Jieli Bluetooth control library ios_sdk development instructions

Statement

- All references and technologies used in this project must come from well-known technical information or independent innovation design。
- This project shall not use any unauthorized third party intellectual property technical information。
- If an individual uses technical information of unauthorized third party intellectual property rights, the economic losses and legal consequences will be borne by the individual。

History

Version	Build Version	Date	Modifier	Modification of records
1.6.4	11382	2022/08/13	EzioChan	1.Add auxiliary listening headphone function
1.6.3	11349	2022/07/13	EzioChan	1.Added neck earphone compatibility 2.Adjust the document structure
1.0.0	10017	2021/12/01	LingXuanfeng	1. Initial version and set structure 2. Describes functions and interfaces

Overview

```
This document is created to facilitate the maintenance and management of
subsequent projects and record the development content。
```

Development

1.Preparation

1.1 Environment

Category	Compatibility range	Remarks
System	The above system is IOS 10.0	Support for BLE functionality
hardware requirements	Jieli BT Product	
Development Platform	Xcode	It is recommended to use the latest version of development

1.2 Premission

Need to add the following informations to the info.plist!

```
<key>NSAppleMusicUsageDescription</key>

<string>Do you allow the App's local concerts to access your music information?
</string>

<key>NSAppleMediaUsageDescription</key>

<string>Do you allow the App's local music to access your media library?
</string>

<key>NSBluetoothPeripheralUsageDescription</key>
```

```
<string>Do you allow the APP to access Bluetooth?</string>

<key>NSBluetoothAlwaysUsageDescription</key>

<string>Do you allow the APP always access Bluetooth?</string>
```

1.3 Import frameworks

The following needs to Frameworks be introduced into the project

File name	version	description
JL_BLEKit.framework	1.6.4	Jieli bluetooth manager
DFUnits.framework	lastest version	Jieli iOS utils framework
JL_RunSDK.h	lastest version	Supporting file in iOS Project

1.4 Attention

- The above system is IOS 10.0;
- Import JL_RunSDK class and Instantiate this class in a singleton;
- In SDK,Mainly used JL_Manager.h API operation SDK;

2. Base interfaces

2.1. Bluetooth connection part

2.1.1 Bluetooth initialization

```
//1, external reference
@property(strong,nonatomic) JL_BLEMultiple *mBleMultiple;
@property(weak ,nonatomic) JL_EntityM *mBleEntityM; //Weak reference is
required, disconnect the device and search again, and the SDK needs to be
released. (Used as the currently operating device)
@property(strong,nonatomic) NSString *mBleUUID;
@property(weak ,nonatomic) NSArray *mFoundArray; //Requires Weak reference,
scanned device.
@property(weak ,nonatomic) NSArray *mConnectedArray;//Requires Weak reference,
connected device.

//2. Instantiate SDK
self.mBleMultiple = [[JL_BLEMultiple alloc] init];
self.mBleMultiple.BLE_FILTER_ENABLE = YES; //Filter device enable
self.mBleMultiple.BLE_PAIR_ENABLE = YES; //pairing enable
```

```
self.mBleMultiple.BLE_TIMEOUT = 7; //Connection timeout

//3. Select the device type to be searched
self.mBleMultiple.bleDeviceTypeArr = @[@(JL_DeviceTypeWatch)];//Only select
Watch

//4. The device searched by the SDK will be added to the bleConnectedArr array
after clicking Connect.
//5. Calling [self.mBleMultiple scanStart] will release the JL_EntityM of
blePeripheralArr.
self.mFoundArray = self.mBleMultiple.blePeripheralArr;

//6. After the connected device of SDk is disconnected, it will be added to the
blePeripheralArr array.
self.mConnectedArray = self.mBleMultiple.bleConnectedArr;

//7. Use mBleEntityM to weakly reference a connected JL_EntityM device in
mConnectedArray, and then use mBleEntity to send commands to [JL_ManagerM].
```

2.1.2 Connect device

```
//Connect one from the list of discovered devices.
JL_EntityM *entity = self.mFoundArray[indexPath.row];
/**
 Connect the device
@param entity Bluetooth device class
 */
[self.mBleMultiple connectEntity:entity
                          Result:^(JL_EntityM_Status status) {
    [JL_Tools mainTask:^{
    /*[status] Error code and error reason
    JL_EntityM_StatusBleOFF = 0, //BLE Bluetooth is not turned on
    JL_EntityM_StatusConnectFail = 1, //BLE connection failed
    JL_EntityM_StatusConnecting = 2, //BLE is connecting
    JL_EntityM_StatusConnectRepeat = 3, //BLE repeat connection
    JL_EntityM_StatusConnectTimeout = 4, //BLE connection timeout
    JL_EntityM_StatusConnectRefuse = 5, //BLE is rejected
    JL_EntityM_StatusPairFail = 6, //Pairing failed
    JL_EntityM_StatusPairTimeout = 7, //Pairing timeout
    JL_EntityM_StatusPaired = 8, //paired
    JL_EntityM_StatusMasterChanging = 9, //Master-slave switching
    JL_EntityM_StatusDisconnectOk = 10, //Disconnected successfully
    JL_EntityM_StatusNull = 11, //Entity is empty */

     if (status == JL_EntityM_StatusPaired) {
        NSString *txt = [NSString stringWithFormat:@"Connection
succeeded:%@",deviceName];
     }else{
```

```
        NSString *txt = [NSString stringWithFormat:@"Connection
failed:%@",deviceName];
     }
    }];
}];

/*--- Precautions
//mBleEntityM is introduced in document 1.3.1;
//After the connection is successful, the device information must be obtained
first;
[self.mBleEntityM.mCmdManager cmdTargetFeatureResult:^(NSArray *array) {
    JL_CMDStatus st = [array[0] intValue];
    if (st == JL_CMDStatusSuccess) {
          /*--- model of device information ---*/
    JLModel_Device *model = [self.mBleEntityM.mCmdManager outputDeviceModel];
          NSLog(@"Get success.");
    }else{
          NSLog(@"Failed to get.");
    }
}];
```

2.1.3 Disconnect device

```
/**
 Connect the device
 @param entity Bluetooth device class
 */
[self.mBleMultiple disconnectEntity:entity Result:^(JL_EntityM_Status status) {
[JL_Tools mainTask:^{
    if (status == JL_EntityM_StatusDisconnectOk) {
           NSString *txt = [NSString
stringWithFormat:@"Disconnected:%@",deviceName];
    }
 }];
}];
```

2.1.4 MAC reconnect device

```
[self.mBleMultiple connectEntityForMac:@"Mac address"
Result:^(JL_EntityM_Status status) {
   [JL_Tools mainTask:^{
      if (status == JL_EntityM_StatusPaired) {
          NSLog(@"----> The connection to the device is successful.");
      }else{
          NSLog(@"----> Failed to connect to the device successfully.");
          }
    }];
  }];
```

