文档版本	V1.3
发布日期	2021/8/25

Genie Mesh SDK 开发手册

天猫精灵

目录

1 前言 1
1.1 文档目的 1
1.2 文档范围 1
1.3 预期读者 1
1.4 版本修订记录 1
2 开发板介绍 2
3 搭建开发环境 2
3.1 软件开发环境 2
3.2 CKLink Lite 调试器 2
3.3 获取 SDK 2
4 灯应用示例 4
4.1 应用示例概况 4
4.2 编译应用示例 5
4.3 例程烧录 6
4.4 宏定义说明 6
4.5 串口命令 7
4.6 创建与调试产品 8
5 通用节点应用示例 8
5.1 应用示例概况 8
5.2 编译固件 10
5.3 例程烧录 11
5.4 宏定义说明 11
5.5 创建与调试产品 13
5.6 产品应用 13
5.7 调试串口 14
5.8 数据串口配置示例 15
5.9 用户自定义串口协议示例 16
6 三键开关应用示例 16
6.1 应用示例概况 16
6.2 编译固件 18
6.3 例程烧录 18
6.4 宏定义说明 18
7 Genie Service 组件接口说明 19
7.1 宏定义说明 19
7.2 初始化 20
7.3 发送数据 21
7.4 接收数据 23
7.5 透传固件收发数据示例 24
7.6 收发数据 TID 说明 29
7.7 SDK Event 说明 30
8 其他参考文档 33
9 AT 指令定义 33

9.1 串口设置与指令格式 33
9.2 AT 指令集说明 34
10 二进制串口协议定义 39
10.1 串口通信协议 39
10.2 控制指令 39
10.3 透传数据示例 40
10.4 控制指令示例 41
10.5 模组回复错误码格式与定义 42

1 前言

1.1 文档目的

本文档介绍如何基于天猫精灵 Genie Mesh SDK 开发应用。

1.2 文档范围

本文档适用于天猫精灵 TG7120B 与 TG7121B 芯片项目。

1.3 预期读者

本文档适用的文档使用对象为使用天猫精灵 TG7120B 与 TG7121B 芯片的产品经理、软件开发人员。

1.4 版本修订记录

表格 1-1 版本修订记录

版本编号	修订日期	修订说明	修改人/日期	审批人/日期
V1.0	2021/5/31	初版	渊峙、远情	
V1.1	2021/6/15	增补灯应用示例部分	渊峙	
V1.2	2021/8/23	增加多节点网络性能宏定义	竹萌	
V1.3	2021/8/25	增加固件中芯片封装相关配置与开发自有品牌项目 Mesh 产品说明	渊峙	

2 开发板介绍

请访问芯片开放社区TG7120B的芯片主页参考TG7120B开发板说明文档。
请访问芯片开放社区TG7121B的芯片主页参考TG7121B开发板说明文档。

3 搭建开发环境

3.1 软件开发环境

首先安装 YoC 开发环境。
TG7120B 和 TG7121B 可以使用 Linux 开发。
安装和使用 Linux 开发环境请参考：

- 环境安装
- YoCTools 使用说明

TG7120B 可以支持在 Windows 下使用 CDK 开发（TG7121B 不支持）

- 使用 CDK 开发快速上手

3.2 CKLink Lite 调试器

TG7120B 芯片可以使用 CDK + CKLink Lite 调试器来调试。

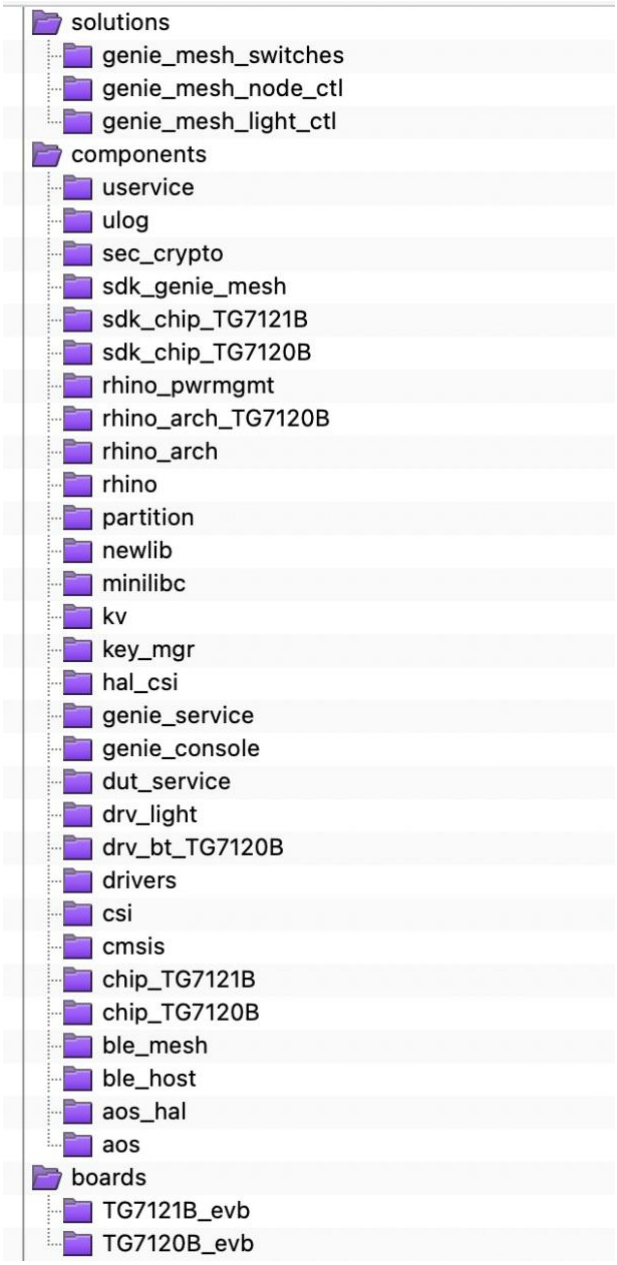
- CKLink Lite 调试器购买链接
- CKLink 设备使用指南

3.3 获取 SDK

Genie Mesh SDK：https://gitee.com/yocop/sdk_genie_mesh.git
SDK 入口是 sdk_genie_mesh 组件。可以通过 yoc 安装 sdk_genie_mesh 来下载整个 SDK。

```
yoc init
yoc install sdk_genie_mesh
```

会自动安装天猫精灵 3 个应用 solution 以及 3 个应用关联的 components 与 boards。



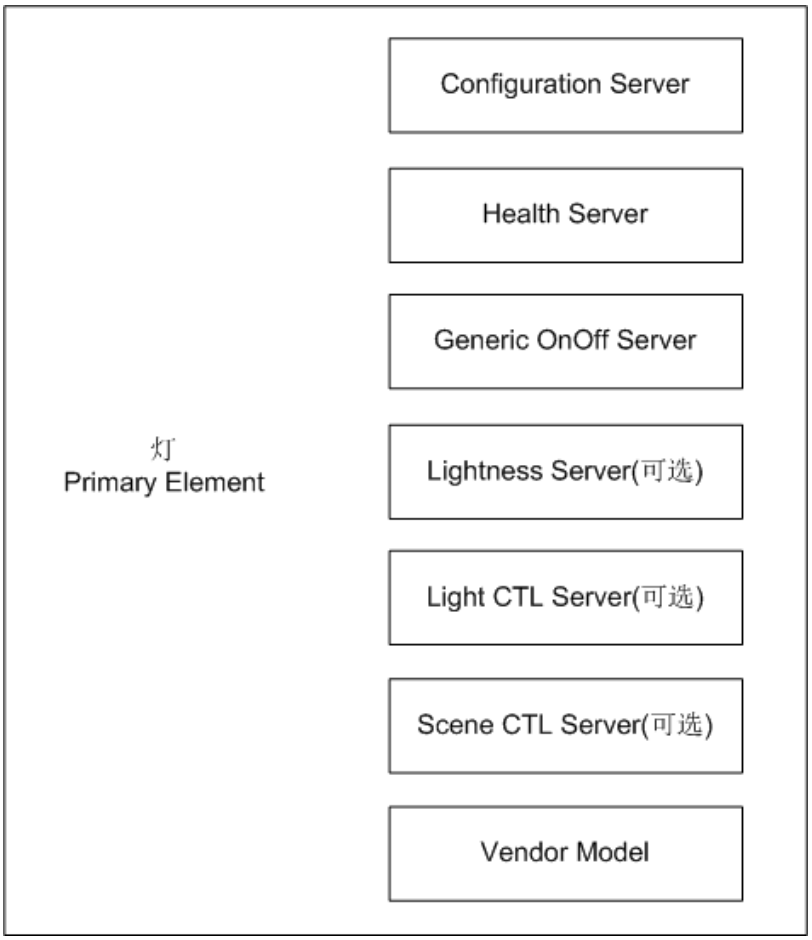
整个 SDK 包含：

- 直接关联天猫精灵 3 个应用 solution：
  - 灯应用：genie_mesh_light_ctl
  - 通用固件：genie_mesh_node_ctl
  - 多键开关应用：genie_mesh_switches
- 通过上述 3 个应用关联天猫精灵 Mesh SDK 涉及到的组件如 bt_mesh，bt_host 等。
- 通过上述 3 个应用关联天猫精灵定制 Mesh 芯片 TG7120B 与 TG7121B 的驱动与板级支持包等。

4 灯应用示例

4.1 应用示例概况

灯应用示例位于 solution/genie_mesh_light_ctl ，是可连接天猫精灵音箱，同时支持天猫精灵 APP 控制的灯应用示例，支持灯的开关、亮度、色温及场景模式的控制，同时支持基于 PWM 的渐变功能控制。会用到 SIG Model 与阿里巴巴 Vendor Model。



