

## 杰理蓝牙控制库\_IOS\_SDK开发说明

- 本项目所参考、使用技术必须全部来源于公知技术信息，或自主创新设计。
- 本项目不得使用任何未经授权的第三方知识产权的技术信息。
- 如个人使用未经授权的第三方知识产权的技术信息，造成的经济损失和法律后果由个人承担。

### 杰理蓝牙控制库\_IOS\_SDK开发说明

#### 版本历史

#### 概述

#### 开发说明

##### 运行环境

##### 前提

##### 注意

#### 1. 蓝牙连接部分

##### 1.1 蓝牙初始化

##### 1.2 蓝牙操作API

##### 1.3 蓝牙状态

#### 2. 语音传输部分

#### 3. 获取设备信息

##### 3.1 获取设备信息

#### 4. 文件浏览

#### 5. 百度语音识别接口:BDKit

#### 6. Deepbrain接口类:DBrain\_Http

##### 6.1 mpush接口

#### 7. 图灵接口

##### 7.1 使用初始化流程图

##### 7.2 MQTT 使用

##### 7.3 API 接入方式相关接口

##### 7.4 API方式请求返回内容

##### 7.5 AI - WIFI 接入参数相关类

##### 7.6 AI-WIFI 请求返回

##### 7.7 TuringKit主要交互请求

##### 7.8 TTS 语音合成返回内容

#### 8. 收音模式（非AI模式）

#### 9. 歌词显示

#### 10. 设备音乐快进或者快退

#### 11. EQ设置

#### 12. OTA

- 12.1 版本校对,并获取升级文件
  - 12.2 执行OTA升级
- 13. 区分AI模式和标准模式（非AI模式）
  - 13.1 标准模式（非AI模式）
  - 13.2 AI模式
- 14. 闹钟功能
  - 14.1 设置/增加闹钟
  - 14.2 删除闹钟
  - 14.3 闹钟正在响或则闹钟停止响
  - 14.4 停止闹钟响声回调
- 15. 服务器OTA升级
  - 15.1 OTA升级文件下载
  - 15.2 OTA升级设备
  - 15.3 OTA的升级状态
  - 15.4 OTA的返回结果
  - 15.5 获取MD5的数据
- 16. 对耳的接口
  - 16.1 获取设备的图片
  - 16.2 设置EDR名字
  - 16.3 按键设置(对耳)
  - 16.4 LED设置(对耳)
  - 16.5 MIC设置(耳机)
  - 16.6 工作模式(耳机)
  - 16.7 同步时间戳(耳机)
  - 16.8 获取设备信息(耳机)
  - 16.9 设备广播通知(耳机)
  - 16.10 关闭或开启设备广播(耳机)
  - 16.11 用于ADV设置同步后需要主机操作的行为。
- 17. 智能充电仓
  - 17.1 通知固件App的信息
  - 17.2 设置通讯MTU
  - 17.3 开启蓝牙扫描
  - 17.4 推送蓝牙扫描结果
  - 17.5 停止蓝牙扫描（APP-->固件）
  - 17.6 停止蓝牙扫描（固件-->APP）
  - 17.7 通知固件连接指定的蓝牙设备
  - 17.8 文件传输 【固件-->APP】
  - 17.9 文件传输 【APP-->固件】
- 18. ID3控制
  - 18.1 播放/暂停
  - 18.2 上一曲
  - 18.3 下一曲
  - 18.4 开启/暂停 音乐信息推送
  - 18.5 主动设置ID3播放状态
- 19. 动态调节EQ段数
  - 19.1 获取EQ的相关数据
- 20. 设置音效的高低音
- 21. 自定义命令
  - 21.1 自定义命令收发接口

- 22. SD卡/U盘的目录浏览
  - 22.1 监听目录数据
  - 22.2 浏览目录
  - 22.3 清除设备音乐缓存记录
  - 22.4 快进快退
- 23. Linein
  - 23.1 切换到Linein模式
  - 23.2 获取Linein的状态
  - 23.3 设置Linein下的播放和暂停
- 24. 手机和设备互找
  - 24.1 设备查找手机的通知，携带了响铃时长
  - 24.2 查找设备命令
- 25. 耳机降噪模式设置
- 26. 多设备发送和接收数据
- 27. 声卡功能配置
- 28. 手表切换表盘(OTA功能)
  - 28.1 前提
  - 28.2 蓝牙控制库的表盘相关接口
  - 28.3 FatsObject操作API的介绍
  - 28.4 外部操作使用流程
    - STEP.1 设置命令中心类
    - STEP.2 获取外挂Flash的信息
    - STEP.3 初始化本地FATFS系统
    - 读取表盘(文件)名字
    - 新增表盘(文件)
    - 删除表盘(文件)
    - 用户设置当前表盘
    - 用户获取当前表盘
    - 设备主动切换的表盘
    - 获取FAT系统剩余空间

## 版本历史

---

版本	日期	撰写者	修改记录
1.11.0	2020/12/17	李放	增加手表切换表盘(OTA功能)
1.10.0	2020/12/14	李放	增加声卡功能配置
1.9.0	2020/10/08	李放	增加多设备发送和接收数据接口
1.8.0	2020/08/03	李放	增加U盘、SD卡浏览，外部音源显示，手机和设备互找，耳机主动降噪命令
1.7.0	2020/06/18	李放	增加ID3控制、动态调节EQ段数、设置音效的高低音、OTA升级添加MD5校验、添加自定义命令
1.6.0	2020/03/22	冯洪鹏	智能充电仓接口
1.5.0	2019/08/15	李放	添加对耳的接口
1.4.0	2019/04/04	李放	添加闹钟功能、服务器OTA升级
1.3.0	2019/02/23	李放	添加OTA升级、收音模式（非AI模式）、EQ、设备音乐的歌词同步、设备音乐的快进快退
1.2.0	2019/01/12	李放	区分AI模式和标准模式（非AI模式）
1.1.0	2018/12/18	冯洪鹏	接口封装优化
1.0.0	2018/11/23	冯洪鹏	初始版本，接口制定

## 概述

本文档是为了方便后续项目维护和管理、记录开发内容而创建。

## 开发说明

# 运行环境

类别	兼容范围	备注
系统	IOS 10.0以上系统	支持BLE功能
硬件要求	杰理蓝牙产品	
开发平台	Xcode10.0以上	建议使用最新版本开发

## 前提

需要在工程中导入以下Frameworks

- 导入以下库：
  - `JL_BLEKit.framework` 所有的API都集中于此，详情请看【`JL_ManagerM`】类；
  - `DFUnits.framework` 依赖的基础库，项目开发中提高高发效率；
  - `SpeexKit.framework` **Space**语音解析器；
  - `JL_RunSDK.h`
- 在Xcode11的Info加入键值 *Privacy - Bluetooth Peripheral Usage Description*；
- 在Xcode11的Info加入键值 *Privacy - Bluetooth Always Usage Description*；
- 将Xcode BuildSetting中的Enable Bitcode设置为No;

## 注意

- 在IOS 10.0以上手机系统运行；
- 导入`JL_RunSDK`类，且在一个单例中实例化此类；
- 此版本SDK中，主要使用`JL_ManagerM.h`的API操作SDK；

## 1. 蓝牙连接部分

### 1.1 蓝牙初始化

```
/*--- 初始化JL_SDK ---*/
self.mBleMultiple = [[JL_BLEMultiple alloc] init];
self.mBleMultiple.BLE_FILTER_ENABLE = YES;
self.mBleMultiple.BLE_PAIR_ENABLE = YES;
self.mBleMultiple.BLE_TIMEOUT = 7;

/*--- 开启蓝牙【搜索】【弹窗】 ---*/
[self noteBleScanOpen:nil];
```

### 1.2 蓝牙操作API

```

//
// JL_BLEMultiple.h
// JL_BLEKit
//
// Created by 杰理科技 on 2020/9/1.
// Copyright © 2020 www.zh-jieli.com. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <AVFoundation/AVFoundation.h>
#import <CoreBluetooth/CoreBluetooth.h>
#import <UIKit/UIKit.h>

#import "JL_EntityM.h"

NS_ASSUME_NONNULL_BEGIN

/**
 * BLE状态通知
 */
extern NSString *kJL_BLE_M_FOUND; //发现设备
extern NSString *kJL_BLE_M_FOUND_SINGLE; //发现单个设备
extern NSString *kJL_BLE_M_ENTITY_CONNECTED; //连接有更新
extern NSString *kJL_BLE_M_ENTITY_DISCONNECTED; //断开连接
extern NSString *kJL_BLE_M_ON; //BLE开启
extern NSString *kJL_BLE_M_OFF; //BLE关闭
extern NSString *kJL_BLE_M_EDR_CHANGE; //经典蓝牙输出通道变化

@interface JL_BLEMultiple : NSObject
@property(nonatomic,strong)NSData *__nullable filterKey; // 过滤码
@property(nonatomic,strong)NSData *__nullable pairKey; // 配对码
@property(nonatomic,assign)BOOL BLE_IS_CONNECTING; // 是否有设备正在连接
@property(nonatomic,assign)BOOL BLE_FILTER_ENABLE; // 是否【开启过滤】
@property(nonatomic,assign)BOOL BLE_PAIR_ENABLE; // 是否【开启配对】
@property(nonatomic,assign)int BLE_TIMEOUT; // 连接超时时间

@property(nonatomic,assign)CBManagerState bleManagerState; // 蓝牙状态
@property(nonatomic,strong)NSMutableArray *blePeripheralArr; // 发现的设备
@property(nonatomic,strong)NSMutableArray *bleConnectedArr; // 已连接的设备

