private long timeStamp;
     //Weather type (0=unknown, 1=sunny, 2=cloudy, 3=rain, 4=snow,
5=haze, 6=lightning)
     private int weatherType;
     //Minimum temperature value (signed, range -128~127)
     private int minDegree;
     //Maximum temperature (with sign, range -128~127)
     private int maxDegree;
     //Humidity value (0~100)
     private int humidity;
     //Whether to bring an umbrella #01=with , others without
     private boolean takeUmbrella;
```

Continuous heart rate, blood oxygen, blood pressure switch

```
     //Read continuous heart rate settings
      CommandHandle.getInstance()
          .executeReqCmd(HeartRateSettingReq.getReadInstance(),
              ICommandResponse<HeartRateSettingRsp> {

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

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

              })
       //Write continuous heart rate switch isEnable: true on,
false: off
      CommandHandle.getInstance().executeReqCmd(
          HeartRateSettingReq.getWriteInstance(boolean isEnable),
          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
```

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
```

Set the screen time of the watch, and customize the data type displayed by the watch face

```
CommandHandle.getInstance().executeReqCmd(DisplayTimeReq.getWriteIn
stance(final int displayTime, final int displayType, final int
alpha),
               ICommandResponse<DisplayTimeRsp> {

               })

Parameter Description
displayTime Bright screen time range【5,8】
displayType Type of data displayed on the main interface 0 steps 1
calories 2 weather 3 heart rate
alpha  Screen brightness, temporarily useless, reserved, write 0 by
default
```

Find equipment

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

Find phone features

```
BleOperateManager.getInstance().addNotifyListener(
       Constants.CMD_FIND_THE_PHONE.toInt(),
       ICommandResponse<FindPhoneRsp>
   )
   FindPhoneRsp Parameter Description
   openOrClose 1: Start search 2: End search
```

Bracelet 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)
```

music control

```
   //Send music information to the watch playing: true play, false
pause progress play progress (default 0)
   //volume  (current * 100 / max) name music song name
    FileHandle.getInstance().executeMusicSend(boolean playing, int
progress, int volume, String name)


BleOperateManager.getInstance().addNotifyListener(Constants.CMD_MUS
IC_COMMAND,new ICommandResponse<MusicCommandRsp>(){

           @Override
           public void onDataResponse(MusicCommandRsp resultEntity)
{

           }
       });
     resultEntity.getAction() Parameter Description
      1: play or pause
      2: Previous song
      3: Next song
      4: Volume up
      5: Volume down
```

- watch hang up

```
BleOperateManager.getInstance().addNotifyListener(Constants.CMD_PHO
NE_NOTIFY,new ICommandResponse<PhoneNotifyRsp>(){

            @Override
            public void onDataResponse(PhoneNotifyRsp resultEntity)
{
                isReject() :true Hang up the phone
            }
        });
```

Set the watch 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
```

Sedentary reminder

```
    //Write a sedentary reminder

CommandHandle.getInstance().executeReqCmd(SetSitLongReq(StartEndTim
eEntity startEndTimeEntity, byte weekMask, int cycle),
ICommandResponse<SimpleStatusRsp> {  })
    //Read sedentary reminders
    CommandHandle.getInstance()

.executeReqCmd(SimpleKeyReq(Constants.CMD_GET_SIT_LONG),
                ICommandResponse<ReadSitLongRsp> { longSit ->

                })

    StartEndTimeEntity parameter description is the same as above
     weekMask repetition period
     Bit0 = 0 means Sunday off, Bit0 = 1 means Sunday open.
     Bit1 = 0 means closed on Monday, Bit1 = 1 means open on
Monday.
     Bit2 = 0 means close on Tuesday, Bit2 = 1 means open on
Tuesday.
     Bit3 = 0 means close on Wednesday, Bit3 = 1 means open on
Wednesday.
     Bit4 = 0 means close on Thursday, Bit4 = 1 means open on
Thursday.
     Bit5 = 0 means close on Friday, Bit5 = 1 means open on Friday.
     Bit6 = 0 means closed on Saturday, Bit6 = 1 means open on
Saturday.
     The cycle reminder interval can only be one of 30, 60, and 90
```

Drink water reminder

```
  //Read the drinking water reminder index is the serial number of
the the drinking water reminder [0, 8]

CommandHandle.getInstance().executeReqCmd(ReadDrinkAlarmReq(index),
ICommandResponse<ReadAlarmRsp>{})
  //write water reminder
  CommandHandle.getInstance().executeReqCmd(
                   SetDrinkAlarmReq(AlarmEntity alarmEntity)
                    null)
             }
  AlarmEntity parameter description
     //Sequence number 0-8
     private int alarmIndex;
     //switch
     private int enable;
     // Reminder hour 0-23
     private int hour;
     //Reminder minute 0-59
     private int minute;
      // Repeat cycle as above
     private byte weekMask;
```

