## IDO sdk 开发文档

**修改记录**：1.0.1版本-----------------第一次创建完成文档-------------------------2022.06.17

SDK 初始化：

在Application 类里面调用：

@Override

public void onCreate() {

BLEManager.onApplicationCreate(this);

InitParam param = new InitParam();

param.isEnableLog = false; //是否开启日志

param.log\_save\_path = LogPathImpl.getInstance().getBleSdkLogPath();//日志存储路径

param.isSaveDeviceDataToDB = false; // 是否开启数据存储到ido 数据库 param.isEncryptedSPData = true; //是否加密sp 数据

param.databaseName = "VeryFitSdk.db";//数据库名字

BLEManager.init(param);

}

## 一 扫描、连接（Scan, connect）

1、在扫描前就要确保有申请以下权限：

Manifest.permission.ACCESS\_FINE\_LOCATION

Manifest.permission.ACCESS\_COARSE\_LOCATION

Manifest.permission.WRITE\_EXTERNAL\_STORAGE

Manifest.permission.READ\_EXTERNAL\_STORAGE

Manifest.permission.BLUETOOTH\_ADMIN

Manifest.permission.BLUETOOTH

2、按以下流程完善修改首次连接绑定的代码：

（1）开始扫描前，要先主动调用断连的方法：BLEManager.disConnect()

（2）5秒钟后开始扫描：

**代码示例：**

//先注册回调监听

BLEManager.registerScanCallBack(new ScanCallBack.ICallBack(){

@Override

Public void onFindDevice(BLEDevice device){

//每扫描到一个手表，就会调用一次，BLEDevice 就是手表的信息

}

})

//调用接口

BLEManager.startScanDevices()

1. 扫描到符合条件的手表后，马上调用停止扫描的方法BLEManager.stopScanDevices()

（4）连接扫描到的手表

**代码示例：**

//先注册回调监听

BLEManager.registerConnectCallBack(new ConnectCallBack.ICallBack(){

@Override

public void onConnectStart(String macAddress) {}

@Override

public void onConnecting(String macAddress) {}

@Override

public void onRetry(int count, String macAddress) {}

@Override

public void onConnectSuccess(String macAddress) {

//连接成功的回调，macAddress是指连接成功的手表mac地址

}

@Override

public void onConnectFailed(ConnectFailedReason failedReason, String macAddress) {}

@Override

public void onConnectBreak(String macAddress) {

//蓝牙连接中断后 就会回调这个方法（比如：手机与手表距离太远）

}

@Override

public void onInDfuMode(BLEDevice bleDevice) {}

@Override

public void onDeviceInNotBindStatus(String macAddress) {}

@Override

public void onInitCompleted(String macAddress) {}

})

//调用接口

BLEManager.connect(bleDevice)

1. 判断设备是否连接的方法：

BLEManager.isConnected() 返回true 代表已连接，false 代表未连接

4. 自动连接

手表与app连接成功后，如果连接出现断线，sdk也会定时去自动重连。如果客户想要自己调用重连方法马上重连，也可以调用如下两个方法：

（1）if (！BLEManager.isConnected()) {

BLEManager.autoConnect();//自动连接最近一次连接的手表

}

（2）if (！BLEManager.isConnected()) {

BLEManager.autoConnect(String macAddress);//自动连接指定mac地址的手表

}

1. 获取当前连接的设备信息：

BLEDevice device = LocalDataManager.getCurrentDeviceInfo();

1. **BT 连接（支持BT 功能的才需要）**

**调用方法：BTManager.connectBT();**

**注意：支持BT功能的手表才调用，此方法用来创建Bt 绑定（sdk 里面已经实现了自动调用，为了防止没有创建绑定成功，app 可以在需要的地方调用）**

supportFunctionInfo.v2\_get\_bt\_addr 的值true 代表支持BT功能

## 二 绑定、解绑(bind ,unbind)

1. **绑定**

连接成功后才去绑定，绑定前可以先判断是否已经绑定，如果未绑定再调用绑定方法。

**代码示例：**

//先注册回调监听

BLEManager.registerBindCallBack(new BindCallBack.ICallBack(){

@Override

Public void onSuccess( ){

//绑定成功的回调

//把设备存储到设备

}

})

//调用接口

if(!BLEManager.isBind()){

BLEManager.bind()

}

1. **解绑**

解绑前如果app与手表处于连接状态，则调用unbind方法，否则调用forceUnbind方法。

**代码示例：**

//先注册回调监听（ 调用 BLEManager.unbind（）这个方法是需要注册）

BLEManager.registerUnbindCallBack(new UnbindCallBack.ICallBack(){

@Override

Public void onSuccess( ){

//解绑成功的回调

//从设备列表删除设备（客户端处理）

BLEManager.disConnect()

}

})

//调用接口

if(BLEManager.isConnected()){

BLEManager.unbind();

} else {

BLEManager.forceUnbind();

BLEManager.disConnect()

//从设备列表删除设备（客户端处理）

}

解绑成功后，正常情况下：手表会有震动。

注意：解绑后，还需要让客户主动调用断连的方法 BLEManager.disConnect()

## 三 同步数据(sync data)

同步所有的数据

1、这个功能只需要使用到两个类:SyncPara 和 BLEManager。下面做具体的说明：

（1）SyncPara

**参数说明：**

|  |  |
| --- | --- |
| **SyncPara** | |
| timeoutMillisecond | 整个同步设置的超时时间，单位毫秒，默认是3分钟 |
| ISyncDataListener | 监听所有数据的回调 |
| ISyncProgressListener | 监听同步进度的回调 |
| isNeedSyncConfigData | 这个成员属性暂时没用到，无需理会 |
|  |  |
|  |  |
|  |  |

由于ISyncDataListener 这个监听的回调方法很多，下面会再进行详细说明：

客户经常用到的就是下面要介绍的四个回调方法，其他方法依此类推。

①onGetSportData 是对当天步数数据的回调。HealthSport是当天所有的步数数据的汇总，这个类里面的成员属性如下表所示：（v2协议）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sportDataId | dId | year | month | day | totalStepCount | totalCalory | totalDistance | totalActiveTime |
|  |  | 年 | 月 | 日 | 总步数 | 总卡路里 | 总距离 | 总活动时间,单位秒 |

|  |  |  |  |
| --- | --- | --- | --- |
| keyword | startTime | timeSpace | date |
| 没有用到 | 开始时间 | 时间间隔 | 日期 |

HealthSportItem 是当天步数的单条数据的详情。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sportDataId | dId | year | month | day | stepCount | activeTime | calory | distance | date |
|  |  | 年 | 月 | 日 | 步数 | 活动时长 | 卡路里 | 运动距离 | 日期 |

isSectionItemData 是否是分页加载的数据。

②onGetSleepData 是对当天睡眠数据的回调。HealthSleep是当天所有睡眠数据的汇总：（V2协议）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| sleepDataId | dId | year | month | day | sleepEndedTimeH | sleepEndedTimeM | totalSleepMinutes |
|  |  | 年 | 月 | 日 | 结束的时间--小时 | 结束的时间--分钟 | 总睡眠时间（分钟） |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lightSleepCount | deepSleepCount | awakeCount | lightSleepMinutes | deepSleepMinutes |
| 浅睡次数 | 深睡次数 | 清醒次数 | 浅睡总时间（分钟） | 深睡总时间（分钟） |

|  |  |
| --- | --- |
| sleepScore | date |
| 睡眠得分 | 日期 |

HealthSleepItem是当天步数的单条数据的详情。

**参数说明：**

|  |  |
| --- | --- |
| **HealthSleepItem** | |
| year | 年 |
| month | 月 |
| day | 日 |
| offsetMinute | 偏移量（单位：分钟），用来计算数据的发生时间（时：分）,  在计算当前时间点时，需要把前面所以偏移量都累加起来（累加值不超过24\*60=1440）;偏移量的最大值为255, 最小值为0 |
| sleepStatus | 当前时间段的睡眠状态 1:清醒 2:浅睡 3：深睡 |
| date | 日期 |
|  |  |

③ onGetHeartRateData 是对当天心率数据的回调。HealthHeartRate是当天所有心率数据的汇总：（v2协议）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| rateDataId | dId | year | month | day | startTime | silentHeart | warmUpThreshold |
|  | 设备id | 年 | 月 | 日 | 开始时间 | 静息心率 | 热身运动阀值 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| burn\_fat\_threshold | aerobic\_threshold | anaerobicThreshold | limit\_threshold | warmUpMins |
| 脂肪燃烧阀值 | 有氧燃烧阀值 | 无氧运动阀值 | 极限燃烧阀值 | 热身运动分钟数 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| burn\_fat\_mins | aerobic\_mins | anaerobicMins | limit\_mins | UserMaxHr |
| 脂肪燃烧分钟数 | 有氧燃烧分钟数 | 无氧燃烧分钟数据 | 极限燃烧分钟数 | 用户最大心率 |

5级心率区间

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Range1 | Range2 | Range3 | Range4 | Range5 | date |
| 等级1 | 等级2 | 等级3 | 等级4 | 等级5 | 日期 |

HealthHeartRateItem是当天心率的单条数据详情：

**参数说明：**

|  |  |
| --- | --- |
| **HealthHeartRateItem** | |
| year | 年 |
| month | 月 |
| day | 日 |
| offsetMinute | 偏移量（单位：分钟），用来计算数据的发生时间（时：分）,  在计算当前时间点时，需要把前面所以偏移量都累加起来（累加值不超过24\*60=1440）;偏移量的最大值为255, 最小值为0 |
| HeartRaveValue | 自动测量模式下，测量得到的心率值,15分钟产生一个值,一天最多96个值 |
| date | 日期 |
|  |  |

④onGetActivityData 是当天所有活动数据的回调。（V2协议运动数据）

HealthActivity：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| activityId | dId | year | month | day | hour | minute | second | hr\_data\_interval\_minute |
|  | 设备id | 年 | 月 | 日 | 时 | 分 | 秒 | 心率数据间隔分钟 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| type | step | durations | calories | distance | avg\_hr\_value | max\_hr\_value |
| 运动类型 | 步数 | 时长(单位：秒) | 卡路里(千卡) | 距离(单位：米) | 平均心率 | 最大心率 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| warmUpMins | burn\_fat\_mins | aerobic\_mins | anaerobicMins | limit\_mins |
| 热身运动分钟数 | 脂肪燃烧时长(分钟) | 心肺锻炼时长(分钟) | 无氧燃烧分钟数据 | 极限锻炼时长(分钟) |

5级心率区间

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| range1 | range2 | range3 | range4 | range5 | date |
| 等级1 | 等级2 | 等级3 | 等级4 | 等级5 | 日期 |

|  |  |
| --- | --- |
| hr\_data\_vlaue | 最大保存2小时,5s一个,最大1440;greendao无法保存这种类型，所以用json字符串方式存储(hr\_data\_vlaue\_json) |
| hr\_data\_vlaue\_json | 存储hr\_data\_vlaue数据的json字符串 |
| items | 保存步数、卡路里、距离的单条数据 |
| items\_json | 保存items数据的json字符串 |

**参数说明：**

|  |  |
| --- | --- |
| **HealthActivity.Item** | |
| steps | 步数 |
| calories | 卡路里 |
| distance | 距离 |
|  |  |
|  |  |
|  |  |
|  |  |

⑤onGetHealthSpO2Data是血氧数据的回调。（v3协议）

HealthSpO2：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | dId | year | month | day | startTime | date |
| 主键 | 设备id | 年 | 月 | 日 | 数据距离0点的开始时间 | 时间 |

HealthSpO2Item：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Id | dId | year | month | day | offset | value | date |
| 主键 | 设备id | 年 | 月 | 日 | 偏移 | 值0-255 | 时间 |

offset 单位默认 s, 但是如果功能表ex\_table\_main8\_v3\_spo2\_off\_change = true, 则单位是min

isSectionItemData ：是否是分页加载数据。

6.onGetHealthPressureData是压力数据的回调。(v3 协议)

HealthPressure：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | dId | year | month | day | startTime | date |
| 主键 | 设备id | 年 | 月 | 日 | 数据距离0点的开始时间 | 时间 |

HealthPressureItem：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Id | dId | year | month | day | offset | value | date |
| 主键 | 设备id | 年 | 月 | 日 | 偏移 | 值0-255 | 时间 |

offset 单位min

isSectionItemData ：是否是分页加载数据。

7.onGetHealthGpsV3Data是对gps健康数据的回调。 (v3协议)

HealthGpsV3：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| year | month | day | hour | minute | second | data\_interval | items | stringItems |
| 年 | 月 | 日 | 时 | 分 | 秒 | 数据间隔 | HealthGpsItemV3 的List集合 | so库返回的原始数据 |

HealthGpsItemV3：

|  |  |
| --- | --- |
| latitude | longitude |
| 纬度 | 经度 |

//注意：gps 数据与运动关联date 关联（把date 转换为long 型的时间戳）

8. onGetHealthSwimmingData是游泳数据的回调。（v3协议）

HealthSwimming：

/\*\*

\* 年

\*/

public int year;

public int month;//月

public int day;//日

public int hour;//时

public int minute;//分

public int second;//秒

/\*\*

\* 类型 0 无效,1 为室内游泳,2 为开阔水域游泳(无详情)

\*/

public int type;

public int calories; //卡路里

public int distance; //距离

/\*\*

\* 游泳总时长（单位：分钟）

\*/

public int duration;

/\*\*

\* 游泳趟数

\*/

public int trips;

/\*\*

\* 平均swolf

\*/

public int averageSWOLF;

/\*\*

\* 总划水次数

\*/

public int totalStrokesNumber;

/\*\*

\* 主泳姿

\* 0x00 : 混合泳; 0x01 : 自由泳; 0x02 : 蛙泳; 0x03 : 仰泳; 0x04 : 蝶泳;

\*/

public int swimmingPosture;

/\*\*

\* 泳池距离

\*/

public int poolDistance;

/\*\*

\* 用户确认的距离

\*/

public int confirmDistance;

public List<HealthSwimmingDetail> items; //详细信息

private long dId;//设备id

private Date date;//时间

public int avg\_speed;//平均配速

public int avg\_frequency;//平均划水频率

HealthSwimmingDetail：

/\*\*

\* 时间,单位秒

\*/

public int duration;//

/\*\*

\* 划水次数

\*/

public int strokesNumber;

/\*\*

\* 划水效率

\*/

public int swolf;

/\*\*

\* 泳姿

\*/

public int swimmingPosture;

/\*\*

\* 距离

\*/

public int distance;

public int frequency; //频率 -- 新增

public int speed; //配速 -- 新增

public int stop\_time;//结束时间

public int difference\_time;//2趟之间的休息时间（单位ms 毫秒

9.onGetHealthHeartRateSecondData v3协议心率数据

HealthHeartRateSecond：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Id | dId | year | month | day | startTime | date | items （Lise<HealthHeartRateSecondItem >） |
| 主键 | 设备id | 年 | 月 | 日 | 起始时间 | 时间 | 心率集合 |

HealthHeartRateSecondItem：

|  |  |  |
| --- | --- | --- |
| offset | heartRateVal |  |
| 偏移值 （单位s） | 心率值 |  |

offset 单位s

1. onGetHealthSleepV3Data 睡眠数据 （V3协议）

HealthSleepV3 字段说明

public int data\_type; //（固件返回）算法定义的返回值： 1 科学睡眠（有REM、 呼吸得分） 2 午睡 4 普通睡眠（非科学睡眠）

//入睡时间

public int fall\_asleep\_year; //日期 年

public int fall\_asleep\_month; // 月

public int fall\_asleep\_day; // 日

public int fall\_asleep\_hour; // //入睡时间 时

public int fall\_asleep\_minte; //入睡时间 分

//起床时间

public int get\_up\_year;

public int get\_up\_month;

public int get\_up\_day;

public int get\_up\_hour; //起床时间 时

public int get\_up\_minte; //起床时间 分

public int total\_sleep\_time\_mins; //睡眠总时长 单位 mins

public int wake\_mins; //总清醒时长, 单位:分

public int light\_mins; //总浅眠时长

public int rem\_mins; //总REM时长

public int deep\_mins; //总深眠时长

public int wake\_count; //清醒次数

public int light\_count; /浅眠次数

public int rem\_count; //眼动次数

public int deep\_count; //深眠次数

public int awrr\_status; //呼吸状况

public int sleep\_score; //睡眠得分

public int breath\_quality; //呼吸质量

public List<HealthSleepV3Item> items;