```

```

@property(nonatomic,strong) NSString                *JL_BLE_SERVICE;           //
服务号
@property(nonatomic,strong) NSString                *JL_BLE_PAIR_W;             //
配对【写】通道
@property(nonatomic,strong) NSString                *JL_BLE_PAIR_R;             //
配对【读】通道
@property(nonatomic,strong) NSString                *JL_BLE_AUIDO_W;            //
音频【写】通道
@property(nonatomic,strong) NSString                *JL_BLE_AUIDO_R;            //
音频【读】通道
@property(nonatomic,strong) NSString                *JL_BLE_RCSP_W;             //
命令【写】通道
@property(nonatomic,strong) NSString                *JL_BLE_RCSP_R;             //
命令【读】通道

+(NSString*)versionOfSDK;

/**
  开始搜索
 */
-(void)scanStart;

/**
  继续搜索
 */
-(void)scanContinue;

/**
  停止搜索
 */
-(void)scanStop;

/**
  通过UUID生成Entity。
 */
-(JL_EntityM*)makeEntityWithUUID:(NSString*)uuid;

/**
  连接设备
  @param entity 蓝牙设备类
 */
-(void)connectEntity:(JL_EntityM*)entity Result:(JL_EntityM_STATUS_BK)result;

/**
  断开连接
 */
-(void)disconnectEntity:(JL_EntityM*)entity Result:
(JL_EntityM_STATUS_BK)result;

```

```

/**
 更新名字信息
*/
-(void)updateHistoryRename:(NSString*)name withUuid:(NSString*)uuid;

/**
 返回经典蓝牙信息
@return @{@"ADDRESS":@"7c9a1da7330e",
        @"TYPE"     :@"BluetoothA2DPOutput",
        @"NAME"     :@"earphone"}

*/
+(NSDictionary*)outputEdrInfo;

@end

NS_ASSUME_NONNULL_END

```

## 1.3 蓝牙状态

```

extern NSString *kUI_JL_SHOW_ID3;
extern NSString *kUI_JL_CARD_MUSIC_INFO;
extern NSString *kUI_JL_ELSATICVIEW_BTN;
extern NSString *kUI_JL_EFCIRCULAR_BEGIN_TOUCH;
extern NSString *kUI_JL_EFCIRCULAR_END_TOUCH;
extern NSString *kUI_JL_REVERBERATION_END_TOUCH;

typedef NS_ENUM(UInt8, JLUuidType) {
    JLUuidTypeDisconnected = 0,    //未连接的UUID
    JLUuidTypeConnected    = 1,    //已连接的UUID
    JLUuidTypeInUse        = 2,    //正在使用的UUID
    JLUuidTypeNeedOTA      = 3,    //UUID需要OTA
    JLUuidTypePreparing    = 4,    //正在准备的UUID
};

typedef NS_ENUM(UInt8, JLDeviceChangeType) {
    JLDeviceChangeTypeConnectedOffline = 0,    //断开已连接的设备
    JLDeviceChangeTypeInUseOffline     = 1,    //断开正在使用的设备
    JLDeviceChangeTypeSomethingConnected = 2,    //有设备连接上
    JLDeviceChangeTypeManualChange     = 3,    //手动切换设备
    JLDeviceChangeTypeBleOFF           = 4,    //蓝牙已关闭
};

extern NSString *kUI_JL_DEVICE_CHANGE;
extern NSString *kUI_JL_DEVICE_PREPARING;
extern NSString *kUI_JL_DEVICE_SHOW_OTA;

extern NSString *kUI_JL_BLE_SCAN_OPEN;
extern NSString *kUI_JL_BLE_SCAN_CLOSE;

@interface JL_RunSDK : NSObject

```



```

@property(strong, nonatomic) JL_BLEMultiple *mBleMultiple;
@property(weak , nonatomic) JL_EntityM * __nullable mBleEntityM;
@property(strong, nonatomic) NSString * __nullable mBleUUID;

+ (id)sharedMe;

/**
    使用UUID切换设备
 */
+ (void)setActiveUUID: (NSString*)uuid;

/**
    使用UUID获取已连接的Entity
 */
+ (JL_EntityM*)getEntity: (NSString*)uuid;

/**
    获取当前设备状态
    0: 未连接的UUID
    1: 已连接的UUID
    2: 正在使用的UUID
    3: UUID需要OTA
    4: 正在准备的UUID
 */
+ (JLUuidType)getStatusUUID: (NSString*)uuid;

/**
    获取连接状态对应中文解析
 */
+ (NSString *)textEntityStatus: (JL_EntityM_Status)status;

+ (BOOL)isCurrentDeviceCmd: (NSNotification*)note;

@end
NS_ASSUME_NONNULL_END

```

## 2. 语音传输部分

```

/**
    语音操作状态
    kJL_MANAGER_KEY_OBJECT ==> JLModel_SPEEX
 */
extern NSString *kJL_MANAGER_SPEEX;

/**
    语音数据
    kJL_MANAGER_KEY_OBJECT ==> NSData
 */

```

```
extern NSString *kJL_MANAGER_SPEEX_DATA;

/**
 发送命令给音箱，允许音箱端开始接收语音，音箱收到这个消息后会发一个提示音
 */
-(void)cmdAllowSpeak;

/** 拒绝录音
 发送命令给音箱，不允许接收语音
 */
-(void)cmdRejectSpeak;

/** 停止语音
 发发送命令给音箱，停止接收数据，即检测到断句
 */
-(void)cmdSpeakingDone;
```

### 3. 获取设备信息

```
#pragma mark ---> 用户自定义数据
/**
  @param data 数据
  @param result 回复
 */
-(void)cmdCustomData:(NSData* __nullable)data
                    Result:(JL_CMD_BK __nullable)result;
#pragma mark ---> 设备返回的自定义数据
extern NSString *kJL_MANAGER_CUSTOM_DATA;

//设备操作API
/**
  获取设备信息
 */
+(void)cmdTargetFeatureResult:(JL_CMD_BK __nullable)result;

/**
  断开经典蓝牙
  @param result 回复
 */
+(void)cmdDisconnectEdrResult:(JL_CMD_BK __nullable)result;

/**
  拨打电话
  @param number 电话号码
  @param result 回复
 */
+(void)cmdPhoneCall:(NSString*)number Result:(JL_CMD_BK __nullable)result;
```

```

/**
    获取系统信息
    @param function JL_FunctionCode
    @param result 回复
    */
+(void)cmdGetSystemInfo:(JL_FunctionCode)function Result:(JL_CMD_BK
__nullable)result;

/**
    设置系统音量
    @param volume 音量值
    */
+(void)cmdSetSystemVolume:(UInt8)volume;

/**
    设置系统时间
    @param date 时间类
    */
+(void)cmdSetSystemTime:(NSDate*)date;

/**
    设置播放模式
    @param mode 模式
    */
+(void)cmdSetSystemPlayMode:(UInt8)mode;

/**
    通用、BT、Music、RTC、Aux
    @param function 功能类型
    @param cmd 操作命令
    @param ext 扩展数据
    @param result 回复
    */
+(void)cmdFunction:(JL_FunctionCode)function
    Command:(UInt8)cmd
    Extend:(UInt8)ext
    Result:(JL_CMD_BK __nullable)result;

/**
    监听目录数据
    @param result 状态回复
    */
+(void)cmdBrowseMonitorResult:(JL_FILE_BK)result;

/**
    浏览目录
    @param model 文件Model
    @param number 读取的数量
    */

```

```

+ (void)cmdBrowseModel:(JLFileModel*)model
    Number:(uint8_t)number;

/**
  用户自定义数据
  @param data 数据
  @param result 回复
  */
+ (void)cmdCustomData:(NSData* __nullable)data
    Result:(JL_CMD_BK __nullable)result;

```

## 3.1 获取设备信息

```

/**
  获取设备信息
  */
+ (void)cmdTargetFeatureResult:(JL_CMD_BK __nullable)result;

/**
  取出设备信息，详细描述请查看：JLDeviceModel
  */
+ (JLDeviceModel *)outputDeviceModel;

```

## 4. 文件浏览

```

[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdBrowseMonitorResult:^(NSArray * __Nullable array, JL_BrowseReason reason) {
    switch (reason) {
        case JL_BrowseReasonReading:{
            [wself updateData:array];

            NSLogEx(@"正在读取:%lu", (unsigned long)array.count);
        }break;
        case JL_BrowseReasonCommandEnd:{
            [wself updateData:array];
            self->reqNum-=10;
            if (self->reqNum>0) {
                [wself request:self->reqModel];
            }
            NSLogEx(@"读取命令结束:%lu delegate:%@", (unsigned
long)array.count, self->_delegate);
        }break;
        case JL_BrowseReasonFolderEnd:{
            [wself updateData:array];

            if (array.count == 0 && self->reqNum != -1) {
                if ([self->_delegate
respondToSelector:@selector(dmHandleWithItemModelArray:)] {

```

```

        [self->_delegate dmHandleWithItemModelArray:array];
    }
}

self->reqNum = -1;
NSLogEx(@"目录读取结束:%lu delegate:%@",(unsigned
long)array.count,self->_delegate);
}break;
case JL_BrowseReasonBusy:{
    NSLogEx(@"设备在忙");
}break;
case JL_BrowseReasonDataFail:{
    NSLogEx(@"数据读取失败");
}break;
case JL_BrowseReasonPlaySuccess:{
    [wself updatePlay];
    self->playItem = NO;
    NSLogEx(@"播放成功");
}break;
case JL_BrowseReasonUnknown:{
    NSLogEx(@"未知错误");
    self->reqNum = -1;
    self->playItem = NO;
    [self requestWith:self->reqModel Number:10];
}
default:
    break;
}
}
}
};

