# SIG Mesh iOS APP介绍

目录

SIG Mesh iOS APP介绍 1

目录 1

一、 使用介绍 2

二、 开发代码介绍 4

a、 运行SIGMeshOCDemo步骤 4

b、 集成SDK步骤 4

c、 通信机制 4

d、 Bluetooth通信过程 5

e、 Provision过程(加灯过程) 5

f、 指令模块 6

g、 OTA过程 7

h、 附 8

i、 修订记录 9

1. 使用介绍

打开app后，点击主界面右上角的“+”按钮进入添加界面，APP会自动添加可入网设备，添加完成后返回主界面如图一。点击灯可进行开关控制设备控制，长按灯图标，进入图2可对设备进行开关、亮度、色温、HSL等的控制，图3可对设备进行分组编辑，图4可对设备进行剔除网络、设置闹钟、设置publish model、单灯OTA等操作。

添加设备后的主界面 设备长按之后，效果如下图

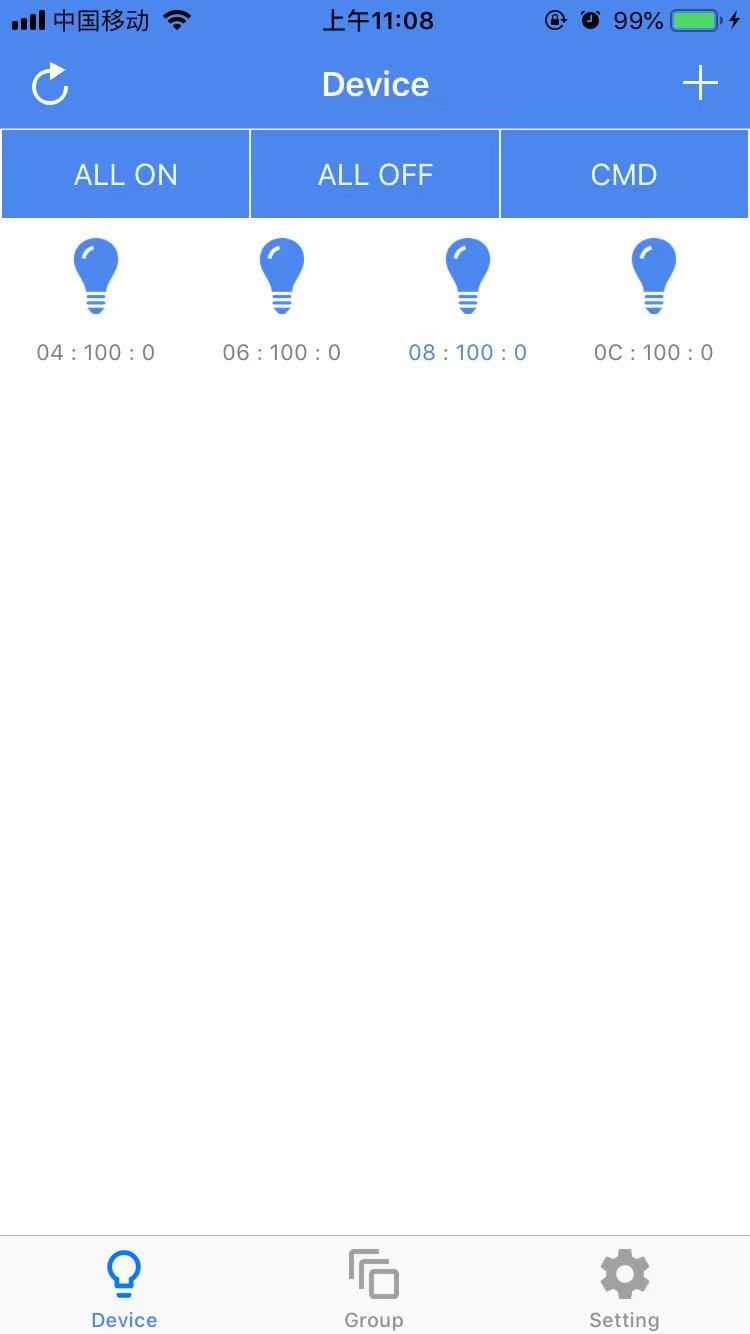
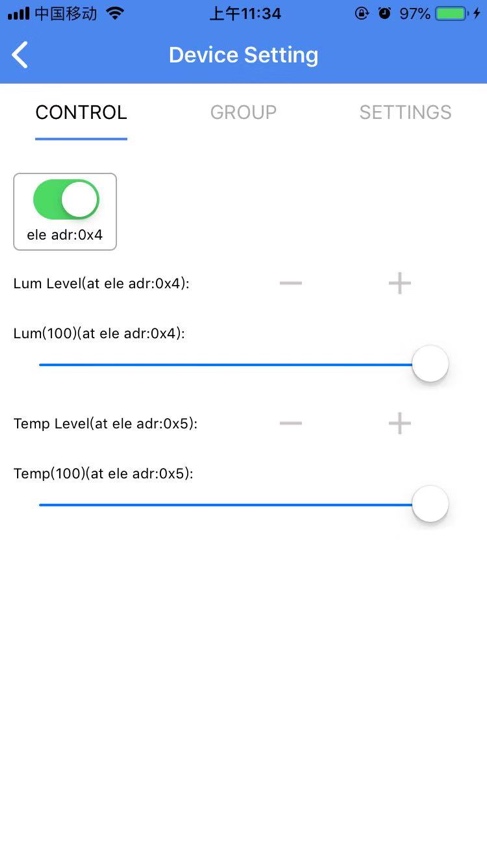
 