HealthSleepV3Item 字段说明： 睡眠详细信息

int stage; // 阶段

int duration; //时长 单位秒

1. onGetHealthSportV3Data（HealthSportV3 healthSportV3）— v3健康数据 （v3协议）

HealthSportV3 字段说明：

public int year;

public int month;

public int day;

public int hour;

public int minute;

public int second;

public int start\_time;

public int per\_minute; ////每条数据产生的间隔 分钟 （15分钟默认） 也 有可能是1min,看具体值

public int total\_step; //总步数

public int total\_rest\_activity\_calories; //总活动+静息卡路里

public int total\_distances; //总距离

public int total\_active\_time; //当天的活动时间 单位min

public int total\_activity\_calories; //总活动卡路里

public int item\_count;

public List<Integer> wear\_flag\_array; /每小时佩戴时长，单位分钟，数组长度24

public List<Integer> type; //24小时的状态数据 0 是未知的，1是达标，2是未达标

public List<HealthSportV3Item> items;

public int walk\_goal\_time; //固件实际走动时长 需要配置 单位小时h

HealthSportV3Item 字段说明

public int mode; // 模式00：安静， 01：轻微，10：中等活跃， 11：激烈

public int step\_count; //步数个数

public int active\_time; // 活跃时长 单位分钟， 最大per\_minute参数决定 目前是15分钟 206定制新增：时长间隔 1分钟

public int activity\_calories; //活动卡路里

public int distance; //距离，单位米

public int rest\_activity\_calories; //活动+静息卡路里

1. onGetHealthActivityV3Data(HealthActivityV3 healthActivityV3）— v3运动数据 （v3协议）

HealthActivityV3 字段说明：

/\*\*

\* 年

\*/

public int year;

/\*\*

\* 月

\*/

public int month;//月

/\*\*

\* 日

\*/

public int day;//日

/\*\*

\* 时

\*/

public int hour;//时

/\*\*

\* 分

\*/

public int minute;//分

/\*\*

\* 秒

\*/

public int second;//秒

public int hr\_data\_interval\_minute;

/\*\*

\* 运动类型

\*/

public int type; //运动类型

/\*\*

\* 计划类型

\*/

public int act\_type; //计划类型 0 表示用type，其余type无效，用act\_type 类型//计划类型：0x01：跑步计划3km ，0x02：跑步计划5km ， 0x03：跑步计划10km ， 0x04：半程马拉松训练（二期） ，0x05：马拉松训练（二期）

//0x40:6分钟轻松跑 ,0x41：10分钟轻松跑 ，0x42：15分钟轻松跑 ，0x43：走跑结合初级 ，0x44：走跑结合进阶 ，0x45：走跑结合强化}

//0x80:跑后拉伸

//0x80:跑后拉伸

/\*\*

\* 步数

\*/

public int step; //步数

/\*\*

\* 时长(单位：秒)

\*/

public int durations; //时长

/\*\*

\* 卡路里(千卡)

\*/

public int calories; //卡路里

/\*\*

\* 距离(单位：米)

\*/

public int distance; //距离

/\*\*

\* 平均心率

\*/

public int avg\_hr\_value;//平均心率

/\*\*

\* 最大心率

\*/

public int max\_hr\_value;//最大心率

/\*\*

\* 热身运动分钟数

\*/

public int warmUpMins; //热身运动分钟数

public int warm\_up\_time;//热身运动时长（秒）

/\*\*

\* 脂肪燃烧时长(单位：分钟)

\*/

public int burn\_fat\_mins; //脂肪燃烧时长

public int fat\_burning\_time; //脂肪燃烧时长（秒）

/\*\*

\* 心肺锻炼时长(单位：分钟)

\*/

public int aerobic\_mins; //心肺锻炼时长

public int aerobic\_exercise\_time;//心肺锻炼时长（秒）

/\*\*

\* 无氧燃烧分钟数据

\*/

public int anaerobicMins; //无氧燃烧分钟数据

public int anaerobic\_exercise\_time;//无氧燃烧时长(秒)

/\*\*

\* 极限锻炼时长(单位：分钟)

\*/

public int limit\_mins; //极限锻炼时长（分钟）

public int extreme\_exercise\_time;//极限锻炼时长(秒)

public int training\_offset;//训练的课程日期偏移 从零开始

public List<ActionTemp> action\_temp;//动作详情

public static class ActionTemp{

public int type;//动作类型（十进制） 1快走；2慢跑；3中速跑；4快跑 ；

//71左腿前测拉伸； 72右腿前测拉伸； 73左腿后侧拉伸；74右腿后侧拉伸；75左腿弓步拉伸；76右腿弓步拉伸；77左腿内侧拉伸；78右腿内侧拉伸；79左侧小腿拉伸；80右侧小腿拉伸

public int actual\_time; //实际运动时间

public int goal\_time; //目标时间 单位秒

public int heart\_value; //心率控制值

}

/\*\*

\* 5s 一个心率数据

\*/

public int[] hr\_data\_vlaue; //最大保存2小时,5s一个,最大1440;greendao无法保存这种类型，所以用json字符串方式存储(hr\_data\_vlaue\_json)

public List<Item> items;

public int grade;////血氧等级 0x00 无等级，0x01:低等 0x02:业余 0x03:一般 0x04：平均 0x05：良好 0x06：优秀 0x07：专业

public static class Item{

public int steps;

public int calories;

public int distance;

}

/\*\*

\* 最快公里配速

\*/

public int fast\_km\_speed;

public List<ItemKMSpeed> items\_km\_speed;

public static class ItemKMSpeed{

public int second;

}

public List<Integer> frequency\_items;

/\*\*

\* 英里配速

\*/

public List<Integer> items\_mi\_speed;

/\*\*

\* 公里配速

\*/

public int km\_speed;

/\*\*

\* 平均配速

\*/

public int avg\_speed;

/\*\*

\* 最大配速

\*/

public int max\_speed;//

/\*\*

\* 平均步频

\*/

public int avg\_step\_frequency;//

/\*\*

\* 最大步频

\*/

public int max\_step\_frequency;//

/\*\*

\* 平均步幅

\*/

public int avg\_step\_stride;//

/\*\*

\* 最大步幅

\*/

public int max\_step\_stride;//

/\*\*

\* 0:无效， 1 : app发起的运动， 2：手表发起的运动

\*/

public int sport\_start\_type; //

/\*\*

\* 手环是否连接app，用app评产生何种报告， 1是连接，0是未连接

\*/

public int connect\_app;

public int avg\_pace\_speed;//平均配速 传过来的是s钟， 比如361 361/60=6分 余数是1s 6''1' , 公里和英里是按照 英里= 公里\*1609/1000f

public int fast\_pace\_speed;//最快配速

public int training\_effect; //训练效果； 单位：无： 范围： 1.0 - 5.0 （扩大10倍传输）

public int vO2max; //最大摄氧量； 单位：毫升/公斤/分钟； 范围 0-80

public int recovery\_time\_year; // 恢复时间点年

public int recovery\_time\_mon; // 恢复时间点月

public int recovery\_time\_day; // 恢复时间点日

public int recovery\_time\_hour; // 恢复时间点时

public int recovery\_time\_min; // 恢复时间点分

public int recovery\_time\_s; // 恢复时间点秒

public int min\_hr\_value; //最小心率值

public List<Integer> pace\_speed\_items;//实时配速

public List<Integer> paddle\_number\_items;//桨次数组 一分钟保存一次

public List<Integer> paddle\_frequency\_items;//桨频数组 一分钟保存一次 存平均值

public List<Integer> tread\_frequency\_items;//踏频数组 一分钟保存一次 存平均值

public int paddle\_number\_count; //桨次个数

public int paddle\_frequency\_count; //桨频个数

public int tread\_frequency\_count; //踏频个数

public int end\_month; //

public int end\_day; //踏频个数

public int end\_hour; //踏频个数

public int end\_minute; //踏频个数

public int act\_count;//action\_temp的个数 动作个数

public int completion\_rate;// 动作完成率 0—100

public int hr\_completion\_rate; //心率控制率 0—100

public int in\_class\_calories; //课程内运动热量 单位千卡

（2）BLEManager：提供给客户使用SDK方法的管理类。

（3）了解以上两个类后，就可以开始实现同步所有数据的功能了：

SyncPara syncPara = new SyncPara();  
syncPara.iSyncDataListener = new ISyncDataListener(){  
 @Override  
 public void onGetSportData(HealthSport healthSport, List<HealthSportItem> items, boolean isSectionItemData) {  
 //todo 拿到数据后，就可以在这里处理相关业务逻辑，数据展示以及数据存储（其他回调方法类似）  
 }

@Override  
public void onGetSleepData(HealthSleep healthSleep, List<HealthSleepItem> items) {  
}  
  
@Override  
public void onGetHeartRateData(HealthHeartRate healthHeartRate, List<HealthHeartRateItem> items, boolean isSectionItemData) {  
 }  
  
@Override  
public void onGetBloodPressureData(HealthBloodPressed healthBloodPressed, List<HealthBloodPressedItem> items, boolean isSectionItemData) {  
 }  
  
@Override  
public void onGetActivityData(HealthActivity healthActivity) {  
 }  
  
@Override  
public void onGetGpsData(HealthGps healthGps, List<HealthGpsItem> healthGpsItems, boolean isSectionItemData) {  
 }  
  
@Override  
public void onGetHealthSpO2Data(HealthSpO2 healthSpO2, List<HealthSpO2Item> itemList, boolean isSectionItemData) {  
 }  
  
@Override  
public void onGetHealthPressureData(HealthPressure healthPressure, List<HealthPressureItem> itemList, boolean isSectionItemData) {  
 }  
  
@Override //v3协议心率回调  
public void onGetHealthHeartRateSecondData(HealthHeartRateSecond healthHeartRateSecond, boolean isSectionItemData) {  
 }  
  
@Override  
public void onGetHealthSwimmingData(HealthSwimming healthSwimming) {  
 }  
  
@Override  
public void onGetHealthActivityV3Data(HealthActivityV3 healthActivityV3) {  
 }  
  
@Override  
public void onGetHealthSportV3Data(HealthSportV3 healthSportV3) {  
 }  
  
@Override  
public void onGetHealthSleepV3Data(HealthSleepV3 healthSleepV3) {  
 }  
  
@Override  
public void onGetHealthGpsV3Data(HealthGpsV3 healthGpsV3) {  
 }  
  
@Override  
public void onGetHealthNoiseData(HealthNoise healthNoise) {  
 }  
  
@Override  
public void onGetHealthTemperature(HealthTemperature healthTemperature) {  
 tvSyncResult.setText("onGetHealthTemperature is ok");  
}  
  
@Override  
public void onGetHealthBloodPressure(HealthBloodPressureV3 healthBloodPressureV3) {  
 tvSyncResult.setText("onGetHealthBloodPressure is ok");  
}  
  
@Override  
public void onGetHealthRespiratoryRate(HealthRespiratoryRate healthRespiratoryRate) {  
  
}

}

syncPara.iSyncProgressListener = new ISyncProgressListener(){  
 @Override  
 public void onStart() { }  
  
 @Override  
 public void onProgress(int progress) { }  
  
 @Override  
 public void onSuccess() { }  
  
 @Override  
 public void onFailed() { }  
};

BLEManager.syncAllData(syncPara);

## 四 获取信息(get info)

### 设备信息(devicein info)

**接口名称：**

BLEManager.getBasicInfo

**参数说明：**

|  |  |
| --- | --- |
| **BasicInfo** | |
| deviceId | 设备id号 |
| firmwareVersion | 固件版本号 |
| battStatus | 电池状态 |
| energe | 剩余电量 |
|  |  |
|  |  |
|  |  |

**代码示例：**

方式一：//先注册回调监听

BLEManager.registerGetInfoCallBack(new GetDeviceInfoCallBack.ICallBack(){

@Override

Public void onGetBasicInfo(BasicInfo basicInfo){

//这里返回设备信息

}

})

//调用接口

BLEManager.getBasicInfo()

方式二：sdk有存储basicinfo 信息

BasicInfo basicInfo = LocalDataManager.getBasicInfo();

### 获取功能表(get Funcition)

**接口名称：**

BLEManager.getFunctionTables()

**代码示例：**

方式一://先注册回调监听

BLEManager.registerGetDeviceInfoCallBack(new GetDeviceInfoCallBack.ICallBack(){

@Override

Public void onGetFunctionTable(SupportFunctionInfo supportFunctionInfo){

//这里返回设备支持的功能表

}

})

//调用接口

BLEManager.getFunctionTables()

方式二：

sdk 也会保存功能表，在首次绑定的时候sdk 会自动获取，用户调用这个

SupportFunctionInfo info = LocalDataManager.getSupportFunctionInfo();就可以拿到功能表（推荐用这个）

### 获取所有的运动类型（24种）

public static List<Integer> getSupportMotionTypeList() {

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

ArrayList<Integer> list = new ArrayList<>();

if (functionInfo == null) {

return list;

}

CommonLogUtil.printAndSave(LogPathImpl.getInstance().getDeviceLogPath(), "获取支持的运动类型列表===" + functionInfo.toString());

//户外跑步

if (functionInfo.outdoor\_run) {

list.add(SportType.SOPRT\_TYPE\_OUTDOOR\_RUN);

}

//跑步或者室内跑步

if (functionInfo.indoor\_run) {

list.add(SportType.SOPRT\_TYPE\_INDOOR\_RUN);

}

//跑步

if (functionInfo.sport\_type0\_run) {

list.add(SportType.SPORT\_TYPE\_RUN);

}

//户外走路

if (functionInfo.outdoor\_walk) {

list.add(SportType.SOPRT\_TYPE\_OUTDOOR\_WALK);

}

//走路

if (functionInfo.sport\_type0\_walk) {

list.add(SportType.SPORT\_TYPE\_WALK);

}

//走路或者室内走路

if (functionInfo.indoor\_walk) {

list.add(SportType.SOPRT\_TYPE\_INDOOR\_WALK);

}

//徒步

if (functionInfo.sport\_type0\_on\_foot) {

list.add(SportType.SPORT\_TYPE\_ONFOOT);

}

//户外骑行

if (functionInfo.outdoor\_cycle) {

list.add(SportType.SOPRT\_TYPE\_OUTDOOR\_CYCLE);

}

//骑行

if (functionInfo.sport\_type0\_by\_bike) {

list.add(SportType.SPORT\_TYPE\_CYCLING);

}

//板球

if (functionInfo.cricket) {

list.add(SportType.SPORT\_TYPE\_CRICKET);

}

//瑜伽

if (functionInfo.sport\_type2\_yoga) {

list.add(SportType.SPORT\_TYPE\_YOGA);

}

//室内骑行

if (functionInfo.indoor\_cycle) {

list.add(SportType.SOPRT\_TYPE\_INDOOR\_CYCLE);

}

//健身

if (functionInfo.sport\_type1\_fitness) {

list.add(SportType.SPORT\_TYPE\_FITNESS);

}

//游泳

if (functionInfo.sport\_type0\_swim) {

list.add(SportType.SPORT\_TYPE\_SWIM);

}

//泳池游泳

if (functionInfo.pool\_swim) {

list.add(SportType.SOPRT\_TYPE\_POOL\_SWIM);

}

//开发性水域游泳

if (functionInfo.open\_water\_swim) {

list.add(SportType.SOPRT\_TYPE\_WATER\_SWIM);