2.1.5 UUID reconnect device

```
//Find the corresponding JL_EntityM connection according to UUID.
JL_EntityM *entity = [bleMp makeEntityWithUUID:@"UUID-xxxx-xxxx-xxxx-xxxx"];

/*--- 1. Direct UUID connection to the device ---*/
[self.mBleMultiple connectEntity:entity Result:^(JL_EntityM_Status status){
    [JL_Tools mainTask:^{
        if (status == JL_EntityM_StatusPaired) {
            NSLog(@"----> UUID connected to the device successfully.");
        }else{
            NSLog(@"----> Failed to connect to the device successfully.");
        }
    }];
}];
```

2.1.6 Initialization Bluetooth connection

```
//Receive the kJL_BLE_M_ENTITY_CONNECTED notification and do the following:
/*--- Turn off the headset information push ---*/
[self.mBleEntityM.mCmdManager.mTwsManager cmdHeadsetAdvEnable:NO];

/*--- sync timestamp ---*/
NSDate *date = [NSDate new];
JL_SystemTime *systemTime = self.mBleEntityM.mCmdManager.mSystemTime;
[systemTime cmdSetSystemTime:date];

/*--- Clear device music cache ---*/
[self.mBleEntityM.mCmdManager.mFileManager cmdCleanCacheType:JL_CardTypeUSB];
[self.mBleEntityM.mCmdManager.mFileManager cmdCleanCacheType:JL_CardTypeSD_0];
[self.mBleEntityM.mCmdManager.mFileManager cmdCleanCacheType:JL_CardTypeSD_1];

__weak typeof(self) wSelf = self;
/*--- get device information ---*/
```

```
[self.mBleEntityM.mCmdManager cmdTargetFeatureResult:^(JL_CMDStatus
status,uint8_t sn,NSData *_Nullable data){
        JL_CMDStatus st = status;
        if(st == JL_CMDStatusSuccess){
            [wSelf startTimeout];

          JLModel_Device *model = [wSelf.mBleEntityM.mCmdManager
outputDeviceModel];
         JL_OtaStatus upSt = model.otaStatus;
         if(upSt == JL_OtaStatusForce){
             wSelf.mBleEntityM.mBLE_NEED_OTA = YES;
             return;
}else{
   if(model.otaHeadset == JL_OtaHeadsetYES){
     wSelf.mBleEntityM.mBLE_NEED_OTA = YES;
     return;
}
}
wSelf.mBleEntityM.mBLE_NEED_OTA = NO;

/*--- Common information ---*/
        [wSelf.mBleEntityM.mCmdManager cmdGetSystemInfo:JL_FunctionCodeCOMMON
Result:^(JL_CMDStatus status,uint8_t sn,NSData *_Nullable data){
            [wSelf.mBleEntityM.mCmdManager cmdGetSystemInfo:JL_FunctionCodeBT
Result:^(JL_CMDStatus status,uint8_t sn,NSData *_Nullable data){

        }];
     }];
   }
}];
```

2.1.7 Bluetooth Status.

```
extern NSString *kJL_BLE_M_FOUND; //1, discover the device
extern NSString *kJL_BLE_M_FOUND_SINGLE; //2, find a single device
extern NSString *kJL_BLE_M_ENTITY_CONNECTED; //3. Device connection
extern NSString *kJL_BLE_M_ENTITY_DISCONNECTED; //4. The device is disconnected
extern NSString *kJL_BLE_M_ON; //5, BLE is turned on
extern NSString *kJL_BLE_M_OFF; //6, BLE off
extern NSString *kJL_BLE_M_EDR_CHANGE; //7, Classic Bluetooth output channel
change
//Listen to the 1st, 3rd, and 4th notifications, check the
mBleMultiple.blePeripheralArr array element changes, and update the UI
interface.
//Listen to the fifth notification, you will know the change of the current
classic Bluetooth connection, and call back the classic Bluetooth information:
            @{@"ADDRESS":@"7c9a1da7330e", //Classic Bluetooth address
            @"TYPE" :@"BluetoothA2DPOutput", //Type
            @"NAME" :@"earphone"} //name
```

3. How to use

```
// 1. Set the main mBleEntityM from all bleEntities
NSMutableArray *mConnectArr = SDK.mBleMultiple.bleConnectedArr;
JL_EntityM *model = nil;
NSString *uuid = @"your device uuid";
for (JL_EntityM *entity in mConnectArr) {
    NSString *inUnid = entity.mPeripheral.identifier.UUIDString;
    if ([uuid isEqual:inUnid] && entity.mBLE_NEED_OTA == NO &&
entity.isCMD_PREPARED== YES) {
        SDK.mBleEntityM = entity;
        SDK.mBleUUID = uuid;
        break;
    }
}
[JL_RunSDK setActiveUUID:model.uuid];

// 2. Send data:
 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){
     }
 }];

// 3. Receive data:
[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) {
        [JL_Tools mainTask:^{
            JLModel_Device *model = [self->bleSDK.mBleEntityM.mCmdManager
outputDeviceModel];
            if(model){
                [[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];
            }
        }];
    }
}
```

4. Function modules

4.1 Voice transmission part

4.1.1 Description

Recognition and transmission of speech on the device side

4.1.2 Sample code

```
-(void)speexTest {
    JL_ManagerM *manager = [[JL_ManagerM alloc] init];
    //Send the command to the speaker to allow the speaker to start receiving
voice, and the speaker will send a prompt tone after receiving this message
    [manager.mSpeexManager cmdAllowSpeak];
    //Send command to speaker, not allowed to receive voice
    [manager.mSpeexManager cmdRejectSpeak];
    //Send a command to the speaker to stop receiving data, that is, a sentence
break is detected
    [manager.mSpeexManager cmdSpeakingDone];
}
```

4.1.3 Related Interfaces

```
/**
    Voice Action Status
    kJL_MANAGER_KEY_OBJECT ==> JLModel_SPEEX
*/
extern NSString *kJL_MANAGER_SPEEX;

/**
    voice data
    kJL_MANAGER_KEY_OBJECT ==> NSData
*/
extern NSString *kJL_MANAGER_SPEEX_DATA;

/**
 Send a command to the speaker to allow the speaker to start receiving voice,
and the speaker will send a prompt tone after receiving the message
 */
-(void)cmdAllowSpeak;

/** reject recording
 Send commands to the speaker, do not allow to receive voice
 */
-(void)cmdRejectSpeak;

/** stop voice
 Send a command to the speaker to stop receiving data, that is, a sentence
segment is detected
 */
-(void)cmdSpeakingDone;
```

4.2. Get device information

4.2.1 Description

Through this function, you can obtain the relevant content of the device status information, and then you can perform other operations.

