阿里云开发者社区 | 芯片开放社区 Open Chip Community

无需从0开发

平头哥教你 1天上手蓝牙Mesh应用方案

- 业内性能领先的Mesh协议栈
- 解决多场景配网难题
- 支持天猫精灵生态

藏经阁

阿里云开发者电子书系列

平头哥OCC钉钉交流群，
进群为你答疑解惑

平头哥芯片开放社区公众号，
扫码关注获取更多信息与资料

扫码注册平头哥OCC官网，
观看各类蓝牙视频及课程

阿里云开发者“藏经阁”
海量免费电子书下载

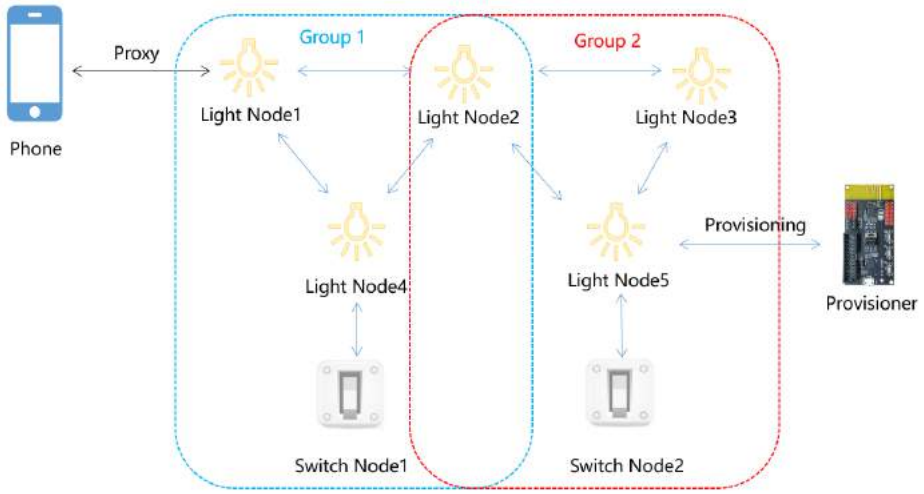
目录

蓝牙 Mesh 网络及 SDK 概述

1. 概述

蓝牙 Mesh SDK 是基于低功耗蓝牙芯片 PHY6212 提供的软件开发套件。该开发套件以 YoC 平台为基础，对蓝牙 MESH 协议栈做了深度优化和整合，为开发者提供了通用的 MESH 组件，涵盖了 SIG MESH Model 和私有 Model 以及丰富的芯片外设驱动。

1.1 Mesh 网络介绍



上图是一个 Mesh 灯控网络的拓扑，以此为例，介绍一下 Mesh 网络的构成。

按照功能来划分，Mesh 设备可以分为两类，一种是 Provisioner，一种是 Node 节点。

Provisioner 负责组建 Mesh 网络，主要功能有发现未入网设备，将未入网的设备加入

Mesh 网络，配置入网设备的特性，比如 Relay 特性，Friend 特性，Proxy 特性等。在上图中，Provisioner 可以是蓝牙开发板，也可以是一个手机，但是当前不支持两个 Mesh 网络中同时存在手机 Provisioner 和蓝牙开发板 Provisioner 的情况。

当一个设备加入特定的 Mesh 网络后，该设备成为 Mesh 网络的 Node 节点。在上图中，节点有 Light 灯控节点和 Switch 开关节点两种。这两种节点默认都支持 Relay 特性和 Proxy 特性。Relay 特性打开的情况下，节点会转发来自别的节点的 Mesh 消息。Proxy 特性打开的情况下，节点会支持手机接入 Mesh 网络。