灯应用支持蓝牙联盟《Mesh Model Profile》中定义的 SIG Model 中的 Generic OnOff Model，平台通过下发 Generic OnOff Set 消息（Opcode 0x8202）来设置开关。

灯应用支持 Lightness Server Model，平台通过下发 Light Lightness Set（Opcode 0x824c）消息来设置灯的亮度。Light Lightness Set 消息中的字段 Lightness（16bit）表示亮度，0xFFFF（65535）表示最大亮度，即 100%。示例截图中的 0xE666(58982)为相对于 0xFFFF （65535）的 90%，0x199A（6554）为相对于 0xFFFF （65535）的 10%。

灯应用支持 Light CTL Model，平台通过下发 Light CTL Set（Opcode 0x825e）消息来设置灯的色温。Light CTL Set 消息中的 CTL Temperature 字段（16bit）表示色温，取值范围为 800～20000（即 0x0320～0x4E20），其中下发 0x320（800）代表色温值取最低（对应色温以百分比为单位的 0%），

0x4E20（20000）代表色温值取最高（对应色温以百分比为单位的 100%）。中间的值按比例计算，如 0x2120（8480）对应 40%。

灯应用支持 Light Scene Model，平台通过下发 Scene Set（Opcode 0x8242）消息来设置灯的模式。Scene Set 消息中的字段 Scene Number（16bit）表示灯的场景模式，是枚举型数据。如 0x0003 对应阅读模式，0x0004 对应影院模式。

4.2 编译应用示例

编译 TG7120B 上运行的程序

根据使用的开发板或者模组上 TG7120B 芯片的实际封装，在 boards/TG7120B_evb/package.yaml 文件中选择正确的配置项。默认使能了 CONFIG_CHIP_PACKAGE_QFN32。

```
## 第五部分：配置信息
def_config:                                    # 组件的可配置项
    BOARD_TG7120B_EVB: 1
    #CONFIG_CHIP_PACKAGE_SOP16: 1
    #CONFIG_CHIP_PACKAGE_SOP24: 1
    CONFIG_CHIP_PACKAGE_QFN32: 1
```

使能了正确的封装配置后，执行编译。

```
cd solutions/genie_mesh_light_ctl/
make clean
make
```

生成固件：
完整的烧录固件：solutions/genie_mesh_light_ctl/generated/total_image.hexf
OTA 固件：solutions/genie_mesh_light_ctl/generated/fota.bin

编译 TG7121B 上运行的程序

根据使用的开发板或者模组上 TG7121B 芯片的实际封装，在 boards/TG7121B_evb/package.yaml 文件中选择正确的配置项。默认使能了 CONFIG_CHIP_PACKAGE_QFN32。

```
## 第五部分：配置信息
def_config:                                    # 组件的可配置项
    BOARD_TG7121B_EVB: 1
    CONFIG_CHIP_PACKAGE_QFN32: 1
    #CONFIG_CHIP_PACKAGE_QFN48: 1
```

使能了正确的封装配置后，执行编译。

```
cd solutions/genie_mesh_light_ctl/
make clean
```

```
make SDK=sdk_chip_TG7121B
```

生成固件：
完整的烧录固件：solutions/genie_mesh_light_ctl/generated/total_image.hex
OTA 固件：solutions/genie_mesh_light_ctl/generated/fota.bin

4.3 例程烧录

烧录 TG7120B 上运行的程序

参考 boards/TG7120B_evb/README.md 中的说明。

烧录 TG7121B 上运行的程序

参考 boards/TG7121B_evb/README.md 中的说明。

4.4 宏定义说明

应用宏定义在 solutions/genie_mesh_light_ctl/package.yaml 文件中配置。

灯功能相关宏定义

宏定义的名称	功能说明
CONFIG_MESH_MODEL_GEN_ONOFF_SRV	支持开关控制功能
CONFIG_MESH_MODEL_LIGHTNESS_SRV	支持灯亮度控制功能
CONFIG_MESH_MODEL_CTL_SRV	支持灯色温控制功能
CONFIG_MESH_MODEL_SCENE_SRV	支持场景模式控制功能
CONFIG_MESH_MODEL_TRANS	支持灯渐变控制功能

| CONFIG_BT_MESH_CTRL_RELAY | 启用 Mesh Controlled Relay 特性(用于多节点场景消减网络泛洪) |

| CONFIG_BT_MESH_NPS_OPT | 启用 Mesh 网络性能优化特性(用于多节点场景组控操作状态同步要求） |

通用功能相关宏定义

宏定义的名称	功能说明
CONFIG_BT_MESH_GATT_PROXY	支持 Proxy 功能
CONFIG_BT_MESH_PB_GATT	支持手机配网功能
CONFIG_BT_MESH_RELAY	支持中继功能

CONFIG_GENIE_OTA	支持手机 OTA 功能
CONFIG_GENIE_RESET_BY_REPEAT	支持连续上电五次进入配网状态功能
CONFIG_BT_MESH_CTRL_RELAY	支持 Mesh Controlled Relay 功能(用于多节点场景消减网络泛洪)
CONFIG_BT_MESH_NPS_OPT	支持 Mesh 网络性能优化功能(用于多节点场景组控操作状态同步要求，组灯类产品建议开启)
PROJECT_SW_VERSION	配置版本号，OTA 功能使用，int32 数据类型
CONFIG_PM_SLEEP	支持低功耗功能（芯片未适配，暂不推荐打开）
CONFIG_GENIE_MESH_GLP	支持 GLP 模式的低功耗功能（芯片尚未完全适配，低功耗暂不推荐打开）
CONFIG_DEBUG	支持 BT_xxx 日志输出
CONFIG_BT_DEBUG_LOG	支持 BT_DBG 日志输出
MESH_DEBUG_PROV	支持配网日志输出
MESH_DEBUG_TX	支持 Access 层发送 Mesh 数据日志输出
MESH_DEBUG_RX	支持 Access 层接收 Mesh 数据日志输出

4.5 串口命令

命令名称	命令说明	使用参考（示例）
set_tt	设置蓝牙 Mesh 设备证书	set_tt 5297793 0c51b11c6ec78b52b803b3bbaae64fba486e704a5bf6
get_tt	查看蓝牙 Mesh 设备证书	无参数
get_info	查看版本和 MAC 等信息	无参数
reboot	系统重启	无参数
reset	设备复位	无参数
mesg	通过 MESH 发送数据	mesg d4 1 f000 010203

1） set_tt 命令格式

set_tt <ProductID> <Device Secret> <Device Name>

2） mesg 命令格式

- 第一个参数d4就是indication发送，其他有D3、CE及CF等；参考蓝牙Mesh设备扩展协议
- 第二个参数是发送模式和重发次数参数
  - 0表示不重发
  - 1-252表示重发次数
  - 253表示使用payload的第一个字节作为时间间隔参数，单位是100ms，例如：mesg d4 253 f000 030201　表示300毫秒发一次0201，mesg d4 253 f000 1e0201是3秒一次0201
  - 254表示收到回复或者发送超时就再次发送
  - 255表示每秒自动发送一次
- 第三个参数是接收者地址，必须是四个字符如果设置为0000会使用音箱默认组播地址F000；
- 第四个参数是发送的内容，例如 010203 就是发送 0x01，0x02，0x03 因此必须是偶数个 0-f 之间的字符。

4.6 创建与调试产品

开发天猫精灵项目产品

在生活物联网平台创建天猫精灵生态的蓝牙 Mesh 产品。参考如下文档。

- 创建项目
- 创建产品并定义产品功能
- 配置 App
- 添加设备

TG7120B 调试串口默认波特率为 256000，开发板 reset 或上电后连接串口工具后串口会有打印输出。
TG7121B 调试串口默认波特率为 115200，且注意默认无打印输出。
使用 set_tt 指令输入设置上述添加设备步骤中获取的蓝牙 Mesh 设备证书。
然后输入 reboot，设备会发送 Unprovisioned Beacon，进入待配网状态。
用天猫精灵音箱或者天猫精灵 App 配网后，可以在生活物联网平台控制台调试设备，请参考如下文档。