Alarm data

```
Note: Connect the watch to initialize:
LargeDataHandler.getInstance().initEnable() once, refer to the demo
  //read the alarm clock
  LargeDataHandler.getInstance().readAlarmWithCallback(new
IReadAlarmCallback() {}
  //write the alarm clock
  LargeDataHandler.getInstance().writeAlarm(entity);

  AlarmNewEntity parameter description
  //Total number of alarms
  private int total;
   //collection of alarm clocks
  private List<AlarmBean> data;

  AlarmBean parameter description
  //The length of the alarm clock: 4+content to byte[], refer to
demo
  private int alarmLength;
  //bit0~bit6 represents the switch corresponding to Monday to
Sunday, bit7 represents whether the current alarm clock is on
  private int repeatAndEnable;
  //The number of minutes of the day, such as 480 represents 8:00
  private int min;
  //Alarm title, up to 30 bytes
  private String content;
```

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 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;
```

2.3.4 watch face

Note: The watch supports up to 6 dial market dials and one custom dial. Among the 6 dial market dials, there are ones that can be deleted and those that cannot be deleted. The ones that can be deleted can be used to replace other dials. The dials cannot be added repeatedly. sex, can only add watch faces that do not exist in the watch

```
  //callback description: the callback of all operations on the
dial
       inner class Callback : SimpleCallback() {
        //List of dial names returned by the device
        override fun onUpdatePlate(array: MutableList<PlateEntity>)
{
            PlateEntity see description below
        }
         //The progress of the APP releasing the disk to the device
0-100
        override fun onProgress(percent: Int) {

        }
        //APP downloads the disk successfully
        override fun onComplete() {

        }
        //APP delete dial successfully
        override fun onDeletePlate() {

        }
        //Download the watch face or delete the watch face
exception
        override fun onUpdatePlateError(code: Int) {

        }
    }

    PlateEntity Description
    //Whether the dial can be deleted
    public boolean mDelete = false;
    //name of the dial
    public String mPlateName = "";
```

Read the list of watch faces in the watch

```
   //Registry disk operation monitoring
    FileHandle.getInstance().registerCallback(callback)
    //initialization
    FileHandle.getInstance().initRegister()
    //Read the watch face list
    FileHandle.getInstance().startObtainPlate()
```

APP writes the dial to the watch

```
   // set file class
     FileHandle.getInstance().currFileType =
FileHandle.TypeMarketWatchFace
     //Registry disk operation monitoring
     FileHandle.getInstance().registerCallback(callback)
     //initialization
     FileHandle.getInstance().initRegister()
     //Execute file check, binPath is the path of the dial file
     val prepare: Boolean =
FileHandle.getInstance().executeFilePrepare(binPath)
     if (prepare) {
         //Start to issue the watch to the watch, the callback will
return the progress and result
         FileHandle.getInstance().executeFileInit(name, 0x36)
     }
```

APP delete watch face

```
   // set file class
    FileHandle.getInstance().currFileType =
FileHandle.TypeMarketWatchFace
     //Registry disk operation monitoring
    FileHandle.getInstance().registerCallback(callback)
    //initialization
    FileHandle.getInstance().initRegister()
    //Delete the dial name The name of the dial, the watch will
return when reading
    FileHandle.getInstance().executeFileDelete(name)
```

2.3.5 custom watch face

```
//Read custom dial coordinates, font color
LargeDataHandler.getInstance().readCustomizeDial(ILargeDataCallback
<CustomizeDialResp> respILargeDataCallback)
//CustomizeDialResp description
    //The distance between the time and the left border is unit
pixels
    private int timeLeft;
     //The distance between time and upper border is unit pixel
    private int timeTop;
```

```
     //The battery and date are unit pixels away from the left
border
    private int batteryLeft;
    //The unit pixel distance from the battery and date to the
upper border
    private int batteryTop;
    // The data and date are unit pixels away from the left border
    private int dataLeft;
    // The data and date are unit pixels away from the upper border
    private int dataTop;
    //font color
    private int r;
    private int g;
    private int b;

  //Write custom dial coordinates, font color
  LargeDataHandler.getInstance().writeCustomWatch(entity)
  CustomWatchFaceEntity parameter description Element type: 1 for
time 2: for battery and date 3: for data
  type: element type
  x: distance from the left border
  y: distance from the upper boundary
  r: font color R
  g: font color G
  b: font color B

//Customize the dial to replace the back image
 FileHandler.getInstance().customizeWatchFace(Bitmap bitmap,int
width,int height,SimpleCallback callback)
 //Parameter Description
 Bitmap: background image
 width: watch width
 height: watch height
 SimpleCallback callback
 //Deliver the picture progress
 override fun onProgress(percent: Int) {

        }
 // Finished uploading the image
        override fun onComplete() {

        }
```

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) {

            }
        });
```

2.3.8 One-click measurement

```
//Start the measurement (it is best to measure for more than 20
seconds and then call the end, it is recommended 30 seconds)
lastFatigueValue The value of the last fatigue
lastFatigueTime Last test time in milliseconds
  BleOperateManager.getInstance().startOneKey(int
lastFatigueValue,int lastFatigueTime,
ICommandResponse<StartCalcDataRsp> callback)
  //end measurement
  BleOperateManager.getInstance().endOneKey()

  StartCalcDataRsp parameter description
     private byte errCode; 1 and 2 represent measurement failure
     private byte heart; heart rate
     private byte sbp; systolic pressure
     private byte dbp; diastolic blood pressure
     private int fatigue;
     private int spo2; blood oxygen
     private int score; score
```