}

//划船机

if (functionInfo.rower) {

list.add(SportType.SOPRT\_TYPE\_ROWER);

}

//椭圆机

if (functionInfo.elliptical) {

list.add(SportType.SOPRT\_TYPE\_ELLIPTICAL);

}

//登山

if (functionInfo.sport\_type0\_mountain\_climbing) {

list.add(SportType.SPORT\_TYPE\_CLIMB);

}

//羽毛球

if (functionInfo.sport\_type0\_badminton) {

list.add(SportType.SPORT\_TYPE\_BADMINTON);

}

//动感单车

if (functionInfo.sport\_type1\_spinning) {

list.add(SportType.SPORT\_TYPE\_DYNAMIC);

}

//跑步机

if (functionInfo.sport\_type1\_treadmill) {

list.add(SportType.SPORT\_TYPE\_TREADMILL);

}

//篮球

if (functionInfo.sport\_type2\_basketball) {

list.add(SportType.SPORT\_TYPE\_BASKETBALL);

}

//足球

if (functionInfo.sport\_type2\_footballl) {

list.add(SportType.SPORT\_TYPE\_SOCKER);

}

//网球

if (functionInfo.sport\_type2\_tennis) {

list.add(SportType.SPORT\_TYPE\_TENNISBALL);

}

//跳舞

if (functionInfo.sport\_type3\_dance) {

list.add(SportType.SPORT\_TYPE\_DANCING);

}

//椭圆球

if (functionInfo.sport\_type1\_ellipsoid) {

list.add(SportType.SPORT\_TYPE\_ELLIPOSID);

}

//仰卧起坐

if (functionInfo.sport\_type1\_sit\_up) {

list.add(SportType.SPORT\_TYPE\_SIT\_UP);

}

//俯卧撑

if (functionInfo.sport\_type1\_push\_up) {

list.add(SportType.SPORT\_TYPE\_PUSHUP);

}

//哑铃

if (functionInfo.sport\_type1\_dumbbell) {

list.add(SportType.SPORT\_TYPE\_DUMBBELLS);

}

//举重

if (functionInfo.sport\_type1\_weightlifting) {

list.add(SportType.SPORT\_TYPE\_LIFTING);

}

//健身操

if (functionInfo.sport\_type2\_bodybuilding\_exercise) {

list.add(SportType.SPORT\_TYPE\_AEROBICS);

}

//跳绳

if (functionInfo.sport\_type2\_rope\_skipping) {

list.add(SportType.SPORT\_TYPE\_ROPE);

}

//排球

if (functionInfo.sport\_type2\_volleyball) {

list.add(SportType.SPORT\_TYPE\_VOLLEYBALL);

}

//乒乓球

if (functionInfo.sport\_type2\_table\_tennis) {

list.add(SportType.SPORT\_TYPE\_PINGPONG);

}

//高尔夫

if (functionInfo.sport\_type3\_golf) {

list.add(SportType.SPORT\_TYPE\_GOLF);

}

//棒球

if (functionInfo.sport\_type3\_baseball) {

list.add(SportType.SPORT\_TYPE\_BASEBALL);

}

//滑雪

if (functionInfo.sport\_type3\_skiing) {

list.add(SportType.SPORT\_TYPE\_SKI);

}

//轮滑

if (functionInfo.sport\_type3\_roller\_skating) {

list.add(SportType.SPORT\_TYPE\_ROLLER);

}

//普拉提

if (functionInfo.pilates) {

list.add(SportType.SPORT\_TYPE\_PILATES);

}

//尊巴舞

if (functionInfo.zumba) {

list.add(SportType.SPORT\_TYPE\_ZUMBA);

}

//核心训练

if (functionInfo.sport\_type3\_core\_training) {

list.add(Sport100Type.SPORT\_TYPE\_CORE\_TRAINING);

}

//传统力量训练

if (functionInfo.sport\_type\_traditional\_strength\_training) {

list.add(Sport100Type.SPORT\_TYPE\_TRADITIONAL\_STRENGTH\_TRAINING);

}

//功能性力量训练

if (functionInfo.sport\_type3\_strength\_training) {

list.add(Sport100Type.SPORT\_TYPE\_FUNCTIONAL\_STRENGTH\_TRAINING);

}

//高强度间歇训练

if (functionInfo.HIIT) {

list.add(SportType.SOPRT\_TYPE\_HIIT);

}

//整理放松

if (functionInfo.sport\_type3\_tidy\_up\_relax) {

list.add(Sport100Type.SPORT\_TYPE\_ORGANIZE\_AND\_RELAX);

}

//其他、自由锻练

if (functionInfo.sport\_type0\_other) {

list.add(SportType.SPORT\_TYPE\_OTHER);

}

return list;

}

### 获取显示的运动类型（v3协议）

**接口名称：**

BLEManager.getSupportSportInfoV3()

**代码示例：**

方式一://先注册回调监听

BLEManager.registerGetDeviceParaCallBack(new GetDeviceParaCallBack.ICallBack(){

@Override

Public void onGetSupportSportInfoV3(SupportSportInfoV3 supportSportInfoV3) {

//这里返回设备支持的运动类型

}

})

//调用接口

BLEManager.getSupportSportInfoV3()

SupportSportInfoV3 类说明：

public int defaultShowNum;//默认显示的数量

public int isSupportsSort;//是否支持默认排序,0x0 不支持,0x01 支持

public int maxShowNum;//最大支持的数量

public int minShowNum;//最小支持的数量

public List<Integer> sportTypes;//运行类型,is\_supports\_sort=1时有效,数量为 default\_show\_num,空位填0

### 获取快捷应用（v3协议）

**接口名称：**

BLEManager.getMenuList()

**代码示例：**

方式一://先注册回调监听

BLEManager.registerGetDeviceParaCallBack(new GetDeviceParaCallBack.ICallBack(){

@Override

Public void onGetMenuList(MenuList.DeviceReturnInfo deviceReturnInfo) {

//这里返回设备的快捷应用

}

})

//调用接口

BLEManager.getMenuList()

DeviceReturnInfo类说明：

public int minShowNum; //最小展示数量

public int maxShowNum; // 最大展示数量

public int maxNum; //最大数量

public int currentShowNum; //当前展示数量,代表items 里面的前面currentShowNum个是当前显示的快捷应用

public List<MenuList.DeviceReturnInfo.Item> items; //快捷应用集合

### 获取GPS状态

**接口名称：**

BLEManager.getGpsStatus()

**代码示例：**

方式一://先注册回调监听

BLEManager.registerGetGpsInfoCallBack(new GpsCallBack.IGetGpsInfoCallBack(){

@Override

public void onGetGpsStatus(GpsStatus gpsStatus) {

//这里返回设备的gps 状态

// GPS 非空闲状态，不处理

if (GpsStatus.STATUS\_IDLE != gpsStatus.gps\_run\_status) {

saveAgpsLog("checkDeviceGpsStatus , gps is not in idle status");

return;

}

//下载GPS文件 ,具体参考（十 Apgs 更新）

}

})

//调用接口

BLEManager.getGpsStatus()

DeviceReturnInfo类说明：

public int minShowNum; //最小展示数量

public int maxShowNum; // 最大展示数量

public int maxNum; //最大数量

public int currentShowNum; //当前展示数量,代表items 里面的前面currentShowNum个是当前显示的快捷应用

public List<MenuList.DeviceReturnInfo.Item> items; //快捷应用集合

### 获取ble bt 版本号（get version）

先根据功能表盘判断是否支持：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_support\_set\_v3\_notify\_add\_app\_name 返回ture 代表支持

**接口名称：**

BLEManager.getFirmwareAndBt3Version();

**代码示例：**

//先注册回调监听

BLEManager.registerGetDeviceParaCallBack(new GetDeviceParaCallBack.ICallBack(){

@Override

public void onGetFirmwareAndBt3Version(FirmwareAndBt3Version firmwareAndBt3Version) {

//这里返回设备的版本号

}

})

//调用接口

BLEManager.getFirmwareAndBt3Version()

FirmwareAndBt3Version 类说明：

public int firmware\_version1;//固件版本version1

public int firmware\_version2;//固件版本version2

public int firmware\_version3;//固件版本version3

public int BT\_flag;//BT版本生效标志位; 0：无效，1：说明固件有对应的BT固件

public int BT\_version1;//BT的版本version1

public int BT\_version2;//BT的版本version2

public int BT\_version3;//BT的版本version3

public int BT\_match\_version1;//BT的所需要匹配的版本version1

public int BT\_match\_version2;//BT的所需要匹配的版本version2

public int BT\_match\_version3;//BT的所需要匹配的版本version3

注意：firwareVersion = firmware\_version1 +firmware\_version2 +firmware\_version3; //firmware\_version2,firmware\_version3 小于10，则前面需要补0显示

BT\_version ,BT\_match\_version 跟firwareVersion 处理方式一样

当BT\_version 与 BT\_mach\_version 不一致时需要进行升级，具体看升级BT 文档(十二 BT升级)

### 手表通知app

DeviceParaChangedCallBack.ICallBack changeC = new DeviceParaChangedCallBack.ICallBack() {

@Override

public void onChanged(DeviceChangedPara deviceParaChange) {

}

};

BLEManager.registerDeviceParaChangedCallBack(changeC);

BLEManager.unregisterDeviceParaChangedCallBack(changeC);(ondestory 的时候调用)

此监听主要用来监听设备的一些状态改变，下面是type 说明，需要用到的场景可以通过对应type 做判断

DeviceChangedPara 对象说明:

/\*\*

\* data\_type

\* | 值 | 说明 | 备注 |

\* | ---- | ------------------------------------- | ---- |

\* | 0x0 | 无效 | |

\* | 1 | 手环已经解绑 | |

\* | 2 | 心率模式改变 | |

\* | 3 | 血氧产生数据，发生改变 | |

\* | 4 | 压力产生数据，发生改变 | |

\* | 5 | Alexa识别过程中退出 | |

\* | 6 | 固件发起恢复出厂设置，通知app弹框提醒 | |

\* | 7 | app需要进入相机界面（TIT01定制） | |

\* | 8 | sos事件通知（205土耳其定制） | |

\* | 9 | alexa设置的闹钟，固件修改，需要发送对应的通知位给app，app收到后发送获取V3的闹钟命令|

\* | 10 | 固件有删除日程提醒，app这边( cmd = 0x33 , cmd\_id = 0x36 )查询列表，要更新对应的列表数据|

\* | 11 | 固件端有修改对应的表盘子样式，通知app获取（command\\_id为0x33， key为 0x5000）|

\* | 12 |固件通知ios更新通知图标和名字|

\* | 13 |固件通知app图标已经更新，通知app获取已经更新的图标状态|

\* | 14 |固件请求重新设置天气，app收到收，下发天气数据|

\* |15 |固件步数每次增加2000步，设备请求app同步数据，app调用同步接口|

\* |16 |固件探测到睡眠结束，请求app同步睡眠数据，app调用同步接口同步|

\* |17 |固件三环数据修改，通知app更新三环数据|

\* |18 |固件充满电完成发送提醒，app收到后通知栏显示设备充电完成|

\* |19 |结束运动后，手动测量心率后，手动测量血氧后，手动测量压力后，设备自动请求同步，先检查链接状态，未连接本次同步不执行，满足下个自动同步条件后再次判断发起同步请求|

\* |20 |固件修改 心率通知状态类型、压力通知状态类型、血氧通知状态类型、生理周期通知状态类型、健康指导通知状态类型、提醒事项通知状态类型通知app更新心率、压力、血氧、生理周期、健康指导、提醒事项通知状态类型|

\* |21 |固件压力值计算完成，通知app获取压力值|

\* |22 |固件通知app，固件压力校准失败(固件退出测量界面/检测失败/检测超时/未佩戴)|

\* |23 |固件产生心率过高或者过低提醒时，通知app获取心率数据|

\* |24 |固件通知app bt蓝牙已连接|

\* |25 |固件通知app bt蓝牙断开连接|

\* |26 |固件蓝牙通话开始|

\* |27 |固件蓝牙通话结束|

\* |28 |新版本固件每隔4分30秒发送一个通知命令用于修复ios 会显示离线的问题|

\* |29 |通知app运动开始（作用于拦截表盘传输同26）|

\* |30 |通知app运动结束（作用于拦截表盘传输同27）|

\* |31 |固件重启后重连发送通知给app （app收到通知需要获取固件版本信息）|

\* |32 |设备空闲时（没有使用aleax），需要上报通知给app（时间间隔为1小时）|

\* |33 |固件整理空间完成通知app 继续下传表盘文件|

\*/

public int dataType;

### 手表控制app

DeviceControlAppCallBack.ICallBack callBack= new DeviceControlAppCallBack.ICallBack() {

@Override

public void onControlEvent(DeviceControlAppCallBack.DeviceControlEventType deviceControlEventType, int var2) {

//控制事件

}

@Override

public void onFindPhone(boolean b, long l) {

//收到寻找手机的请求

b true:开始寻找手机 false : 停止寻找手机

l 超时时间

}

@Override

public void onOneKeySOS(boolean b, long l) {

//sos 事件

b true:开始sos 事件 false : 停止sos事件

}

@Override

public void onAntiLostNotice(boolean b, long l) {

}

};

BLEManager.registerDeviceControlAppCallBack(callBack); BLEManager.unregisterDeviceControlAppCallBack(callBack);(ondestory 的时候调用)

DeviceControlEventType ：说明

/\*\*

\* 控制事件

\* <br/>

\* Control type

\*/

public enum DeviceControlEventType {

/\*\*

\* 音乐开始

\* <br/>

\* Start

\*/

START,

/\*\*

\* 音乐暂停

\* <br/>

\* Pause

\*/

PAUSE,

/\*\*

\* 音乐停止

\* <br/>

\* Stop

\*/

STOP,

/\*\*

\* 前一首

\* <br/>

\* Previous

\*/

PREVIOUS,

/\*\*

\* 下一首

\* <br/>

\* Next

\*/

NEXT,

/\*\*

\* 拍照，单拍

\* <br/>

\* Take one photo

\*/

TAKE\_ONE\_PHOTO,

/\*\*

\* 连拍

\* <br/>

\* Take multi photo

\*/

TAKE\_MULTI\_PHOTO,

/\*\*

\* 音量加

\* <br/>

\* Volume up

\*/

VOLUME\_UP,

/\*\*

\* 音量减

\* <br/>

\* Volume down

\*/

VOLUME\_DOWN,

/\*\*

\* 打开相机

\* <br/>

\* Open camera

\*/

OPEN\_CAMERA,

/\*\*

\* 关闭相机

\* <br/>

\* Close camera

\*/

CLOSE\_CAMERA,

/\*\*

\* 接听电话

\* <br/>

\* Answer phone

\*/

ANSWER\_PHONE,

/\*\*

\* 拒接电话

\* <br/>

\* Reject phone

\*/

REJECT\_PHONE,

/\*\*

\* 手机静音

\* <br/>

\* Reject phone

\*/

MUTE\_PHONE,

/\*\*

\* 请求系统配对

\*/

REQUEST\_PAIRED,

/\*\*

\* 控制音乐百分比

\*/

VOLUME\_PERCENTAGE

}

### 获取屏幕亮度信息

**接口名称：**

BLEManager.getScreenBrightness()

**代码示例：**

方式一://先注册回调监听

BLEManager.registerGetDeviceParaCallBack(new GetDeviceParaCallBack.ICallBack(){

@Override

public void onGetScreenBrightness(ScreenBrightness screenBrightness) {

}

})

//调用接口

BLEManager.getScreenBrightness()

## 五 设置参数（parameter setting）

### 设置时间(settime)

**接口名称：**

（1）BLEManager.setTime() //此方法会直接获取手机的时间设置给设备

（2）BLEManager.setTime(SystemTime time)

**example：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.TIME) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

@Override

public void onFailed(SettingCallBack.SettingType settingType) {

}

})