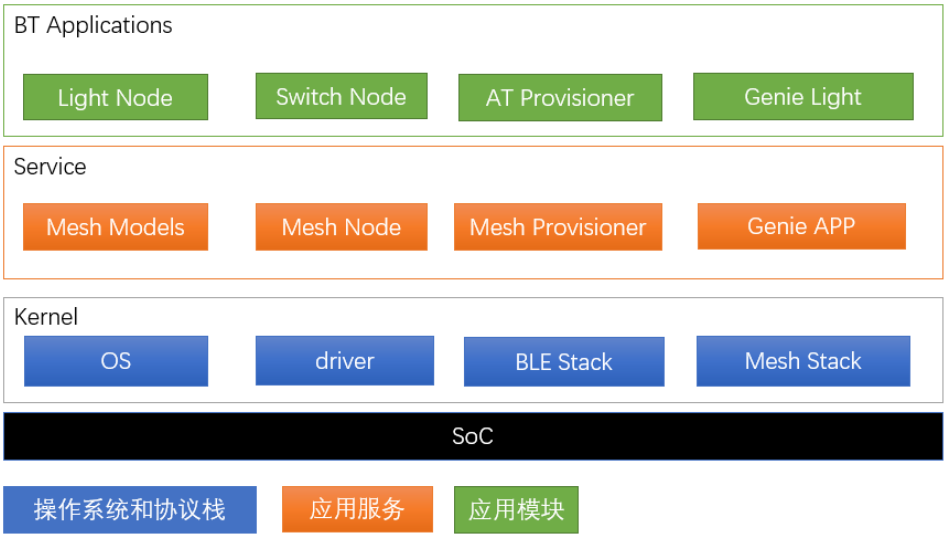
在上图中，存在 Group1 和 Group2 两个分组，这个也是 Mesh 网络的一个重要特性，Mesh 网络支持将不同的节点分配到相同的组中，同一个节点也可以同时存在于两个组中。上图 Light Node1，Light Node2，Light Node4，Swtich Node1 位于 Group1，3 个 Light Node 可以被 Switch Node1 控制。Light Node2，Light Node3，Light Node5，Swtich Node2 位于 Group2，3 个 Light Node 可以被 Switch Node2 控制。而 Light Node2 同时位于 Group1 和 Group2，它可以被 Switch Node1 和 Swicth Node2 控制。

在蓝牙 Mesh SDK 中，以上的特性均已实现，开发者可以以下的介绍了解到如何实现一个这样的 Mesh 网络。

2. SDK 介绍

2.1 架构介绍

蓝牙 Mesh SDK 总体分为四个层次，自下而上分别为 SoC 蓝牙芯片层，Kernel 内核层，Service 服务层和 BT Applications 蓝牙应用层。



- Kernel

  内核层，包含标准的 BLE（5.0）/MESH（1.0）协议栈，Ali OS 操作系统，芯片驱动。

- Service

  蓝牙服务层

  Mesh Model 组件，其中支持的 Sig Models 有 Generic Level Model，Generic Onoff Model，，Light Level Model，Light CTL Model 等，此外还有一个 Vendor Model，支持数据透传。

  Mesh Node 组件，提供 Mesh 节点入网和配置功能，开发者可以基于该组件开发 Mesh 节点应用。

  Mesh Provisioner 组件，支持 MESH 节点管理，包括新增 Mesh 节点，删除 Mesh 节点，获取节点信息等功能。

  Genie APP 组件，天猫精灵 Mesh 组件，支持天猫精灵网络协议。

- BT Applications

  蓝牙应用层，包含灯控节点，开关节点，Provisioner AT 模组和天猫精灵灯控节点。

2.2 目录介绍

下面是蓝牙 Mesh SDK 的目录结构，表格中介绍了各个目录的功能。

文件夹名称	内容描述
applications	应用示例目录
--bluetooth	蓝牙示例
----mesh*light*node	Mesh 灯控示例
----mesh*switch*node	Mesh 开关示例
----genie_light	天猫精灵灯控示例
----at*mesh*provsioner	基于 AT 命令实现的 Mesh Provisioner 示例
--driver_examples	芯片驱动示例
--kernel_example	内核接口示例
boards	板级配置信息
csi	芯片驱动
defconfigs	SDK 配置文件目录
--defconfig*ch6121*evb*mesh*node	Mesh 节点配置文件，用于灯控和开关等节点应用
--defconfig*ch6121*evb*mesh*at_provisioner	Provisioner 配置文件，用于 at*mesh*provsioner
defconfig*ch6121*evb*genie*mesh	天猫精灵 MESH 配置文件，用于天猫精灵方案
drivers	外设驱动
include	SDK 头文件
kernel	YoC 框架核心，包括 BLE Host, Misc, KV 文件系
libs	外部引用库文件
modules	SDK 组件，
tools	系统工具，包括编译脚本和一些开发小工具