```

## 5. 百度语音识别接口:BDKit

```
#import <Foundation/Foundation.h>
#import <AVFoundation/AVFoundation.h>
#import <DFUnits/DFUnits.h>

typedef void(^BD_SPEAK_END) (void);
typedef void(^BD_SPEAK_DATA)(NSData*data);
typedef void(^BD_SPEAK_ASR) (NSDictionary*info);

@interface BDKit : NSObject

@property(n nonatomic, strong) NSString * BD_API_KEY;
@property(n nonatomic, strong) NSString * BD_SECRET_KEY;
@property(n nonatomic, strong) NSString * BD_APP_ID;
@property(n nonatomic, assign) int BD_PER; //0为普通女声, 1为普通男生, 3为情感合成-度逍遥, 4为情感合成-度丫丫
@property(n nonatomic, assign) int BD_SPD; //语速, 取值0-15, 默认为5中语速
```

```

@property(nonatomic,assign)int          BD_PIT;          //音调,取值0-15,默认为5中语
调
@property(nonatomic,assign)int          BD_VOL;          //音量,取值0-15,默认为5中音
量
@property(nonatomic,assign)BOOL         BD_CACHE;        //是否保存PCM缓存
#pragma mark - 语音合成 TTS
-(void)sayWords:(NSString*)text;
-(void)sayWords:(NSString *)text End:(BD_SPEAK_END)end;
-(void)sayWords:(NSString *)text End:(BD_SPEAK_END)end IsCache:(BOOL)cache;
-(void)sayWords:(NSString *)text Data:(BD_SPEAK_DATA)data;

-(void)sayStop;
-(void)sayQuit;
-(BOOL)hasSpeakEnd;

#pragma mark - 语音识别 ASR
-(void)asrPath:(NSString*)path Asr:(BD_SPEAK_ASR)asr;
@end
```

```

## 6. Deepbrain接口类:DBrain\_Http

```

#import "DBrain_Http.h"
#import "Mpush.h"

@implementation DBrain_Http

+ (void)postWithTxt:(NSString*)txt
result:(void(^)(NSDictionary *response,
NSDictionary *head))result
{
    NSString *requestUrl = @"http://api.deepbrain.ai:8383/deep-brain-api/ask";
    NSURLSessionConfiguration *cf = [NSURLSessionConfiguration
defaultSessionConfiguration];
    AFURLSessionManager *manager = [[AFURLSessionManager alloc]
initWithSessionConfiguration:cf];
    NSMutableURLRequest *request = [[AFHTTPRequestSerializer serializer]
requestWithMethod:@"POST"
URLString:requestUrl
parameters:nil
error:nil];
    request.timeoutInterval= 10;
    [request setValue:@"application/json" forHTTPHeaderField:@"Content-Type"];

    /* 设置请求头 */
    NSString *nonceStr = [self getNonceStr];
    NSString *dateStr = [self getDataFommortStr];

```

```

NSString *keyStr = [NSString
stringWithFormat:@"%@@%@",nonceStr,dateStr,APP_ROBOTID];
NSString *privateKey = [self sha1:keyStr isRepeat:NO];

NSDictionary *accessToken = @{@"createdTime":dateStr,
@"nonce":nonceStr,
@"privateKey":privateKey};

NSDictionary *apiAccount = @{@"appId":APP_APPID,
@"deviceId":APP_LICENSE,
@"robotId":APP_ROBOTID,
@"userId":APP_LICENSE};
//设置body
NSDictionary *body = @{@"location"::@{@"cityName":[[JL_Listen sharedMe]
myCity]},
@"nlpData"::@{@"inputText":txt},
@"requestHead"::@{@"accessToken":accessToken,
@"apiAccount":apiAccount},
@"simpleView":@YES};
NSString *bodyStr = [DFTools dictionaryToJson:body];
NSData *bodyData = [bodyStr dataUsingEncoding:NSUTF8StringEncoding];
[request setHTTPBody:bodyData];

AFHTTPResponseSerializer *responseSerializer = [AFHTTPResponseSerializer
serializer];
responseSerializer.acceptableContentTypes = [NSSet
setWithObjects:@"application/json",
@"text/html",
@"text/json",
@"text/javascript",
@"text/plain",
nil];
manager.responseSerializer = responseSerializer;
NSURLSessionDataTask *task = [manager dataTaskWithRequest:request
completionHandler:^(NSURLResponse * _Nonnull response,
id _Nullable responseObject,
NSError * _Nullable error)
{
if (!error) {
NSDictionary *ret = [DFTools jsonWithData:responseObject];
NSLog(@"%@",ret);

NSHTTPURLResponse *response_1 = (NSHTTPURLResponse *)response;
NSDictionary *allHeaders = response_1.allHeaderFields;
if (result) result(ret,allHeaders);

} else {
NSLog(@"%@",error);
}
}

```

```

if (result) result(nil, nil);
}
}];
[task resume];

}

//日期格式
+ (NSString *)getDataFommortStr {
NSDate *date = [NSDate date];
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];

//    //创建日期格式化对象
[formatter setDateFormat:@"%yyyy-MM-dd"];
//    //将格式串转换为日期对象
NSString *dateStr1 = [formatter stringFromDate:date];
[formatter setDateFormat:@"%HH:mm:ss"];
NSString *dateStr2 = [formatter stringFromDate:date];
NSString *dateStr = [NSString stringWithFormat:@"%T%", dateStr1, dateStr2];
return dateStr;
}

//nonce 随机串
+ (NSString *)getNonceStr {
NSString *uuidStr = [NSUUID UUID].UUIDString;
NSString *encodeBase64StringSig = [self sha1:uuidStr isRepeat:YES];
return encodeBase64StringSig;
}

////sha1加密
+ (NSString *)sha1:(NSString *)str isRepeat:(BOOL)isRepeat {
const char *cstr = [str cStringUsingEncoding:NSUTF8StringEncoding];
NSData *data = [NSData dataWithBytes:cstr length:str.length];

uint8_t digest[CC_SHA1_DIGEST_LENGTH];
CC_SHA1(data.bytes, (unsigned int)data.length, digest);
//NSLog(@"%lu", sizeof(digest));
if (isRepeat) {
uint8_t digest1[CC_SHA1_DIGEST_LENGTH];
CC_SHA1(digest, sizeof(digest), digest1);
NSString *base64Str = [self encodeBase64:[NSData dataWithBytes:digest1
length:16] ];
return base64Str;
} else {
NSString *base64Str = [self encodeBase64:[NSData dataWithBytes:digest
length:sizeof(digest)]];
}
}

```



```
return base64Str;
}
}

+(NSString*)encodeBase64:(NSData*)data{
NSData *base64Data = [data base64EncodedDataWithOptions:0];
NSString *baseString = [[NSString alloc] initWithData:base64Data
encoding:NSUTF8StringEncoding];
return baseString;
}
@end
```
```

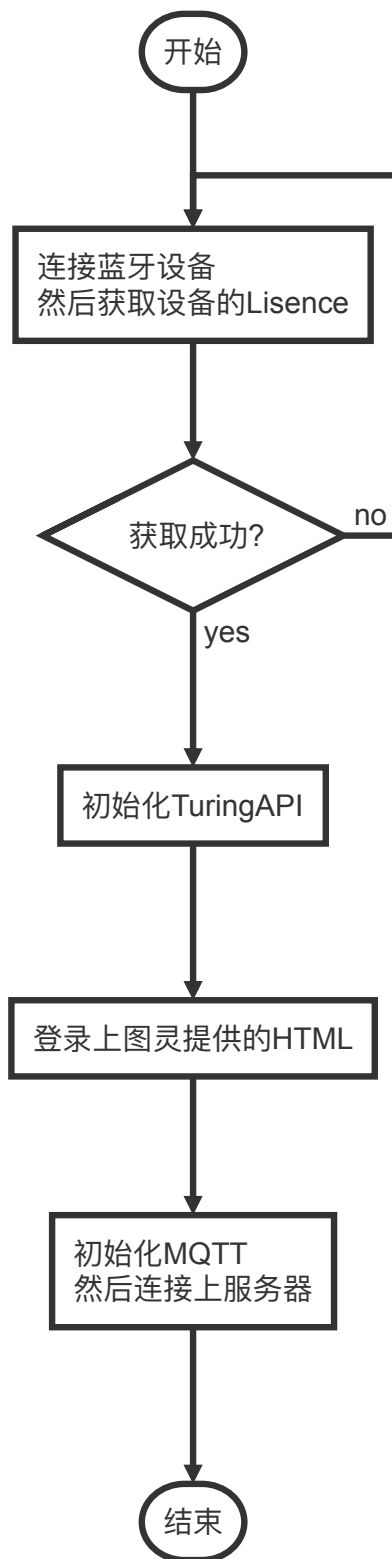
## 6.1 mpush接口

```
//saveUUID
[self saveUUID];
//配置app参数
[JSConfigureApp configureApp];
//配置mpush
[self setupMpush];
@end
```
```

## 7. 图灵接口

---

### 7.1 使用初始化流程图



## 7.2 MQTT 使用

```
//  
// MqttClientManager.h  
// IntelligentBox  
//  
// Created by Ezio on 2018/4/7.  
// Copyright © 2018 Zhuhia Jieli Technology. All rights reserved.
```

```

//

#import <Foundation/Foundation.h>

#define MQTT_HOST @"mqtt.ai.tuling123.com"
#define MQTT_PORT 10883
#define MQTT_USER @"37141b13150b44a5bad1222163f60bef"
#define MQTT_PWD @"51a84ef7"

#define MQTT_DID_PAUSE @"DID_PAUSE_MQTT" //暂停播放
#define MQTT_DID_PLAY @"DID_PLAY_MQTT" //播放链接

@interface MqttClientManager : NSObject

@property(nonatomic,strong) NSString *topic;

+(instancetype)shareInstanced;

/**
设置ClientID And Connect

@param clientid clientID
*/
-(void)stepUpClientID:(NSString *)clientid;

/**
订阅Topic

@param topic Topic
*/
-(void)mqSubscribeToTopic:(NSString *)topic;

/**
向Topic发送数据

@param data 数据内容
@param topic Topic
*/
-(void)mqPublishAndWaitData:(NSData *)data toTopic:(NSString *)topic;

@end

```

## 7.3 API 接入方式相关接口

```
//
//  TuringAPIEntry.h
//  IntelligentBox
//
//  Created by Ezio on 2018/4/9.
//  Copyright © 2018 Zhuhia Jieli Technology. All rights reserved.
//

#import <Foundation/Foundation.h>

typedef void(^RespBlock)(id result);

@interface TuringAPIEntry : NSObject

@property(nonatomic,strong) NSString *apikey;
@property(nonatomic,strong) NSString *secret;
@property(nonatomic,strong) NSString *userId;