//调用接口

SystemTime systemTime = new SystemTime();

systemTime.year = 2021;

systemTime.monuth = 08;

systemTime.day = 26;

systemTime.hour = 17;

systemTime.minute = 30;

systemTime.second = 50;

BLEManager.setTime(systemTime);

### 寻找手机开关(setfindPhone)

**接口名称：**

（1）BLEManager.setFindPhoneSwitch(boolean isOpen)给设备

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.FIND\_PHONE\_SWITCH) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

@Override

public void onFailed(SettingCallBack.SettingType settingType) {

}

})

//调用接口

BLEManager.setFindPhoneSwitch(true);

### 设置闹钟(set alarm)

是否支持V3闹钟： true 支持，false 不支持

LocalDataManager.getSupportFunctionInfo().ex\_table\_main8\_v3\_sync\_alarm

**v2协议：（**ex\_table\_main8\_v3\_sync\_alarm = false **）**

1. **获取闹钟列表：**

public List<Alarm> getAlarmList() {  
 List<Alarm> alarmList = LocalDataManager.*getAlarm*();  
 List<Alarm> list = new ArrayList<>();  
 if (alarmList != null) {  
 for (Alarm alarm : alarmList) {  
 if (alarm.getAlarmStatus() == Alarm.*STATUS\_DISPLAY*) {  
 list.add(alarm);  
 }  
 }  
 }

return list;  
}

1. **设置闹钟**

**接口名称：**

BLEManager.setAlarm(List<Alarm> alarmList)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.ALARM) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

Alarm mAlarm = new Alarm();

mAlarm.setAlarmHour(hour);

mAlarm.setAlarmMinute(min);

mAlarm.setAlarmType(alarmType);//闹钟类型

mAlarm.setOn\_off(onOff);//闹钟开关

mAlarm.setWeekRepeat(weeks);//周重复的星期

List<Alarm> alarms = new ArrayList<>();// 这个集合用上面获取到的集合（getAlarmList），不然原来的闹钟就不存在了

alarms.add(mAlarm);

设置闹钟的方法，需要对闹钟定义不重复的id,然后再下发给设备

public void sendAlarmClock2Device(@NonNull List<Alarm> alarmList) {  
 for (int i = 0; i < alarmList.size(); i++) {  
 Alarm alarm = alarmList.get(i);  
 if (alarm == null) {  
 continue;  
 }  
 alarm.setAlarmId(i + 1);  
 }  
 BLEManager.*setAlarm*(alarmList);  
}

**v3协议：（**ex\_table\_main8\_v3\_sync\_alarm = true**）**

1. 获取闹钟

private GetDeviceParaCallBack.ICallBack getParaCallBack = new GetDeviceParaCallBack.ICallBack() {  
 @Override  
 public void onGetAlarmV3(List<AlarmV3> alarmV3List) {

//闹钟列表

}

}

BLEManager.*registerGetDeviceParaCallBack*(mICallBack);  
BLEManager.*getAlarmV3*();

1. 设置闹钟

**接口名称：**

BLEManager.setAlarmV3(List<Alarm> alarmList)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.ALARM\_V3) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

Alarm mAlarm = new Alarm();

mAlarm.setAlarmHour(hour);

mAlarm.setAlarmMinute(min);

mAlarm.setAlarmType(alarmType);//闹钟类型

mAlarm.setOn\_off(onOff);//闹钟开关

mAlarm.setWeekRepeat(weeks);//周重复的星期

List<Alarm> alarms = new ArrayList<>();// 这个集合用上面获取到的集合（alarmV3List），不然原来的闹钟就不存在了

alarms.add(mAlarm);

设置闹钟的方法，需要对闹钟定义不重复的id,然后再下发给设备

public void sendAlarmClock2Device(@NonNull List<Alarm> alarmList) {  
 for (int i = 0; i < alarmList.size(); i++) {  
 Alarm alarm = alarmList.get(i);  
 if (alarm == null) {  
 continue;  
 }  
 alarm.setAlarmId(i + 1);  
 }  
 BLEManager.*setAlarmv3*(alarmList);  
}

注意事项：闹钟的个数有限制， functionInfo.alarmCount 这个是闹钟限制的个数，超过了会设置不成功

### 设置步数目标(set step goal)

**接口名称：**

BLEManager.setGoal(Goal goal)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.GOAL) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

Goal goal = new Goal();

goal.sport\_step = 10000;

BLEManager.setGoal(goal);

### 重启设备(reboot)

**接口名称：**

BLEManager.reBoot()

**代码示例：**

//先注册回调监听

BLEManager.registerRebootCallBack(new registerRebootCallBack.ICallBack(){

@Override

public void onSuccess() {

}

public void onFailed() {

}

})

//调用接口

BLEManager.reBoot()

### 恢复出厂设置(RestoreFactory)

**接口名称：**

BLEManager.setRestoreFactory();

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.RESTORE\_FACTORY) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

BLEManager.setRestoreFactory();

### 设置抬腕亮屏(setUpHandGesture)

**接口名称：**

BLEManager.setUpHandGesture(UpHandGesture upHandGesture）

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.UP\_HAND\_GESTURE) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

UpHandGesture upHandGesture = LocalDataManager.getUpHandGesture();

if (upHandGesture == null) {

upHandGesture = new UpHandGesture();

}

upHandGesture.onOff=UpHandGesture.STATE\_ON ;

BLEManager.setUpHandGesture(upHandGesture);

### 设置音乐控制开关(setMusicSwitch)

**接口名称：**

BLEManager.setMusicSwitch(boolean isSwitchOn）

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.MUSIC\_SWITCH) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

BLEManager.setMusicSwitch(true);

### 设置音乐信息(setMusicControlInfo)

1.先根据功能表判断是否支持设置音乐名称：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_support\_set\_v3\_music\_name 返回ture 代表支持

**接口名称：**

BLEManager.setMusicControlInfo(MusicControlInfo musicControlInfo)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.MUSIC\_CONTROL\_INFO) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

MusicControlInfo musicInfo = new MusicControlInfo();

musicInfo.status = MusicControlInfo.STATUS\_PLAY;

musicInfo.musicName = "boy";

//musicInfo.singerName = "";//V3\_music\_control\_02\_add\_singer\_name = true 才支

BLEManager.setMusicControlInfo(musicInfo);

MusicControlInfo 字段说明：

public static final int STATUS\_INVALID = 0x00;

public static final int STATUS\_PLAY = 0x01;

public static final int STATUS\_PAUSE = 0x02;

public static final int STATUS\_STOP = 0x03;

public int status;// 音乐状态

public int curTimeSecond;// 当前音乐时间,单位秒

public int totalTimeSecond;// 总音乐时间

public String musicName;// 音乐名称 字符串【最大64字节】

public String singerName;//歌手名信息

### 设置设备音量(setPhoneVoice)

**接口名称：**

BLEManager.setPhoneVoice(PhoneVoice phoneVoice）

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.PHONE\_VOICE) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

PhoneVoice phoneVoice = new PhoneVoice();

phoneVoice.total\_voice = 100;//最大音量

phoneVoice.now\_voice = 50;//当前音量

BLEManager.setPhoneVoice(phoneVoice);

### 勿扰模式设置(not disturb mode setting)

1. 根据功能表判断设备是否支持设置勿扰模式：

public SupportFunctionInfo getSupportFunctionInfo() {

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

return functionInfo == null ? new SupportFunctionInfo() : functionInfo;

}

public boolean isSupportGetDndMode() {

return getSupportFunctionInfo().ex\_main3\_get\_do\_not\_disturb; //true 代表支持设置

}

2、获取勿扰模式：

**接口名称：**

BLEManager.getDoNotDisturbPara()

**代码示例：**

**NotDisturbPara notDisturbPara;**

//先注册回调监听

BLEManager.registerGetDeviceParaCallBack(new BaseDeviceParaCallBack(){

@Override

public void onGetDoNotDisturbPara(NotDisturbPara notDisturbPara) {

if (notDisturbPara == null) { //初始化默认的值

notDisturbPara = new NotDisturbPara();

notDisturbPara.onOFf = NotDisturbPara.STATE\_OFF;

notDisturbPara.startHour = 9;

notDisturbPara.startMinute = 0;

notDisturbPara.endHour = 18;

notDisturbPara.endMinute = 0;

notDisturbPara.noontimeRestOnOff = NotDisturbPara.STATE\_OFF;

notDisturbPara.noontimeRestStartHour = 22;

notDisturbPara.noontimeRestStartMinute = 0;

notDisturbPara.noontimeRestEndHour = 7;

notDisturbPara.noontimeRestEndMinute = 0;

notDisturbPara.setWeeks(new boolean[]{true, true, true, true, true, true, true});

}

}

}

})

//调用接口

BLEManager.getDoNotDisturbPara();

3.设置勿扰模式：

**接口名称：**

BLEManager.getDoNotDisturbPara()

**代码示例：**

**NotDisturbPara notDisturbPara;**

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.NOT\_DISTURB) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

BLEManager.setNotDisturbPara(**notDisturbPara**);//把上面获取到的值设置给设备，可以修

### 设置用户信息(set userinfo)

**接口名称：**

BLEManager.setUserInfo(UserInfo userInfo)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.USER\_INFO) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

UserInfo userInfo = new UserInfo();

userInfo.year = 2001;

userInfo.month = 12;

userInfo.day = 7;

userInfo.weight = 60;

userInfo.height = 180;

userInfo.gender = 1;

BLEManager.setUserInfo(userInfo);

### 设置单位，语言(set unit ，language)

**接口名称：**

BLEManager.setUnit(Units unit)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.UNIT) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

Units mUnits = new Units();

mUnits.weight = Units.WEIGHT\_UNIT\_KG;

mUnits.language = Units.UNIT\_DEFAULT;

mUnits.timeMode = Units.UNIT\_DEFAULT;

BLEManager.setUnit(mUnits);

//常量

WEIGHT\_UNIT\_KG//公斤

WEIGHT\_UNIT\_LB//磅

WEIGHT\_UNIT\_ST//英石

LANG\_ZH//中文

LANG\_DE//德语

LANG\_JA//日语（语言有很多，就不一一列举了）

TIME\_MODE\_12//12小时制

TIME\_MODE\_24//24 小时制

### 设置屏幕亮度(set screenBrightness)

**接口名称：**

BLEManager.setScreenBrightness(int level)

**代码示例：**

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.SCREEN\_BRIGHTNESS) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

//调用接口

BLEManager.setScreenBrightness(3);

### 设置经期（set period）

**接口名称：**

BLEManager.setMenstrual(Menstrual menstrual)

**参数说明：**

|  |  |
| --- | --- |
| **Menstrual** | |
| on\_off | 标识开关状态 |
| menstrual\_length | 经期长度 |
| menstrual\_cycle | 经期周期 |
| last\_menstrual\_year | 最近一次经期开始时间-年 |
| last\_menstrual\_month | 最近一次经期开始时间-月 |
| last\_menstrual\_day | 最近一次经期开始时间-日 |
| STATUS\_ON | 常量：表示开关打开 |
| STATUS\_OFF | 常量：表示开关关闭 |
| ovulation\_interval\_day | 从下一个经期开始前到排卵日的间隔,一般为14天 |
| ovulation\_before\_day | 排卵日之前易孕期的天数,一般为5 |
| ovulation\_after\_day | 排卵日之后易孕期的天数,一般为5 |
| notify\_flag | 0无效 ； 1：允许通知； 2：静默通知； 3：关闭通知 |

代码示例：

//先注册回调监听

BLEManager.registerOtherProtocolCallBack(new OtherProtocolCallBack.ICallBack(){

@Override

Public void onSuccess(OtherProtocolCallBack.SettingType type){

if (settingType == SettingCallBack.SettingType.ALARM) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

Menstrual menstrual = new Menstrual();

menstrual .menstrual\_length = 7;

menstrual .menstrual\_cycle = 30;

menstrual .last\_menstrual\_year = 2021;

menstrual .last\_menstrual\_month = 9;

menstrual .last\_menstrual\_day = 3;

//调用接口

BLEManager.setMenstrual(menstrual);

### 设置久坐提醒(set long sit remind)

**接口名称：**

BLEManager.setLongSit(LongSit longSit)

**参数说明：**

|  |  |
| --- | --- |
| **LongSit** | |
| startHour | 开始时间：时 |
| startMinute | 开始时间：分 |
| endHour | 结束时间：时 |
| endMinute | 结束时间：分 |
| interval | 间隔 |
| repetitions | 整型数，bit【1】到bit【7】分别表示周一到周日的重复状态 |
| onOff | 标识开关状态 |
| weeks | 布尔值数组：周重复，七个布尔值分别表示周一到周日 |

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.ALARM) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

LongSit longSit = new LongSit();

longSit.setStartHour(startHour);

longSit.setStartMinute(startMinute);

longSit.setEndHour(endHour);

longSit.setEndMinute(endMinute);

longSit.setInterval(lenth);

longSit.setOnOff(onOff);

longSit.setWeeks(weeks);

//调用接口

BLEManager.setLongSit(longSit);

### 设置喝水提醒(set drink water remind)

**接口名称：**

BLEManager.setDrinkWaterReminder(DrinkWaterReminder drinkWaterReminder)

**参数说明：**

|  |  |
| --- | --- |
| **DrinkWaterReminder** | |
| startHour | 开始时间：时 |
| startMinute | 开始时间：分 |
| endHour | 结束时间：时 |
| endMinute | 结束时间：分 |
| interval | 间隔 |
| repeat | 整型数，bit【1】到bit【7】分别表示周一到周日的重复状态 |
| onOff | 标识开关状态 |
| weeks | 布尔值数组：周重复，七个布尔值分别表示周一到周日 |
| STATUS\_ON | 常量：表示开关打开 |
| STATUS\_OFF | 常量：表示开关关闭 |
|  |  |

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.ALARM) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

DrinkWaterReminder waterReminder = new DrinkWaterReminder();

waterReminder.setStartHour(9);

waterReminder.setStartMinute(0);

waterReminder.setEndHour(18);

waterReminder.setEndMinute(0);

waterReminder.setInterval(30);

waterReminder.setWeeks(new boolean[]{true, true, true, true, true, false, false});

//调用接口

BLEManager.setDrinkWaterReminder(waterReminder);

### 设置天气(set weather)

1. 开关设置

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.WEATHER\_SWITCH) {

// send weather to device

}

}

})

//true 开启 false 关闭

BLEManager.setWeatherSwitch(isSwitchOn);

**接口名称：**

BLEManager.setWeatherData(WeatherInfo weatherInfo)

1. 天气下发

//支持v3 协议

if (supportFunctionInfo.V3\_support\_set\_v3\_weather) {

BLEManager.*setWeatherDataV3*(weatherInfoV3)

}else {//v2 协议

BLEManager.*setWeatherData*(*mWeatherInfo*);

}

**v2 协议**

**参数说明：**

|  |  |
| --- | --- |
| **WeatherInfo** | |
| type | 天气类型 |
| temp | 当前温度 |
| max\_temp | 当天最高气温 |
| min\_temp | 当天最低气温 |
| humidity | 当前湿度 |
| uv\_intensity | 当前紫外线强度 |
| aqi | 当天空气指数 |
| future | 未来3天天气气状况 |
|  |  |

|  |  |
| --- | --- |
| **WeatherFutureInfo** | |
| type | 天气类型 |
| min\_temp | 当天最低气温 |
| max\_temp | 当天最高气温 |

type 已经在WeatherInfo类中定义天气的常量。例如：

public static int WEATHER\_TYPE\_UNKNOWN = 0x00;//未知

public static int WEATHER\_TYPE\_CLEAR = 0x01;//晴

public static int WEATHER\_TYPE\_CLOUDY = 0x02;//多云

public static int WEATHER\_TYPE\_OVERCASTSKY = 0x03;//阴

public static int WEATHER\_TYPE\_RAIN = 0x04;//雨

public static int WEATHER\_TYPE\_RAINSTORM = 0x05;//暴雨

/\*\*

\*阵雨

\*/

public static int WEATHER\_TYPE\_SHOWER = 0x06;