4.2.2 Sample code

```
NSLog(@"--->(3) GET Device infomation.");
//JL_ManagerM
[self.mBleEntityM.mCmdManager cmdTargetFeatureResult:^(JL_CMDStatus status,
uint8_t sn, NSData * _Nullable data) {
//TODO: After getting here, you can pass
//#pragma mark ---> Get device information
//-(JLModel_Device *)outputDeviceModel;
}];
```

4.2.3 Related Interfaces

```
#pragma mark ---> Get system information (full acquisition)
/**
 @param function JL_FunctionCode
 @param result reply
 */
-(void)cmdGetSystemInfo:(JL_FunctionCode)function
                 Result:(JL_CMD_RESPOND __nullable)result;
-(void)cmdGetSystemInfoResult;

#pragma mark ---> Get system information (optional)
/**
 @param function JL_FunctionCode
 @param result reply
 */
-(void)cmdGetSystemInfo:(JL_FunctionCode)function
           SelectionBit:(uint32_t)bits
                 Result:(JL_CMD_RESPOND __nullable)result;
-(void)cmdGetSystemInfoResult_1;

#pragma mark ---> System information actively returned by the device
extern NSString *kJL_MANAGER_SYSTEM_INFO;

#pragma mark ---> Generic, BT, Music, RTC, Aux
/**
 @param function function type
 @param cmd operation command
 @param ext extension data
 @param result reply
 */
-(void)cmdFunction:(JL_FunctionCode)function
           Command:(UInt8)cmd
            Extend:(UInt8)ext
            Result:(JL_CMD_RESPOND __nullable)result;
```

4.3 Alarm

4.3.1 Description

Synchronize mobile phone time to device, manage alarms including: read, modify, delete, listen to the alarm, etc.

4.3.2 Sample code

4.3.2.1 The real time clock model

```
@interface JLModel_RTC : NSObject
@property (assign,nonatomic) uint16_t       rtcYear;
@property (assign,nonatomic) uint8_t        rtcMonth;
@property (assign,nonatomic) uint8_t        rtcDay;
@property (assign,nonatomic) uint8_t        rtcHour;
@property (assign,nonatomic) uint8_t        rtcMin;
@property (assign,nonatomic) uint8_t        rtcSec;
@property (assign,nonatomic) BOOL           rtcEnable;
@property (assign,nonatomic) uint8_t        rtcMode;//bell type(0: default  1:
device file)
@property (assign,nonatomic) uint8_t        rtcIndex;
@property (copy  ,nonatomic) NSString       *rtcName;
@property (strong,nonatomic) RTC_RingInfo   *ringInfo;
@property (strong,nonatomic) NSData         *RingData;
@end
```

4.3.2.2 Time synchronization

```
    JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
    [entity.mCmdManager cmdSetSystemTime:[NSDate new]];
```

4.3.2.3 Get alarm clock's list

```
    JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
    [entity.mCmdManager cmdGetSystemInfo:JL_FunctionCodeRTC SelectionBit:0xF2
Result:^(NSArray * _Nullable array) {
        //TODO: do something with array
    }];
```

4.3.2.4 Add/Modify alarm clock's setting

```
JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
   JLModel_RTC *rtcmodel = [JLModel_RTC new];
        rtcmodel.rtcName = @"Alarm name";
        NSDateFormatter *formatter = [NSDateFormatter new];
        formatter.dateFormat = @"yyyy:MM:DD:HH:mm:ss";
        NSString *nowStr = [formatter stringFromDate:[NSDate new]];
        NSArray *timeArr = [nowStr componentsSeparatedByString:@":"];
        rtcmodel.rtcYear = [timeArr[0] intValue];
        rtcmodel.rtcMonth = [timeArr[1] intValue];
        rtcmodel.rtcDay = [timeArr[2] intValue];
        rtcmodel.rtcHour = [timeArr[3] intValue];
        rtcmodel.rtcMin = [timeArr[4] intValue];
        rtcmodel.rtcSec = [timeArr[5] intValue];
        rtcmodel.rtcMode = 0x00;//bell type
        rtcmodel.rtcEnable = YES;
        rtcmodel.rtcIndex = 0;//current alarm's index
        if (device.rtcDfRings.count>0) {// when device supports custom bell
            JLModel_Ring *ring = device.rtcDfRings[0];//default bell
            rtcmodel.ringInfo = [RTC_RingInfo new];
            rtcmodel.ringInfo.type = 0;//alarm type: default or custom
            rtcmodel.ringInfo.dev = 0;//bell's location
            rtcmodel.ringInfo.clust = 0;//The custom file's cluster number
            rtcmodel.ringInfo.data = [ring.name
dataUsingEncoding:NSUTF8StringEncoding];//bell name
            rtcmodel.ringInfo.len =
(uint8_t)self.rtcmodel.ringInfo.data.length;//bell file's size
        }
        // set new alarm's bell
   [entity.mCmdManager cmdRtcSetArray:@[rtcmodel] Result:^(NSArray * _Nullable
array) {
           //TODO: to do something with alarm array
   }];
```

4.3.2.5 Delete alarm clock's setting

```
JLModel_RTC *rtcmodel = ...;
//JLModel_RTC
int rtcIndex = rtcmodel.rtcIndex;
  JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
    [entity.mCmdManager cmdRtcDeleteIndexArray:@[@(rtcIndex)] Result:^(NSArray
* _Nullable array) {
        JL_CMDStatus state = (UInt8)[array[0] intValue];
        if(state == JL_CMDStatusSuccess){
            //TODO: do something...
        }
        if(state == JL_CMDStatusFail){
            //ERR: delete failed
        }
    }];
```

4.3.2.6 Get the list of default ringtones

```
 JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
 JLModel_Device *model = [entity.mCmdManager outputDeviceModel];
 NSArray *defaultRings =  model.rtcDfRings;
```

4.3.2.7 Bell's audition

```
JLModel_RTC *rtcmodel = ...;
JLModel_Ring *ring = [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
outputDeviceModel].rtcDfRings.firstObject;
rtcModel.ringInfo.type = 0;
/*
typedef NS_ENUM(UInt8, JL_CardType) {
    JL_CardTypeUSB                  = 0,    //USB
    JL_CardTypeSD_0                 = 1,    //SD_0
    JL_CardTypeSD_1                 = 2,    //SD_1
    JL_CardTypeFLASH                = 3,    //FLASH
    JL_CardTypeLineIn               = 4,    //LineIn
    JL_CardTypeFLASH2               = 5,    //FLASH2
};
*/
rtcModel.ringInfo.dev = JL_CardTypeUSB; //ring
rtcModel.ringInfo.clust = ring.index;//file's cluster
rtcModel.ringInfo.data = [ring.name
dataUsingEncoding:NSUTF8StringEncoding];//file's name
rtcModel.ringInfo.len = rtcModel.ringInfo.data.length;//file's length
        JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
```

```
        [entity.mCmdManager cmdRtcAudition:rtcModel Option:YES result:^(NSArray
* _Nullable array) {
       }];
```