图-1 图-2

分组配置 单灯设置

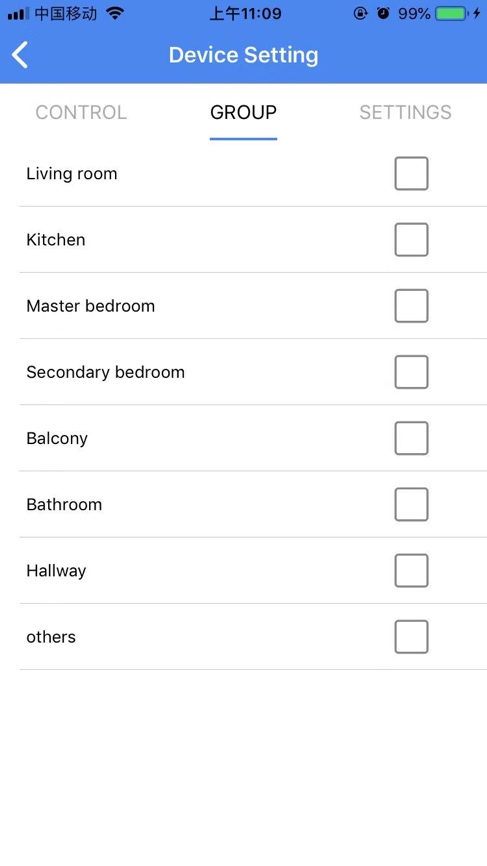
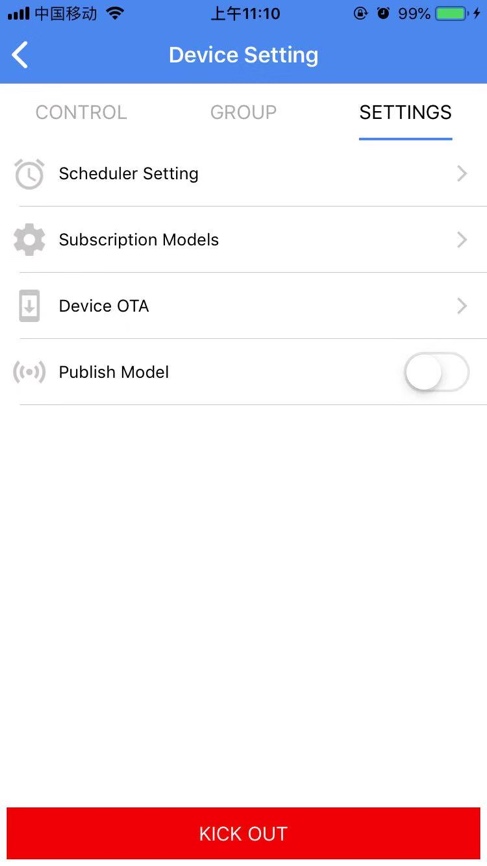
 