- 调试设备

开发自有品牌项目产品

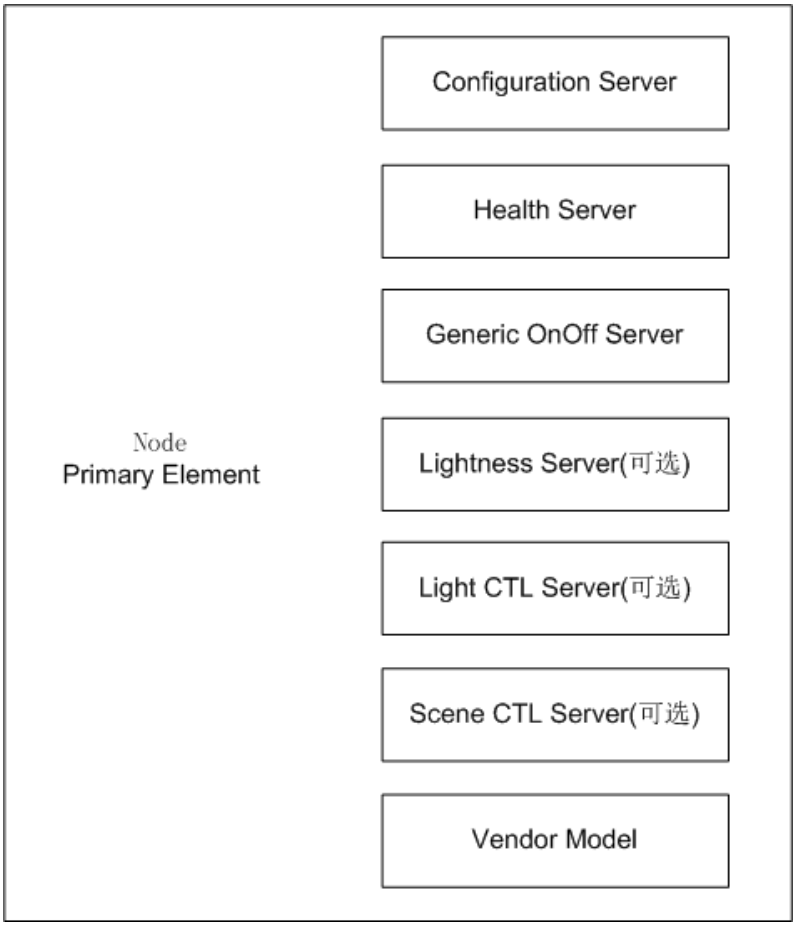
可以参考最佳实践-开发自有品牌项目蓝牙 Mesh 灯产品。

5 通用节点应用示例

5.1 应用示例概况

genie_mesh_node_ctl 是可连接天猫精灵音箱，同时支持天猫精灵 APP 控制的 Mesh 节点应用示例，可以用于开发支持开关、场景模式控制的灯产品；或者作为支持 AT 指令/串口协议的蓝牙 Mesh 模组，外接 MCU 通过串口与蓝牙 Mesh 模组通信来开发各品类智能家居产品；也可以作为基于单蓝牙 Mesh 芯片二次开发的基础代码。会用到 SIG Model 与阿里巴巴 Vendor Model。

genie_mesh_node_ctl 中默认配置的模型如下图所示，其中保留了 Lightness Server（用于灯亮度控制），Light CTL Server（用于灯色温）控制，即可以给 genie_mesh_node_ctl 配置灯品类的设备证书，灯的功能基本可以工作，便于调试。也可以通过宏 CONFIG_MESH_MODEL_LIGHTNESS_SRV 与 CONFIG_MESH_MODEL_CTL_SRV，把这两个模型关闭。



从应用开发角度看，绝大多数品类都可以通过通用开关服务模型（Generic OnOff Server）、场景控制服务模型（Scene CTL Server）与阿里巴巴厂商自定义模型（Vendor Model）组合来实现。

模型配置对应的代码在 genie_mesh_node_ctl/node_ctl/node_ctl.c 文件中，如下。

```
/* 节点产品中的 SIG 通用模型定义 */
static struct bt_mesh_model primary_element[] = {
    MESH_MODEL_CFG_SRV_NULL(),              /* Configuration Server */
    MESH_MODEL_HEALTH_SRV_NULL(),           /* Health Server */

    MESH_MODEL_GEN_ONOFF_SRV(&node_ctl_elem_stat[0]),       /* Generic OnOff Server */
    MESH_MODEL_LIGHTNESS_SRV(&node_ctl_elem_stat[0]),       /* Lightness Server */
    MESH_MODEL_CTL_SRV(&node_ctl_elem_stat[0]),             /* Light CTL Server */
    MESH_MODEL_SCENE_SRV(&node_ctl_elem_stat[0])            /* Scene Server */
};
```

```
/* 节点产品中的厂商自定义模型定义 */
static struct bt_mesh_model primary_vendor_element[] = {
    MESH_MODEL_VENDOR_SRV(&node_ctl_elem_stat[0]),
};

/* 单 element 节点注册 SIG 通用模型和厂商自定义模型*/
struct bt_mesh_elem node_ctl_elements[] = {
    BT_MESH_ELEM(0, primary_element, primary_vendor_element, 0),
    /*BT_MESH_ELEM 宏最后一个参数为 0，即节点默认不订阅品类组播，二次开发时可以修改*/
};
```

5.2 编译固件

编译 TG7120B 上运行的程序

根据使用的开发板或者模组上 TG7120B 芯片的实际封装，在 boards/TG7120B_evb/package.yaml 文件中选择正确的配置项。默认使能了 CONFIG_CHIP_PACKAGE_QFN32。

```
## 第五部分：配置信息
def_config:                                    # 组件的可配置项
    BOARD_TG7120B_EVB: 1
    #CONFIG_CHIP_PACKAGE_SOP16: 1
    #CONFIG_CHIP_PACKAGE_SOP24: 1
    CONFIG_CHIP_PACKAGE_QFN32: 1
```

使能了正确的封装配置后，执行编译。

```
cd solutions/genie_mesh_node_ctl/
make clean
make
```

生成固件：
完整的烧录固件：solutions/genie_mesh_node_ctl/generated/total_image.hexf
OTA 固件：solutions/genie_mesh_node_ctl/generated/fota.bin

编译 TG7121B 上运行的程序

根据使用的开发板或者模组上 TG7121B 芯片的实际封装，在 boards/TG7121B_evb/package.yaml 文件中选择正确的配置项。默认使能了 CONFIG_CHIP_PACKAGE_QFN32。

```
## 第五部分：配置信息
def_config:                                    # 组件的可配置项
    BOARD_TG7121B_EVB: 1
```

```
CONFIG_CHIP_PACKAGE_QFN32: 1
#CONFIG_CHIP_PACKAGE_QFN48: 1
```

使能了正确的封装配置后，执行编译。

```
cd solutions/genie_mesh_node_ctl/
make clean
make SDK=sdk_chip_TG7121B
```

生成固件：
完整的烧录固件：solutions/genie_mesh_node_ctl/generated/total_image.hex
OTA 固件：solutions/genie_mesh_node_ctl/generated/fota.bin

5.3 例程烧录

烧录 TG7120B 上运行的程序

参考 boards/TG7120B_evb/README.md 中的说明。

烧录 TG7121B 上运行的程序

参考 boards/TG7121B_evb/README.md 中的说明。

5.4 宏定义说明

应用宏定义在 solutions/genie_mesh_node_ctl/package.yaml 文件中配置。

模型相关宏定义

宏定义的名称	功能说明
CONFIG_MESH_MODEL_GEN_ONOFF_SRV	支持开关控制功能（所有支持开关属性的品类）
CONFIG_MESH_MODEL_LIGHTNESS_SRV	支持亮度控制功能（主要灯品类使用）
CONFIG_MESH_MODEL_CTL_SRV	支持色温控制功能（主要灯品类使用）
CONFIG_MESH_MODEL_SCENE_SRV	支持场景模式控制功能（所有支持场景模式属性的品类）

透传固件相关宏定义

宏定义的名称	功能说明
CONFIG_GENIE_MESH_AT_CMD	使用 AT 指令协议，详情请参考 9.1

CONIFG_GENIE_MESH_BINARY_CMD	使用二进制串口协议，详情请参考 10.1
CONIFG_GENIE_MESH_USER_CMD	使用用户自定义串口协议，详情请参考 5.9

注意：

- **可以自定义串口协议，实现代码用 CONIFG_GENIE_MESH_USER_CMD 宏开关。**
- **以上三个宏定义只能开启其中一个。**

通用功能相关宏定义

宏定义的名称	功能说明
CONFIG_BT_MESH_GATT_PROXY	支持 Proxy 功能
CONFIG_BT_MESH_PB_GATT	支持手机配网功能
CONFIG_BT_MESH_RELAY	支持中继功能
CONFIG_GENIE_OTA	支持手机 OTA 功能
CONFIG_GENIE_RESET_BY_REPEAT	支持连续上电五次进入配网状态功能
CONFIG_BT_MESH_CTRL_RELAY	支持 Mesh Controlled Relay 功能(用于多节点场景消减网络泛洪)
CONFIG_GENIE_MESH_NO_AUTO_REPLY	控制在透传模式下，是否自动回复 Attribute Status 与属性变化后的 Attribute Indication。这个可以由应用自行决定是否打开，默认会自动回复
PROJECT_SW_VERSION	配置版本号，OTA 功能使用，int32 数据类型
CONFIG_PM_SLEEP	支持低功耗功能（芯片未适配，暂不推荐打开）
CONFIG_GENIE_MESH_GLP	支持 GLP 模式的低功耗功能（芯片尚未完全适配，低功耗暂不推荐打开）
CONFIG_DEBUG	支持 BT_xxx 日志输出
CONFIG_BT_DEBUG_LOG	支持 BT_DBG 日志输出
MESH_DEBUG_PROV	支持配网日志输出
MESH_DEBUG_TX	支持 Access 层发送 Mesh 数据日志输出
MESH_DEBUG_RX	支持 Access 层接收 Mesh 数据日志输出

5.5 创建与调试产品

在生活物联网平台创建天猫精灵生态的蓝牙 Mesh 产品。参考如下文档。

- 创建项目
- 创建产品并定义产品功能
- 配置 App
- 添加设备

注意以下事项：

- **是否接入网关**配置为**是**，天猫精灵生态项目下创建的产品，接入网关协议可以选择 **BLE Mesh** 或者 **BLE GATT**。
- **接入网关协议**配置为 **BLE Mesh** 时，可以根据产品类型选择**低功耗**与**非低功耗**。如选择**低功耗**，按照精灵低功耗(GLP)方案，天猫精灵音箱在给此产品设备发送数据的时候，会在 1.2s 的时间内持续不断地发送数据。
- 数据格式如果选择**透传/自定义**，需要在云端实现自定义格式数据转换的脚本，并且设备与云端通信时使用 Vendor message 透传消息。

TG7120B 调试串口默认波特率为 256000，开发板 reset 或上电后连接串口工具后串口会有打印输出。
TG7121B 调试串口默认波特率为 115200，且注意默认无打印输出。
使用 set_tt 指令输入设置上述添加设备步骤中获取的蓝牙 Mesh 设备证书。
然后输入 reboot，设备会发送 Unprovisioned Beacon，进入待配网状态。