4.3.2.8 Stop bell's audition

```
JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
    [entity.mCmdManager cmdRtcStopResult:^(NSArray * _Nullable array) {
    }];
```

4.3.2.9 Set the alarm parameters

```
@interface JLModel_AlarmSetting : NSObject
@property(assign,nonatomic)uint8_t index;       //Index of the alarm clock
@property(assign,nonatomic)uint8_t isCount;     //Whether the [alarm times] can
be set
@property(assign,nonatomic)uint8_t count;       //The alarm number
@property(assign,nonatomic)uint8_t isInterval;  //Whether the [time interval]
can be set
@property(assign,nonatomic)uint8_t interval;    //The time interval
@property(assign,nonatomic)uint8_t isTime;      //Whether the [time length] can
be set
@property(assign,nonatomic)uint8_t time;        //Length of time
-(NSData*)dataModel;
@end

    JL_RunSDK *bleSDK = [JL_RunSDK sharedMe] ;
    JL_ManagerM *mCmdManager = bleSDK.mBleEntityM.mCmdManager;
    JLModel_Device *model = [mCmdManager outputDeviceModel];

    // can set the alarm mode
    if (model.rtcAlarmType == YES) {
        JLModel_RTC *rtcModel = itemArray[indexPath.section];
        uint8_t bit = 0x01;
        uint8_t bit_index = bit << rtcModel.rtcIndex;

        [mCmdManager cmdRtcOperate:0x00 Index:bit_index Setting:nil
                           Result:^(NSArray<JLModel_AlarmSetting *> *
_Nullable array, uint8_t flag) {
            JLModel_AlarmSetting *setting = nil;
            if (array.count > 0) setting = array[0];

            if (setting == nil) {
                setting = [JLModel_AlarmSetting new];
                setting.index = rtcModel.rtcIndex;
```

```
                setting.isCount = 1;
                setting.count = 1;
                setting.isInterval = 1;
                setting.interval = 5;
                setting.isTime = 1;
                setting.time = 5;
            }
        }];
  }
```

4.3.2.10 Alarm clock's Notification

```
/*Alarm clock's notifications*/
extern NSString *kJL_MANAGER_RTC_RINGING;       // alarm clock is ringing
extern NSString *kJL_MANAGER_RTC_RINGSTOP;      // alarm clock stop
extern NSString *kJL_MANAGER_RTC_AUDITION;      // listen to the alarm clock
```

4.4 Music Control

4.4.1 Description

When the device is playing SD card or U disk/TF card, it is used when the mobile phone controls

4.4.2 Sample code

```
 JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
[entity.mCmdManager.mMusicControlManager cmdFastPlay:JL_FCmdMusicFastBack
                                        Second: (uint16_t)fabsf((pg * f_tott -
f_curt))
                                        Result:^(JL_CMDStatus status, uint8_t
sn, NSData * _Nullable data) {
                    //progress_sec = (int)(f_tott*pg);
                    //NSLog(@"---------------> To Progress Second:
%d",progress_sec);

                    if (sv_tott > 60*60) {
                        [JL_Tools delay:0.6 Task:^{
                            NSLog(@"----> delay get music progess 0");
                            [self getDeviceMusicProgress];
                        }];
                    }else{
                        [self getDeviceMusicProgress];
                    }
}];
```

4.4.3 Related Interfaces

```
typedef NS_ENUM(UInt8, JL_FCmdMusic) {
    JL_FCmdMusicPP = 0x01, //PP button
    JL_FCmdMusicPREV = 0x02, //Previous song
    JL_FCmdMusicNEXT = 0x03, //Next song
    JL_FCmdMusicMODE = 0x04, //Switch the playback mode
    JL_FCmdMusicEQ = 0x05, //EQ
    JL_FCmdMusicFastBack = 0x06, //Fast backward
    JL_FCmdMusicFastPlay = 0x07, //Fast forward
};


NS_ASSUME_NONNULL_BEGIN

@interface JL_MusicControlManager : JL_FunctionBaseManager

#pragma mark ---> set play mode
/**
 @param mode mode
 0x01: All loop; 0x02: Device loop; 0x03: Single loop; 0x04: Shuffle; 0x05:
Folder loop
 */
-(void)cmdSetSystemPlayMode:(JL_MusicMode)mode;

#pragma mark ---> fast forward and rewind
/**
 @param cmd fast forward or rewind enumeration
 @param sec time
 @param result return result
 */
-(void)cmdFastPlay:(JL_FCmdMusic)cmd
            Second: (uint16_t)sec
            Result:(JL_CMD_RESPOND __nullable)result;

@end
```

4.5 Equalizer settings

4.5.1 Description

Set the EQ of the device

4.5.2 Sample code

4.5.2.1 Get relevant data of EQ

```
// JLModel_Device
@property (assign,nonatomic) JL_EQMode          eqMode;          //EQ Mode
@property (copy,  nonatomic) NSArray            *eqArray;        //EQ parameter
value (EQ mode = = custom only)
@property (copy,  nonatomic) NSArray            *eqCustomArray; //Custom EQ
@property (copy,  nonatomic) NSArray            *eqFrequencyArray; //EQ
frequency
@property (assign,nonatomic) JL_EQType          eqType;           //EQ segment
type F (JL_Eqtype10 < fixed 10 segment >, JL_Eqtypemutable < dynamic EQ segment
>)
@property (strong,nonatomic) NSArray            *eqDefaultArray;  //Default
value array element type of EQ--> 【JLEQModel】

// usage
JLModel_Device *model = [[JL_RunSDK sharedMe].mBleEntityM.mCmdManager
outputDeviceModel];
...
model.eqMode;     // get EQ Mode
...
```

4.5.2.2 Config equalizer settings

```
typedef NS_ENUM(UInt8, JL_EQMode) {
    JL_EQModeNORMAL                 = 0,
    JL_EQModeROCK                   = 1,
    JL_EQModePOP                    = 2,
    JL_EQModeCLASSIC                = 3,
    JL_EQModeJAZZ                   = 4,
    JL_EQModeCOUNTRY                = 5,
    JL_EQModeCUSTOM                 = 6,
    JL_EQModeLATIN                  = 7,
    JL_EQModeDANCE                  = 8,
};
/**
@param eqMode EQ Mode
@param params EQ Params, only used in JL_EQModeCUSTOM
*/
-(void)cmdSetSystemEQ:(JL_EQMode)eqMode Params:(NSArray* __nullable)params;

// usage
JLModel_Device *model = [[JL_RunSDK sharedMe].mBleEntityM.mCmdManager
outputDeviceModel];
if (model.eqType == JL_EQType10) {
    NSArray *defaultArr =
@[@(32),@(64),@(125),@(250),@(500),@(1000),@(2000),@(4000),@(8000),@(16000)];
```

```
     [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdSetSystemEQ:JL_EQModeCUSTOM Params:defaultArr];
} else {
    [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdSetSystemEQ:JL_EQModeCUSTOM Params:model.eqFrequencyArray];
}
```