图-3 图-4

OTA升级界面 分组控制

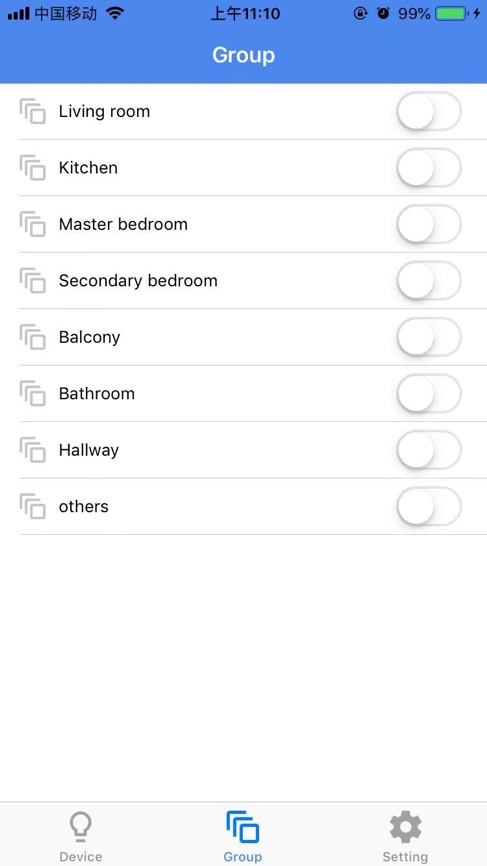
 

图-5 图-6

1. 开发代码介绍

须知

语言环境：Object-C

编译器环境：Xcode 9.0及以上

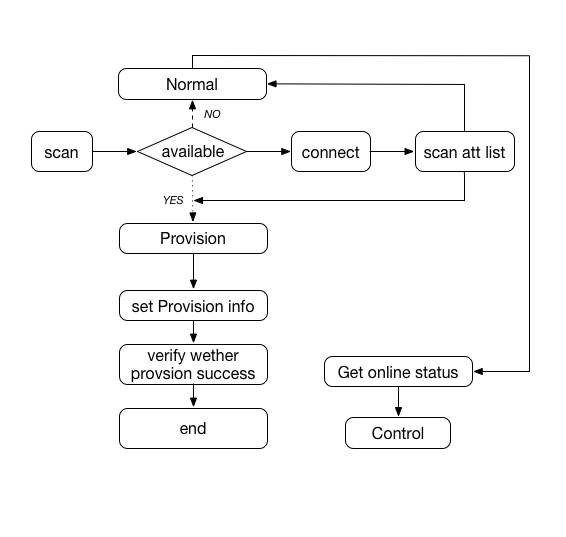
涉及框架：CoreBluetooth

开放的API文件：SDKLibCommand.h

1. 运行SIGMeshOCDemo步骤
2. 到Telink下载到SDK源代码TelinkSigMesh\_iOS\_Source\_V2.7.0\_0131，解压，命令行cd进入TelinkSigMesh\_iOS\_Source\_V2.7.0\_0131/release/TelinkSigMesh\_iOS\_Source/SigMeshOC这个目录，执行pod update。
3. 打开demo工程文件SigMeshOCDemo.xcworkspace即可运行。
4. 集成SDK步骤
5. 把SDK代码文件夹TelinkSigMesh\_iOS\_Source\_V2.7.0\_0131/SigMeshOC/ SigMeshOC物理拷贝到开发者项目文件夹中，打开开发者工程，并把SDK工程SIGMeshOC.xcodeproj拖到xcode打开的开发者工程中，再链接library静态库(项目->targets->SigMeshOCDemo->Build phases->Link Binary With Libraries->点击+->添加libSigMeshOC.a)。
6. SDK工程SIGMeshOC.xcodeproj使用到了第三方库YYModel，需要开发者在自己的项目中集成pod并安装YYModel。
7. 打开项目--->Targets--->Build Settings--->Preprocessor Macros，添加两个全局预编译宏WIN32=1 、IOS\_APP\_ENABLE=1，不添加的话，整个项目会编译不通过。
8. 打开项目--->Targets--->Build Settings--->Header Search Paths，添加头文件搜索路径(即Bluetooth.h所在的文件夹的路径)。
9. 参考demo，在开发者工程的AppDelegate.m中#import <YYModel/YYModel.h>，并调用宏START\_MESH\_SDK初始化SDK。
10. 导入相关的头文件，如#import “SDKLibCommand.h”，即可调用该头文件的API接口。
11. 注意事项
12. 开发者在自己的项目中添加其它第三方库是需要注意：泰凌提供的SigMeshOC里面只导入了部分.h头文件，所以可以在自己工程中添加编译选项“-all\_load”时会出现链接错误。解决办法：将编译选项“-all\_load”修改为“-force\_load”加其它第三方库静态库的路径的方式。 异常实例：集成了SigMeshOC的项目，再集成微信的SDK时，微信需要开发者添加编译选项“-all\_load”，就会出现连接错误。
13. 当前SigMeshOC集成了第三方库YYModel用于数据模型与JSON的快速转换。在AppDelegate调用宏START\_MESH\_SDK进行SEL导入。开发者可以通过pod的方式集成YYModel，如果开发者需要集成YYKit，则需要修改宏START\_MESH\_SDK的实现，因为YYModel与YYKit之间的方法名存在一些差异。