2.3 Mesh 协议栈特性介绍

Mesh Profile v1.0 特性支持

- Provisioner role(Advertising bearer and GATT bearer)
- Node role(Advertising bearer and GATT bearer)

- Relay feature
- Proxy feature (proxy server)
- Foundation Models

  Config server and client

  Health server and client

Mesh Models 支持

Models	Client	Server	说明
Generic OnOff	√	√	通用开关
Generic Level	√	√	通用等级
Light Lightness	√	√	灯亮度
Light Lightness Setup	–	√	灯亮度
Light CTL	√	√	灯色温控制
Light CTL Temperature	–	√	灯色温控制
Light CTL Setup	–	√	灯色温控制
Vendor Model(0x01A800)	√	√	透传 自动配网

3. Mesh 组件

本章节介绍 SDK 中包含的几个关键 Mesh 组件的功能和使用方法。

- Mesh Node 组件
- Mesh Models 组件
- Genie APP 组件

3.1 Mesh Node

Mesh Node 支持 Mesh 节点配置和 Model 消息的接收功能。

组件的主要 API 如下表：

API	说明
ble*mesh*node_init	Mesh Node 组件的初始接口
ble*mesh*node*OOB*input_num	配网过程中，要求输入数字的接口
ble*mesh*node*OOB*input_string	配网过程中，要求数据字符串的接口

3.1.1 示例代码

- 初始化

  初始化 Node 节点，主要配置节点的设备名称，UUID，事件的回调。

```
/* 定义设备名称，使用手机 APP 扫描时可以看到该名称 */
#define DEVICE_NAME "YoC Light NODE"

/* 定义设备 UUID */
#define LIGHT_DEV_UUID {0xcf, 0xa0, 0xe3, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86,
0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0x02}

node_config_t g_node_param = {
    /* 设置当前为设备节点 */
    .role = NODE,
    /* 设置节点的 UUID */
    .dev_uuid = LIGHT_DEV_UUID,
    /* 设置节点的 Device Name */
    .dev_name = DEVICE_NAME,
    /* 注册 Model 的事件处理回调函数 */
    .user_model_cb = app_event_cb,
    /* 注册 Provision 的事件处理回调函数 */
    .user_prov_cb = app_prov_event_cb,
    /* RSSI 上报功能使能标识，默认关闭 */
    .rssi_report_enable = 0,
};

/* MESH 节点初始化，设置节点 UUID、Device Name 并注册 Model 事件回调函数 */
ret = ble_mesh_node_init(&g_node_param);
```

- 回调处理

  Node 节点在配网或者复位过程中会产生如下事件，开发者应当在 userprovcb 回调中处理

Event	说明
BT*MESH*EVENT*NODE*REST,	节点复位事件
BT*MESH*EVENT*NODE*PROV_COMP	节点入网成功事件
BT*MESH*EVENT*NODE*OOB*INPUT*NUM	配网过程中，要求输入数字
BT*MESH*EVENT*NODE*OOB*INPUT*STRING	配网过程中，要求输入字符串

```
void app_prov_event_cb(mesh_prov_event_en event, void *p_arg)
{
    switch (event) {
        /* 配网成功，亮绿灯 */
        case BT_MESH_EVENT_NODE_PROV_COMP: {
            if (p_arg) {
                mesh_node_local_t *node = (mesh_node_local_t *)p_arg;
                LOGI(TAG, "prov complete %04x", node->prim_unicast);
                prov_succeed_flag = 1;
                led_control(led_dev, COLOR_GREEN, -1, -1);
            }
        }
        break;
              /* 节点复位，绿灯闪烁 */
        case BT_MESH_EVENT_NODE_REST : {
            LOGI(TAG, "node reset");
            led_control(led_dev, COLOR_GREEN, 200, 200);
        }
        break;

        /* 配网要求输入数字，通过其他方式比如按键，获取数字后，调用 ble_mesh_node_OOB_
input_num 接口输入 */
        case BT_MESH_EVENT_NODE_OOB_INPUT_NUM : {
            if (p_arg) {
                LOGI(TAG, "oob input num size:%d", *(uint8_t *)p_arg);
            }
        }
        break;
        /* 配网要求输入字符串，通过其他方式比如按键，获取字符串后，调用   ble_mesh_
node_OOB_input_string 接口输入 */
        case BT_MESH_EVENT_NODE_OOB_INPUT_STRING : {
            LOGI(TAG, "oob input string size:%d", *(uint8_t *)p_arg);
        }
        break;

        default:
            break;
    }

}
```