4.6 FM Control

4.6.1 Description

Control the related operations of the firmware radio module **FM related operations**

4.6.2 Sample code

```
typedef NS_ENUM(UInt8, JL_FCmdFM) {
    JL_FCmdFMPP                     = 0x01, //FM pause/play
    JL_FCmdFMPonitBefore            = 0x02, //last FM frequency
    JL_FCmdFMPonitNext              = 0x03, //next FM frequency
    JL_FCmdFMChannelBefore          = 0x04, //last FM channel
    JL_FCmdFMChannelNext            = 0x05, //next FM channel
    JL_FCmdFMSearch                 = 0x06, //search
    JL_FCmdFMChannelSelect          = 0x07, //select FM channel
    JL_FCmdFMChannelDelete          = 0x08, //delete FM channel
    JL_FCmdFMFrequencySelect        = 0x09, //select FM frequency
    JL_FCmdFMFrequencyDelete        = 0x0a, //delete FM frequency
};
typedef NS_ENUM(UInt8, JL_FMSearch) {
    JL_FMSearchALL                  = 0x00,
    JL_FMSearchForward              = 0x01,
    JL_FMSearchBackward             = 0x02,
    JL_FMSearchStop                 = 0x03,
};
/**
FM Control
@param cmd FM JL_FCmdFM
@param search JL_FMSearch, only in used when cmd == JL_FCmdFMSearch
@param channel FM's channel
@param frequency FM's frequency
@param result result's callback
*/
-(void)cmdFm:(JL_FCmdFM)cmd Saerch:(JL_FMSearch)search Channel:(uint8_t)channel
Frequency:(uint16_t)frequency Result:(JL_CMD_BK __nullable)result;
```

4.7 Linein

4.7.1 Description

change to linein model or control linein music player

4.7.2 Switch to Linein mode

```
/**
 Universal, BT, Music, RTC, Aux
 @param function
 @param cmd
 @param ext
 @param result
 */
-(void)cmdFunction:(JL_FunctionCode)function
           Command:(UInt8)cmd
            Extend:(UInt8)ext
            Result:(JL_CMD_BK __nullable)result;

// usage:
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdFunction:JL_FunctionCodeCOMMON Command:JL_FunctionCodeLINEIN Extend:0x00
Result:nil]; //Switch to Linein mode
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdGetSystemInfo:JL_FunctionCodeLINEIN Result:nil]; //Gets the information in
Linein mode
```

4.7.3 Gets the status of Linein

```
 JL_LineInStatus    lineInStatus;   //LineIn State
```

4.7.4 Set play and pause under Linein

```
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdFunction:JL_FunctionCodeLINEIN Command:JL_FCmdLineInPP Extend:0 Result:nil];
```

4.8 Voice Control

4.8.1 Description

Control device's voice

4.8.2 Get current volume

```
[[JL_RunSDK sharedMe].mBleEntityM.mCmdManager outputDeviceModel].currentVol;
```

4.8.3 Set volume

```
-(void)cmdSetSystemVolume:(UInt8)volume;
-(void)cmdSetSystemVolume:(UInt8)volume Result:(JL_CMD_BK __nullable)result;
```

4.8.4 Get the treble or bass of the sound effect

```
[[JL_RunSDK sharedMe].mBleEntityM.mCmdManager outputDeviceModel].pitchLow;
[[JL_RunSDK sharedMe].mBleEntityM.mCmdManager outputDeviceModel].pitchHigh;
```

4.8.5 Set the treble or bass of the sound effect

```
#pragma mark set the treble or bass [-12,+12]
-(void)cmdSetLowPitch:(int)p_low HighPitch:(int)p_high;
```

4.9 Tws interface

4.9.1 Description

Control earphone tws funcation status or action

4.9.2 Get pictures of the device

```
-(void)cmdRequestDeviceImageVid:(NSString*)vid Pid:(NSString*)pid Result:
(JL_IMAGE_RT __nullable)result;

// usage
    JL_RunSDK  *bleSDK = [JL_RunSDK sharedMe];
    NSString *uid = bleSDK.mBleEntityM.mVID;
    NSString *pid = bleSDK.mBleEntityM.mPID;
    if (uid.length == 0) uid = @"0000";
    if (pid.length == 0) pid = @"0000";
```

```
    NSNumber *vidNumber = [NSNumber numberWithLong:strtoul(uid.UTF8String, 0,
16)];
    NSString *vidStr = [vidNumber stringValue];

    NSNumber *pidNumber = [NSNumber numberWithLong:strtoul(pid.UTF8String, 0,
16)];
    NSString *pidStr = [pidNumber stringValue];

    NSArray *itemArr = @[@"PRODUCT_SOUND_CARD"];
    [bleSDK.mBleEntityM.mCmdManager cmdRequestDeviceImageVid:vidStr Pid:pidStr
                                                    ItemArray:itemArr
Result:^(NSMutableDictionary * _Nullable dict) {}];
```

4.9.3 Set the name's of device

```
//NSData *nameData =[@"name" dataUsingEncoding:NSUTF8StringEncoding];
-(void)cmdHeatsetEdrName:(NSData*)name;
```

4.9.4 Button setting

```
-(void)cmdHeatsetKeySettingKey:(uint8_t)key Action:(uint8_t)act Function:
(uint8_t)fuc;

// usage
@property(assign,nonatomic)int      funType; //0:click 1:click twice 2:earphone
3:mic mode 4:flashlight functions' list
@property(assign,nonatomic)int      directionType; //0:left ear 1:right ear
2:not paired 3: disconnected 4:connected

    if(self->_funType == 0 && self->_directionType == 0){
        [[JL_RunSDK sharedMe].mBleEntityM.mCmdManager
cmdHeatsetKeySettingKey:0x01 Action:0x01 Function:255];
    }
    if(self->_funType == 0 && self->_directionType == 1){
        [JL_RunSDK sharedMe].mBleEntityM.mCmdManager
cmdHeatsetKeySettingKey:0x02 Action:0x01 Function:255];
    }
    if(self->_funType == 1 && self->_directionType == 0){
        [[JL_RunSDK sharedMe].mBleEntityM.mCmdManager
cmdHeatsetKeySettingKey:0x01 Action:0x02 Function:255];
    }
    if(self->_funType == 1 && self->_directionType == 1){
```

```
        [[JL_RunSDK sharedMe].mBleEntityM.mCmdManager
cmdHeatsetKeySettingKey:0x02 Action:0x02 Function:255];
    }
```