+(instancetype)shareInstanced;

/**
获取API的UserID

@param uniqueId licence
*/
-(void)apiGetUserIdWith:(NSString *)uniqueId;

/**
循环获取某个关键字段的内容，比方说歌曲

@param text 关键字
@param index 循环获取次数
@param block 返回列表
*/
-(void)apiOpenApiRequest:(NSString *)text WithIndex:(int) index Result:
(RespBlock) block;

@end
```

## 7.4 API方式请求返回内容

```

//
//  TuringAPIResponse.h
//  IntelligentBox
//
//  Created by Ezio on 2018/4/9.
//  Copyright © 2018 Zhuhia Jieli Technology. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface TuringAPIResponse : NSObject

@property(nonatomic,strong) NSString *singer;
@property(nonatomic,strong) NSString *url;
@property(nonatomic,strong) NSString *song;
@property(nonatomic,strong) NSString *name;
@property(nonatomic,strong) NSString *originalSong;
@property(nonatomic,assign) int songId;

+(TuringAPIResponse *)apiResponseToObject:(NSDictionary *)dict;

+(NSDictionary *)apiResponseToDictionary:(TuringAPIResponse *)response;

@end

```

## 7.5 AI - WIFI 接入参数相关类

```

//
//  TuringParameters.h
//  TuringDemo
//
//  Created by Ezio on 2018/1/5.
//  Copyright © 2018年 Ezio. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface TuringParameters : NSObject

/**
//apiKey,用于权限验证
*/
@property(nonatomic,strong) NSString *ak;

```

```
/**
//设备ID加密后的字符串
*/
@property(nonatomic,strong) NSString *uid;

/**
//请求令牌，首次请求可以为空
*/
@property(nonatomic,strong) NSString *token;

/**
//针对上传音频字段设置控制， 当asr=0时： pcm_16K_16bit（默认）；
当asr=1时： pcm_8K_16bit；
当asr=2时： amr_8K_16bit；
当asr=3时： amr_16K_16bit；
当asr=4时： opus；
当asr=5时： speex（需要用特定的编码工具）。
*/
@property(nonatomic,strong) NSString *asr;

/**
//需要合成的文本
*/
@property(nonatomic,strong) NSString *text;

/**
//语义解析输出结果音频格式控制，

当tts=0时： pcm_8K_16bit（默认）；
当tts=1时： mp3_64；
当tts=2时： mp3_24；
当tts=3时： mp3_16；
当tts=4时： amr_nb.

*/
@property(nonatomic,strong) NSString *tts;

/**
结果输出控制标识，
当flag=0时： 不输出文本（默认）；
当flag=1时： 输出asr文本信息；
当flag=2时： 输出tts文本信息；
当flag=3时： 输出asr&tts文本信息；
*/
@property(nonatomic,strong) NSString *flag;

/**
流式识别控制字段，
```

```
当realTime=0时：非流式识别（默认）；
当realTime=1时：流式识别。
*/
@property(nonatomic,strong) NSString *realTime;

/**
当realTime=1为流式识别时，此字段必选，用以标识音频片段索引。
index从1开始计数，且最后一个音频片段索引必须为负数，如index=1、2、3、-4。
*/
@property(nonatomic,strong) NSString *index;

/**
当realTime=1时，该字段有效，用于标识一个流式识别过程，所以每个流式识别过程该identify值必须
保证唯一性。
*/
@property(nonatomic,strong) NSString *identify;

/**
上传音频编码方式，主要用于自定义编码支持，非自定义编码可忽略该字段，
当encode=0时：通用编码，即asr字段支持的编码方式（默认）；
当encode=1时：自定义编码，若使用该编码方式，则需要提供转码工具，且转码目标格式必须是asr字段支
持的格式。
*/
@property(nonatomic,strong) NSString *encode;

/**
请求类型标识。
type=0时：智能聊天（默认）；
type=1时：主动交互；
type=2时：提示语
*/
@property(nonatomic,strong) NSString *type;

/**
tts语速设置，取值范围1~9， 默认5
*/
@property(nonatomic,strong) NSString *speed;

/**
tts语调设置，取值范围1~9， 默认5
*/
@property(nonatomic,strong) NSString *pitch;

/**
tts发音人选择 取值范围0~2，默认 0
```

```

*/
@property(nonatomic,strong) NSString *tone;

/**
TuringParameters对象转字典

@param objc TuringParameters
@return dict
*/
+(NSDictionary *)turingTTSPParametersToDic:(TuringParameters *)objc;

/**
TuringParameters对象转字典

@param objc TuringParameters
@return dict
*/
+(NSDictionary *)turingParametersToDic:(TuringParameters *)objc;

/**
字典转TuringParameters

@param dict 字典
@return TuringParameters
*/
+(instancetype)turingParmDictToObject:(NSDictionary *)dict;

@end

```

## 7.6 AI-WIFI 请求返回

```

//
// TuringResponse.h
// TuringDemo
//
// Created by Ezio on 2018/1/5.
// Copyright © 2018年 Ezio. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface TuringResponse : NSObject

/**

```



返回码:

2xxxx 功能码; 4xxxx 错误码

\*/

@property(nonatomic,strong) NSString \*code;

/\*\*

请求令牌

\*/

@property(nonatomic,strong) NSString \*token;

/\*\*

分段音频输出, 具体含义根据请求type定义:

当type=0时: 普通聊天音频;

当type=1时: 主动交互音频;

当type=2时: 开机等提示语音频。

\*/

@property(nonatomic,strong) NSArray \*nlp;

/\*\*

语义解析结束语音地址

\*/

@property(nonatomic,strong) NSString \*end;

/\*\*

语音识别文本结果, 根据输入字段flag控制是否输出

\*/

@property(nonatomic,strong) NSString \*asr;

/\*\*

语义解析文本结果, 根据输入字段flag控制是否输出

\*/

@property(nonatomic,strong) NSString \*tts;

/\*\*

返回类型及对应数据结构根据功能码定义, 详细参考功能码列表

\*/

@property(nonatomic,strong) NSDictionary \*func;

/\*\*

知识库中设置的动作

\*/

@property(nonatomic,strong) NSString \*action;

/\*\*

情绪ID返回数据, 具体ID对应表见 5.情绪ID对应表; 如果在知识库中设置表情, 会覆盖系统返回的表情值

\*/

@property(nonatomic,strong) NSString \*emotion;

```

/**
Res转换成Dict

@param objc TuringResponse
@return 字典
*/
+(NSDictionary *)turingResToDict:(TuringResponse *)objc;

/**
Dict转换成Res

@param dict TuringResponse
@return 字典
*/
+(TuringResponse *)turingResDicToObjc:(NSDictionary *)dict;

@end

```

## 7.7 TuringKit主要交互请求

```

//
// TuringSession.h
// TuringDemo
//
// Created by Ezio on 2018/1/8.
// Copyright © 2018年 Ezio. All rights reserved.
//

#import <Foundation/Foundation.h>

#define TURING_INIT_OK @"TURING_INIT_FINISH_OK"

@class TuringParameters;

@interface TuringSession : NSObject

@property(nonatomic, strong) NSString *text;

@property(nonatomic, strong) NSString *title;
@property(nonatomic, strong) NSString *mediaId;

```

```

@property(n nonatomic, strong) TuringParameters *t_parameters;

+ (instancetype) sharedInstance;

/**
读取uid

@return uid
*/
- (NSString *) getUserIdNormal;

/**
读取aes加密过的id

@return uid (aes)
*/
- (NSString *) getUserId;

/**
推荐列表内容相关的URL地址

@return NSURL
*/
- (NSURL *) getRecommandDataURL;

/**
发起请求

@param spData Speech Data
@param block 返回结果
*/
- (void) turingRequestActionwithSpeech: (NSData *) spData Result: (void (^)(id result, NSError *error)) block;

/**
发起TTS请求

@param spData 请求数据
@param block 返回结果
*/
- (void) turingRequestTTSwithSpeech: (NSData *) spData Result: (void (^)(id result, NSError *error)) block;

/**
请求Topic && ClientId

```

```

@param block 返回的结果
*/
-(void)getTopicAndClientId:(void(^)(id result, NSError *error)) block;

/**
切换上下曲

@param type 当type=0: 下一首, payload为歌曲相关信息
当type=1: 上一首
@param block 返回内容
*/
-(void)ExChangeSongBy:(int)type Result:(void(^)(id result, NSError *error))
block;

/**
请求播放状态
*/
-(void)requestMusicPlay:(int)type Result:(void(^)(id result, NSError *error))
block;

/**
绑定设备

@param deviceId deviceId
@param name 用户名
@param url 头像url
@param block result
*/
-(void)bindServiceWithDeviceId:(NSString *)deviceId withName:(NSString *)name
imageUrl:(NSString *)url Result:(void(^)(id result, NSError *error)) block;

/**
检查绑定状态

@param deviceId deviceId
@param block resule
*/
-(void)checkBindWithDeviceId:(NSString *)deviceId Result:(void(^)(id
result, NSError *error)) block;

/**
解除绑定

@param deviceId deviceId
@param block resule
*/
-(void)unBindWithDeviceId:(NSString *)deviceId Result:(void(^)(id
result, NSError *error)) block;

```

```
@end
```

## 7.8 TTS 语音合成返回内容

```
//
//  TuringResponse.h
//  TuringDemo
//
//  Created by Alex on 2018/3/20.
//  Copyright © 2018年 Ezio. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface TuringTTSResponse : NSObject

/**
 返回码:
 2xxxxx 功能码; 4xxxxx 错误码
 */
@property(nonaatomic,strong) NSString *code;

/**
 请求令牌
 */
@property(nonaatomic,strong) NSString *token;

/**
 返回合成音频的地址
 */
@property(nonaatomic,strong) NSArray *url;

/**
 Res转换成Dict

 @param objc TuringTTSResponse
 @return 字典
 */
+(NSDictionary *)turingResToDict:(TuringTTSResponse *)objc;