14. 通信机制

基于CoreBluetooth框架下，单一BLE连接的网络结构，各设备间的通信基于SIG标准协议。

通信过程，如图



1. Bluetooth通信过程

Bluetooth通信过程，开启了一个runloop ，整个APP运行过程中，与main loop并行存在，这是lib需要， Bluetooth(包括父类) 提供了很多API，常用的API有function：

- (void)startWorkNormal:(NSArray <NSString \*>\*)availableList canControl:(workWithPeripheralCallBack)canControl ---function 001

- (void)startAddDeviceWithNextAddress:(UInt16)address networkKey:(NSData \*)networkKey netkeyIndex:(UInt16)netkeyIndex unicastAddress:(UInt16)unicastAddress uuid:(NSData \*)uuid keyBindType:(KeyBindTpye)type isAutoAddNextDevice:(BOOL)isAuto provisionSuccess:(addDevice\_prvisionSuccessCallBack)provisionSuccess keyBindSuccess:(addDevice\_keyBindSuccessCallBack)keyBindSuccess fail:(prvisionFailCallBack)fail finish:(prvisionFinishCallBack)finish ---function 002

1. Provision过程(加灯过程)

被添加到某个网络的设备和没有被添加过的设备，advertisementData内容中的CBAdvertisementDataServiceUUIDsKey对应的values是不一样的，被添加过的id为“1828”，没有添加过的id为“1827” ，没有被添加过的可以被加到新的网络当中，调用function 002即可；

调用function 002后，在call back中处理单个添加成功、单个添加失败、以及完成加灯过程；调用上面API后，是一个自动连续Provision过程，如果搜索到可添加设备，则进行Provision；当检测到可添加到新的网络时，会设置一个network key(不大于16个字节的key)，Demo example中 class DataSource中有提供属性metworkKey获取到。

**由于SIG网络中，把手机也当做一个节点，因此，如果没有给本地手机配过网络(后续称本地网络)，则需要先配置本地网络(如果配置过则可省去此步骤)，此时则会用到之前设置的network key，涉及SDK function：**

int provisionLocal(NSData \*netkeydata) ---function 003

**由于配置本地网络是一个赋值过程，没有回调，配置完后，可进行外围设备的配置动作，配置外围设备时，除了传network key之外，还需要一个地址，这个地址用来唯一标识整个网络的设备，后续控制指令都是通过该地址进行，因此地址管理要保证不能出现同样的；**