4.9.5 Led setting

```
/**
 @param scene  0x01   not paired
               0x02   disconnected
               0x03   connected
               0x04:  play music
               0x05:  pause musica
               0x06:  external audio source playing
               0x07:  pause external audio source
 @param effect  0x00    all close
                0x01    red light on
                0x02    blue light on
                0x03    red light is flashing slowly
                0x04    blue light is flashing slowly
                0x05    red light is flashing
                0x06    blue light is flashing
 */
-(void)cmdHeatsetLedSettingScene:(uint8_t)scene Effect:(uint8_t)effect;
```

4.9.6 Microphone setting

```
/**
 @param mode 0: left ear
             1: right ear
             2: custom
 */
-(void)cmdHeatsetMicSettingMode:(uint8_t)mode;
```

4.9.7 Work mode setting

```
/**
 @param mode 1: common mode
             2: game mode
 */
-(void)cmdHeatsetWorkSettingMode:(uint8_t)mode;
```

4.9.8 Time setting

```
-(void)cmdHeatsetTimeSetting:(NSDate*)date;
```

4.9.9 Get Headset's info

```
/**
 Get Headset's info
 @param flag  BIT0   get electric quantity
              BIT1    get Edr's name
              BIT2    button's function
              BIT3    LED's status
              BIT4    MIC's mode
              BIT5    work's mode
              BIT6    product's info
              BIT7    connect time
              BIT8    The ear detection
              BIT9    language
 @param result Dictionary keys:
                @"ISCHARGING_L"
                @"ISCHARGING_R"
                @"ISCHARGING_C"
                @"POWER_L"
                @"POWER_R"
                @"POWER_C"
                @"EDR_NAME"
                @"KEY_LR"
                @"KEY_ACTION"
                @"KEY_FUNCTION"
                @"LED_SCENE"
                @"LED_EFFECT"
                @"MIC_MODE"
                @"WORK_MODE"
                @"VID"
                @"UID"
                @"PID"
                @"LINK_TIME"
                @""IN_EAR_TEST"
                @"DEVICE_LANGUAGE"
                @"KEY_ANC_MODE" : NSArray
 */
-(void)cmdHeatsetGetAdvFlag:(uint32_t)flag Result:(JL_HEADSET_BK
__nullable)result;
```

4.9.10 Dev's adv notification

```
// NSNotification’s Name
extern NSString *kJL_MANAGER_HEADSET_ADV;
```

4.9.11 Dev's adv notification switch

```
-(void)cmdHeatsetAdvEnable:(BOOL)enable;
```

4.9.12 App needs to refresh setting's info after synchronization

```
// Notification's name: app need to update UI when app receiving this
notification
extern NSString *kJL_MANAGER_HEADSET_TIPS;
```

4.9.13 Neck earphone

The function of the neck-mounted earphone is similar to that of the paired ear and TWS earphones mentioned above, but the product modeling method is inconsistent. The APP analyzes it according to the method of the TWS paired ear.

See the following properties of the **JL_EntityM** class for details:

```
/**
    JL_DeviceTypeSoundBox = 0, //AI speaker type
    JL_DeviceTypeChargingBin = 1, //Charging bin type
    JL_DeviceTypeTWS = 2, //TWS headset type
    JL_DeviceTypeHeadset = 3, //normal headphone type
    JL_DeviceTypeSoundCard = 4, //sound card type
    JL_DeviceTypeWatch = 5, //watch type
    JL_DeviceTypeTradition = -1, //traditional device type
 */
@property(assign,nonatomic) JL_DeviceType mType; //This is marked as: TWS
headset type

@property(assign,nonatomic) BOOL isCharging_L; //The status of the left ear is
displayed as the power of the entire headset
@property(assign,nonatomic) BOOL isCharging_R;//zero or no reference meaning
@property(assign,nonatomic) BOOL isCharging_C;//zero or no reference meaning
@property(assign,nonatomic) uint8_t mPower; //The left ear power is displayed
as the power of the entire headset
```

```
@property(assign,nonatomic) uint8_t mPower_L; //The left ear power is displayed as the power of the entire headset
@property(assign,nonatomic) uint8_t mPower_R; //zero or no reference meaning
@property(assign,nonatomic) uint8_t mPower_C;//zero or no reference meaning

@property(assign,nonatomic) uint8_t mProtocolType; //When it is a neck-mounted headset, the display value of this property is 0x03, when the device is a normal TWS headset, it is 0x02
```

Other properties and settings are the same as Interface to the ear

4.9.13 Auxiliary fitting

The auxiliary listening function is the content that acts on the TWS headset settings and is not supported by other devices. Implementations need to rely on the following properties of the `JL_EntityM` class: `mCmdManager` properties.

- Determine whether the current device supports the assistive listening function:

> Whether the hearing aid function is currently supported can be judged by the following properties of the `JLModel_Device` class: `isSupportDhaFitting` properties.

> `JLModel_Device` class objects are obtained through the `JL_EntityM` class's following properties: `mCmdManager` properties, `outputDeviceModel` methods.

```
    /// Whether to support auxiliary listening settings
    @property (assign,nonatomic) BOOL isSupportDhaFitting;
    ///Interaction of fitting information: version, channel number, channel frequency
    @property (strong,nonatomic) DhaFittingInfo *dhaFitInfo;
    /// Fitting interrupted/opened object, only for monitoring
    @property (strong,nonatomic) DhaFittingSwitch *dhaFitSwitch;
```

- Get the basic information of the current listening device of the device: version, number of channels, channel frequency

```
    [JLDhaFitting auxiGetInfo:^(DhaFittingInfo * _Nonnull info, NSArray<NSNumber *> * _Nonnull gains) {
        if (info) {
            if (info.ch_num != 0) {
                [self.navigationController pushViewController:vc animated:true];
            }else{
                Dialog()
                //Location
                    .wToastPositionSet(DialogToastBottom)
                    .wTypeSet(DialogTypeToast)
```

```
                    .wMessageSet(kJL_TXT("msg_read_file_err_reading"))
                // adjust the width
                    .wMainOffsetXSet(30)
                    .wStart();
            }
        }else{
            Dialog()
            //Location
                .wToastPositionSet(DialogToastBottom)
                .wTypeSet(DialogTypeToast)
                .wMessageSet(kJL_TXT("msg_read_file_err_reading"))
            // adjust the width
                .wMainOffsetXSet(30)
                .wStart();
        }
    } Manager:[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager];
```

- To perform a hearing aid fitting:

```
    dhaManager = [[JLDhaFitting alloc] init];
    JL_ManagerM *cmd = [[JL_RunSDK sharedMe] mBleEntityM].mCmdManager;
     DhaFittingData *dataInfo = [DhaFittingData new];
    dataInfo.leftOn = NO;//Whether the left ear is turned on
    dataInfo.rightOn = NO;//Whether the right ear is turned on
    dataInfo.freq = 0;//frequency
    dataInfo.gain = 0.0;//Gain
    [dhaManager auxiCheckByStep:dataInfo Manager:cmd Result:^(JL_CMDStatus
status, uint8_t sn, NSData * _Nullable data) {
        NSLog(@"Fitting: channecl:%d freq:%d
gains:%.2f",dataInfo.channel,dataInfo.freq,dataInfo.gain);

    }];
```