3.2 Mesh Models

Mesh Models 组件提供 Generic Onoff，Generic Level，Light Lightness，Light CTL 的 Server/Client Models，此外还有一个 Vendor Model。

Mesh Model 的 API 较多，下表列出几个主要的，其他的请参考《蓝牙 Mesh SDK API 说明》。

Event	说明
ble*mesh*model_init	Mesh Model 组件的初始接口
ble*mesh*model*get*comp_data	获取 Mesh Model 组件的 Models 结构
ble*mesh*model*set*cb	设置 Mesh Model 组件的回调函数
ble*mesh*model_find	查找指定的 Mesh Model
ble*mesh*model*status*get	获取指定 model 的状态

3.2.1 示例代码

- 初始化

  初始化 Mesh Model 组件，主要配置需要使用的 Models。

```
/* 定义设备的 Models */
static struct bt_mesh_model elem0_root_models[] = {
    /* Configuration Server Model, 必选 Model */
    BT_MESH_MODEL_CFG_SRV_NULL(),
    /* Health Server Model, 可选 Model */
    BT_MESH_MODEL_HEALTH_SRV_NULL(),
    /* Generic OnOff Server Model */
    BT_MESH_MODEL_GEN_ONOFF_SRV_NULL(),
};

/* 定义设备的 Composition data */
static const struct bt_mesh_comp mesh_comp = {
    .cid = 0x01A8, //厂商 ID
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};

/* MESH Model 组件初始化 */
ret = ble_mesh_model_init(&mesh_comp);

/* Mesh Model 回调函数注册，需要注意如果使用 Mesh Node 组件，不需要注册该回调，通过 Node 组件的中的 user_model_cb 回调获取 Model 信息 */
ble_mesh_model_set_cb(app_event_cb);
```

- 回调处理

  Mesh Model 组件根据使用的 Models 不同会产生不同的事件回调，下面是一个灯控的设置的事件。

```
/*
  设备 Models 定义完成后，将通过回调函数返回相关 Models 的事件，应用需要处理相关事件。
  以 Generic OnOff Server Modle 为例，
*/
void app_event_cb(mesh_model_event_en event, void *p_arg)
{
    switch (event) {
        /* Generic OnOff Server Model 的 SET 事件，
           说明 Generic OnOff Client 来设置开关状态 */
        case BT_MESH_MODEL_ONOFF_SET: {
            if (p_arg) {
                model_message message = *(model_message *)p_arg;
                S_ELEM_STATE *elem_state = (S_ELEM_STATE *)message.user_data;
                                /* 解析数据，记录开关状态，并控制 LED 灯的开和关 */
                if (elem_state->state.onoff[T_TAR]) {
                    /* 开灯 */
                    led_set_status(led2, true);
                } else {
                    /* 关灯 */
                    led_set_status(led2, false);
                }
            }

        }
        break;
        default:
            break;
    }
}
```