用天猫精灵音箱或者天猫精灵 App 配网后，可以在生活物联网平台控制台调试设备，请参考如下文档。

- 调试设备

5.6 产品应用

基于 genie_mesh_node_ctl 可以二次开发不同品类的产品，需要处理的包括：

- 不同产品有不同的属性、事件、服务的定义，属性、事件、服务会对应 Vendor Model 中不同 Attribute Type。在应用代码里面需要对这些做处理。详细的数据格式参考蓝牙 Mesh 设备扩展协议。
- 不同品类产品有不同的品类组播地址，需要通过代码内部配置，或者通过 AT 指令/二进制串口协议等方式从模组外部配置。

接入天猫精灵的蓝牙 mesh 智能设备的属性、事件和场景模式等定义请参考：

- 设备属性表
- 设备事件表
- 设备场景模式表

品类组播地址定义请参考：

- 设备组播地址

外接 MCU 模式开发注意事项：

- 如果用外接 MCU 通过 AT 指令/二进制串口协议与蓝牙模组交互的方式来开发产品，注意打开宏 CONFIG_GENIE_MESH_NO_AUTO_REPLY，这样 Generic OnOff Get（获取开关状态）、Generic OnOff Set（设置开关状态）、Scene Get（获取模式状态）、Scene Set（设置模式状态）这些指令，蓝牙模组不会自动回复，而是会透传给外部的 MCU，由外部 MCU 来解析这些报文并根据设备实际的开关状态和模式状态组装对应的 Generic OnOff Status 或者 Scene Status 报文来回复。即把开关状态和模式状态统一放到外部 MCU 来管理。
- 如果要把开关状态和模式状态甚至其他 Vendor Model 中 Attribute Type 对应的属性统一放到蓝牙模组来管理，可以二次开发 genie_mesh_node_ctl，通过使能 CONIFG_GENIE_MESH_USER_CMD 宏，并在用户自定义串口协议中设计 MCU 把相关属性同步到蓝牙模组的机制来实现。例如 MCU 把开关状态及时同步到 genie_mesh_node_ctl 中的对应 element 的开关属性（如 node_ctl_elem_stat[index].state.onoff[TYPE_PRESENT],index 索引 element，单 element 设备即为 0），这样可以不打开宏 CONFIG_GENIE_MESH_NO_AUTO_REPLY，由蓝牙模组根据存储的最新开关状态，自动回复 Generic OnOff Status 报文。

5.7 调试串口

调试串口配置

genie_mesh_node_ctl 默认以 UART0 作为调试串口。

```
#define CONSOLE_UART_IDX    0    /* 指定 UART0 作为 console UART，即调试串口*/
```

通过调用 console_init 函数指定 console UART 与波特率。

串口命令说明

命令名称	命令说明	使用参考（示例）
set_tt	设置蓝牙 Mesh 设备证书	set_tt 5297793 0c51b11c6ec78b52b803b3bbaae64fba 486e704a5bf6
get_tt	查看蓝牙 Mesh 设备证书	无参数
get_info	查看版本和 MAC 等信息	无参数
reboot	系统重启	无参数
reset	设备复位	无参数
mesg	通过 MESH 发送数据	mesg d4 1 f000 010203

1） set_tt 命令格式：

set_tt <ProductID> <Device Secret> <Device Name>

2） mesg 命令格式

- 第一个参数d4就是indication发送，其他有D3、CE及CF等；参考蓝牙Mesh设备扩展协议
- 第二个参数是发送模式和重发次数参数
  - 0表示不重发
  - 1-252表示重发次数
  - 253表示使用payload的第一个字节作为时间间隔参数，单位是100ms，例如：mesg d4 253 f000 030201　表示300毫秒发一次0201，mesg d4 253 f000 1e0201是3秒一次0201
  - 254表示收到回复或者发送超时就再次发送
  - 255表示每秒自动发送一次
- 第三个参数是接收者地址，必须是四个字符如果设置为0000会使用音箱默认组播地址F000；
- 第四个参数是发送的内容，例如010203就是发送0x01，0x02，0x03因此必须是偶数个0-f之间的字符。

5.8 数据串口配置示例

TG7120B 数据串口配置示例

引脚定义参考：boards/TG7120B_evb/include/board_config.h

```
#if defined(CONIFG_GENIE_MESH_BINARY_CMD) || defined(CONFIG_GENIE_MESH_AT_CMD)
#define MCU_UART_IDX 1
#define MCU_TXD P18
#define MCU_RXD P20
#define MCU_TXD_FUNC FMUX_UART1_TX
#define MCU_RXD_FUNC FMUX_UART1_RX
#endif
```

引脚配置参考：boards/TG7120B_evb/board_init.c

```
#if defined(CONIFG_GENIE_MESH_BINARY_CMD) || defined(CONFIG_GENIE_MESH_AT_CMD)
    drv_pinmux_config(MCU_TXD, MCU_TXD_FUNC);
    drv_pinmux_config(MCU_RXD, MCU_RXD_FUNC);
    uart_csky_register(MCU_UART_IDX);
#endif
```

TG7121B 数据串口配置示例

参考 boards/TG7121B_evb/src/board_init.c

```
#if defined(CONIFG_GENIE_MESH_BINARY_CMD) || defined(CONFIG_GENIE_MESH_AT_CMD)
    uart2_io_init(PB08, PB09);      /*配置 PB08 引脚为 TX，PB09 引脚为 RX*/
#endif
```

5.9 用户自定义串口协议示例

在 package.yaml 文件中打开宏 ONIFG_GENIE_MESH_USER_CMD，关闭宏定义 CONFIG_GENIE_MESH_AT_CMD 和 CONIFG_GENIE_MESH_BINARY_CMD。

模组接收 MCU 的数据并上行发送

模组接收 MCU 的数据需要用户自己开发解析代码，可以参考 CONIFG_GENIE_MESH_BINARY_CMD 宏管理的二进制串口协议的实现方式。
模组中发送 Mesh 数据调用函数 genie_sal_ble_send_msg。如果没有特殊需求，请将 **request_msg.tid** 的值设置为 0,即 request_msg.tid = p_data[3]。

```
int genie_sal_ble_send_msg(uint8_t element_id, uint8_t* p_data, uint8_t len);
```

参数 element_id 可指定多 element 设备的哪一个 element 发送。单 element 设备则取值为 0。
参数 p_data 指向要发送的数据包，如要发送 vendor model 消息，格式参考蓝牙 Mesh 设备扩展协议。

模组收到下行数据并转发给 MCU

在 user_event 的 GENIE_EVT_DOWN_MSG 事件中接收下行数据。

```
#ifdef CONIFG_GENIE_MESH_USER_CMD
    case GENIE_EVT_DOWN_MSG:
    {
        genie_down_msg_t *p_msg = (genie_down_msg_t *)p_arg;
        //User handle this msg,such as send to MCU
        if (p_msg)
        {
        }
    }
    break;
#endif
```

6 三键开关应用示例

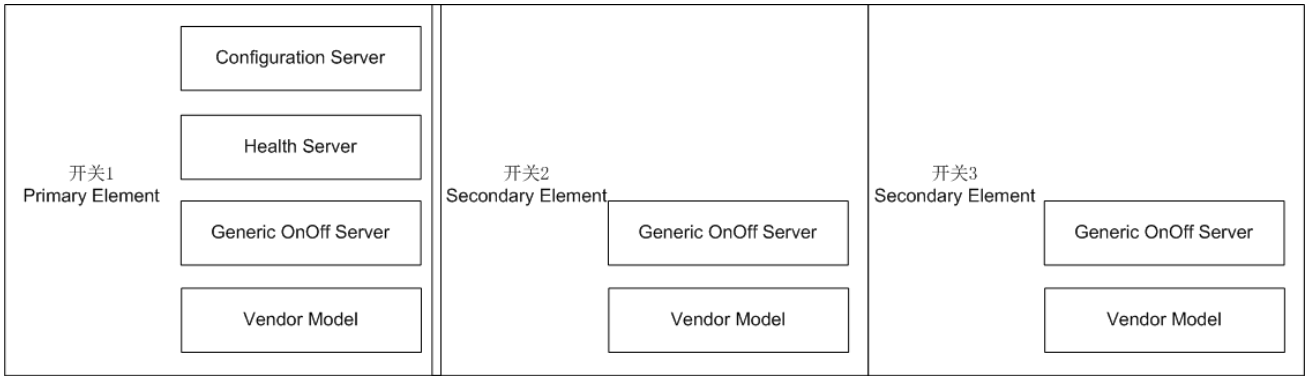
6.1 应用示例概况

genie_mesh_switches 是可连接天猫精灵的三键开关的参考应用，可以作为零火开关或者其他多 Element Mesh 设备的开发示例。目前只适配了 TG7120B，且由于 TG7120B 芯片暂未适配低功耗，所以不推荐做单火

开关应用。如果开发风扇灯、多孔插座等产品可以参考这个应用进行产品开发。
示例可以在 TG7120B QFN32 封装的开发板上运行。这个开发板上只有两个独立按键，其中 P14 按键对应 Element 0，P15 按键对应 Element 1，另外引脚 P24 对应 Element 2，用一个跳线短接 GND 地线与 P24 引脚，即可模拟按键。

GPIO 定义	短按键	灯状态
P14（Element 0）	翻转 Element 0 开关属性	翻转开发板红灯开关状态
P15（Element 1）	翻转 Element 1 开关属性	翻转开发板绿灯开关状态
P24（Element 2）	翻转 Element 2 开关属性	翻转开发板蓝灯开关状态

genie_mesh_switches 中默认配置的模型如下图所示，三键开关对应 3 个 element，只有主 element 包括 Configuration Server Model 与 Health Server Model。从 element 只有 Generic OnOff Server Model 和阿里巴巴 Vendor Model。