- Save fitting settings:

> It should be noted here that, to save the fitting settings, you need to rearrange and send the N pieces of `DhaFittingData` data previously fitted to the device.

```
NSMutableArray *newArray = [NSMutableArray new];
for (FittingMgr *item in self.results) {
    for (DhaFittingData *item2 in item.dhaList) {
        [newArray addObject:item2];
    }
}
JL_ManagerM *mgr = [[JL_RunSDK sharedMe] mBleEntityM].mCmdManager;

[fitting auxiSaveGainsList:newArray Manager:mgr Type:[self getType]
Result:^(JL_CMDStatus status, uint8_t sn, NSData * _Nullable data) {
    // save success or failure
}];
```

- To exit hearing aid fitting:

```
JL_ManagerM *cmd = [[JL_RunSDK sharedMe] mBleEntityM].mCmdManager;
[[JLDhaFitting new] auxiCloseManager:cmd Result:^(JL_CMDStatus status, uint8_t
sn, NSData * _Nullable data) {
    NSLog(@"exit fitting");
}];
```

> In the current protocol version, only the auxiliary listening switch can be set through the firmware button, so the device will automatically enter the auxiliary listening fitting state when the auxiliary listening switch is turned on. Every time the device information is requested and the reply is correct, the device will enter the auxiliary listening fitting state. Under this specific condition, it is necessary to pay attention to the operation of exiting the auxiliary hearing fitting in time.

- The monitoring device launches the auxiliary listening fitting status:

Use the KVO method to monitor the device to launch the auxiliary listening fitting state. When the device launches the auxiliary listening fitting state, the callback will be triggered.

```
    @interface FittingBasicVC(){
        JLModel_Device *devModel;
    }
    @end

    @implementation FittingBasicVC
    - (void)viewDidLoad {
        static void *dhaFitSwitchContext = &dhaFitSwitchContext;

        devModel = [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
getDeviceModel];
        [devModel addObserver:self forKeyPath:@"dhaFitSwitch"
options:NSKeyValueObservingOptionNew context:dhaFitSwitchContext];
    }
```

```
    - (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context
    {

        if (context == dhaFitSwitchContext) {
            if ([change objectForKey:@"new"]) {

                JL_ManagerM *manager = [[JL_RunSDK sharedMe]
mBleEntityM].mCmdManager;
                TwsElectricity *electricity = manager.mTwsManager.electricity;
                DhaFittingSwitch *sw = [change objectForKey:@"new"];
                if (electricity.powerLeft > 0 && electricity.powerRight>0) {
                    // binaural
                    if (sw.rightOn == NO || sw.leftOn == NO) {
                        [self goBackToRoot];
                    }
                }
                if (electricity.powerLeft == 0 && electricity.powerRight>0) {
                    // right ear
                    if (sw.rightOn == NO) {
                        [self goBackToRoot];
                    }

                }
                if (electricity.powerRight == 0 && electricity.powerLeft>0) {
                    //left ear
                    if (sw.leftOn == NO) {
                        [self goBackToRoot];
                    }
                }

            }
        } else {
            [super observeValueForKeyPath:keyPath ofObject:object change:change
context:context];
            }
        }
        -(void)viewDidDisappear:(BOOL)animated{
            [super viewDidDisappear:animated];
            [devModel removeObserver:self forKeyPath:@"dhaFitSwitch"
context:dhaFitSwitchContext];
        }
        -(void)goBackToRoot{
            [self.navigationController popToRootViewControllerAnimated:YES];
        }
        @end
```

4.10 Search device

4.10.1 Description

Control device to search device as ble center.

4.10.2 Search device

```
// search device's notification
extern NSString *kJL_MANAGER_FIND_DEVICE;
// dict = @{@"op":@(op type),@"timeout":@(timeout)};

// search device
// @param isVoice play voice
// @param timeout timeout
// @param isIphone Is device search phone? default is false;
// @param opDict option value
// opDict description:
// play mode: 0 all play
//       1 left play
//       2 right play
// player source: 0 APP
//        1 device
// etc.all play && APP play
// opDict: {@"way":@"0",@"player":@"0"}
-(void)cmdFindDevice:(BOOL)isVoice timeOut:(uint16_t)timeout findIphone:
(BOOL)isIphone Operation:( NSDictionary * _Nullable )opDict;
```

4.10.3 Search phone

```
// search phone's notification

// dict = @{@"op":@(op type),@"timeout":@(timeout)};

extern NSString *kJL_MANAGER_FIND_PHONE;
```

4.11 ID3 Control

4.11.1 Get ID3 status

```
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
outputDeviceModel].model.ID3_Status;
```

4.11.2 Play/Pause

```
-(void)cmdID3_PP;
```

4.11.3 Play previous

```
-(void)cmdID3_Before;
```

4.11.4 Play next

```
-(void)cmdID3_Next;
```

4.11.5 Start/Stop Push ID3 Music Info

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

4.11.6 Set ID3 playing status active

```
/**
*   @param st; play:0x01, pause:0x00
*/
-(void)setID3_Status:(uint8_t)st;
```

4.12 ANC settings

4.12.1 Earphone active noise reduction's support mode

```
@property (copy, nonatomic) NSMutableArray     *mAncModeArray;          //Array
of ANC patterns<JLModel_ANC *>
```

4.12.2 Headset active noise reduction's current mode

```
@property (copy, nonatomic) JLModel_ANC        *mAncModeCurrent;       //The
current ANC model
```

4.12.3 Headset ANC's setting

```
@interface JLModel_ANC : NSObject
@property(assign,nonatomic)JL_AncMode       mAncMode;               //Earphone
noise reduction mode
@property(assign,nonatomic) uint16_t        mAncMax_L;              //Maximum
left ear gain
@property(assign,nonatomic) uint16_t        mAncCurrent_L;          //Left ear
current gain
@property(assign,nonatomic) uint16_t        mAncMax_R;              //Maximum
right ear gain
@property(assign,nonatomic) uint16_t        mAncCurrent_R;          //Right ear
current gain
-(NSData*)dataModel;

#pragma mark ---> Earphone active noise abatement ANC setting
-(void)cmdSetANC:(JLModel_ANC*)model;
```

4.12.4 Headset noise reduction mode setting

```
 The current interface only works for headphones with noise reduction function
and belongs to the public interface, The related field properties in the
'JLDeviceModel' object of the * JL_Manager.h * class are:


 @property (assign,nonatomic) uint8_t            mAncMode;        //Earphone
noise reduction mode
 @property (strong,nonatomic) NSArray            *mAncModeTypes;  //The headset
active noise reduction mode is supported
```