/**
 Dict转换成Res
 */
```

```
@param dict TuringTTSResponse
@return 字典
*/
+(TuringTTSResponse *)turingResDicToObjc:(NSDictionary *)dict;
@end
```

## 8. 收音模式（非AI模式）

```
/**
FM相关操作
@param cmd FM功能
@param search FM搜索
@param channel FM频道
@param frequency FM频点
@param result 返回结果
*/
+ (void)cmdFm:(JL_FCmdFM)cmd Saerch:(JL_FMSearch)search Channel:
(uint8_t)channel Frequency:(uint16_t)frequency Result:(JL_CMD_BK
__nullable)result;
```

## 9. 歌词显示

```
/**
获取LRC歌词
@param result 返回LRC数据
*/
+ (void)cmdLrcMonitorResult:(JL_LRC_BK)result
```

## 10. 设备音乐快进或者快退

```
/**
/**
快进快退
@param cmd 快进或者快退枚举
@param sec 时间
@param result 返回结果
*/
+ (void)cmdFastPlay:(JL_FCmdMusic)cmd
Second:(uint16_t)sec
Result:(JL_CMD_BK __nullable)result;
```

## 11. EQ设置

```

/**
/**
/**
设置系统EQ
@param eqMode EQ模式
@param params EQ参数(10个参数,仅适用于JL_EQModeCUSTOM情况)
*/
+ (void)cmdSetSystemEQ:(JL_EQMode)eqMode Params:(NSArray* __nullable)params;

```

## 12. OTA

### 12.1 版本校对,并获取升级文件

```

- (void)checkVersion{

#if (LT==0)

//      //有新版本
//      self->shouldUp = YES;
//      self->downloadUrl = [JL_Tools find:@"update_yx_hp_123.ufw"];
//      savePath = [JL_Tools find:@"update_yx_hp_123.ufw"];
//      dispatch_async(dispatch_get_main_queue(), ^{
//          [self->upgradeView initWithNews:@"1.0.0.0" tips:@"升级测试"];
//          [self.view addSubview:self->upgradeView];
//      });
//      [self.upgradeTable reloadData];
//      return;

JLModel_Device *model = [self.otaEntity.mCmdManager outputDeviceModel];
if (model.md5Type == YES) {
    /*----- OTA升级使用MD5校验 -----*/
    [self.otaEntity.mCmdManager cmdGetMD5_Result:^(NSArray * __Nullable
array) {
        if (array.count >= 3) {
            NSData *data_md5 = array[2];
            NSString *str_md5 = [[NSString alloc] initWithData:data_md5
encoding:NSUTF8StringEncoding];
            NSLog(@"MD5 ----> %@", str_md5);
            //NSString* test = @"eb5eaa7e89664adc2c840230fc494656";
            [self.otaEntity.mCmdManager cmdGetOtaFileKey:model.authKey
Code:model.proCode hash:str_md5

Result:^(JL_OTAUrlResult result,
        NSString * __Nullable version,
        NSString * __Nullable url,
        NSString * __Nullable explain) {
                [self updateWithOTAResult:result Version:version Url:url
Explain:explain];

```

```

        }];
    }
}];
}else{
    /*---- 传统OTA升级 ----*/
    NSString *authKey = @"";
    NSString *proCode = @"";
    if ([model.authKey isEqualToString:@""] || [model.proCode
isEqualToString:@""] ) {

        DeviceModel *m1 = [[SqliteManager sharedInstance]
checkoutDeviceModelBy:self.otaEntity.mUUID];
        authKey = m1.authKey;
        proCode = m1.proCode;
    }else{
        authKey = model.authKey;
        proCode = model.proCode;
    }
    [self.otaEntity.mCmdManager cmdGetOtaFileKey:authKey Code:proCode
        Result:^(JL_OTAUrlResult result,
                NSString * _Nullable version,
                NSString * _Nullable url,
                NSString * _Nullable explain) {
        [self updateWithOTAResult:result Version:version Url:url
        Explain:explain];
    }];
}
#else
    JLModel_Device *model = [self.otaEntity.mCmdManager outputDeviceModel];
    NSString *authKey = @"";
    NSString *proCode = @"";
    if ([model.authKey isEqualToString:@""] || [model.proCode
isEqualToString:@""] ) {

        DeviceModel *m1 = [[SqliteManager sharedInstance]
checkoutDeviceModelBy:self.otaEntity.mUUID];
        authKey = m1.authKey;
        proCode = m1.proCode;
    }else{
        authKey = model.authKey;
        proCode = model.proCode;
    }
    //有新版本
    shouldUp = YES;
    NSString *path = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) firstObject];
    path = [path stringByAppendingPathComponent:@"update.ufw"];
    savePath = path;

```



```

[upgradeView initWithNews:@"Max" tips:@"无限制升级"];
[self.view addSubview:upgradeView];

#endif

}

```

## 12.2 执行OTA升级

```

[self.otaEntity.mCmdManager cmdOTAData:data Result:^(JL_OTAResult result,
float progress) {
    if (result == JL_OTAResultSuccess) {
        [self->transportView update:1.0 Text:nil];
        self->transportView.alpha = 0.0;

        [[JLUI_Cache sharedInstance] setOtaUUID:nil];

        [weakSelf upgradeFinish];
    }
    if (result == JL_OTAResultFail) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA升级失败")];
    }
    if (result == JL_OTAResultDataIsNull) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA升级数据为空!")];
    }
    if (result == JL_OTAResultCommandFail) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA指令失败!")];
    }
    if (result == JL_OTAResultSeekFail) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA标示偏移查找失败!")];
    }
    if (result == JL_OTAResultInfoFail) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA升级固件信息错误!")];
    }
    if (result == JL_OTAResultLowPower) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA升级设备电压低!")];
    }
    if (result == JL_OTAResultEnterFail) {
        [weakSelf failedWithAction:kJL_TXT(@"未能进入OTA升级模式!")];
    }
    if (result == JL_OTAResultUnknown) {
        [weakSelf failedWithAction:kJL_TXT(@"OTA未知错误!")];
    }
    if (result == JL_OTAResultFailSameVersion) {
        [weakSelf failedWithAction:kJL_TXT(@"相同版本! ")];
    }
}

```

```

if (result == JL_OTAResultFailTWSDisconnect) {
    [weakSelf failedWithAction:kJL_TXT("TWS耳机未连接")];
}
if (result == JL_OTAResultFailNotInBin) {
    [weakSelf failedWithAction:kJL_TXT("耳机未在充电仓")];
}

if (result == JL_OTAResultPreparing ||
    result == JL_OTAResultUpgrading)
{
    if (result == JL_OTAResultUpgrading) [self->transportView
update:progress Text:kJL_TXT("正在升级")];
    if (result == JL_OTAResultPreparing) [self->transportView
update:progress Text:@"检验文件"];
    [self otaTimeCheck]; //增加超时检测
}

if (result == JL_OTAResultPrepared) {
    [self otaTimeCheck]; //增加超时检测
}
if (result == JL_OTAResultReconnect) {
    [self otaTimeCheck]; //增加超时检测

    NSLog(@"----> OTA正在回连设备... %@",self.otaEntity.mItem);
    [self->bleSDK.mBleMultiple connectEntity:self.otaEntity
Result:^(JL_EntityM_Status status) {
        if (status != JL_EntityM_StatusPaired) {
            [weakSelf failedWithAction:kJL_TXT("OTA升级超时")];
        }
    }]];
}
}];
}];

```

## 13. 区分AI模式和标准模式（非AI模式）

本Demo根据固件的SDK类型自动适配功能：AI模式和标准模式（非AI模式），两种模式采用不同的UI风格。两种模式的功能皆根据固件的配置参数适配功能

### 13.1 标准模式（非AI模式）

功能：

1. 手机音乐
2. 设备音乐
3. 外部音源
4. 收音模式
5. EQ设置
6. OTA升级（Build Phases -> Copy Bundle Resources下添加update.bfu）

## 13.2 AI模式

功能：

1. 手机音乐
2. 设备音乐
3. 外部音源
4. 智能语音
5. EQ设置
6. OTA升级 (Build Phases -> Copy Bundle Resources下添加update.bfu)

```
-(void) noteJL_Device:(NSNotification*)note{
    JLDeviceModel *model = [note object];
    if(model.sdkType == JL_SDKTypeAI){
        [JL_Tools setUser:@"AI" forKey:kJL_Product_Type];
    }
    if(model.sdkType == JL_SDKTypeST){
        [JL_Tools setUser:@"BTMate" forKey:kJL_Product_Type];
    }
}
```

## 14. 闹钟功能

### 14.1 设置/增加闹钟

```
/**
 设置/增加闹钟
@param array 闹钟模型数组
@param result 回复
*/
+(void)cmdRtcSetArray:(NSArray*)array Result:(JL_CMD_BK)result;
```

### 14.2 删除闹钟

```
/**
删除闹钟
@param array 闹钟序号数组
@param result 回复
*/
+(void)cmdRtcDeleteIndexArray:(NSArray*)array Result:(JL_CMD_BK)result;
```

### 14.3 闹钟正在响或则闹钟停止响

```
extern NSString *kJL_RTC_RINGING;      //闹钟正在响
extern NSString *kJL_RTC_RINGSTOP;    //闹钟停止响
```

## 14.4 停止闹钟响声回调

```
/**
 停止闹钟响声
@param result 回复
*/
+ (void)cmdRtcStopResult: (JL_CMD_BK)result;
```

## 15. 服务器OTA升级

### 15.1 OTA升级文件下载

```
/**
OTA升级文件下载
@param key 授权key
@param code 授权code
@param result 回复
*/
+ (void)cmdGetOtaFileKey: (NSString*)key
Code: (NSString*)code
Result: (JL_OTA_URL __nullable)result;

/**
OTA升级文件下载【MD5】
@param key 授权key
@param code 授权code
@param hash MD5值
@param result 回复
*/
+ (void)cmdGetOtaFileKey: (NSString*)key
                        Code: (NSString*)code
                        hash: (NSString*)hash
Result: (JL_OTA_URL __nullable)result;
```

### 15.2 OTA升级设备