3.3 Genie APP

天猫精灵组件，支持天猫 MESH Models，天猫精灵的接入，三要素写入 / 存储 / 获取功能。

组件的主要 API 如下表：

API	说明
genie_init	Genie APP 组件初始化

3.3.1 示例代码

- 初始化

  初始化天猫精灵组件

```
/* 初始化 */
genie_init();
```

- 处理函数

  以下内容需要在应用层定义和实现，可以参考 applications\bluetooth\genie_light\src\light.c 中实现。

```
/* 定义使用的 Models */
static struct bt_mesh_model element_models[] = {
    MESH_MODEL_CFG_SRV_NULL(),
    MESH_MODEL_HEALTH_SRV_NULL(),

#ifdef CONFIG_MESH_MODEL_GEN_ONOFF_SRV
    MESH_MODEL_GEN_ONOFF_SRV(&g_elem_state[0]),
#endif

#ifdef CONFIG_MESH_MODEL_GEN_LEVEL_SRV
    MESH_MODEL_GEN_LEVEL_SRV(&g_elem_state[0]),
#endif

#ifdef CONFIG_MESH_MODEL_LIGHTNESS_SRV
    MESH_MODEL_LIGHTNESS_SRV(&g_elem_state[0]),
#endif

#ifdef CONFIG_MESH_MODEL_CTL_SRV
    MESH_MODEL_CTL_SRV(&g_elem_state[0]),
#ifndef CONFIG_ALI_SIMPLE_MODLE
    MESH_MODEL_CTL_SETUP_SRV(&g_elem_state[0]),
#endif
#endif

};

/* 定义使用的 Vendor Models */
static struct bt_mesh_model g_element_vendor_models[] = {

#ifdef CONFIG_MESH_MODEL_VENDOR_SRV
    MESH_MODEL_VENDOR_SRV(&g_elem_state[0]),
#endif
```

```
};

struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, element_models, g_element_vendor_models, 0),
};

// 设置广播时间
uint32_t get_mesh_pbadv_time(void)
{
    return MESH_PBADV_TIME*1000;
}

/* 返回 vendor model element 的数量 */
uint8_t get_vendor_element_num(void)
{
    return MESH_ELEM_COUNT;
}

/* 设置默认绑定地址 */
void mesh_sub_init(u16_t *p_sub)
{
    uint16_t sub_list[CONFIG_BT_MESH_MODEL_GROUP_COUNT];
    memset(sub_list, 0, sizeof(sub_list));

#ifdef DEFAULT_MESH_GROUP1
    sub_list[0] = DEFAULT_MESH_GROUP1;
#endif

#ifdef DEFAULT_MESH_GROUP2
    sub_list[1] = DEFAULT_MESH_GROUP2;
#endif

    memcpy(p_sub, sub_list, sizeof(sub_list));
}

/* 入网成功事件 */
void user_prov_complete(u16_t net_idx, u16_t addr)
{
    //flash 3 timers
}

/* 节点复位事件 */
void user_prov_reset(void)
{

}

/* 用户数据初始化 */
void user_init()
```

```
{

}

/* 用户处理事件回调 */
void user_event(E_GENIE_EVENT event, void *p_arg)
{
    E_GENIE_EVENT next_event = event;

    switch(event) {
        case GENIE_EVT_SW_RESET:
        case GENIE_EVT_HW_RESET_START:
            ....
            break;

        case GENIE_EVT_SDK_ACTION_DONE:
        {
            elem_state_t *p_elem = (elem_state_t *)p_arg;
#if defined(CONFIG_MESH_MODEL_CTL_SRV)
            /* 色温灯控对接 */
            _led_set(p_elem->elem_index, p_elem->state.onoff[T_CUR], p_elem-
>state.actual[T_CUR], p_elem->state.temp[T_CUR]);
#elif defined(CONFIG_MESH_MODEL_LIGHTNESS_SRV)
            /* 亮度灯控对接 */
            _led_set(p_elem->elem_index, p_elem->state.onoff[T_CUR], p_elem-
>state.actual[T_CUR]);
#elif defined(CONFIG_MESH_MODEL_GEN_ONOFF_SRV)
            /* 简单开关灯控对接 */
            _led_set(p_elem->elem_index, p_elem->state.onoff[T_CUR]);
#endif
            if(event == GENIE_EVT_SDK_ACTION_DONE)
                save_light_state(p_elem);
            break;
        }
        .....
    }

    if(next_event != event) {
        genie_event(next_event, p_arg);
    }

}

/* 天猫精灵 Vendor Models 对接 */
u16_t vendor_model_msg_handle(vnd_model_msg *p_msg)
{
    .....
    switch (p_msg->opid) {
        case VENDOR_OP_ATTR_GET_STATUS:
```

```
            /* report VENDOR_OP_ATTR_STATUS */
            //_light_report_status();
            break;
        case VENDOR_OP_ATTR_SET_ACK:
            /* TODO: set status
             * report VENDOR_OP_ATTR_STATUS
             * */
            //_light_report_status();
            break;
        case VENDOR_OP_ATTR_SET_UNACK:
            /* TODO: set status */
            break;
        case VENDOR_OP_ATTR_CONFIME:
            /* clear indicate status */

            break;
        case VENDOR_OP_ATTR_TRANS_MSG:
            break;
        default:
            break;
    }

    return 0;
}
```