模型配置对应的代码在 genie_mesh_switches/switches/switches.c 文件中。

```
struct bt_mesh_elem switches_elements[] = {
    BT_MESH_ELEM(0, primary_element, primary_vendor_element, GENIE_ADDR_SWITCH),
    BT_MESH_ELEM(0, secondary_element, secondary_vendor_element, GENIE_ADDR_SWITCH),
    BT_MESH_ELEM(0, third_element, third_vendor_element, GENIE_ADDR_SWITCH),
};
```

其中每个 element 通过宏定义 GENIE_ADDR_SWITCH 配置了开关的品类组播地址。

```
typedef enum _genie_addr
{
    GENIE_ADDR_MIN = 0xc000,
    GENIE_ADDR_LIGHT = 0xc000,          /*灯品类组播地址*/
    GENIE_ADDR_SWITCH = 0xc001,         /*开关品类组播地址*/
    GENIE_ADDR_SOCKET = 0xc002,         /*插座品类组播地址*/
    GENIE_ADDR_CURTAIN = 0xc003,        /*窗帘品类组播地址*/
    GENIE_ADDR_BODY_SCALE = 0xc004,
    GENIE_ADDR_BUTTON = 0xc005,
    …..
```

```
} genie_addr_e;
```

如果开发别的类型的设备，注意每个 Element 可以定义不同的组播地址。例如如果开发风扇灯，两个 element 可以分别填灯的品类组播地址和一个风扇的品类组播地址。

6.2 编译固件

编译 TG7120B 上运行的程序

根据使用的开发板或者模组上 TG7120B 芯片的实际封装，在 boards/TG7120B_evb/package.yaml 文件中选择正确的配置项。默认使能了 CONFIG_CHIP_PACKAGE_QFN32。

```
## 第五部分：配置信息
def_config:                                    # 组件的可配置项
    BOARD_TG7120B_EVB: 1
    #CONFIG_CHIP_PACKAGE_SOP16: 1
    #CONFIG_CHIP_PACKAGE_SOP24: 1
    CONFIG_CHIP_PACKAGE_QFN32: 1
```

使能了正确的封装配置后，执行编译。

```
cd solutions/genie_mesh_switches/
make clean
make
```

生成固件：
完整的烧录固件：solutions/genie_mesh_switches/generated/total_image.hexf
OTA 固件：solutions/genie_mesh_switches/generated/fota.bin

注意：本例程暂未适配 TG7121B 芯片。

6.3 例程烧录

烧录 TG7120B 上运行的程序

参考 boards/TG7120B_evb/README.md 中的说明。

6.4 宏定义说明

应用宏定义在 solutions/genie_mesh_switches/package.yaml 文件中配置。

模型相关宏定义

宏定义的名称	功能说明

CONFIG_MESH_MODEL_GEN_ONOFF_SRV	支持开关控制功能（所有支持开关属性的品类）

通用功能相关宏定义

宏定义的名称	功能说明
CONFIG_BT_MESH_GATT_PROXY	支持 Proxy 功能
CONFIG_BT_MESH_PB_GATT	支持手机配网功能
CONFIG_BT_MESH_RELAY	支持中继功能
CONFIG_GENIE_OTA	支持手机 OTA 功能
CONFIG_GENIE_RESET_BY_REPEAT	支持连续上电 5 次进入配网状态功能
PROJECT_SW_VERSION	配置版本号，OTA 功能使用，int32 数据类型
CONFIG_DEBUG	支持 BT_xxx 日志输出
CONFIG_BT_DEBUG_LOG	支持 BT_DBG 日志输出
MESH_DEBUG_PROV	支持配网日志输出
MESH_DEBUG_TX	支持 Access 层发送 Mesh 数据日志输出
MESH_DEBUG_RX	支持 Access 层接收 Mesh 数据日志输出

7 Genie Service 组件接口说明

genie_service 组件包含 SIG Model 中 Generic On Off 等 Model 的解析处理和数据应答，和阿里巴巴 Vendor Model 的实现。

7.1 宏定义说明

开发者可配置的宏定义的配置文件：components/genie_service/package.yaml

宏定义名称	功能说明	备注
CONFIG_BT_MESH_PB_ADV	定义支持基于 PB-ADV 的广播承载配网	所有 genie mesh 设备都需要支持 PB-ADV
CONFIG_GENIE_MESH_ENABLE	定义 genie mesh SDK 使能	默认使能

CONFIG_GENIE_MESH_PORT	定义在 genie mesh SDK 中使用 genie event	默认使能
CONFIG_AOS_CLI	定义支持串口命令	默认支持
CONFIG_BT_MESH_PROV_TIMEOUT	定义关闭 bt_mesh 中 CONFIG_BT_MESH_PROV_TIMEOUT	provision timeout 由 genie_provision 中的定时器接管
CONFIG_BT_MESH_MODEL_GROUP_COUNT	定义支持的组播地址个数	默认为 8
GENIE_DEFAULT_DURATION	定义 ADV 发包时长（ms）	默认为 205ms

7.2 初始化

函数定义：int genie_service_init(genie_service_ctx_t* p_ctx)
参数说明：
结构体 genie_service_ctx_t 的成员参数如下：

参数名称	参数类型	功能说明	备注
p_mesh_elem	structbt_mesh_elem	Element 的地址	
mesh_elem_counts	uint8_t	Element 的数目	
event_cb	user_event_cb	SDK 上传给应用的事件	用户事件回调，系统初始化和复位通知及接收 Mesh 数据通知，另外还有定时事件和用户自定义串口透传数据通知
prov_timeout	uint32_t	默认 10 分钟，超时之后进入静默广播状态	静默广播可以进行设备发现，但是不能直接配网
lpm_conf	genie_lpm_conf_t	低功耗相关参数配置	低功耗相关，由宏定义 CONFIG_PM_SLEEP 开关该功能

结构体 genie_lpm_conf_t 的成员参数如下：

参数名称	参数类型	功能说明	备注
lpm_wakeup_io	uint8_t	是否支持 GPIO 唤醒，0:不支持，>0:支持	
lpm_wakeup_io_config	genie_lpm_wakeup_io_t	配置多个唤醒 GPIO	多键开关可以使用
is_auto_enable	uint8_t	上电之后是否自动进入低功耗状态	大于 0 自动进入低功耗

delay_sleep_time	uint32_t	上电多久之后进入低功耗状态，默认是 10 秒	
sleep_ms	uint16_t	低功耗睡眠时长	GLP 模式的时候 1100ms
wakeup_ms	uint16_t	低功耗唤醒时长	GLP 模式的时候 60ms
genie_lpm_cb	genie_lpm_cb_t	睡眠唤醒状态通知用户的回调函数	

7.3 发送数据

SDK 为 Genie Vendor Model 的数据发送提供了两个 API 接口，
genie_transport_send_model 可以在参数中指定发送的 element，genie_transport_send_payload 不可指定 element，直接使用 primary element，两个函数的详细介绍如下。

genie_transport_send_model

函数名称：int genie_transport_send_model(genie_transport_model_param_t* p_vendor_msg)
参数说明：
结构体 genie_transport_model_param_t 的成员参数如下：

参数名称	参数类型	功能说明	备注
p_elem	struct bt_mesh_elem*	指定发送数据的 Element，主要是获取源地址	
p_model	struct bt_mesh_model*	指定发送数据的 model，主要是获取 app key	
result_cb	transport_result_cb	发送结果回调给应用	发送超时或者发送成功（D3 和 CF 消息不会执行回调）
retry	uint8_t	没有收到应答执行重发的次数，默认重发两次	如果是 GLP 设备默认重发五次
opid	uint8_t	opcode id，例如：0xD4	
tid	uint8_t	传输 ID，通常传入值 0，为 0 的时候 SDK 会自动生成 TID	Mesh 设备发送数据的 TID 从 0x80 到 0xBF
dst_addr	uint16_t	数据目的地址	
retry_period	uint16_t	重传的时间间隔	
len	uint16_t	发送数据的长度	

data	uint8_t*	发送数据的起始地址	

函数调用的参考代码如下，可以调用 bt_mesh_elem_find_by_id 函数获取对应的 element 指针。

```
...

genie_transport_model_param_t transport_model_param = {0};

memset(&transport_model_param, 0, sizeof(genie_transport_model_param_t));
transport_model_param.opid = VENDOR_OP_ATTR_INDICATE;
transport_model_param.data = payload;
transport_model_param.len = MIN(SIG_MODEL_INDICATE_PAYLOAD_MAX_LEN, payload_len);
transport_model_param.p_elem = bt_mesh_elem_find_by_id(p_elem->element_id);
transport_model_param.retry_period = GENIE_TRANSPORT_EACH_PDU_TIMEOUT *
genie_transport_get_seg_count(transport_model_param.len);
transport_model_param.retry = GENIE_TRANSPORT_DEFAULT_RETRY_COUNT;

genie_transport_send_model(&transport_model_param);

...
```

genie_transport_send_payload

函数名称：int genie_transport_send_payload(genie_transport_payload_param_t* payload_param)
结构体 genie_transport_payload_param_t 的成员参数如下：