/\*\*

\*雪

\*/

public static int WEATHER\_TYPE\_SNOW = 0x07;

/\*\*

\*雨夹雪

\*/

public static int WEATHER\_TYPE\_SLEET = 0x08;

/\*\*

\*台风

\*/

public static int WEATHER\_TYPE\_TYPHOON = 0x09;

/\*\*

\*沙尘暴

\*/

public static int WEATHER\_TYPE\_SANDSTORMS = 0x0A;

/\*\*

\*夜间晴

\*/

public static int WEATHER\_TYPE\_CLEAR\_NIGHT = 0x0B;

/\*\*

\*夜间多云

\*/

public static int WEATHER\_TYPE\_CLOUDY\_NIGHT = 0x0C;

/\*\*

\*热

\*/

public static int WEATHER\_TYPE\_HOT = 0x0D;

/\*\*

\* 冷

\*/

public static int WEATHER\_TYPE\_COLD = 0x0E;

/\*\*

\*清风

\*/

public static int WEATHER\_TYPE\_BREEZE = 0x0F;

/\*\*

\*大风

\*/

public static int WEATHER\_TYPE\_GALE = 0x10;

/\*\*

\*雾霾

\*/

public static int WEATHER\_TYPE\_HAZE = 0x11;

/\*\*

\*阵雨

\*/

public static int WEATHER\_TYPE\_ZHENYU = 0x12;

/\*\*

\*多云转晴

\*/

public static int WEATHER\_TYPE\_YUN = 0x13;

/\*\*

\*新月

\*/

public static int WEATHER\_TYPE\_NEW\_MOON = 0x14;

/\*\*

\*蛾眉月

\*/

public static int WEATHER\_TYPE\_MOTH\_EYEBROW\_MOON = 0x15;

/\*\*

\*上弦月

\*/

public static int WEATHER\_TYPE\_FIRST\_QUARTER\_MOON = 0x16;

代码示例：

//先注册回调监听

BLEManager.registerAppSendDataCallBack(new AppSendDataCallBack.ICallBack(){

@Override

Public void onSuccess(DataType type){

if (settingType == AppSendDataCallBack.DataType.WEATHER) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

WeatherInfo mWeatherInfo = new WeatherInfo ();

。。。。

//调用接口

BLEManager.setWeatherData(mWeatherInfo );

V3 协议

**参数说明：（Refer to SDK demo for details）**

|  |  |
| --- | --- |
| **WeatherInfoV3 （v3天气）** | |
| weather\_type | 天气类型 |
| today\_emp | 当前温度 |
| today\_max\_temp | 当天最高气温 |
| today\_min\_temp | 当天最低气温 |
| city\_name | 城市名称 |
| hours\_weather\_items | 24小时天气 List<Hour24> |
| humidity | 湿度0-100 |
| today\_uv\_intensity | 当前紫外线强度 |
| air\_quality | 空气质量 |
| precipitation\_probability | 降水率 |
| wind\_speed | 风速 |
| future\_items | 7天的天气数据 List<Future> |

|  |  |
| --- | --- |
| Hour24 （24小时天气） | |
| weather\_type | 天气类型 |
| temperature | 温度 |
| probability | 出现的概率 |

|  |  |
| --- | --- |
| Future（7天的天气数据） | |
| weather\_type | 天气类型 |
| max\_temp | 最大温度 |
| min\_temp | 最小温度 |

type 已经在WeatherInfoV3类中定义天气的常量。例如：

public static int WEATHER\_TYPE\_UNKNOWN = 0x00;//未知

public static int WEATHER\_TYPE\_CLEAR = 0x01;//晴

public static int WEATHER\_TYPE\_CLOUDY = 0x02;//多云

public static int WEATHER\_TYPE\_OVERCASTSKY = 0x03;//阴

public static int WEATHER\_TYPE\_RAIN = 0x04;//雨

public static int WEATHER\_TYPE\_RAINSTORM = 0x05;//暴雨

/\*\*

\*阵雨

\*/

public static int WEATHER\_TYPE\_SHOWER = 0x06;

/\*\*

\*雪

\*/

public static int WEATHER\_TYPE\_SNOW = 0x07;

/\*\*

\*雨夹雪

\*/

public static int WEATHER\_TYPE\_SLEET = 0x08;

/\*\*

\*台风

\*/

public static int WEATHER\_TYPE\_TYPHOON = 0x09;

/\*\*

\*沙尘暴

\*/

public static int WEATHER\_TYPE\_SANDSTORMS = 0x0A;

/\*\*

\*夜间晴

\*/

public static int WEATHER\_TYPE\_CLEAR\_NIGHT = 0x0B;

/\*\*

\*夜间多云

\*/

public static int WEATHER\_TYPE\_CLOUDY\_NIGHT = 0x0C;

/\*\*

\*热

\*/

public static int WEATHER\_TYPE\_HOT = 0x0D;

/\*\*

\* 冷

\*/

public static int WEATHER\_TYPE\_COLD = 0x0E;

/\*\*

\*清风

\*/

public static int WEATHER\_TYPE\_BREEZE = 0x0F;

/\*\*

\*大风

\*/

public static int WEATHER\_TYPE\_GALE = 0x10;

/\*\*

\*雾霾

\*/

public static int WEATHER\_TYPE\_HAZE = 0x11;

/\*\*

\*阵雨

\*/

public static int WEATHER\_TYPE\_ZHENYU = 0x12;

/\*\*

\*多云转晴

\*/

public static int WEATHER\_TYPE\_YUN = 0x13;

/\*\*

\*新月

\*/

public static int WEATHER\_TYPE\_NEW\_MOON = 0x14;

/\*\*

\*蛾眉月

\*/

public static int WEATHER\_TYPE\_MOTH\_EYEBROW\_MOON = 0x15;

/\*\*

\*上弦月

\*/

public static int WEATHER\_TYPE\_FIRST\_QUARTER\_MOON = 0x16;

代码示例

//先注册回调监听

BLEManager.registerAppSendDataCallBack(new AppSendDataCallBack.ICallBack(){

@Override

Public void onSuccess(DataType type){

if (settingType == AppSendDataCallBack.DataType.WEATHER) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

WeatherInfoV3 weatherInfoV3 = new WeatherInfoV3();

。。。。

//调用接口

BLEManager.setWeatherData(weatherInfoV3 );

### 设置一键呼救(set sos)

**接口名称：**

BLEManager.setOneKeySOSSwitch(boolean isOpen)

**参数说明：**

|  |  |
| --- | --- |
| isOpen | true是打开，false是关闭 |

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.ONE\_KEY\_SOS) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

BLEManager.setOneKeySOSSwitch(true);

### 设置走动提醒(set walk reminder)

1. 获取走动提醒状态

*/\*\*  
 \* 获取走动提醒状态  
 \*  
 \** ***@return*** *\*/*public WalkReminder getWalkReminderState() {  
 WalkReminder walkReminder = LocalDataManager.*getWalkReminder*();  
 if (walkReminder == null) {  
 walkReminder = new WalkReminder();  
 walkReminder.setOnOff(WalkReminder.*OFF*);  
 walkReminder.setWeeks(new boolean[]{true, true, true, true, true, true, true});  
 walkReminder.setStartHour(9);  
 walkReminder.setStartMinute(0);  
 walkReminder.setEndHour(21);  
 walkReminder.setEndMinute(0);  
 walkReminder.setGoalStep(Constants.*WALK\_REMINDER\_GOAL\_DEFAULT*);//默认的小时目标，用户自己定义  
 }  
 return walkReminder;  
}

1. 设置走动提醒

**接口名称：**

BLEManager.setWalkReminder(WalkReminder walkReminder)

**参数说明：**

|  |  |
| --- | --- |
| WalkReminder | |
| onOff | 开关 |
| goalStep | 目标步数 |
| startHour | 开始时间：时 |
| startMinute | 开始时间：分 |
| endHour | 结束时间：时 |
| endMinute | 结束时间：分 |
| goalTime | 走动提醒的目标时间(单位：小时） |
| weeks | 布尔值数组：周重复，七个布尔值分别表示周一到周日 |
| notifyFlag | //通知类型 ： 0无效 ； 1：允许通知； 2：静默通知； 3：关闭通知 |

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.WALK\_REMINDER) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

WalkReminder walkReminder = getWalkReminderState()//上面获取到的状态，下面是用户修改参数设置

walkReminder .setStartMinute(startMinute);

walkReminder .setEndHour(endHour);

walkReminder .setEndMinute(endMinute);

walkReminder .setGoalStep(goalStep);

walkReminder .setOnOff(onOff);

walkReminder .setWeeks(weeks);

//调用接口

BLEManager.setWalkReminder(walkReminder );

### 设置经期提醒(set menstrual Remind)

**接口名称：**

BLEManager.setMenstrualRemind(MenstrualRemind menstrualRemind)

**参数说明：**

|  |  |
| --- | --- |
| MenstrualRemind | |
| start\_day | 开始日提醒 提前天数 |
| ovulation\_day | 排卵日提醒 提前天数 |
| hour | 提醒时间：时 |
| minute | 提醒时间：分 |
| pregnancy\_day\_before\_remind | 易孕期 开始的时候 提前多少天提醒 |
| pregnancy\_day\_end\_remind | 易孕期 结束的时候 提前多少天提醒 |
| menstrual\_day\_end\_remind | 经期结束 提前多少天提醒 |

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new OtherProtocolCallBack.ICallBack(){

@Override

Public void onSuccess(OtherProtocolCallBack.SettingType type){

if (settingType == SettingCallBack.SettingType.ALARM) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

MenstrualRemind menstrualRemind = new MenstrualRemind();

menstrualRemind.start\_day = 3;

menstrualRemind.ovulation\_day = 3;

menstrualRemind.hour = 9;

menstrualRemind.minute = 30;

menstrualRemind.pregnancy\_day\_before\_remind = 5;

menstrualRemind.pregnancy\_day\_end\_remind = 2;

menstrualRemind.menstrual\_day\_end\_remind = 1;

//调用接口

BLEManager.setMenstrualRemind(menstrualRemind);

### 运动智能识别设置(Motion intelligent recognition set)

接口名称：

BLEManager.setActivitySwitch(activitySwitch);

参数说明：

|  |  |
| --- | --- |
| ActivitySwitch | |
| autoIdentifySportWalk | 步行自动识别 |
| autoIdentifySportRun | 跑步自动识别 |
| SWITCH\_ON | 开启 |
| SWITCH\_OFF | 关闭 |

1. getState

public ActivitySwitch getMotionRecognitionState() {  
 ActivitySwitch activitySwitch = LocalDataManager.*getActivitySwitch*();  
 return activitySwitch == null ? new ActivitySwitch() : activitySwitch;  
}

1. send to device

example:

//先注册回调监听

BLEManager.registerSettingCallBack(new OtherProtocolCallBack.ICallBack(){

@Override

Public void onSuccess(OtherProtocolCallBack.SettingType type){

if (settingType == SettingCallBack.SettingType.ACTIVITY\_SWITCH) {

//设置成功后的回调

}

}

})

ActivitySwitch mMotionRecognitionState = getMotionRecognitionState();

mMotionRecognitionState .autoIdentifySportRun = ActivitySwitch .SWITCH\_ON;

BLEManager.setActivitySwitch(mMotionRecognitionState );

### 心率设置(HeartRate set) v3协议

接口名称：

BLEManager.setHeartRateMeasureModeV3(heartRateMode);

HeartRateMeasureModeV3 heartRateMode;

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

public void onSuccess(SettingCallBack.SettingType type, Object returnData) {

heartRateMode= (HeartRateMeasureModeV3) o;

}

@Override

public void onFailed(SettingCallBack.SettingType type) {

}})

1.HeartRateMeasureModeV3 modeV3 = new HeartRateMeasureModeV3();

modeV3.updateTime = 0;

BLEManager.setHeartRateMeasureModeV3(modeV3);

先这样调用一下，就可以拿到固件返回的heartRateMode

2.

heartRateMode.measurementInterval = 5;

heartRateMode.startHour = 9;

heartRateMode.startMinute = 30;

heartRateMode.endHour = 18;

heartRateMode.endMinute = 0;

heartRateMode.updateTime = (int) System.currentTimeMillis();

heartRateMode.mode = HeartRateMeasureModeV3.MODE\_CUSTOM;

BLEManager.setHeartRateMeasureModeV3(heartRateMode);

这里可以修改具体的参数

### 设置压力开关(PRESSURE on\_off set)

接口名称：

BLEManager.setPressureParam(pressureParam);

PressureParam 对象说明：

|  |  |
| --- | --- |
| PressureParam | |
| onOff | 总开关 0xAA开,0x55关闭 |
| startHour | //开关时间段和提醒时间段是一致的 |
| startMinute | 开始时间：分 |
| endHour | 结束时间：时 |
| endMinute | 结束时间：分 |
| remind\_on\_off | //压力提醒 开关 0xAA开,0x55关，on\_off为关，提醒不起作用 |
| interval | //提醒间隔,单位分钟 默认60分钟 |
| notifyFlag | //通知类型 ： 0无效 ； 1：允许通知； 2：静默通知； 3：关闭通知 |
| repeat | /预留 bit0 无效,bit1-bit7 分别是星期1到星期7,同闹钟 |

example:

BLEManager.registerOtherProtocolCallBack(mOtherSettingCallback);

private final OtherProtocolCallBack.ICallBack mOtherSettingCallback = new OtherProtocolCallBack.ICallBack() {

@Override

public void onSuccess(OtherProtocolCallBack.SettingType settingType) {

if (settingType == OtherProtocolCallBack.SettingType.PRESSURE) {

//设置成功

}

}

@Override

public void onFailed(OtherProtocolCallBack.SettingType settingType) {

}

};

BLEManager.setPressureParam(pressureParam);

### 快捷应用设置(set setMenuList)

**接口名称：**

BLEManager.setMenuList(MenuList menuList)

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new OtherProtocolCallBack.ICallBack(){

@Override

Public void onSuccess(OtherProtocolCallBack.SettingType type){

if (settingType == SettingCallBack.SettingType.MENU\_LIST\_SET) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

MenuList menuList = new MenuList();

menuList.items.add(MenuList.MENU\_ALARM);

menuList.items.add(MenuList.MENU\_PRESSURE);//调用接口

BLEManager.setMenuList(menuList);

### 常用联系人(FrequentContacts)

**接口名称：**

1.获取常用联系人：

BLEManager.queryFrequentContactsV3()

1. 设置常用联系人：

BLEManager.setFrequentContactsV3(List<FrequentContactsV3> frequentContactsV3List)

//先注册回调监听

OperateCallBack.ICallBack callBack = new OperateCallBack.ICallBack() {

@Override

public void onSetResult(OperateCallBack.OperateType type, boolean isSuccess) {

//设置结果

}

@Override

public void onAddResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onDeleteResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onModifyResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onQueryResult(OperateCallBack.OperateType type, Object returnData) {

}

}

BLEManager.registerOperateCallBack(mOperateCallback);

BLEManager.unregisterOperateCallBack(mOperateCallback);（ondestroy 调用）

FrequentContactsV3 ：

/\*\*

\*联系人的名字

\*/

public String name;

/\*\*

\*联系人的手机

\*/

public String phone;

### 通话开关设置(CallReminderSwitch)

1.先根据功能表判断是否支持蓝牙通话：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_support\_sync\_contact 返回ture 代表支持

**接口名称：**

BLEManager.setNoticeReminderSwitchStatus(status);

代码示例：

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingCallBack.SettingType type){

if (settingType == SettingCallBack.SettingType.NOTICE\_REMINDER\_SWITCH\_STATUS) {

//设置成功后的回调，SettingType 设置的类型，data返回的数据

}

}

})

boolean switch;

NoticeReminderSwitchStatus status = new NoticeReminderSwitchStatus();

status.notify\_switch = NoticeReminderSwitchStatus.NOTIFY\_88;

if (switch) {

status.call\_switch = NoticeReminderSwitchStatus.SWITCH\_ON;