```
/**
OTA升级设备
@param data 升级数据
@param result 升级结果
*/
+ (void)cmdOTAData: (NSData*)data
Result: (JL_OTA_RT __nullable)result;
```

### 15.3 OTA的升级状态

```
JL_OtaStatusNormal      = 0,    //正常升级
JL_OtaStatusForce       = 1,    //强制升级
```

## 15.4 OTA的返回结果

```
JL_OTAResultSuccess      = 0x00, //OTA升级成功
JL_OTAResultFail         = 0x01, //OTA升级失败
JL_OTAResultDataIsNull   = 0x02, //OTA升级数据为空
JL_OTAResultCommandFail  = 0x03, //OTA指令失败
JL_OTAResultSeekFail     = 0x04, //OTA标示偏移查找失败
JL_OTAResultInfoFail     = 0x05, //OTA升级固件信息错误
JL_OTAResultLowPower     = 0x06, //OTA升级设备电压低
JL_OTAResultEnterFail    = 0x07, //未能进入OTA升级模式
JL_OTAResultUpgrading    = 0x08, //OTA升级中
JL_OTAResultReconnect    = 0x09, //OTA需重连设备
JL_OTAResultReboot       = 0x0a, //OTA需设备重启
JL_OTAResultPreparing    = 0x0b, //OTA准备中
JL_OTAResultPrepared     = 0x0f, //OTA准备完成
JL_OTAResultFailVerification = 0xf1, //升级数据校验失败
JL_OTAResultFailCompletely = 0xf2, //升级失败
JL_OTAResultFailKey       = 0xf3, //升级数据校验失败
JL_OTAResultFailErrorFile = 0xf4, //升级文件出错
JL_OTAResultFailUboot     = 0xf5, //uboot不匹配
JL_OTAResultFailLenght   = 0xf6, //升级过程长度出错
JL_OTAResultFailFlash     = 0xf7, //升级过程flash读写失败
JL_OTAResultFailCmdTimeout = 0xf8, //升级过程指令超时
JL_OTAResultFailSameVersion = 0xf9, //相同版本
JL_OTAResultFailTWSDisconnect = 0xfa, //TWS耳机未连接
JL_OTAResultFailNotInBin  = 0xfb, //耳机未在充电仓
JL_OTAResultUnknown,      //OTA未知错误
```

## 15.5 获取MD5的数据

```
#pragma mark 获取MD5数据
+ (void)cmdGetMD5_Result:(JL_CMD_BK __nullable)result;
```

## 16. 对耳的接口

### 16.1 获取设备的图片

```

/**
获取设备的图片。
@param vid 设备vid
@param pid 设备pid
@param result 图片数据
*/
+ (void)cmdRequestDeviceImageVid:(NSString*)vid
Pid:(NSString*)pid
Result:(JL_IMAGE_RT __nullable)result;

```

## 16.2 设置EDR名字

```

//对耳相关API
/**
设置EDR名字
@param name
*/
+ (void)cmdHeatsetEdrName:(NSData*)name;

```

## 16.3 按键设置(对耳)

```

/**
按键设置(对耳)
@param key 左耳0x01 右耳0x02
@param act 单击0x01 双击0x02
@param fuc 0x00 无作用
0x01 开机
0x02 关机
0x03 上一曲
0x04 下一曲
0x05 播放/暂停
0x06 接听/挂断
0x07 拒听
0x08 拍照
*/
+ (void)cmdHeatsetKeySettingKey:(uint8_t)key
Action:(uint8_t)act
Function:(uint8_t)fuc;

```

## 16.4 LED设置(对耳)

```

/**
LED设置(对耳)
@param scene 未配对 0x01
未连接 0x02
连接 0x03

```

```

@param effect 0x00 全灭
0x01 红灯常亮
0x02 蓝灯常亮
0x03 红灯呼吸
0x04 蓝灯呼吸
0x05 红蓝交替快闪
0x06 红蓝交替慢闪
*/
+ (void)cmdHeatsetLedSettingScene:(uint8_t)scene
Effect:(uint8_t)effect;

```

## 16.5 MIC设置(耳机)

```

/**
MIC设置(耳机)
@param mode 0: 仅左耳
1: 仅右耳
2: 自动选择
*/
+ (void)cmdHeatsetMicSettingMode:(uint8_t)mode;

```

## 16.6 工作模式(耳机)

```

/**
工作模式(耳机)
@param mode 0: 普通模式
1: 游戏模式
*/
+ (void)cmdHeatsetWorkSettingMode:(uint8_t)mode;

```

## 16.7 同步时间戳(耳机)

```

/**
同步时间戳(耳机)
@param date 当前系统时间
*/
+ (void)cmdHeatsetTimeSetting:(NSDate*)date;

```

## 16.8 获取设备信息(耳机)

```

/**
获取设备信息(耳机)
@param flag BIT0 小机电量获取 格式为3个字节 参考广播包格式
BIT1 Edr 名称
BIT2 按键功能
BIT3 LED 显示状态

```

```

BIT4      MIC 模式
BIT5      工作模式

@param result 返回字典:
@"ISCHARGING_L"
@"ISCHARGING_R"
@"ISCHARGING_C"
@"POWER_L"
@"POWER_R"
@"POWER_C"
@"EDR"
@"KEY_LR"
@"KEY_ACTION"
@"KEY_FUNCTION"
@"LED_SCENE"
@"LED_EFFECT"
@"MIC_MODE"
@"WORK_MODE"
*/
+ (void)cmdHeatsetGetAdvFlag:(uint32_t)flag
Result:(JL_HEADSET_BK __nullable)result;

```

## 16.9 设备广播通知(耳机)

```

/**
设备广播通知(耳机)
@{@"JLID": 杰理ID,
@"VID": ,
@"PID": ,
@"EDR": ,
@"SCENE": ,
@"ISCHARGING_L": ,
@"ISCHARGING_R": ,
@"ISCHARGING_C": ,
@"POWER_L": ,
@"POWER_R": ,
@"POWER_C": ,
@"CHIP_TYPE": ,
@"PROTOCOL_TYPE": ,
@"SEQ": };
*/
extern NSString *kJL_HEADSET_ADV;

```

## 16.10 关闭或开启设备广播(耳机)



```
/**
 关闭或开启设备广播(耳机)
@param enable 使能位
*/
+(void)cmdHeatsetAdvEnable:(BOOL)enable;
```

## 16.11 用于ADV设置同步后需要主机操作的行为。

```
/**
 用于ADV设置同步后需要主机操作的行为。
 1: 更新配置信息, 需要重启生效。
*/
extern NSString *kJL_HEADSET_TIPS;
```

## 17. 智能充电仓

### 17.1 通知固件App的信息

```
/// 通知固件App的信息
/// @param flag 未知
+(void)cmdSetAppInfo:(uint8_t)flag;
```

### 17.2 设置通讯MTU

```
/// 设置通讯MTU
/// @param mtu app请求mtu大小
/// @param result 实际设置的Mtu大小
+(void)cmdSetMTU:(uint16_t)mtu Result:(JL_CMD_VALUE_BK)result;
```

### 17.3 开启蓝牙扫描

```
/// 开启蓝牙扫描
/// @param timeout 超时时间
/// @param result 0:成功 1:失败
+(void)cmdBTScanStartTimeout:(uint16_t)timeout Result:(JL_CMD_VALUE_BK)result;
```

### 17.4 推送蓝牙扫描结果

```
/// 推送蓝牙扫描结果
/// 返回【蓝牙数据结构】数组
/// @see JLBTModel
extern NSString *kJL_BT_LIST_RESULT;
```

## 17.5 停止蓝牙扫描（APP-->固件）

```
/// 停止蓝牙扫描（APP-->固件）
/// @param reason 0: 超时结束 1: 打断结束 2: 开启扫描失败 3: 正在扫描
/// @param result 0: 成功 1: 失败
+ (void)cmdBTScanStopReason:(uint8_t)reason Result:(JL_CMD_VALUE_BK)result;
```

## 17.6 停止蓝牙扫描（固件-->APP）

```
/// 停止蓝牙扫描（固件-->APP）
/// 0: 超时结束 1: 打断结束 2: 开启扫描失败 3: 正在扫描
extern NSString *kJL_BT_SCAN_STOP_NOTE;
```

## 17.7 通知固件连接指定的蓝牙设备

```
/// 通知固件连接指定的蓝牙设备
/// @param addr 蓝牙设备地址
/// @param result 0: 成功 1: 失败
+ (void)cmdBTConnectAddress:(NSData*)addr Result:(JL_CMD_VALUE_BK)result;
```

## 17.8 文件传输 【固件-->APP】

```
//1.监听文件数据
+ (void)cmdFileDataMonitorResult:(JL_FILE_DATA_BK)result;

//2.允许传输文件数据
+ (void)cmdAllowFileData;

//3.拒绝传输文件数据
+ (void)cmdRejectFileData;

//4.停止传输文件数据
+ (void)cmdStopFileData;
```

## 17.9 文件传输 【APP-->固件】

```
//5.请求传输文件给设备
+ (void)cmdFileDataSize:(uint8_t)size
    SavePath:(NSString*)path;

//6.推送文件数据给设备
+ (void)cmdPushFileData:(NSData*)data;
```

## 18. ID3控制

## 18.1 播放/暂停

```
+(void)cmdID3_PP;
```

## 18.2 上一曲

```
+(void)cmdID3_Before;
```

## 18.3 下一曲

```
+(void)cmdID3_Next;
```

## 18.4 开启/暂停 音乐信息推送

```
+(void)cmdID3_PushEnable:(BOOL)enable;
```

## 18.5 主动设置ID3播放状态

```
+(void)setID3_Status:(uint8_t)st;
```

# 19. 动态调节EQ段数

## 19.1 获取EQ的相关数据

```
@property (assign, nonatomic) JL_EQMode      eqMode;           //EQ模式
@property (copy, nonatomic) NSArray          *eqArray;        //EQ参数值（只适用于EQ Mode == CUSTOM情况）
@property (copy, nonatomic) NSArray          *eqCustomArray;  //自定义EQ
@property (copy, nonatomic) NSArray          *eqFrequencyArray; //EQ频率
@property (assign, nonatomic) JL_EQType      eqType;          //EQ段数类型F
(JL_EQType10<固定10段式>、JL_EQTypeMutable<动态EQ段>)
@property (strong, nonatomic) NSArray        *eqDefaultArray; //EQ的预设值数组
数组元素类型--> 【JLEQModel】
```

# 20. 设置音效的高低音