参数名称	参数类型	功能说明	备注
opid	uint8_t	opcode id，例如：0xD4	
tid	uint8_t	传输 ID，通常传入值 0，为 0 的时候 SDK 会自动生成 TID	Mesh 设备发送数据的 TID 从 0x80 到 0xBF
retry_cnt	uint8_t	没有收到应答执行重发的次数，默认重发两次	如果是 GLP 设备默认重发五次
dst_addr	uint16_t	数据目的地址	
p_payload	uint8_t*	发送数据的起始地址	
payload_len	uint16_t	发送数据的长度	

result_cb	transport_result_cb	发送结果回调给应用	发送超时或者发送成功（D3 和 CF 消息不会执行回调）

调用参考代码如下。

```
...

genie_transport_payload_param_t transport_payload_param;

memset(&transport_payload_param, 0, sizeof(genie_transport_payload_param_t));
transport_payload_param.opid = VENDOR_OP_ATTR_INDICATE;
transport_payload_param.p_payload = payload;
transport_payload_param.payload_len = sizeof(payload);
transport_payload_param.retry_cnt = GENIE_TRANSPORT_DEFAULT_RETRY_COUNT;
transport_payload_param.result_cb = report_poweron_callback;

genie_transport_send_payload(&transport_payload_param);

...
```

7.4 接收数据

应用程序接收 Mesh 数据是通过初始化的时候注册的用户回调函数接收的，如下两个 Event 分别接收 SIG Model 的数据和 Vendor Model 的数据。

接收 SIG Model 的数据

```
…

case GENIE_EVT_SIG_MODEL_MSG:
{
    sig_model_msg *p_msg = (sig_model_msg *)p_arg;

    if (p_msg)
    {
        GENIE_LOG_INFO("SIG mesg ElemID(%d)", p_msg->element_id);
    }
}
break;
…
```

接收 Vendor Model 的数据

```
…

case GENIE_EVT_VENDOR_MODEL_MSG:
{
    genie_transport_model_param_t *p_msg = (genie_transport_model_param_t *)p_arg;

    if (p_msg && p_msg->p_model && p_msg->p_model->user_data)
    {
        sig_model_element_state_t *p_elem_state = (sig_model_element_state_t
*)p_msg->p_model->user_data;
        GENIE_LOG_INFO("ElemID(%d) TID(%02X)", p_elem_state->element_id, p_msg->tid);
        (void)p_elem_state;
    }
}
break;
…
```

7.5 透传固件收发数据示例

数据上行示例

当外部有 MCU，使用透传模式时，模组发送收到的 MCU 的数据给网关，可以参考如下代码示例。

```
int genie_sal_ble_send_msg(uint8_t element_id, uint8_t *p_data, uint8_t len)
{
    int ret = -1;
    uint8_t i = 0;
    uint8_t seg_count = 0;
    struct bt_mesh_model *p_model = NULL;
    struct net_buf_simple *msg = NULL;
    struct bt_mesh_msg_ctx ctx;
    struct bt_mesh_elem *p_element = NULL;
    genie_transport_model_param_t request_msg;

    if (len > GENIE_HAL_BLE_SEND_MAX_DATA_LEN)
    {
        return -1;
    }

    p_element = bt_mesh_elem_find_by_id(element_id);
    if (p_element == NULL)
```

```
    {
        printf("not find element by id:%d\r\n", element_id);
        return -1;
    }

    if ((len >= 4) && (p_data[0] == VENDOR_OP_ATTR_INDICATE || p_data[0] ==
VENDOR_OP_ATTR_INDICATE_TG))
    {
        seg_count = genie_transport_get_seg_count(len - 4 + 4);
        memset(&request_msg, 0, sizeof(genie_transport_model_param_t));
        request_msg.opid = p_data[0];
        request_msg.tid = p_data[3];
        request_msg.data = (u8_t *)(p_data + 4);
        request_msg.len = len - 4;
        request_msg.p_elem = p_element;
        request_msg.retry_period = GENIE_TRANSPORT_EACH_PDU_TIMEOUT * seg_count;
        request_msg.retry = VENDOR_MODEL_MSG_MAX_RETRY_TIMES;

        ret = genie_transport_send_model(&request_msg);
        if (ret != 0)
        {
            printf("vendor model msg send fail\n");
            return -1;
        }
        return 0;
    }
    else
    {
        ctx.app_idx = 0;
        ctx.net_idx = 0;
        ctx.addr = genie_transport_src_addr_get();//发送的目的地址是上一个接收数据的发送者
        ctx.send_ttl = GENIE_TRANSPORT_DEFAULT_TTL;
        ctx.send_rel = 0;

        msg = NET_BUF_SIMPLE(GENIE_HAL_BLE_SEND_MAX_DATA_LEN + 4);
        if (msg == NULL)
        {
            printf("no mem\n");
            return -1;
        }

        net_buf_simple_init(msg, 0);
        while (i < len)
        {
```

```
            net_buf_simple_add_u8(msg, p_data[i]);
            i++;
        }

        if ((p_data[0] & 0xC0) == 0xC0)
        {
            p_model = p_element->vnd_models;
        }
        else
        {
            p_model = p_element->models;
        }

        if (p_model == NULL)
        {
            printf("no model\n");
            return -1;
        }

        if (bt_mesh_model_send(p_model, &ctx, msg, NULL, NULL))
        {
            printf("mesh send fail\n");
            return -1;
        }

        return 0;
    }
}
```

数据下行示例

当外部有 MCU，使用透传模式时，模组将收到的网关下发的数据转发给 MCU，可以参考如下代码示例。

```
int genie_down_msg(genie_down_mesg_type msg_type, uint32_t opcode, void *p_msg)
{
    uint8_t *p_data = NULL;
    uint16_t data_len = 0;
    genie_service_ctx_t *p_context = NULL;

#ifdef CONIFG_GENIE_MESH_USER_CMD
    uint8_t element_id = 0;
    genie_down_msg_t down_msg;
#endif
```

```
    p_context = genie_service_get_context();

    if (p_msg == NULL || !p_context || !p_context->event_cb)
    {
        GENIE_LOG_ERR("param err");
        return -1;
    }

    if (GENIE_DOWN_MESG_VENDOR_TYPE == msg_type)
    {
        genie_transport_model_param_t *p_vnd_mesg = (genie_transport_model_param_t *)p_msg;
        data_len = 4 + p_vnd_mesg->len;
        p_data = (uint8_t *)aos_malloc(data_len);
        if (p_data == NULL)
        {
            return -1;
        }

        p_data[0] = p_vnd_mesg->opid;
        p_data[1] = opcode;
        p_data[2] = opcode >> 8;
        p_data[3] = p_vnd_mesg->tid;
        if (p_vnd_mesg->len > 0)
        {
            memcpy(&p_data[4], p_vnd_mesg->data, p_vnd_mesg->len);
        }

#ifdef CONIFG_GENIE_MESH_USER_CMD
        sig_model_element_state_t *p_elem_state = (sig_model_element_state_t
*)p_vnd_mesg->p_model->user_data;
        if (p_elem_state)
        {
            element_id = p_elem_state->element_id;
        }
#endif
    }
    else
    {
        sig_model_msg *p_net_buf = (sig_model_msg *)p_msg;

        p_context->event_cb(GENIE_EVT_SIG_MODEL_MSG, (void *)p_msg);

        if (opcode < 0x7F) //one byte opcode
        {
```

```
            data_len = 1 + p_net_buf->len;
            p_data = (uint8_t *)aos_malloc(data_len);
            if (p_data == NULL)
            {
                return -1;
            }
            p_data[0] = opcode & 0xFF;
            memcpy(&p_data[1], p_net_buf->data, p_net_buf->len);
        }
        else
        {
            data_len = 2 + p_net_buf->len;
            p_data = (uint8_t *)aos_malloc(data_len);
            if (p_data == NULL)
            {
                return -1;
            }
            p_data[0] = (opcode >> 8) & 0xFF;
            p_data[1] = opcode & 0xFF;
            memcpy(&p_data[2], p_net_buf->data, p_net_buf->len);
        }
#ifdef CONIFG_GENIE_MESH_USER_CMD
        element_id = p_net_buf->element_id;
#endif
    }

#ifdef CONFIG_GENIE_MESH_AT_CMD
    genie_at_cmd_send_data_to_mcu(p_data, data_len);          /* AT 指令转发给 MCU*/
#endif

#ifdef CONIFG_GENIE_MESH_BINARY_CMD
    genie_bin_cmds_send_data_to_mcu(p_data, data_len);       /*  二进制协议报文转发给 MCU*/
#endif

#ifdef CONIFG_GENIE_MESH_USER_CMD
    down_msg.len = data_len;
    down_msg.data = p_data;
    down_msg.element_id = element_id;

    p_context->event_cb(GENIE_EVT_DOWN_MSG, (void *)&down_msg);
#endif

    aos_free(p_data);
```

```
    return 0;
}
```

7.6 收发数据 TID 说明

发送数据

当调用上述接口发送数据，但 TID 字段为 0 的时候，SDK 会调用 genie_transport_gen_tid 函数
设备发送数据的 TID 取值范围在 128～191 之间，上电第一次发送的 TID 采用随机数。后续发送数据 TID 在 128～191(即 0x80～0xBF)范围内递增。

```
uint8_t genie_transport_gen_tid(void)
{
    static uint8_t tid = GENIE_TRANSPORT_TID_MAX;

    if (tid == GENIE_TRANSPORT_TID_MAX) //When bootup use rand tid
    {
        bt_rand(&tid, 1);
        tid &= 0x3F;
    }
    else
    {
        tid = (tid + 1) & 0x3F;
    }

    return (tid | 0x80);
}
```