} else {

status.call\_switch = NoticeReminderSwitchStatus.SWITCH\_OFF;

}

BLEManager.setNoticeReminderSwitchStatus(status);

### 事项提醒(ScheduleReminder)

1.先根据功能表判断是否支持事项提醒：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_schedule\_reminder 返回ture 代表支持

**接口名称：**

1. 查询事项提醒

BLEManager.queryScheduleReminderV3()

1. 删除事项提醒

BLEManager.deleteScheduleReminderV3(List<ScheduleReminderV3> scheduleReminderV3List)

1. 添加事项提醒

BLEManager.addScheduleReminderV3(List<ScheduleReminderV3> scheduleReminderV3List)

代码示例：

//先注册回调监听

//先注册回调监听

OperateCallBack.ICallBack callBack = new OperateCallBack.ICallBack() {

@Override

public void onSetResult(OperateCallBack.OperateType type, boolean isSuccess) {

//设置结果

}

@Override

public void onAddResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onDeleteResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onModifyResult(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onQueryResult(OperateCallBack.OperateType type, Object returnData) {

}

}

BLEManager.registerOperateCallBack(mOperateCallback);

BLEManager.unregisterOperateCallBack(mOperateCallback);（ondestroy 调用）

### 时间时钟设置( world time set)

**接口名称：**

BLEManager.setWorldTime(List<WorldTime.Item> itemList)

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.WORLD\_TIME) {

}

}

})

example:

List<WorldTime.Item> list = new ArrayList<>();

WorldTime.Item item = new WorldTime.Item();

item.city\_name = "Beijing";

item.id = 31;

item.min\_offset = 480;

item.sunset\_hour = 4;

item.sunrise\_min = 45;

item.sunset\_hour = 19;

item.sunrise\_min = 44;

item.latitude\_flag = 1;

item.latitude = 3390;// 33.90 \*100

item.longitude\_flag = 1;

item.longitude = 11641; // 116.41\*100

list.add(item);

BLEManager.setWorldTime(list);

Item 说明：

public int id;

public int min\_offset;

public String city\_name; //城市名称

public int sunrise\_hour; //日出

public int sunrise\_min;

public int sunset\_hour; //日落

public int sunset\_min;

public int longitude\_flag; // 1: 东经； 2：西经

public int longitude; //经度 保留2位小数扩大100倍传输， app需要将对应的分转换成度

public int latitude\_flag; // 1: 北纬； 2：南纬

public int latitude; //纬度 保留2位小数扩大100倍传输， app需要将对应的分转换成度

### 设置夜间亮度( Set night brightness)

**接口名称：**

BLEManager.setSCreenBrightnessConfig(screenBrightness);

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.SCREEN\_BRIGHTNESS) {

}

}

})

ScreenBrightness brights = new ScreenBrightness();

brights.level = 60;

brights.mode = 2;

brights.autoAdjustNight = 3;

brights.startHour = 19;

brights.startMinute = 0;

brights.endHour = 6;

brights.endMinute = 0;

brights.nightLevel = 0;

brights.showInterval = 5;

BLEManager.setSCreenBrightnessConfig(brights);

ScreenBrightness类说明：

public int opera = OPERA\_TYPE\_USER;//0x0 自动,0x01 手动 (如果是自动同步配置,请发送00,如果是用户调节请发送01)

public int level = 100;

public int mode = MODE\_ADJUST\_OFF; //0x00 关闭自动调整,使用level的值 ;0x01 使用环境光传感器;0x02同步配置时sdk内部使用

public int autoAdjustNight = AUTO\_ADJUST\_NIGHT\_TYPE\_DEVICE\_DEFAULT\_TIME\_RANGE; //0x00无效 由固件定义 0x01 关闭 0x02 夜间降亮度使用手环预设的时间 0x03 夜间降亮度使用设置的时间

public int startHour;

public int startMinute;

public int endHour;

public int endMinute;

public int showInterval;//显示间隔

### 权限设置( 目前只有拍照权限设置)

1.先根据功能表判断是否支持拍照权限设置：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.v2\_support\_disable\_func 返回ture 代表支持

**接口名称：**

BLEManager.setNoticeDeviceDiableFunc(permmsion);

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.CAMERA\_PERMISSION) {

}

}

})

example:

NoticeDevicePermmsion permmsion = new NoticeDevicePermmsion();

permmsion.type = NoticeDevicePermmsion.CAMERA\_PERMMSION;

permmsion.enable = NoticeDevicePermmsion.STATUS\_ON ; BLEManager.setNoticeDeviceDiableFunc(permmsion);

### 噪音设置( noise set)

1.先根据功能表判断是否支持噪音功能：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_health\_sync\_noise 返回ture 代表支持

**接口名称：**

BLEManager.setNoisePara(param);

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.NOISE) {

}

}

})

example:

NoisePara para = new NoisePara();

para.mode = NoisePara.MODE\_OFF;

BLEManager.setNoisePara(para);

### 夜间体温开关设置( temperature set)

1.先根据功能表判断是否支持皮温功能：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_support\_set\_temperature\_switch 返回ture 代表支持

**接口名称：**

BLEManager.setNightTemperatureMonitoringPara(param);

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.NIGHT\_TEMPERATURE\_MONITORING) {

}

}

})

example:

NightTemperatureMonitoringPara para = new NightTemperatureMonitoringPara(); param.mode = NightTemperatureMonitoringPara.SWITCH\_OFF;

param.unit=NightTemperatureMonitoringPara.UNIT\_F; BLEManager.setNightTemperatureMonitoringPara(para);

### 血氧自动检测设置( BloodOxygen set)

1.先根据功能表判断是否支持血氧自动检测功能：

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.V3\_support\_set\_spo2\_all\_day\_on\_off 返回ture 代表支持

**接口名称：**

BLEManager.setSPO2Param(param);

//先注册回调监听

BLEManager.registerOtherProtocolCallBack(new OtherProtocolCallBack.ICallBack() {

@Override

public void onSuccess(OtherProtocolCallBack.SettingType settingType) {

if (settingType == SettingCallBack.SettingType.SPO2) {

}

}

@Override

public void onFailed(OtherProtocolCallBack.SettingType settingType) {

}

});

example:

SPO2Param spO2Param = spO2Param = new SPO2Param();

spO2Param.startHour = 0;

spO2Param.startMinute = 0;

spO2Param.endHour = 23;

spO2Param.endMinute = 59;

spO2Param.lowSpo2OnOff = SPO2Param.STATE\_OFF;

spO2Param.onOff = SPO2Param.STATE\_OFF;

spO2Param.lowSpo2OnValue = 85;

BLEManager.setSPO2Param(spO2Param )

### 设置距离卡路里目标(set distanc calorie goal)

**接口名称：**

BLEManager.setCalorieAndDistanceGoal(CalorieAndDistanceGoal calorieAndDistanceGoal)

**代码示例：**

//先注册回调监听

//先注册回调监听

BLEManager.registerOtherProtocolCallBack(new OtherProtocolCallBack.ICallBack() {

@Override

public void onSuccess(OtherProtocolCallBack.SettingType settingType) {

if(settingType==SettingCallBack.SettingType.CALORIE\_DISTANCE\_GOAL) {

}

}

@Override

public void onFailed(OtherProtocolCallBack.SettingType settingType) {

}

});

//调用接口

CalorieAndDistanceGoal calorieAndDistanceGoal = new CalorieAndDistanceGoal();

calorieAndDistanceGoal .calorie = 100;

calorieAndDistanceGoal .distance = 100;

BLEManager.setCalorieAndDistanceGoal(calorieAndDistanceGoal);

CalorieAndDistanceGoal 对象说明：

public int calorie; //卡路里

public int distance; // 距离

public long mid\_high\_time\_goal; //中高运动时长目标设置

public int calorie\_min; //卡路里最小值

public int calorie\_max; // 卡路里最大值

public int walk\_goal\_time; //走动目标时间

### 设置左右手模式(setHandWearMode)

**接口名称：**

BLEManager.setHandWearMode(HandWearMode handWearMode)；

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

if (settingType == SettingCallBack.SettingType.HAND\_MODE) {

}

}

})

example:

HandWearMode handWearMode = new HandWearMode();

handWearMode.hand = HandWearMode.HAND\_MODE\_LEFT;

BLEManager.setHandWearMode(handWearMode);

### 设置心率区间(setHeartRateInterval)

**接口名称：**

BLEManager.setHeartRateInterval(HeartRateInterval interval);

//先注册回调监听

BLEManager.registerSettingCallBack(new SettingCallBack.ICallBack(){

@Override

Public void onSuccess(SettingType type, Object data){

}

})

example:

HeartRateInterval heartRateInterval = heartRateInterval = new HeartRateInterval();

if (heartRateInterval == null) {

heartRateInterval = new HeartRateInterval();

}

heartRateInterval.setRemindStartHour(0);

heartRateInterval.setRemindStartMinute(0);

heartRateInterval.setRemindStopHour(23);

heartRateInterval.setRemindStopMinute(59);

heartRateInterval.setMinHr(20);

int maxValue = 220;

heartRateInterval.setUserMaxHR(maxValue);

//热身运动

heartRateInterval.setWarmUpThreshold((int) (maxValue \* 0.5));

//脂肪燃烧

heartRateInterval.setBurnFatThreshold((int) (maxValue \* 0.6));

//有氧运动

heartRateInterval.setAerobicThreshold((int) (maxValue \* 0.7));

//无氧运动

heartRateInterval.setAnaerobicThreshold((int) (maxValue \* 0.8));

//极限运动

heartRateInterval.setLimintThreshold((int) (maxValue \* 0.9));

BLEManager.setHeartRateInterval(heartRateInterval);

## 六 表盘 （dial）

### 获取表盘列表(get dial list)

WatchPlateCallBack.IOperateCallBack iOperateCallBack = new WatchPlateCallBack.IOperateCallBack() {

@Override

public void onGetPlateFileInfo(WatchPlateFileInfo watchPlateFileInfo) {

if (watchPlateFileInfo == null || watchPlateFileInfo.fileNameList == null){

return;

}

tvOperateTv.setText(watchPlateFileInfo.toString()); //f返回的表盘列表

}

@Override

public void onGetScreenInfo(WatchPlateScreenInfo screenInfo) {

tvOperateTv.setText(GsonUtil.toJson(screenInfo)); //返回屏幕信息

}

@Override

public void onGetCurrentPlate(String uniqueID) {

tvOperateTv.setText("当前使用的表盘：" + uniqueID);

}

@Override

public void onSetPlate(boolean isSuccess) {

tvOperateTv.setText(isSuccess ? "设置成功" : "设置失败");

}

@Override

public void onDeletePlate(boolean isSuccess) {

tvOperateTv.setText(isSuccess ? "删除成功" : "删除失败");

}

@Override

public void onGetDialPlateParam(DialPlateParam dialPlateParam) {

}

};

BLEManager.registerWatchOperateCallBack(iOperateCallBack);

if (getSupportFunctionInfo().V3\_get\_watch\_list\_new) { //根据功能表盘判断，调用不同的方法

saveDialLog("getInstalledDialInfo start, V3\_get\_watch\_list\_new");

BLEManager.getDialPlateParam();

} else {

saveDialLog("getInstalledDialInfo start");

BLEManager.getWatchPlateList();

}

### 壁纸表盘(wallpaper dial)

**设置流程：**

**1.getWatchPlateScreenInfo(); 获取屏幕大小**

**2. 选择图片，截图图片大小（根据watchPlateScreenInfo里面的大小）**

**3. 制作图片**

**Wallpaper wallpaper = new Wallpaper();**

**wallpaper.setFileName(path);// 裁剪的图片存储的地方**

**wallpaper.setSaveFileName(outputpath);**

**wallpaper.setFormat(5);**

**String json = GsonUtil.toJson(wallpaper);**

**BLEManager.setParaToDeviceByTypeAndJson(5500, json);//请求制作照片**

**4.安装**

**BLEManager.startTranCommonFile(FileTransferConfig.getDefaultTransPictureConfig(outputpath.concat(".lz"),, new IFileTransferListener() {**

**@Override**

**public void onStart() { }**

**@Override**

**public void onProgress(int progress) { }**

**@Override**

**public void onSuccess() {**

**//传输成功，设置颜色参数**

**5.BLEManager.photoWallpaperOperate(params)**

**}**

**@Override**

**public void onFailed(String errorMsg) { }**

**});**

**接口名称：**

BLEManager.createPlateWallpaperFile(WallpaperFileCreateConfig config);

BLEManager.startTranCommonFile(FileTransferConfig config);

**代码示例：**

WallpaperFileCreateConfig config = new WallpaperFileCreateConfig();

//App裁剪用户选择的图片，生成的PNG原始图片地址

config.setSourceFilePath("/asdcard/demo/test.png");

//经过sdk处理之后会输出新的壁纸文件

config.setOutFilePath("/adcard1/demo/outFile.png");

//开始制作壁纸文件

BLEManager.createPlateWallpaperFile(config);

//制作完成，传输壁纸文件到设备

FileTransferConfig fileConfig = FileTransferConfig.getDefaultTransPictureConfig(config.getOutFilePath(), new IFileTransferListener() {

@Override

public void onStart() { }

@Override

public void onProgress(int progress) { }

@Override

public void onSuccess() { }

@Override

public void onFailed(String errorMsg) { }

});

BLEManager.stopTranCommonFile();

BLEManager.startTranCommonFile(fileConfig);

### 云端表盘(cloud dial)

**接口名称：**

BLEManager.startSetPlateFileToWatch(WatchPlateSetConfig config)

**参数说明：**

|  |  |
| --- | --- |
| **WatchPlateSetConfig** | |
| filePath | 表盘zip包路径 |
| uniqueID | 表盘唯一ID，App-服务器-手表 统一 |
| isOnlyTranslateWatchFile | 是否只是传输表盘 |
| stateListener | 状态监听 |
| maxRetryTimes | 最大重试次数 |
|  |  |
|  |  |

**代码示例：**

WatchPlateSetConfig config = new WatchPlateSetConfig();

File file = new File(filePath);

config.uniqueID = file.getName().replace(".zip", "");

config.filePath = filePath;

config.stateListener = new WatchPlateCallBack.IAutoSetPlateCallBack() {

@Override

public void onStart() {

tvState.setText("start");

}

@Override

public void onProgress(int progress) {

tvState.setText("progress = " + progress);

}

@Override

public void onSuccess() {

tvState.setText("success");

}

@Override

public void onFailed() {

tvState.setText("failed");

}

};

BLEManager.startSetPlateFileToWatch(config);

注意：安装表盘的时候要确认手表里面没有相同名称的表盘，如果有了，则需要先删除才能重新安装

### 设置壁纸表盘颜色

注册监听：

BLEManager.registerPhotoWallpaperOperateCallBack(new PhotoWallpaperOperateCallBack.ICallBack() {

@Override

public void onOperateResult(PhotoWallpaperOperation.ResponseInfo responseInfo) {

}

});

/\*\*

\* 初始化壁纸表盘颜色集合

\*/

public List<String> getWallpaperColorList() {

if (mColorList == null) {

mColorList = new ArrayList<>();

}

if (mColorList.isEmpty()) {

mColorList.add("#F2F2F2");

mColorList.add("#000000");

mColorList.add("#FC1E58");

mColorList.add("#FF9501");

mColorList.add("#0091FF");

mColorList.add("#44D7B6");

}

return mColorList;

}

开始设置颜色：

PhotoWallpaperOperation.SetParams params = new PhotoWallpaperOperation.SetParams();

params.operate = PhotoWallpaperOperation.SetParams.OPERATE\_SET;

Integer color =Integer.parseInt(getWallpaperColorList().get(colorIndex).replace("#",""),16);

params.time\_color = color;

params.widget\_icon\_color = color;

params.widget\_num\_color = color;

params.location = 2;

BLEManager.photoWallpaperOperate(params);