- 三要素写入

  Genie APP 组件需要用到三要素才能接入天猫精灵，三要素的获取请参考天猫精灵开发者网站。

  在 Genie APP 内部有一个调试三要素，位于 modules\genieapp\base\triple_default.h。

  Genie APP 组件支持通过 CLI 命令写入和读取三要素。

命令如下：

```
/* 设置三要素 */
set_tt <product id> <key> <mac address>
/* 获取三要素 */
get_tt
```

4. 应用开发

4.1 开发环境

环境搭建和烧录方法，请参考《CB6121 快速上手手册》。

4.2 Mesh 灯控

Mesh 灯控示例请参考《MESH 灯控开发指南》。

4.3 Mesh 开关

Mesh 开关示例请参考《MESH 开关开发指南》。

5. AT Provisioner

Mesh SDK 中提供一个了 AT Provisoner 模组解决方案，开发者可以使用该方案对其他 Mesh 节点进行配网和控制。

相关的 AT 命令和使用方法，请参考《MESH 配网模块用户手册》。

6. API 说明

SDK API 请参考《蓝牙 Mesh SDK API 说明》。

蓝牙 Mesh SDK 快速上手

1. 简介

本文介绍如何使用 CB6121 开发板进行 MESH Light 的功能演示。通过本文的指引，开发者可以学会开发环境的搭建、SDK 的编译与烧录、基本调试方法，快速上手 CB6121 的开发。

2. 开发环境搭建

2.1 准备

栏目	名称	版本	说明
SDK 包	blemeshsdk_v1.x.x	V1.x.x	从 https://occ.t-head.cn/ 获取
烧录软件	PhyPlusKit.zip	2.3.7	从 https://occ.t-head.cn/ 获取
工具链	gcc-arm-none-eabi-8-2018-q4-major-linux.tar.bz2	8.2.1 20181213	工具链下载
仿真器驱动	JLinkWindowsV620f.exe	V6.20f	JLink 仿真器驱动下载
开发板	CB6121 开发板	–	–
仿真器	JLink V9	–	自备
串口驱动	CP210xUniversalWindows_Driver	–	串口驱动下载
nRF Mesh	NRF Mesh APP	v1.2.1 及以上	IOS 可以从 APP Store 获取 安卓版本可以 Github 获取

2.2 Linux 开发环境搭建

- Linux 环境搭建

  Win10 用户，建议前往应用商店下载安装 Ubuntu18.04 LTS；其他用户可自行

选择安装 Linux 或者虚拟机。

- 工具链安装

```
$ tar -jxvf gcc-arm-none-eabi-8-2018-q4-major-linux.tar.bz2
```

- 环境变量中添加工具链路径并使其立即生效，其中 toolchain_path 为工具链解压的目录

```
$ vi ~/.bashrc
...
PATH={toolchain_path}/bin:$PATH
...
$ source ~/.bashrc
```