接收数据

收到数据包会根据 TID 做消息的去重，做法如下：
缓存最近接收的 5 个消息的 TID，如果在 6 秒内收到相同 TID 的消息那么认为是重复的消息，忽略重复的消息。参考 genie_transport_check_tid 函数实现。

```
E_MESH_ERROR_TYPE genie_transport_check_tid(u16_t src_addr, uint8_t tid, uint8_t elem_id)
{
    static uint8_t cur_index = 0;
    uint8_t i = cur_index;
    uint8_t ri = 0;
    uint32_t cur_time = k_uptime_get();
    uint32_t end_time = 0;

    if (src_addr >= TMALL_GENIE_UADDR_START && src_addr <= TMALL_GENIE_UADDR_END)
    {
```

```
        src_addr = TMALL_GENIE_UADDR_START;
    }

    while (i < cur_index + RECV_MSG_TID_QUEUE_SIZE)
    {
        ri = i % RECV_MSG_TID_QUEUE_SIZE;
        if ((tid_queue[ri].tid == tid) && (tid_queue[ri].addr == src_addr) && (tid_queue[ri].elemid == elem_id))
        {
            end_time = tid_queue[ri].time + GENIE_TRANSPORT_DEDUPLICATE_DURATION;
            if (cur_time < end_time)
            {
                break;
            }
        }
        i++;
    }
    if (i < cur_index + RECV_MSG_TID_QUEUE_SIZE)
    {
        return MESH_TID_REPEAT;
    }
    else
    {
        tid_queue[cur_index].tid = tid;
        tid_queue[cur_index].elemid = elem_id;
        tid_queue[cur_index].addr = src_addr;
        tid_queue[cur_index].time = cur_time;
        cur_index++;
        cur_index %= RECV_MSG_TID_QUEUE_SIZE;

        return MESH_SUCCESS;
    }
}
```

7.7 SDK Event 说明

SDK 中事件处理分为 Genie Event 和 Sig Event 两部分，各自定义如下。

Genie Event 定义

定义在 components/genie_service/core/inc/genie_event.h 文件中。

```
typedef enum
{
    GENIE_EVT_NONE = 0,
    GENIE_EVT_SW_RESET,             /* triggered from cloud */
    GENIE_EVT_HW_RESET_START, /* triggered from user */
```

```
    GENIE_EVT_BT_READY,
    GENIE_EVT_MESH_READY, // Used to sync device's state with cloud

    GENIE_EVT_SDK_AIS_DISCON = 10,
    GENIE_EVT_SDK_AIS_CONNECT,
    GENIE_EVT_SDK_MESH_PBADV_START,
    GENIE_EVT_SDK_MESH_PBADV_TIMEOUT,
    GENIE_EVT_SDK_MESH_SILENT_START,

    GENIE_EVT_SDK_MESH_PROV_START = 20,
    GENIE_EVT_SDK_MESH_PROV_DATA,
    GENIE_EVT_SDK_MESH_PROV_TIMEOUT,
    GENIE_EVT_SDK_MESH_PROV_SUCCESS,
    GENIE_EVT_SDK_MESH_PROV_FAIL,

    GENIE_EVT_SDK_APPKEY_ADD = 30,
    GENIE_EVT_SDK_APPKEY_DEL,
    GENIE_EVT_SDK_APPKEY_UPDATE,
    GENIE_EVT_SDK_NETKEY_ADD,
    GENIE_EVT_SDK_NETKEY_DEL,
    GENIE_EVT_SDK_NETKEY_UPDATE,

    GENIE_EVT_SDK_HB_SET = 40,
#ifdef CONFIG_BT_MESH_CTRL_RELAY
    GENIE_EVT_SDK_CTRL_RELAY_SET,
#endif
    GENIE_EVT_SDK_SEQ_UPDATE,

    GENIE_EVT_TIMEOUT = 50,
    GENIE_EVT_DOWN_MSG,
    GENIE_EVT_VENDOR_MODEL_MSG,
    GENIE_EVT_SIG_MODEL_MSG,
    GENIE_EVT_USER_TRANS_CYCLE,
    GENIE_EVT_USER_ACTION_DONE
} genie_event_e;
```

Sig Event 定义

定义在 components/genie_service/core/inc/sig_models/sig_model_event.h 文件中。

```
typedef enum
{
    SIG_MODEL_EVT_NONE = 0,
    SIG_MODEL_EVT_ANALYZE_MSG,
    SIG_MODEL_EVT_TIME_OUT,
```

```
    SIG_MODEL_EVT_DOWN_MSG,
    SIG_MODEL_EVT_ACTION_DONE,
    SIG_MODEL_EVT_INDICATE,

    SIG_MODEL_EVT_DELAY_START = 10,
    SIG_MODEL_EVT_DELAY_END,

#ifdef CONFIG_MESH_MODEL_TRANS
    SIG_MODEL_EVT_TRANS_START,
    SIG_MODEL_EVT_TRANS_CYCLE,
    SIG_MODEL_EVT_TRANS_END,
#endif

    SIG_MODEL_EVT_GENERIC_MESG = 20,
} sig_model_event_e;
```

处理函数说明

两个 event 处理函数都是递归调用函数，所以扩展 event 时需要注意保证递归深度不要太深，避免线程栈溢出。

```
void genie_event(genie_event_e event, void *p_arg)
void sig_model_event(sig_model_event_e event, void *p_arg)
```

日志查看 Event 值

日志示例如下：

```
[    0.354]<I>AOSBT sig_model_event:SigE:10
[    0.439]<I>AOSBT genie_event:GenieE:3
[    0.859]<I>AOSBT sig_model_event:SigE:11
[    0.864]<I>AOSBT sig_model_event:SigE:12
[    1.819]<I>AOSBT sig_model_event:SigE:14
[    1.824]<I>AOSBT sig_model_event:SigE:4
[    1.828]<I>AOSBT sig_model_event:SigE:5
```

根据上述定义可以查出 Event 依次如下：

SigE：10 即 SIG_MODEL_EVT_DELAY_START

GenieE：3 即 GENIE_EVT_BT_READY

SigE：11 即 SIG_MODEL_EVT_DELAY_END

SigE：12 即 SIG_MODEL_EVT_TRANS_START

SigE：14 即 SIG_MODEL_EVT_TRANS_END

SigE：4 即 SIG_MODEL_EVT_ACTION_DONE

SigE：5 即 SIG_MODEL_EVT_INDICATE

8 其他参考文档

天猫精灵蓝牙 Mesh 与 BLE OTA 相关文档请参考生活物联网开发文档。

- 蓝牙 Mesh 模组软件规范
- 蓝牙 Mesh 设备扩展协议
- 蓝牙 mesh 设备开发 FAQ
- 蓝牙 BLE 基础规范
- 蓝牙 BLE OTA 规范
- 蓝牙 mesh 智能家居产品规范
- 蓝牙设备属性表

9 AT 指令定义

如果在应用固件中使能 CONFIG_GENIE_MESH_AT_CMD 宏，可以使用 AT 指令与设备交互，参考以下说明。

9.1 串口设置与指令格式

串口设置

默认波特率 9600bps，8 位数据位，无奇偶校验，1 位停止位。

AT 指令格式

符号	描述
AT	每个 AT 指令，都以 AT 开头，ASCII 码，不区分大小写
+X1	+X1 为命令
=	说明当前为设置操作，例如：AT+MESHADV=0
?	说明当前为读取操作，例如：AT+MESHADV?
,	参数分隔符，可能带多个参数，例如：AT+MESHGRP=0xC000,0xCFFF
<CR><LF>	回车换行,命令结束

AT 指令返回值

返回结果	描述
OK	串口命令执行成功
+ERROR:Y1	执行错误，Y1 为错误码，如下

	-1 - 表示命令错误 -2 - 表示参数错误 -3 - 表示执行错误
+X1:Y1..	命令 X1 对应的响应结果

9.2 AT 指令集说明

重启设备指令

描述	mesh 设备重启	
命令	AT+REBOOT	
响应 1	OK	重启成功
响应 2	+ERROR:Y1	返回错误
示例	AT+REBOOT OK	重启设备成功

清除配网信息指令

描述	清除 mesh 设备配网信息	
命令	AT+MESHRST	
响应 1	OK	清除成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHRST OK	复位清除配网信息成功，并重启设备

获取固件版本号指令

描述	获取 mesh 设备固件版本号	
命令	AT+MESHVER?	
响应 1	+MESHVER:Y1 OK	获取成功，返回结果 Y1 Y1 为当前固件版本号（4 个字节）
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHVER? +MESHVER:00010104 OK	成功获取设备固件版本号：00010104

获取设备配网状态指令

描述	获取 mesh 设备当前配网状态	
命令	AT+MESHINF?	
响应 1	+MESHINF:Y1 OK	获取成功，返回结果 Y1，表示当前配网状态，返回值范围： 0 - 表示未配网 1 - 表示已配网 |
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHINF? +MESHINF:1 OK	返回结果 1，表示设备已配网

设置蓝牙 mesh 广播类型指令

描述	控制 mesh 设备 mesh 广播类型	
命令	AT+MESHADV=<mode>	<mode>为 mesh 广播类型，取值范围： 0 - 关闭 mesh 广播 1 - 开启 mesh 广播 2 - 开启静默广播
响应 1	OK	设置成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHADV=1 OK	设置开启 mesh 广播成功

获取蓝牙 mesh 广播类型

描述	获取 mesh 设备当前 mesh 广播类型	
命令	AT+MESHADV?	
响应 1	+MESHADV:Y1 OK	获取成功，返回结果 Y1，表示设备当前 mesh 广播类型，返回值范围： 0 - 表示已关闭 mesh 广播 1 - 表示已开启 mesh 广播 2 - 表示已开启静默广播
响应 2	+ERROR:Y1	返回错误

示例	AT+MESHADV? +MESHADV:1 OK	获取设备已开启 mesh 广播

设置测试模式

设置测试模式		
描述	设置 mesh 设备进入测试模式，进行设备扫描，并获取指定 MAC 地址设备的信号强度	
命令	AT+MESHTEST=<testmode>,[param]	<testmode>为测试类型，目前可支持的测试类型值： 0 - 表示进入测试模式并获取指定 MAC 地址设备的信号强度 [param]为其对应的测试类型所需的参数，当 testmode 为 0 时，此值为设备 MAC 地址，格式为：xx:xx:xx:xx:xx:xx
响应 1	+MESHTEST:Y1,Y2 OK	获取成功，返回结果 Y1，表示设备当前测试类型 Y2，表示所指定 MAC 地址设备的信号强度
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHTEST=0,88:8c:f3:70:5f:55 +MESHTEST:0,-93 OK	设备进入测试模式，进行设备扫描，并成功获取到 88:8c:f3:70:5f:55 设备的信号强度为-93

设置组播地址

命令	AT+MESHGRP=<addr1>,[addr2]...	<addr1>为组播地址，格式为：0xXXXX，例如：0xC000，取值范围：[0xC000~0xCFFF]
响应 1	OK	设置成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHGRP=0xC000,0xCFFF OK	成功设置设备两个组播地址 0xC000、0xCFFF

获取组播地址

描述	获取 mesh 设备组播地址	
命令	AT+MESHGRP?	