```
#pragma mark 设置高低音 [-12,+12]
+(void)cmdSetLowPitch:(int)p_low HighPitch:(int)p_high;
```

# 21. 自定义命令

## 21.1 自定义命令收发接口

```
/**
  用户自定义数据的发送接口
  @param data 数据
  @param result 回复
  */
+ (void)cmdCustomData:(NSData* __nullable)data
                  Result:(JL_CMD_BK __nullable)result;

#pragma mark ---> 设备返回的自定义数据
extern NSString *kJL_MANAGER_CUSTOM_DATA;
```

## 22. SD卡/U盘的目录浏览

### 22.1 监听目录数据

```
/**
  监听目录数据
  @param result 状态回复
  */
+ (void)cmdBrowseMonitorResult:(JL_FILE_BK __nullable)result;
```

### 22.2 浏览目录

```
浏览目录
@param model 文件Model
@param number 读取的数量
*/
+ (void)cmdBrowseModel:(JLFileModel*)model
                  Number:(uint8_t)number
                  Result:(JL_CMD_BK __nullable)result;
```

### 22.3 清除设备音乐缓存记录

```
/**
  清除设备音乐缓存记录
  @param type 卡的类型
  */
+ (void)cmdCleanCacheType:(JL_CardType)type;
```

### 22.4 快进快退

```

/**
 快进快退
@param cmd 快进或者快退枚举
@param sec 时间
@param result 返回结果
*/
+ (void)cmdFastPlay:(JL_FCmdMusic)cmd
    Second:(uint16_t)sec
    Result:(JL_CMD_BK __nullable)result;

```

## 23. Linein

### 23.1 切换到Linein模式

```

[JL_Manager cmdFunction:JL_FunctionCodeCOMMON Command:JL_FunctionCodeLINEIN
Extend:0x00 Result:nil]; //切换到Linein模式
[JL_Manager cmdGetSystemInfo:JL_FunctionCodeLINEIN Result:nil]; //获取Linein模
式下的信息

```

### 23.2 获取Linein的状态

```

JL_LineInStatus    lineInStatus;    //LineIn状态

```

### 23.3 设置Linein下的播放和暂停

```

[JL_Manager cmdFunction:JL_FunctionCodeLINEIN Command:JL_FCmdLineInPP
Extend:0 Result:nil];

```

## 24. 手机和设备互找

### 24.1 设备查找手机的通知，携带了响铃时长

```

extern NSString *kJL_BT_FIND_PHONE;

```

### 24.2 查找设备命令

```

/// 查找设备命令
/// @param isVoice 是否发声
/// @param timeout 超时时间
/// @param isIphone 是否设备查找手机（默认是手机找设备）
+ (void)cmdFindDevice:(BOOL)isVoice timeOut:(uint16_t)timeout findIphone:
(BOOL)isIphone;

```

## 25. 耳机降噪模式设置

当前接口只针对具备降噪功能的耳机起作用，属于公共接口，在 `*JL_Manager.h` 类的 `~JLDeviceModel`` 对象里面的相关字段属性为：

```
@property (assign, nonatomic) uint8_t          mAncMode;          //耳机降噪模式
@property (strong, nonatomic) NSArray          *mAncModeTypes;    //耳机主动降噪
所支持模式
```

目前分为两个接口，一个是用来设置耳机变成什么降噪模式的，一个是设置耳机可支持多少种降噪模式。

```
/// 耳机主动降噪ANC
/// @param mode 模式 (0x01:普通模式 0x02:降噪模式 0x03:通透模式)
+ (void)cmdSetAncMode:(uint8_t)mode;

/// 耳机主动降噪ANC (模式使能)
/// @param modeTypes 支持的模式
@[(JL_ANCType_Normal), (JL_ANCType_NoiseReduction).....]
/// JL_ANCType_Normal = 0, //普通模式
/// JL_ANCType_NoiseReduction = 1, //降噪模式
/// JL_ANCType_Transparent = 2, //通透模式
+ (void)cmdSetAncModeTypes:(NSArray *)modeTypes;
```

## 26. 多设备发送和接收数据

发送数据：

```
JL_RunSDK *bleSDK = [JL_RunSDK sharedMe];
[self->bleSDK.mBleEntityM.mCmdManager cmdSetSystemVolume:self->cVol
Result:^(NSArray *
_Nullable array)
{
    JL_CMDStatus state = (UInt8)[array[0] intValue];
    if (state == JL_CMDStatusFail) {
        [[DFUITools showText:kJL_TXT("设置失败") onView:self delay:1.0];
    }
}];
```

接收数据：

```
监听:[JL_Tools add:kJL_MANAGER_SYSTEM_INFO Action:@selector(noteSystemInfo:)
Own:self];

-(void)noteSystemInfo:(NSNotification*)note{
    BOOL isOK = [JL_RunSDK isCurrentDeviceCmd:note];
    if (isOK == NO) return;
```

```

        if (bleSDK.mBleEntityM.mType != 1) { //音箱
            [JL_Tools mainTask:^(
                JLModel_Device *model = [self->bleSDK.mBleEntityM.mCmdManager
outputDeviceModel];
                if(model){
                    //设备音量
                    [[JLCacheBox cacheUuid:self->bleUUID]
setP_Cvol:model.currentVol];
                    [[JLCacheBox cacheUuid:self->bleUUID]
setP_Mvol:model.maxVol];

                    self->_volSlider.value = model.currentVol;
                    self->_volLabel.text = [NSString stringWithFormat:@"%lu",
(unsigned long)model.currentVol];
                }
            }];
        }
    }
}

```

## 27. 声卡功能配置

```

extern NSString *kJL_MANAGER_KALAOK_Data;

#pragma mark ---> 设置卡拉OK 【index、value】
-(void)cmdSetKalaokIndex:(uint8_t) index Value:(uint16_t) value;

#pragma mark ---> 设置卡拉OK 【MIC EQ增益】
-(void)cmdSetKaraokeMicEQ:(NSArray*)array;

```

## 28. 手表切换表盘(OTA功能)

### 28.1 前提

- 需导入JL\_Fatfs文件内代码;
- UI界面上引用FatsObject.h的APIs即可;
- 使用文档开头部分的蓝牙连接操作;

### 28.2 蓝牙控制库的表盘相关接口

```

// Watch OTA
typedef void(^JL_FlashInfo_BK)(JLModel_Flash* __nullable model);
typedef void(^JL_FlashWrite_BK)(uint8_t flag);
typedef void(^JL_FlashWriteSize_BK)(uint8_t flag,uint32_t size);

```

```

typedef void(^JL_FlashRead_BK)(uint8_t flag,NSData *__nullable data);
typedef void(^JL_FlashAddOrDel_BK)(uint8_t flag);
typedef void(^JL_FlashWatch_BK)(uint8_t flag, uint32_t size, NSString
*__nullable path,
NSString *__nullable describe);
typedef void(^JL_FlashClean_BK)(uint8_t flag);
typedef void(^JL_FlashProtect_BK)(uint8_t flag);

// 获取外置Flash信息
/**
获取外置Flash信息 @param result 回复
*/
-(void)cmdGetFlashInfoResult:(JL_FlashInfo_BK __nullable)result;

#pragma mark ---> 写数据到Flash
/**
写数据到Flash
@param data 数据
@param offset 偏移
@param mtu 每包大小
@param result 回复
*/
-(void)cmdWriteToFlashAllData:(NSData*)data
                        Offset:(uint32_t)offset
                        Mtu:(uint16_t)mtu
                        Result:(JL_FlashWriteSize_BK __nullable)result;

// 读数据从Flash
/**
读数据从Flash
@param offset 偏移
@param size 大小
@param mtu 每包大小
@param result 回复
*/
-(void)cmdReadFromFlashAllDataOffset:(uint32_t)offset
                        Size:(uint16_t)size
                        Mtu:(uint16_t)mtu
                        Result:(JL_FlashRead_BK __nullable)result;

// [开始/结束]增加表盘(文件)
/**
开始/结束 插入文件
@param path 路径
@param size 大小
@param flag 开始:0x01 结束:0x00
@param result 回复
*/
-(void)cmdInsertFlashPath:(NSString* __nullable)path

```



```

        Size:(uint32_t)size
        Flag:(uint8_t)flag
        Result:(JL_FlashAddOrDel_BK __nullable)result;

// 设置表盘(文件)
/**
表盘操作
@param path 路径
@param flag 读取:0x00 设置:0x01
@param result 回复
*/
-(void)cmdWatchFlashPath:(NSString*__nullable)path
        Flag:(uint8_t)flag
        Result:(JL_FlashWatch_BK __nullable)result;

// 设备更新表盘(文件) 【kJL_MANAGER_WATCH_FACE】
// 返回 字符串
extern NSString *kJL_MANAGER_WATCH_FACE;

// [开始/结束]删除表盘(文件)
/**
开始/结束 删除文件
@param path 路径
@param flag 开始:0x01 结束:0x00
@param result 回复
*/
-(void)cmdDeleteFlashPath:(NSString* __nullable)path
        Flag:(uint8_t)flag
        Result:(JL_FlashAddOrDel_BK __nullable)result;

// 外挂Flash【写保护】操作
/**
开始/结束
@param flag 开始:0x01 结束:0x00
*/
-(void)cmdWriteProtectFlashFlag:(uint8_t)flag Result:
(JL_FlashProtect_BK)result;

// 断开连接, 对FATFS处理。
-(void)cmdFlashActionDisconnect;

```

## 28.3 FatsObject操作API的介绍

```

typedef void(^FatsCreateFile_BK)(float progress);