**其中，地址的取值范围1~0x7eff，其中0x0001作为本地网络地址，0x7f00以后的地址保留，暂时不确定用途。**

当添加完成后(超时时间不再搜到可添加至网络的设备)，可调用function 001,该function用于一般通信过程，详细使用，然后在call back中，你可以获取在线的设备，获取完成后，可对设备进行控制，详情见Demo example中的处理；

1. 指令模块

指令内容，见SDK中class SDKLibCommand，其中用block属性保存传进来的block声明，有的指令有response，response会在SDK class Bluetooth中 function：

- (void)responseBack:(ResponseModel \*)m ---function 004

进行处理，SDKLibCommand中保存的block都会在这儿call back，并携带参数ResponseModel。

由于demo中，要呈现log内容，因此对控制command添加了一层，Demo example调用的是class DemoCommand中的function。

由于SDK lib里面有对指令进行retry处理，当一个指令response还未回来时，是不会响应其他指令，因此在SDKLibCommand中添加了一个“判忙机制”，采用了一个定时器busyTimer等待response，当定时器被销毁后才可让发其他指令，否则直接return掉；其中，采用Notify通知形式告知外界Notify(NotificationName: NotifyCommandIsBusyOrNot)，同时回传参数userInfo:@{CommandIsBusyKey : @BOOL}；

如开关指令对应的function：

- (void)switchOnOffWithExecuteCommand:(BOOL)executeCommand on:(BOOL)on address:(UInt16)address resMax:(int)resMax Completation:(responseModelCallBack)complete ---function 005

Demo example中是在这个的基础上再次封装(便于看log)，因此，Demo example中调用的是class DemoCommand的function：

+ (void)switchOnOff:(BOOL)executeCommand on:(BOOL)on address:(UInt16)address resMax:(int)resMax complete:(responseModelCallBack)complete ---function 006

如用于获取当前网络在线设备的指令：

- (void)getOnlineStatus:(BOOL)executeCommand reqCount:(int)reqCount complete:(responseModelCallBack)complete ---function 007

SDK中SDKLibCommand还提供有其他的指令，具体介绍，请见代码注释。

1. OTA过程

SDK中class OTAManager封装了OTA过程，外面通过单例调用function 008即可，call back中获取OTA结果（完成写数据、写数据中断），使用example见Demo中class SingleOTAViewController的function：

- (IBAction)clickStartOTA:(UIButton \*)sender，该方法调用了OTAManager中：

- (BOOL)startOTAWithOtaData:(NSData \*)otaData models:(NSArray <BTDevice \*>\*)models singleSuccessAction:(singleDeviceCallBack)singleSuccessAction singleFailAction:(singleDeviceCallBack)singleFileAction singleProgressAction:(singleProgressCallBack)singleProgressAction finishAction:(finishCallBack)finishAction ---function 008

由于上述function中models只测试了单个灯的情况，OTA是否成功的判断标准是，OTA数据包发送完成后设备重启并能够再次搜索到该设备;

1. 私有定制获取设备状态模式

SDK从v2.8.1版本开始添加了getOnlinestatus功能，原理是新增自定义的CBService和UUID为@"00010203-0405-0607-0809-0A0B0C0D1A11"的CBCharacteristic。 返回的状态数据解析如下：

OnlineStatus解密前：<6206dce6 91a96d89 6032b8c3 00859c29 f4e3>

OnlineStatus解密后：<6206dce6 0e004600 64ff0000 00000000 f4e3>

1、解析说明：第1字节0x62为控制码，第2字节0x06为一个设备的状态数据的长度，第3~4字节为snunber(APP可忽略)，第5~10字节(共6字节数据)为第一个node的状态数据，第11~16字节(共6字节数据)为第二个node的状态数据，第17~18字节为CRC校验码。

2、6字节数据组成说明：第1、2字节为短地址address；第3字节为在线状态，0为离线，非0为在线；第4个字节为亮度，范围0-100，其中0表示关灯状态，非0表示开灯状态；第5字节为色温，范围0-100；第六字节为保留数据。