响应 1	+MESHGRP:Y1,Y2 OK	获取成功，返回结果 Y1，表示组播地址 1 Y2，表示组播地址 2
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHGRP? +MESHGRP:0xC000,0xCFFF OK	成功返回设备两个组播地址 0xC000、0xCFFF

透传数据包发送

描述	模组接收外部 MCU 串口透传数据包，通过蓝牙 mesh 转发该包数据	
命令	AT+MESHMSGTX=<data>	<data>为二进制格式的数据包，例如： D4A801A500010021010000
响应 1	OK	发送成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHMSGTX=D4A801A500010021010000 OK	接收到串口透传数据包 D4A801A500010021010000 并通过蓝牙 mesh 转发该包数据

注意：详细的数据格式定义参考蓝牙Mesh设备扩展协议。

透传数据包接收

描述	模组接收应用层透传数据包，通过串口转发给外部 MCU	
命令	+MESHMSGRX:<len>,<data>	<len>为接收到数据包的长度（字节数），<data>为二进制格式的数据包
示例	+MESHMSGRX:6,820201624100	成功接收到应用层透传数据包：820201624100， 通过串口转发

设备事件通知

描述	通过 AT 通知蓝牙 mesh 模组的设备事件	
命令	+MESHEVT:<state>	<state>为设备事件，取值范围： 0x04:设备启动 0x00:配网成功 0x05:配网失败
示例	+MESHEVT:0x04	设备启动事件通知

获取设备 MAC 地址

描述	获取设备 MAC 地址	
命令	AT+MESHMAC?	
响应 1	+MESHMAC:Y1 OK	获取成功，返回结果 Y1 Y1 为设备 MAC 地址
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHMAC? +MESHMAC: 28:fa:7a:33:d9:cf OK	成功获取设备 MAC 地址：28:fa:7a:33:d9:cf

设置串口日志输出开关

设置串口日志输出开关		
描述	控制串口日志输出开关	
命令	AT+MESHLOG=<mode>	<mode>为日志输出开关，取值范围： 0 - 关闭串口日志输出 1 - 开启串口日志输出 默认为关闭，且设置后断电不保存
响应 1	OK	设置成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHLOG=1 OK	设置开启串口日志输出功能

获取串口日志输出开关

获取串口日志输出开关		
描述	获取串口日志输出开关	
命令	AT+MESHLOG?	获取成功，返回结果 Y1，表示设备当前串口日志输出，返回值范围： 0 - 表示已关闭串口日志输出 1 - 表示已开启串口日志输出
响应 1	OK	设置成功
响应 2	+ERROR:Y1	返回错误
示例	AT+MESHLOG? +MESHLOG:0 OK	已关闭串口日志输出

10 二进制串口协议定义

如果在应用固件中使能 CONFIG_GENIE_MESH_BINARY_CMD 宏，可以使用二进制串口协议与设备交互，参考以下说明。

10.1 串口通信协议

串口设置

波特率 9600bps，8 位数据位，无奇偶校验，1 位停止位。

指令帧结构

帧构成	占用字节数	备注
数据类型	2	0xFF00：mesh 透传数据 0xFE00：控制指令 0x5555：错误码返回
数据长度	2	
数据内容/参数	1-N	
校验码	1	累加求和，计算范围包括数据长度和数据内容

注：每条指令之间间隔至少 60ms。当设备主动发送指令时，必须等待接收到模块的回复指令后才可以发送下一条指令。

10.2 控制指令

	控制指令	参数
0x01	控制蓝牙 mesh 广播	0x00:关闭 0x01:开启 mesh 广播 0x02:开启静默广播
0x02	清除配网信息	无
0x03（这个是模组的返回数据）	设备事件/设备信息	0x00:配网完成 0x01:清除配网信息（0x01 成功，0x00 失败） 0x02+MAC 地址+RSSI:对应 MAC 地址的信号强度 0x03+配网状态:设备配网状态（已配网为 0x01，未配网为 0x00） 0x04:设备启动 0x05:配网失败

		0x06+版本号:模组固件版本号 0xFF:设备错误
0x04	设置获取蓝牙信号强度	0x00+MAC 地址:获取设备扫描对应 MAC 地址的信号强度
0x05	查询设备信息	0x00:设备配网状态 0x01:模组版本号
0x06	重启设备	无
0x07	更新组播地址	8 字节，每 2 个字节组成一个组播地址，小端格式，有效的组播地址要求最高 2 位为 1
0x08	更新组播地址返回	更新组播地址的返回指令。 0x00:更新组播地址成功 0x01:更新组播地址失败
0x09	读取组播地址	无
0x0A	读取组播地址返回	byte0: 0x00:读取组播地址成功。 0x01:读取组播地址失败。 byte1-byte8: 8 字节，每 2 个字节组成一个组播地址，小端格式。
0x0B	设置进入产测模式	无参数。 回复： 成功：0x0B 0x00 失败：0x0B 0x01
0x0C	开关串口日志输出功能	0x01：允许输出串口日志 0x00：禁止输出串口日志 回复 成功：0x0C 0x00 失败：0x0C 0x01

10.3 透传数据示例

上报开关属性（开）

0xFF	0x00	0x00	0x07	0xd4	0xa8	0x01	0xff	0x00	0x01	0x01	0x85
mesh 透传数据		数据长度		opcode：d401a8			tid	0x0100 开关属性		开关开	求和校验

上报开关属性（关）

0xFF	0x00	0x00	0x07	0xd4	0xa8	0x01	0x09	0x00	0x01	0x00	0x8E
mesh 透传数据		数据长度		opcode：d401a8			tid	0x0100 开关属性		开关关	求和校验

注意：详细的数据格式定义参考蓝牙Mesh设备扩展协议。

10.4 控制指令示例

打开蓝牙广播

0xFE	0x00	0x00	0x02	0x01	0x01	0x04
控制指令		数据长度		控制蓝牙 mesh 广播	开启 mesh 广播	求和校验

清除配网信息

0xFE	0x00	0x00	0x01	0x02	0x03
控制指令		数据长度		清除配网信息	求和校验

测试模式获取信号强度

0xFE	0x00	0x00	0x08	0x04	0x00	0x11	0x22	0x33	0x44	0x55	0x66	0x70
控制指令		数据长度		测试模式	获取信号强度	MAC 地址						求和校验

测试模式信号强度

0xFE	0x00	0x00	0x09	0x03	0x02	0x11	0x22	0x33	0x44	0x55	0x66	0x10	0x84
控制指令		数据长度		设备信息	信号强度	MAC 地址						RSSI	求和校验

查询设备配网状态

0xFE	0x00	0x00	0x02	0x05	0x00	0x07
控制指令		数据长度		查询设备信息	配网状态	求和校验

更新组播地址

0xFE	0x00	0x00	0x09	0x07	0x12	0xc1	0x00	0xc0	0x11	0xc1	0x75
控制指令		数据长度		更新组播地址	组播地址						求和校验

更新组播地址返回

0xFE	0x00	0x00	0x02	0x08	0x00	0x0a
控制指令		数据长度		更新组播地址状态	更新状态	求和校验

读取组播地址

0xFE	0x00	0x00	0x01	0x09	0x0a
控制指令		数据长度		读取组播地址	求和校验

读取组播地址返回

0xFE	0x00	0x00	0x0a	0x0a	0x00	0x12	0xc1	0x00	0xc0	0x11	0xc1	0x00	0x00	0x79

控制指令	数据长度	读取组播地址返回	返回状态	组播地址	求和校验

10.5模组回复错误码格式与定义

错误码数据格式

0x55	0x55	0x00	0x01	XX	XX
控制指令		数据长度		错误码	校验和

错误码定义

如上所示第五个字节为错误码，具体错误码内容定义如下：

错误码	含义	备注
0x01	数据总长度小于五个字节	数据总长度错误
0x02	数据长度错误	解析 len 的值与数据总的长度不一致
0x03	校验和错误	
0x04	获取 RSSI 的输入有误	
0x05	获取 RSSI 失败	通常是 BLE 错误
0x06	获取 RSSI MAC 不匹配	
0x07	更新组播地址长度错误	
0x08	包含无效的组播地址	组播地址范围在 C000-CFFF 之间，字节顺序是低字节在前面
0x09	更新组播地址 Flash 错误	
0x0A	启动产测模式失败	
0x80	透传发送失败	
0xFF	未知错误	