@interface FatsObject : NSObject
/// 输入命令中心类, 在JL_Entity内。
/// @param manager 命令中心类 +(void)makeCmdManager:(JL_ManagerM*)manager;

```

```

/// 设置外挂Flash的大小, FATFS的大小。
/// @param flashSize Flash大小
/// @param fatsSize FATFS的大小 +(BOOL)makeFlashSize:(uint32_t)flashSize
FatsSize:(uint32_t)fatsSize;

/// 获取Flash上的文件
/// @param path 文件名数组, 其中"JL", "FONT"不能操作。 +(NSArray*)makeListPath:
(NSString*)path;

/// 新增表盘(文件)
/// @param path "/文件名"
/// @param data 文件数据内容
/// @param result 进度 +(BOOL)makeCreateFile:(NSString*)path
Content:(NSData* __nullable)data Result:(FatsCreateFile_BK)result;

/// 删除表盘(文件)
/// @param path "/文件名" +(BOOL)makeRemoveFile:(NSString*)path;

/*-----
-----
FatSize -- FAT文件系统认为自己的大小 因为小机的FAT项按4K对齐了大小, 所以实际上会认为FAT系
统占用的空间比FLASH实际大小还要大
用 f_getfree 获取的是 FAT文件系统认为自己剩余的空间的簇个数, 其中有一部分是在Flash上实际不
存在的 所以需要减掉这部分
计算方式如下:
FatUsed = FatSize - FreeSize
FlashRemainSize = FlashSize - FatUsed
-----
----*/
/// 获取FATFS系统剩余空间, "FreeSize"。
+(uint32_t)makeGetFree;

```

## 28.4 外部操作使用流程

### STEP.1 设置命令中心类

```

//流程需要用到蓝牙的命令交互, 需传入mCmdManager。
bleSDK = [JL_RunSDK sharedMe];
[FatsObject makeCmdManager:bleSDK.mBleEntityM.mCmdManager];

```

### STEP.2 获取外挂Flash的信息

```

//调用该API需要用异步, [JL_Tools subTask:^({})]为异步代码块
[JL_Tools subTask:^(
JL_ManagerM *mCmdManager = self->bleSDK.mBleEntityM.mCmdManager;
[mCmdManager cmdGetFlashInfoResult:^(JLModel_Flash * __Nullable model) {

```

```

        [JL_Tools mainTask:^(
            //mFlashSize; //flash大小
            //mFlashType; //系统类型 0:FAT
            //mFlashMtu; //发包窗口大小
            //mFlashStatus; //系统当前状态,0x00正常, 0x01异常
            //mFlashLeftSize; //外挂Flash剩余空间
            //mFlashCluster; //扇区大小
            //mFatfsSize; //FAT系统大小
            [DFUITools showText:@"FATFS信息已更新" onView:self delay:1.0];
        });
    }];
}];
}];

```

## STEP.3 初始化本地FATFS系统

```

//调用该API需要用异步
[JL_Tools subTask:^(
    JLModel_Device *model = [self->bleSDK.mBleEntityM.mCmdManager
outputDeviceModel];
    uint32_t flashSize = model.flashInfo.mFlashSize; //外挂Flash剩余空间
    uint32_t fatsSize = model.flashInfo.mFatfsSize; //系统大小

    BOOL isOk = [FatsObject makeFlashSize:flashSize FatsSize:fatsSize];
    [JL_Tools mainTask:^(
        NSString *txt = @"FATFS Mount OK !";
        if (isOk == NO) txt = @"FATFS Mount Fail~";
        [DFUITools showText:txt onView:self delay:1.0];
    });
}];
}];

```

## 读取表盘(文件)名字

```

//调用该API需要用异步
[JL_Tools subTask:^(
    self->dataArray = [FatsObject makeListPath:@""];
    NSLog(@"Fats List ---> %@",self->dataArray);
    [JL_Tools mainTask:^(
        [self->subTableView reloadData];
    });
}];
}];

```

## 新增表盘(文件)

```

//调用该API需要用异步
JL_ManagerM *mCmdManager = bleSDK.mBleEntityM.mCmdManager;
NSData *data = [@"文件数据内容" dataUsingEncoding:NSUTF8StringEncoding];
NSLog(@"-->创建的文件大小:%lld", (long long)data.length);

```

```

NSString *path = [NSString stringWithFormat:@"%/%@", @"FILE.txt"];
[JL_Tools subTask:^(
#if IS_FROM_BLE
    /*--- 开始写文件 ---*/
    __block uint8_t mFlag_0 = 0;
    NSLog(@"--->Fats Insert stat.");
    [mCmdManager cmdInsertFlashPath:path Size:(uint32_t)data.length
                                     Flag:0x01 Result:^(uint8_t flag) { //[wSelf
fatsThreadContinue];
    }]];
    //[wSelf fatsThreadWait];
    if (mFlag_0 != 0) {
        NSLog(@"--->Fats Insert Fail."); [JL_Tools mainTask:^(
            [self setLoadingText:@"创建文件失败!" Delay:0.5]; }]);
        return; }
    BOOL isOk = [FatsObject makeCreateFile:path Content:data Result:^(float
progress) { [JL_Tools mainTask:^(
        //NSLog(@"---> Progress: %.1f",progress*100.0f);
        NSString *txt = [NSString stringWithFormat:@"正在升
级:%.1f%%",progress*100.0f]; self->progressLabel.text = txt;
        self->progressView.progress = progress;
    }]]; }]);
    NSLog(@"Fats Add ---> %d",isOk);
    [JL_Tools mainTask:^( if (isOk) {
        mFlag_0 = flag;
        [self setLoadingText:@"创建文件成功!" Delay:0.5];
        [JL_Tools subTask:^(
            /*--- 结束写文件 ---*/
            NSLog(@"--->Fats Insert end.");
            [mCmdManager cmdInsertFlashPath:nil Size:0 Flag:0x00 Result:nil];
        }]];
    }else{
        [self setLoadingText:@"创建文件失败!" Delay:0.5];
    }

    [JL_Tools delay:1.0 Task:^(
        self->progressLabel.hidden = YES; self->progressView.progress =
0.0f;

        /*--- 读取列表 ---*/
        [wSelf btn_List:nil];
    }]];
}]];
}]]

```

## 删除表盘(文件)

//调用该API需要用异步

```

NSString *path = [NSString stringWithFormat:@"%/%@", @"FILE.txt"];
JL_ManagerM *mCmdManager = bleSDK.mBleEntityM.mCmdManager;

```

```

[JL_Tools subTask:^( { #if IS_FROM_BLE
/*---- 开始删除文件 ----*/
    __block uint8_t m_flag = 0;
    NSLog(@"---->Fats Delete stat.");
    [mCmdManager cmdDeleteFlashPath:path Flag:0x01 Result:^(uint8_t flag) {
        m_flag = flag;
        //[wSelf fatsThreadContinue];
    }];
    //[wSelf fatsThreadWait];
    if (m_flag != 0) {
        NSLog(@"---->Fats Delete Fail."); [JL_Tools mainTask:^( {
            [self setLoadingText:@"删除文件失败!" Delay:0.5]; }];
        return; }
    BOOL isOk = [FatsObject makeRemoveFile:path]; NSLog(@"Fats Remove ---->
%d",isOk);
    [JL_Tools mainTask:^( { if (isOk) {
#endif
        [self setLoadingText:@"删除文件成功!" Delay:0.5];
        [JL_Tools subTask:^( {
            /*---- 结束删除文件 ----*/
            NSLog(@"---->Fats Delete end.");
            [mCmdManager cmdDeleteFlashPath:nil Flag:0x00 Result:nil];
        }];
    }else{
        [self setLoadingText:@"删除文件失败!" Delay:0.5];
    }

    /*---- 读取列表 ----*/
    [wSelf btn_List:nil]; }];
}];

```

## 用户设置当前表盘

```

//该API不需要异步
NSString *path = [NSString stringWithFormat:@"%/%@",selectText];
JL_ManagerM *mCmdManager = bleSDK.mBleEntityM.mCmdManager;
[mCmdManager cmdWatchFlashPath:path
    Flag:0x01
    Result:^(uint8_t flag, uint32_t size,
    NSString * _Nullable path,
    NSString * _Nullable describe) {
    [JL_Tools mainTask:^( {
        NSString *txt = @"设置表盘成功!";
        if (flag != 0) txt = @"设置表盘失败~";
        [DFUITools showText:txt onView:self delay:1.0];
    }];
}];

```

## 用户获取当前表盘

//该API不需要异步

```
JL_ManagerM *mCmdManager = bleSDK.mBleEntityM.mCmdManager;
[mCmdManager cmdWatchFlashPath:nil Flag:0x00
                                Result:^(uint8_t flag, uint32_t size,
                                NSString * _Nullable path, NSString *
                                _Nullable describe) {
    [JL_Tools mainTask:^(
        NSString *txt = @"获取表盘成功!";
        if (flag != 0) txt = @"获取表盘失败~";
        [DFUITools showText:txt onView:self delay:1.0];
        self->deviceText = [path stringByReplacingOccurrencesOfString:@"/"
withString:@""];
        [self->subTableView reloadData]; }]];
}];
```

## 设备主动切换的表盘

//监听通知

```
[JL_Tools add:kJL_MANAGER_WATCH_FACE Action:@selector(noteWatchFace:)
Own:self];

#pragma mark - 手表切换表盘
-(void)noteWatchFace:(NSNotification*)note{
    NSDictionary *dict = [note object];
    NSString *text = dict[kJL_MANAGER_KEY_OBJECT];
    deviceText = [text stringByReplacingOccurrencesOfString:@"/"
withString:@""];

    NSString *str = [NSString stringWithFormat:@"Watch update
face 【%@】 ",deviceText];
    [DFUITools showText:str onView:self delay:1.0];
    [subTableView reloadData];
}
```

## 获取FAT系统剩余空间

```
//调用该API需要用异步 [JL_Tools subTask:^(
    uint32_t size = [FatsObject makeGetFree];
}];
```