3、特别说明：

1.解析到的短地址address为0时，表示该6个字节的数据为无效的状态数据。

2.APP需要根据第2字节获取到的长度进行字符串截取，因为客户可以修改该长度和自定义数据拼接，提高兼容性。

1. remote添加模式

SDK从v2.8.1版本开始添加了remote add功能，原理是通过直连灯上报特定短地址扫描到的unprovision beacon数据包，通过广播包的方式对非直连的设备进行添加操作。

RemoteAddVC.m代码逻辑说明如下：

1.需要一个当前mesh的直连设备，将直连设备的短地址添加到需要扫描UUID的短地址列表scanAddressList中。

2.获取直连设备的单次扫描周期的最大可上报UUID的个数。

3.调用扫描UUID接口，传入最大可上报UUID的个数和超时上报时间，扫描地址为scanAddressList的第一个地址。如果scanAddressList的个数为0则remote添加完成了。

4.使用扫描到的UUID列表返回了，选中其中一个UUID，调用setRemoteAddDeviceInfoWithUnicastAddress进行该UUID的初始化和linker open。如果超时时间+2s都没有UUID返回则删除当前的短地址，转跳步骤3。

5.原provision流程，等待provision\_end\_callback。

6.provision\_end\_callback返回成功，无需断开蓝牙连接，直接调用keyBind接口，等待App\_key\_bind\_end\_callback。

7.App\_key\_bind\_end\_callback返回成功则当前UUID的添加已经完成了。UUID列表还存在UUID则转跳步骤4，UUID列表不存在UUID则转跳步骤3。

1. 自定义ini指令组包结构说明

demo中的CMD界面为自定义的ini指令发送界面，当前APP提供的控制API、配置API等接口在lib内部都是通过组建ini指令来实现的。Lib的发包结构体为mesh\_bulk\_cmd\_par\_t，代码路径：**app\_mesh.h/line.384。**代码如下：

typedef struct{

u16 nk\_idx;

u16 ak\_idx;

u8 retry\_cnt; // only for reliable command

u8 rsp\_max; // only for reliable command

u16 adr\_dst;

u8 op;

u8 par[MESH\_CMD\_ACCESS\_LEN\_MAX];

}**mesh\_bulk\_cmd\_par\_t**;

typedef struct{

u8 op;

u16 vendor\_id;

u8 op\_rsp;

u8 tid\_pos;

u8 par[MESH\_CMD\_ACCESS\_LEN\_MAX];

}**mesh\_vendor\_par\_ini\_t**;

例如cmd界面的vendor\_onoff\_get可以为以下两条指令：

**指令1：**a3 ff 00 00 00 00 00 00 ff ff e2 11 02 c4 00

**指令2：**a3 ff 00 00 00 00 02 00 ff ff e2 11 02 00 01 01 00

上面的ini指令1中，a3 ff为固定开头，0000为nk\_idx，0000为ak\_idx，0为retry\_cnt，0为rsp\_max，ffff为adr\_dst，e2为op，1102为vendor\_id，c4为op\_rsp，00为tid\_pos。

上面的ini指令2中，a3 ff为固定开头，0000为nk\_idx，0000为ak\_idx，02为retry\_cnt，0为rsp\_max，ffff为adr\_dst，e2为op，1102为vendor\_id，00为op\_rsp，01为tid\_pos。

**注意：**op\_rsp不为00时，即APP需要等待该op\_rsp的返回才将isBusy置为NO，否则，op\_rsp的未返回时就发送下一个指令，app就打印tx cm busy，实际不会发出指令。

参考文档：更加详细的ini说明文档请参考《SIG\_MESH\_INI格式补充说明\_20190529.docx》。

1. 附
2. 修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Content | Date | Author |
| 1.0.0 | SIG Mesh使用介绍，以及开发介绍 | 2017/11/24 | Qinglu.shi |
| 2.6.0 | SIG Mesh项目的demo代码与库代码分离 | 2018/9/5 | Jiazhi.Liang |
| 2.7.A | mesh库与YYModel分离，优化添加接口 | 2019/3/13 | Jiazhi.Liang |