### 删除表盘(delete dial)

BLEManager.deleteWatchPlate（String name）;

name: 表盘的名称 ，固件返回的表盘名称， 比如： watcht.iwf 一定要带上.iwf

### 6. 其它相关表盘操作（other dial operation）

BLEManager.getCurrentWatchPlate();

BLEManager.getWatchPlateScreenInfo();

回调跟前面一样， BLEManager.registerWatchOperateCallBack(iOperateCallBack);

## 七 OTA升级 (upgrade OTA)

### Nordic平台升级

（1）需要在AndroidManifest.xml 文件中注册服务：

<service

android:name="com.realsil.sdk.dfu.DfuService"

android:enabled="true"

android:exported="false" />

<service

android:name="com.ido.ble.dfu.DFUService"

android:enabled="true"

android:exported="false"

tools:ignore="Instantiatable" />

（2）libs 需要依赖 dfu.aar

**接口名称：**

|  |  |
| --- | --- |
| BLEManager.startDFU(BleDFUConfig dfuConfig) | 开始升级 |
| BLEManager.cancelDFU() | 取消升级 |
| BLEManager.addDFUStateListener(BleDFUState.IListener iListener) | 监听升级状态 |

**参数说明：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BleDFUConfig** | | | |
| 字段名称 | 是否必须 | 说明 |
| filePath | 是 | ota文件路径 |
| macAddresss | 是 | 设备的mac地址 |
| deviceID | 是 | 设备ID |
| PRN | 否 |  |
| isNeedReOpenBluetoothSwitchIfFailed | 否 | 失败重试时，是否需要重启蓝牙 |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **BleDFUState.IListener** | |
| onPrepare() | 开始升级 |
| onProgress(int progress) | 升级进度 |
| onSuccess() | 升级成功 |
| onFailed(FailReason failReason) | 升级失败 |
| onCanceled() | 升级取消 |
|  |  |

**example：**

BleDFUState.IListener mDFUStateListener = new BleDFUState.IListener() {

@Override

public void onPrepare() {

}

@Override

public void onDeviceInDFUMode() {

}

@Override

public void onProgress(int i) {

}

@Override

public void onSuccess() {

}

@Override

public void onSuccessAndNeedToPromptUser() {

}

@Override

public void onFailed(BleDFUState.FailReason failReason) {

}

@Override

public void onCanceled() {

}

@Override

public void onRetry(int i) {

}

};

BleDFUConfig config = new BleDFUConfig();

config.setDeviceId(String.valueOf(mDeviceId));//设备id ,设备的基本信息里面有

config.setMacAddress(mDeviceAddress);// 设备mac

config.setFilePath(mFirmwareFilePath);//文件路径

config.setPlatform(BleDFUConfig.PLATFORM\_NORDIC);

config.setMaxRetryTime(3);//重试次数

BLEManager.addDFUStateListener(mDFUStateListener);

BLEManager.startDFU(config);

### Apollo平台升级,文件传输

直接通过文件传输的方式传输文件就可以

example:

IFileTransferListener mIFileTransferListener = new IFileTransferListener() {

@Override

public void onStart(){};

@Override

public void onProgress(int progress){};

@Override

public void onSuccess(){};

@Override

public void onFailed(String errorMsg){};

}

FileTransferConfig config = FileTransferConfig.getDefaultApolloOTAConfig(updatePath, mIFileTransferListener);//updatePath升级文件保存的路径

BLEManager.stopTranCommonFile();

BLEManager.startTranCommonFile(config);

### 3.Realtek平台升级

**接口名称：**

|  |  |
| --- | --- |
| BLEManager.startDFU(BleDFUConfig dfuConfig) | 开始升级 |
| BLEManager.cancelDFU() | 取消升级 |
| BLEManager.addDFUStateListener(BleDFUState.IListener iListener) | 监听升级状态 |

**参数说明：**

|  |  |  |  |
| --- | --- | --- | --- |
| **BleDFUConfig** | | | |
| 字段名称 | 是否必须 | 说明 |
| filePath | 是 | ota文件路径 |
| macAddresss | 是 | 设备的mac地址 |
| deviceID | 是 | 设备ID |
| PRN | 否 |  |
| isNeedReOpenBluetoothSwitchIfFailed | 否 | 失败重试时，是否需要重启蓝牙 |
| setPlatform | 是 | 值：BleDFUConfig.PLATFORM\_REALTEK （平台设置） |
|  |  |  |

|  |  |
| --- | --- |
| **BleDFUState.IListener** | |
| onPrepare() | 开始升级 |
| onProgress(int progress) | 升级进度 |
| onSuccess() | 升级成功 |
| onFailed(FailReason failReason) | 升级失败 |
| onCanceled() | 升级取消 |
|  |  |

example:

**example：**

BleDFUState.IListener mDFUStateListener = new BleDFUState.IListener() {

@Override

public void onPrepare() {

}

@Override

public void onDeviceInDFUMode() {

}

@Override

public void onProgress(int i) {

}

@Override

public void onSuccess() {

}

@Override

public void onSuccessAndNeedToPromptUser() {

}

@Override

public void onFailed(BleDFUState.FailReason failReason) {

}

@Override

public void onCanceled() {

}

@Override

public void onRetry(int i) {

}

};

BleDFUConfig config = new BleDFUConfig();

config.setDeviceId(String.valueOf(mDeviceId));//设备id ,设备的基本信息里面有

config.setMacAddress(mDeviceAddress);// 设备mac

config.setFilePath(mFirmwareFilePath);//文件路径

config.setMaxRetryTime(3);//重试次数

BLEManager.addDFUStateListener(mDFUStateListener);

BLEManager.startDFU(config);

## 八 多运动数据交换(sport data exchange)

**一.v2协议：(任何手表都要接入)**

1.注册监听

AppExchangeDataCallBack.ICallBack iCallBack = new AppExchangeDataCallBack.ICallBack() {

@Override //手环回复:App发送的“开始”命令

public void onReplyExchangeDataStart(AppExchangeDataStartDeviceReplyData data) {

}

@Override //手环回复:App发送的“发送数据”命令

public void onReplyExchangeDateIng(AppExchangeDataIngDeviceReplyData data) {

}

@Override //手环回复:App发送的“停止”命令

public void onReplyExchangeDateStop(AppExchangeDataStopDeviceReplyData data) {

}

@Override //手环回复:App发送的“暂停”命令

public void onReplyExchangeDatePause(AppExchangeDataPauseDeviceReplyData data) {

}

@Override // 手环回复:App发送的“恢复”命令

public void onReplyExchangeDateResume(AppExchangeDataResumeDeviceReplyData data) {

}

@Override // App收到手环主动发起的“停止”命令

public void onDeviceNoticeAppStop(DeviceNoticeAppExchangeDataStopPara para) {

}

@Override // App收到手环主动发起的“暂停”命令

public void onDeviceNoticeAppPause(DeviceNoticeAppExchangeDataPausePara para) {

}

@Override // App收到手环主动发起的“恢复”命令

public void onDeviceNoticeAppResume(DeviceNoticeAppExchangeDataResumePara para) {

}

};

**BLEManager.registerAppExchangeDataCallBack(iCallBack);**

2、由App发起的交换数据的指令类型：

(1)开始运动：

接口：

BLEManager.appExchangeDataStart(para);

example:

AppExchangeDataStartPara para = new AppExchangeDataStartPara();

Para这个参数需要设置时间、运动类型等参数，具体视需求来决定，例如：

para.day = mDay =Calendar.getInstance().get(Calendar.DAY\_OF\_MONTH);

para.hour = mHour = Calendar.getInstance().get(Calendar.HOUR\_OF\_DAY);

para.minute = mMin = Calendar.getInstance().get(Calendar.MINUTE);

para.second = mSecond = Calendar.getInstance().get(Calendar.SECOND);

para.sportType = AppExchangeDataStartPara.SPORT\_TYPE\_DANCING;

para.target\_type = AppExchangeDataStartPara.TARGET\_TYPE\_DURATIONS;

para.target\_value = 1;

para.force\_start = AppExchangeDataStartPara.FORCE\_START\_VALID;（只是示例）

如果要强制开始就设置：

para.force\_start = AppExchangeDataStartPara.FORCE\_START\_VALID;

不要强制开始就设置：

para.force\_start = AppExchangeDataStartPara.FORCE\_START\_INVALID;

设置好para参数后，再调用如下代码就可以开始交换数据：

BLEManager.appExchangeDataStart(para);

AppExchangeDataStartDeviceReplyData中已经定义好5种状态：

public static final int CODE\_SUCCESS = 0x00;//成功

public static final int CODE\_FAILED\_DEVICE\_ALREADY\_IN\_SPORT\_MODE = 0x01;//失败，因为设备已经进入运动模式

public static final int CODE\_FAILED\_DEVICE\_LOW\_BATTARY = 0x02;//失败，因为设备电量低

public static final int CODE\_FAILED\_DEVICE\_CHARGING = 0x03;//充电中

public static final int CODE\_FAILED\_DEVICE\_IN\_VOICE\_STATE = 0x04;//alexa语音识别中

public int ret\_code;//返回状态码

1. 发送数据,交换数据：用来发送距离，时间给手表(IDO APP 2秒发送一次，可参考)

接口：

BLEManager.appExchangeDataIng(para);

example:

AppExchangeDataIngPara para = new AppExchangeDataIngPara();

para.duration= 20; //运动时间

para.distance= 100; // 运动距离 ，app 根据经纬度计算，单位m

BLEManager.appExchangeDataIng(para);

1. 暂停运动：

接口：

BLEManager.appExchangeDataPause(para);

example:

AppExchangeDataIngPara para = new AppExchangeDataIngPara();

para.day = mDay ;

para.hour = mHour ;

para.minute = mMin ;

para.second = mSecond ;

BLEManager.appExchangeDataPause(para);

1. **恢复运动：**

**接口：**

**BLEManager.appExchangeDataResume(para);**

example:

AppExchangeDataIngPara para = new AppExchangeDataIngPara();

para.day = mDay ;

para.hour = mHour ;

para.minute = mMin ;

para.second = mSecond ;

BLEManager.appExchangeDataResume(para);

1. **结束运动：**

**接口：**

**BLEManager.appExchangeDataStop(para);**

example:

AppExchangeDataIngPara para = new AppExchangeDataIngPara();

para.day = mDay ;

para.hour = mHour ;

para.minute = mMin ;

para.second = mSecond ;

para.duration= 20; //运动时间

para.distance= 100; // 运动距离 ，app 根据经纬度计算，单位m

BLEManager.appExchangeDataStop(para);

注意:

①发送数据、暂停、恢复和结束的参数para 设置的时间是要跟发起开始指令时的时间一样的。

**②app向手表发起的开始、发送数据、暂停、恢复和结束命令成功后，app都会收到手表返回的回调方法，开发人员就可以在对应的回调方法中处理UI的显示和业务逻辑。以下是回调方法的对应关系：**

**开始 ---> onReplyExchangeDataStart(AppExchangeDataStartDeviceReplyData data)**

**发送数据 ---> onReplyExchangeDateIng(AppExchangeDataIngDeviceReplyData data)**

**暂停 ---> onReplyExchangeDatePause(AppExchangeDataPauseDeviceReplyData data)**

**恢复 ---> onReplyExchangeDateResume(AppExchangeDataResumeDeviceReplyData data)**

**结束 ---> onReplyExchangeDateStop(AppExchangeDataStopDeviceReplyData data)**

3、监听的销毁：

在Activity 的onDestroy()方法中销毁监听：

BLEManager.unregisterAppExchangeDataCallBack(iCallBack);

**v3协议：**

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

functionInfo.ex\_table\_main9\_v3\_activity\_exchange\_data

**返回ture 代表支持V3协议，需要接入下面的v3协议，用来交互数据，上面的v2 协议也需要（运动开始，暂停等指令都是走v2这个协议）**

1. 注册监听：

BLEManager.registerV3AppExchangeDataCallBack(new V3AppExchangeDataCallBack.ICallBack() {

@Override

public void onReplyExchangeDateIng(V3AppExchangeDataIngDeviceReplyData v3AppExchangeDataIngDeviceReplyData) {

//步数，距离，时长等数据

}

@Override

public void onReplyExchangeDataEndData(V3AppExchangeDataDeviceReplayEndData v3AppExchangeDataDeviceReplayEndData) {

// BLEManager.v3getEndActivityData(); 调用时数据的回调

}

@Override

public void onReplyExchangeHeartRateData(V3AppExchangeDataHeartRate v3AppExchangeDataHeartRate) {

//心率数据BLEManager.v3AppExchangeDataGetHeartRate(); 调用时的回调

}

});

2. v3 数据交互：APP 可以把距离等数据发送给设备（ido app 2秒发送一次）

BLEManager.v3AppExchangeDataIng(data)

example:

V3AppExchangeDataIngPara mV3SwitchDataAppIng = new V3AppExchangeDataIngPara();

mV3SwitchDataAppIng.distance =30; //距离app 根据经纬度计算

BLEManager.v3AppExchangeDataIng(mV3SwitchDataAppIng );

3.

v3 协议获取运动数据详情：（ido app 40秒调用一次）

BLEManager.v3getEndActivityData();

4 . v3 协议获取心率数据：（ido app 30秒调用一次）

BLEManager.v3AppExchangeDataGetHeartRate();

V3AppExchangeDataIngDeviceReplyData 对象说明：

public int type;

public int hour;

public int minute;

public int second;

public int heart\_rate;

public int distance;//距离 按照设置的单位数据显示

public int real\_time\_speed;//实时速度 单位km/h 扩大100传

public int real\_time\_speed\_pace;//实时的配速 s

public int km\_speed;//实时公里配速 单位s/公里

public int real\_time\_calories;//动态卡路里

public int steps;

public int swim\_posture;//主泳姿,游泳数据用到

public int status;// 状态 开始1 , 手动暂停2， 结束 3， 自动暂停4 ， 0是无效状态 (4是新增的)

public int duration;//持续时间

//新增，version = 3，新增对应的结构数据

public int te; //有氧训练效果等级 单位无 范围 0-50 扩大10倍传输

public int tean; // 无氧运动训练效果等级 单位无 范围 0-50 扩大10倍传输

//运动倒计时（注：递减）， 或课程结束后计时（（注：递增）） action\_type = 1—5时递减 ， action\_type = 6递增

public int count\_hour;

public int count\_minute;

public int count\_second;

## 九 通知提醒 (notice) （app发送提醒给手表）

先通过下面的方法判断协议

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

if (functionInfo.ex\_table\_main10\_v3\_notify\_msg ){

BLEManager.setV3MessageNotice(V3MessageNotice v3MessageNotice)

} else {

BLEManager.setNewMessageDetailInfo()

}

**1、接口名称：**

BLEManager.setNewMessageDetailInfo() v2协议

**参数说明：**

|  |  |
| --- | --- |
| **NewMessageInfo** | |
| type | 信息类型 |
| name | 姓名 |
| content | 内容 |
|  |  |
|  |  |
|  |  |

type 的值已经在NewMessageInfo类中定义好常量，例如:

public static final int TYPE\_FACEBOOK = 0x06;

public static final int TYPE\_TWITTER = 0x07;

public static final int TYPE\_WHATSAPP = 0x08;

public static final int TYPE\_MESSENGER = 0x09;

public static final int TYPE\_INSTAGRAM = 0x0A;

public static final int TYPE\_LINKEDIN = 0x0B;

**代码示例：**

//发送通知不用注册回调监听

NewMessageInfo newMessageInfo = new NewMessageInfo();

newMessageInfo.type = NewMessageInfo.TYPE\_FACEBOOK;

newMessageInfo.name = “名字”;

newMessageInfo.content = “通知内容”;

//调用接口

BLEManager.setNewMessageDetailInfo(newMessageInfo);