At present, it is divided into two interfaces, one is used to set what noise reduction mode the headset becomes, the other is to set how many kinds of noise reduction mode the headset can support.

```
 /// Headphones actively reduce NOISE ANC
 /// @param mode (0x01:Normal mode 0x02:The noise reduction mode 0x03:Bridge
Mode)
 -(void)cmdSetAncMode:(uint8_t)mode;

 /// Earphone active noise Reduction ANC (mode enabled)
 /// @param modeTypes Support Mode
@[@(JL_ANCType_Normal),@(JL_ANCType_NoiseReduction).....]
 /// JL_ANCType_Normal                = 0,    //Normal mode
 /// JL_ANCType_NoiseReduction        = 1,    //The noise reduction mode
 /// JL_ANCType_Transparent           = 2,    //Bridge Mode
 -(void)cmdSetAncModeTypes:(NSArray *)modeTypes;
```

4.13 Sound Card control

```
extern NSString *kJL_MANAGER_KALAOK_Data;

#pragma mark ---> Set Sound card【index、value】
///@param index 0:reverberation; 1:delayed; 2:volumn
///@param value 1-100
-(void)cmdSetKalaokIndex:(uint8_t) index Value:(uint16_t)value;

#pragma mark ---> Set Sound card【MIC EQ】
-(void)cmdSetKaraokeMicEQ:(NSArray*)array;

// usage:
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdSetKaraokeMicEQ:@[@(32),@(64),@(125),@(250),@(500),@(1000),@(2000),@(4000),@
(8000),@(16000)]];
```

5. Files browsing

5.1 Description

Brows device file APIs

5.2 Listen for directory data

```
/**
 Listen for directory data
 @param Result
 */
-(void)cmdBrowseMonitorResult:(JL_FILE_BK __nullable)result;
```

5.3 Browse the directory

```
 Browse the directory
 @param model FileModel
 @param number Read count
 */
-(void)cmdBrowseModel:(JLFileModel*)model Number:(uint8_t)number Result:
(JL_CMD_BK __nullable)result;
```

5.4 Clear device music cache records

```
/**
 Clear device music cache records
 @param type CardType
 */
-(void)cmdCleanCacheType:(JL_CardType)type;
```

5.5 How to use

```
// add files' callback, get the file request callback after request
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdBrowseMonitorResult:^(NSArray<JLModel_File *> * _Nullable array,
JL_BrowseReason reason) {
    switch (reason) {
        case JL_BrowseReasonReading:{
        }break;
        case JL_BrowseReasonCommandEnd:{
        }break;
        case JL_BrowseReasonFolderEnd:{
        }break;
        case JL_BrowseReasonBusy:{
        }break;
        case JL_BrowseReasonDataFail:{
        }break;
        case JL_BrowseReasonPlaySuccess:{
        }break;
```

```
        case JL_BrowseReasonUnknown:{
        }
        default:
            break;
    }
}];

// request file folders or play music
    JL_EntityM *entity = [[JL_RunSDK sharedMe] mBleEntityM];
    JLModel_Device *model = [entity.mCmdManager outputDeviceModel];
    JLModel_File *fileModel = [JLModel_File new];
    fileModel.fileType   = JL_BrowseTypeFolder;
    NSMutableArray *mutbA = [NSMutableArray new];
    NSMutableDictionary *sDict = saveDict[bleUuid];
    switch (nowType) {
        case JL_CardTypeUSB:{
            fileModel.cardType   = JL_CardTypeUSB;
            fileModel.fileName   = @"USB";
            fileModel.folderName = @"USB";
            fileModel.fileHandle = model.handleUSB;
            fileModel.fileClus   = 0;
            [mutbA addObject:fileModel];
            [sDict setValue:mutbA forKey:USB_Card];
        }break;
        case JL_CardTypeSD_0:{
            fileModel.cardType   = JL_CardTypeSD_0;
            fileModel.fileName   = @"SD Card";
            fileModel.folderName = @"SD Card";
            fileModel.fileHandle = model.handleSD_0;
            fileModel.fileClus   = 0;
            [mutbA addObject:fileModel];
            [sDict setValue:mutbA forKey:SD_0_Card];
        }break;
        case JL_CardTypeSD_1:{
            fileModel.cardType   = JL_CardTypeSD_1;
            fileModel.fileName   = @"SD Card 2";
            fileModel.folderName = @"SD Card 2";
            fileModel.fileHandle = model.handleSD_1;
            fileModel.fileClus   = 0;
            [mutbA addObject:fileModel];
            [sDict setValue:mutbA forKey:SD_1_Card];
        }break;
        case JL_CardTypeFLASH:{
            fileModel.cardType   = JL_CardTypeFLASH;
            fileModel.fileName   = @"FLASH";
            fileModel.folderName = @"FLASH";
            fileModel.fileHandle = model.handleFlash;
            fileModel.fileClus   = 0;
            [mutbA addObject:fileModel];
```

```
            [sDict setValue:mutbA forKey:FLASH_Card];
        }break;
        default:
            break;
    }
    if (fileModel.fileType == JL_BrowseTypeFile) {
        // play music
        [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdBrowseModel:fileModel Number:1 Result:nil];
    } else {
        // request files
        [[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager
cmdBrowseModel:fileModel Number:10 Result:nil];
    }
```

5.6 Lyrics display

```
/**
Get LRC lyrics
@param result Return LRC data
*/
-(void)cmdLrcMonitorResult:(JL_LRC_BK)result
```

6. Custom command

6.1 Description

Custom data can be sent and received through this interface.

6.2 User defined command sending and receiving interface

```
/**
 User defined data sending interface
 @param data
 @param result
 */
-(void)cmdCustomData:(NSData* __nullable)data
              Result:(JL_CMD_BK __nullable)result;

// Notification's name: app can receives custom data from this notification
extern NSString *kJL_MANAGER_CUSTOM_DATA;
```

6.3 Custom command examples

```
//Send custom commands
uint8_t buf[2] = {0x00,0x08};
NSData *cmd_buf = [NSData dataWithBytes:buf length:2];
[[[JL_RunSDK sharedMe] mBleEntityM].mCmdManager cmdCustomData:cmd_buf
Result:^(NSArray * _Nullable array) {
    JL_CMDStatus state = (UInt8)[array[0] intValue];
    NSLog(@"Send custom command 1:%d",state);

    if(state == JL_CMDStatusSuccess){
        NSLog(@"Send custom command 2:%d",state);
    }
    if(state == JL_CMDStatusFail){
        NSLog(@"Send custom command 3:%d",state);
    }
}];

//Receive custom commands
1.In JL_RunSDK.m,Register [JL_Manager setManagerDelegate:self];
2.Monitor onManagerCommandCustomData Method, Just process the received NSData。
The details are as follows:
  - (void)onManagerCommandCustomData:(nonnull NSData *)data {
      NSLog(@"---> Receive custom data");
  }
```