V3协议：

**接口名称：**

BLEManager.setV3MessageNotice(V3MessageNotice v3MessageNotice) v3 协议

**参数说明：**

|  |  |
| --- | --- |
| **V3MessageNotice** | |
| evtType | 信息类型 |
| name | 姓名 |
| phoneNumber | 电话号码 |
| dataText | 内容 |
|  |  |
|  |  |

type 的值已经在V3MessageNotice 类中定义好常量，例如:

/\*\*

\* 来电

\*/

public static final int TYPE\_CALL = 0x1000;

private static final int TYPE\_MSG\_BASE = 0x2000;

/\*\*

\* 短信

\*/

public static final int TYPE\_SMS = TYPE\_MSG\_BASE + 0x01;

/\*\*

\* 邮件

\*/

public static final int TYPE\_EMAIL = TYPE\_MSG\_BASE + 0x02;

/\*\*

\* 微信

\*/

public static final int TYPE\_WX = TYPE\_MSG\_BASE + 0x03;

public static final int TYPE\_QQ = TYPE\_MSG\_BASE + 0x04;

public static final int TYPE\_WEIBO = TYPE\_MSG\_BASE + 0x05;

public static final int TYPE\_FACEBOOK = TYPE\_MSG\_BASE + 0x06;

public static final int TYPE\_TWITTER = TYPE\_MSG\_BASE + 0x07;

public static final int TYPE\_WHATSAPP = TYPE\_MSG\_BASE + 0x08;

public static final int TYPE\_MESSENGER = TYPE\_MSG\_BASE + 0x09;

public static final int TYPE\_INSTAGRAM = TYPE\_MSG\_BASE + 0x0A;

public static final int TYPE\_LINKEDIN = TYPE\_MSG\_BASE + 0x0B;

public static final int TYPE\_CALENDAR = TYPE\_MSG\_BASE + 0x0C;

public static final int TYPE\_SKYPE = TYPE\_MSG\_BASE + 0x0D;

public static final int TYPE\_ALARM = TYPE\_MSG\_BASE + 0x0E;

public static final int TYPE\_VKONTAKTE = TYPE\_MSG\_BASE + 0x10;

public static final int TYPE\_LINE = TYPE\_MSG\_BASE + 0x11;

public static final int TYPE\_VIBER = TYPE\_MSG\_BASE + 0x12;

public static final int TYPE\_KAKAO\_TALK = TYPE\_MSG\_BASE + 0x13;

public static final int TYPE\_GMAIL = TYPE\_MSG\_BASE + 0x14;

public static final int TYPE\_OUTLOOK = TYPE\_MSG\_BASE + 0x15;

public static final int TYPE\_SNAPCHAT = TYPE\_MSG\_BASE + 0x16;

public static final int TYPE\_TELEGRAM = TYPE\_MSG\_BASE + 0x17;

public static final int TYPE\_OTHER = TYPE\_MSG\_BASE + 0X18;

public static final int TYPE\_CHATWORK = TYPE\_MSG\_BASE + 0x20;

public static final int TYPE\_SLACK = TYPE\_MSG\_BASE + 0x21;

public static final int TYPE\_MAIL\_YAHOO = TYPE\_MSG\_BASE + 0x22;

public static final int TYPE\_TUMBLR = TYPE\_MSG\_BASE + 0x23;

public static final int TYPE\_YOUTUBE = TYPE\_MSG\_BASE + 0x24;

public static final int TYPE\_PINTEREST\_YAHOO = TYPE\_MSG\_BASE + 0x25;

public static final int TYPE\_TIKTOK = TYPE\_MSG\_BASE + 0x26;

public static final int TYPE\_REDBUS = TYPE\_MSG\_BASE + 0X27;

public static final int TYPE\_DAILYHUNT= TYPE\_MSG\_BASE + 0X28;

public static final int TYPE\_HOTSTAR = TYPE\_MSG\_BASE + 0X29;

public static final int TYPE\_INSHORTS = TYPE\_MSG\_BASE + 0X2A;

public static final int TYPE\_PAYTM = TYPE\_MSG\_BASE + 0X2B;

public static final int TYPE\_AMAZON = TYPE\_MSG\_BASE + 0X2C;

public static final int TYPE\_FLIPKART = TYPE\_MSG\_BASE + 0X2D;

public static final int TYPE\_PRIME = TYPE\_MSG\_BASE + 0X2E;

public static final int TYPE\_NETFLIX = TYPE\_MSG\_BASE + 0X2F;

public static final int TYPE\_GPAY = TYPE\_MSG\_BASE + 0X30;

public static final int TYPE\_PHONPE = TYPE\_MSG\_BASE + 0X31;

public static final int TYPE\_SWIGGY = TYPE\_MSG\_BASE + 0X32;

public static final int TYPE\_ZOMATO = TYPE\_MSG\_BASE + 0X33;

public static final int TYPE\_MAKEMYTRIP = TYPE\_MSG\_BASE + 0X34;

public static final int TYPE\_JIOTV = TYPE\_MSG\_BASE + 0X35;

public static final int TYPE\_KEEP = TYPE\_MSG\_BASE + 0X36;

public static final int TYPE\_MICROSOFT = TYPE\_MSG\_BASE + 0X37;

public static final int TYPE\_WHATSAPP\_BUSINESS = TYPE\_MSG\_BASE + 0x38;

public static final int TYPE\_MISSED\_CALL = TYPE\_MSG\_BASE + 0X3A;

public static final int TYPE\_GPAP = TYPE\_MSG\_BASE + 0X3B;

public static final int TYPE\_YT\_MUSIC = TYPE\_MSG\_BASE + 0X3C;

public static final int TYPE\_UBER = TYPE\_MSG\_BASE + 0X3D;

public static final int TYPE\_OLA = TYPE\_MSG\_BASE + 0X3E;

public static final int TYPE\_MATTER = TYPE\_MSG\_BASE + 0X3F;

**代码示例：**

//发送通知不用注册回调监听

V3MessageNotice v3MessageNotice = new V3MessageNotice();

v3MessageNotice.evtType = V3MessageNotice.TYPE\_CALL;

v3MessageNotice.contact = contactName;

v3MessageNotice.phoneNumber = incomingNumber;

//调用接口

BLEManager.setV3MessageNotice(v3MessageNotice);

## 十 AGPS 更新

**1.先获取gps 状态**

**BLEManager.getGpsStatus()**

**代码示例：**

//先注册回调监听

BLEManager.registerGetGpsInfoCallBack(new GetDeviceInfoCallBack.ICallBack(){

@Override

Public void onGetGpsStatus(GpsStatus gpsStatus){

//这里返回GPS 状态

//GPS非空闲状态，不处理

if (GpsStatus.STATUS\_IDLE != gpsStatus.gps\_run\_status) {

return;

}

//下载Agps文件

downloadAgpsFile();//按协议实现

}

})

//调用接口

BLEManager.getGpsStatus()

/\*\*

\* 下载Agps文件处理

\*/

private void downloadAgpsFile() {

long timeMillis = System.currentTimeMillis();

//支持offline,距离上一次offline升级超过24小时，下载offline文件

if (isSupportOfflineUpgrade() && Math.abs(SPHelper.getLastAgpsOfflineUpgradeTime() - timeMillis) >= TIME\_MILLIONS\_OF\_DAY) {

String AGPS\_OFFLINE\_FILE\_URL = "http://offline-live1.services.u-blox.com/GetOfflineData.ashx?token=vB6zs0P4F0ayAYBMCzx4rw&gnss=gps,glo&period=1&resolution=1";//offline file url

//下载offline文件

//注意，需要把文件名字存储为 **agps.ubx （必须）**

}

//支持online,距离上一次online升级超过4小时，下载online文件

else if (isSupportOnlineUpgrade() && Math.abs(SPHelper.getLastAgpsOnlineUpgradeTime() - timeMillis) >= TIME\_MILLIONS\_OF\_4\_HOUR) {

String AGPS\_ONLINE\_FILE\_URL = "http://online-live1.services.u-blox.com/GetOnlineData.ashx?token=vB6zs0P4F0ayAYBMCzx4rw&gnss=gps,qzss,glo,bds,gal&datatype=eph&format=mga";//online file url

//下载online文件

//注意，需要把文件名字存储为**online.ubx （必须）**

}

}

/\*\*

\* 是否支持Agpe offline升级

\*

\* @return

\*/

private boolean isSupportOfflineUpgrade() {

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

return functionInfo != null && functionInfo.ex\_gps && functionInfo.agps\_offline;

}

/\*\*

\* 是否支持Agpe online升级

\*

\* @return

\*/

private boolean isSupportOnlineUpgrade() {

SupportFunctionInfo functionInfo = LocalDataManager.getSupportFunctionInfo();

return functionInfo != null && functionInfo.ex\_gps && functionInfo.agps\_online; }

2.文件传输（下载完gps 文件后进行文件传输）

IFileTransferListener mIFileTransferListener = new IFileTransferListener() {

@Override

public void onStart(){};

@Override

public void onProgress(int progress){};

@Override

public void onSuccess(){

//传输成功，

};

@Override

public void onFailed(String errorMsg){};

}

FileTransferConfig config = FileTransferConfig.getDefaultUbloxAGpsFileConfig(updatePath, mIFileTransferListener); //updatePath升级文件保存的路径

config.firmwareSpecName = fileName; //fileName 文件名称

config.maxRetryTimes = 0;

BLEManager.stopTranCommonFile();

BLEManager.startTranCommonFile(config);

## 十 音乐传输 （music）

1.注册监听

OperateCallBack.IMusicCallBack callBack = new OperateCallBack.IMusicCallBack() {

@Override

public void onInvalid(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onDeleteMusic(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onAddMusic(OperateCallBack.OperateType type, boolean isSuccess, int musicId) {

// BLEManager.addMusicFile(musicFile) result

//isSuccess 是否成功 musicId : 返回的音乐id

if(isSuccess && musicId>0){

//传输音乐

**transMusicFile();**

}

);

@Override

public void onDeleteFolder(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onAddFolder(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onModifyFolder(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onImportFolder(OperateCallBack.OperateType type, boolean isSuccess) {

}

@Override

public void onDeleteFolderMusic(OperateCallBack.OperateType type, boolean isSuccess) {

}

}

注册监听： BLEManager.registerOperateMusicCallBack(callback)

移除监听： BLEManager.unregisterOperateMusicCallBack(callback) (onDestory 回调里面)

**transMusicFile()**{

SPPFileTransferConfig fileTransferConfig =

SPPFileTransferConfig.getDefaultMusicFileConfig(

music.filePath,

this

);

fileTransferConfig.firmwareSpecName = filename;

BLEManager.startSppTranFile(fileTransferConfig, new ISPPConnectStateListener() {

@Override

public void onStart() {

}

@Override

public void onSuccess() {

//传输成功

}

@Override

public void onFailed() {

}

@Override

public void onBreak() {

}

)

}

1. 相关方法：

1. 添加音乐文件夹

BLEManager.addMusicFolder(MusicOperate.MusicFolder musicFolder)

（2）删除音乐文件夹

BLEManager.deleteMusicFolder(MusicOperate.MusicFolder musicFolder)

(3)更新音乐文件夹

BLEManager.updateMusicFolder(MusicOperate.MusicFolder musicFolder)

（4）查询音乐文件夹

BLEManager.queryMusicAndFolderInfo()

（5）添加音乐到文件夹

BLEManager.moveMusicIntoFolderMusicOperate.MusicFolder musicFolder()

1. 从文件夹删除音乐

BLEManager.removeMusicFromFolder()

(7) 添加音乐文件

BLEManager.addMusicFile(MusicOperate.MusicFile musicFile)

for example:

MusicOperate.MusicFile musicFile = new MusicOperate.MusicFile();

musicFile .music\_memory = 111;// music size

musicFile.music\_name = " boby"; //music name

musicFile.singer\_name = "zhangsan"; // singer name

BLEManager.addMusicFile(musicFile);

(8) 删除音乐文件

BLEManager.deleteMusicFile(MusicOperate.MusicFile musicFile)

设置后返回的成功和失败结果在OperateCallBack.IMusicCallBack;

MusicOperate 类数据信息：

public class MusicOperate {

public int music\_operate;//1：删除音乐； 2增加音乐

public int folder\_operate;//1：删除文件夹； 2增加文件夹 3更新文件夹 4把歌曲添加到文件夹（歌单） 5把歌曲移除文件夹

public MusicFolder folder\_items;//folder\_operate为0的时候这个字段不起作用，不占用空间

public MusicFile music\_items;//music\_operate为0时候这个字段就不起作用

public static class MusicFolder implements Serializable {

private static final long serialVersionUID = 1L;

public int folder\_id;

public String folder\_name;

public int music\_num;

public List<Integer> music\_index;

}

public static class MusicFile implements Serializable {

private static final long serialVersionUID = 1L;

public int music\_id;

public long music\_memory;

public String music\_name;

public String singer\_name;

}

public static class OperateResponse{

public int err\_code;//0成功，非0失败

public int operate\_type;//0：无效操作； 1:删除音乐； 2:增加音乐 3:删除文件夹； 4:增加文件夹；5：修改歌单 6：导入歌单 7：歌单中音乐删除

public int music\_id;//音乐id 当操作为 2:增加音乐成功时返回音乐id

}

public static class MusicAndFolderInfo {

public long all\_memory;//总的可用空间

public long useful\_memory;//可用的空间-可能存在空间的碎片，可用空间不一定是总的空间-已经使用的空间

public long used\_memory;//已经使用的空间

public int folder\_num;

public int music\_num;

public List<MusicFolder> folder\_items;

public List<MusicFile> music\_items;

}

}

## 十一 图库升级

1. 获取flash 信息

调用接口：

BLEManager.getFlashBinInfo();

**代码示例：**

//先注册回调监听

BLEManager.registerGetDeviceInfoCallBack(new BaseDeviceInfoCallback(){

@Override

public void onGetFlashBinInfo(FlashBinInfo flashBinInfo) {

if(flashBinInfo.version != flashBinInfo.mactchBVersion){

//需要升级

}

}

}

})

BLEManager.getFlashBinInfo();

FlashBinInfo对象说明：

public int version; // 当前图库版本

public int matchVersion;//匹配的图库版本

注意：version 与 matchVersion 不相等需要升级

1. 字库升级
2. 从服务器请求获取matchVersion 图库信息（需要在后台服务器配置）
3. 下载图库包
4. 发送给设备

调用接口：跟文件发送方法一样

BLEManager.startTranCommonFile(FileTransferConfig config);

example:

IFileTransferListener mIFileTransferListener = new IFileTransferListener() {

@Override

public void onStart(){};

@Override

public void onProgress(int progress){};

@Override

public void onSuccess(){};

@Override

public void onFailed(String errorMsg){};

}

FileTransferConfig config = FileTransferConfig.getDefaultApolloOTAConfig(updatePath, mIFileTransferListener);//updatePath升级文件保存的路径

config.macRetryTimes = 0;

config.zipType = FileTransferConfig.ZIP\_TYPE\_FAST\_LZ;`

BLEManager.startTranCommonFile(config);

## 十二 BT升级

1. 获取bt 版本号，当BT\_version 与 BT\_mach\_version 不一致时需要进行升级,具体获取参考（四--6获取bt 版本号）
2. 升级BT

（1）从服务器请求获取BT\_mach\_version的BT信息（需要在后台服务器配置）

(2)下载BT 固件包

(3)发送给设备

调用接口：跟文件发送方法一样

BLEManager.startTranCommonFile(FileTransferConfig config);

example:

IFileTransferListener mIFileTransferListener = new IFileTransferListener() {

@Override

public void onStart(){};

@Override

public void onProgress(int progress){};

@Override

public void onSuccess(){};

@Override

public void onFailed(String errorMsg){};

}

FileTransferConfig config = FileTransferConfig.getDefaultApolloOTAConfig(updatePath, mIFileTransferListener);//updatePath升级文件保存的路径

config.macRetryTimes = 0;

config.zipType = FileTransferConfig.ZIP\_TYPE\_FAST\_LZ;

BLEManager.startTranCommonFile(config);