- 验证工具链是否安装成功

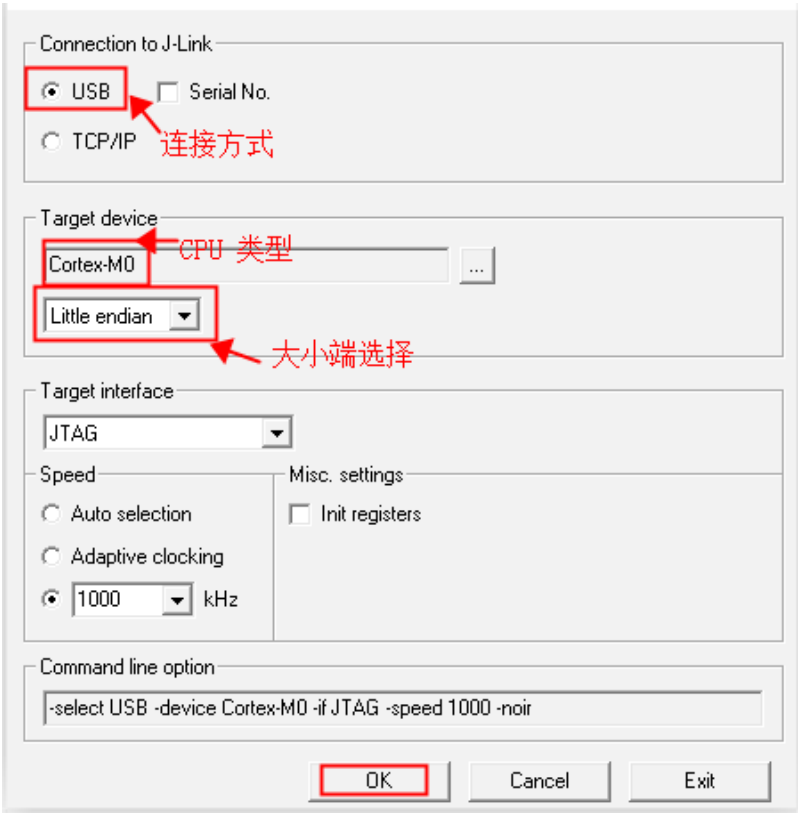
```
$ arm-none-eabi-gcc -v
...
gcc version 8.2.1 20181213 (release) [gcc-8-branch revision 267074] (GNU
Tools for Arm Embedded Processors 8-2018-q4-major)
```

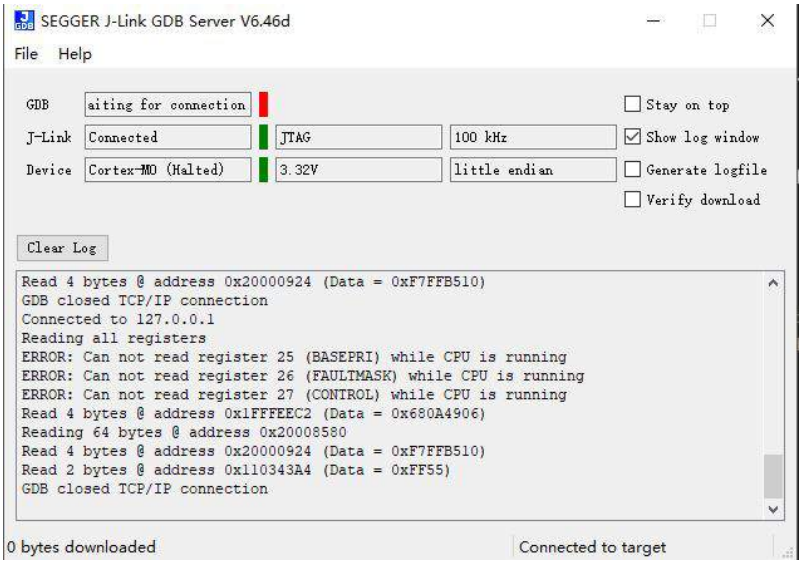
- 安装 make 工具

```
$ sudo apt-get install make
```

2.3 仿真器驱动安装

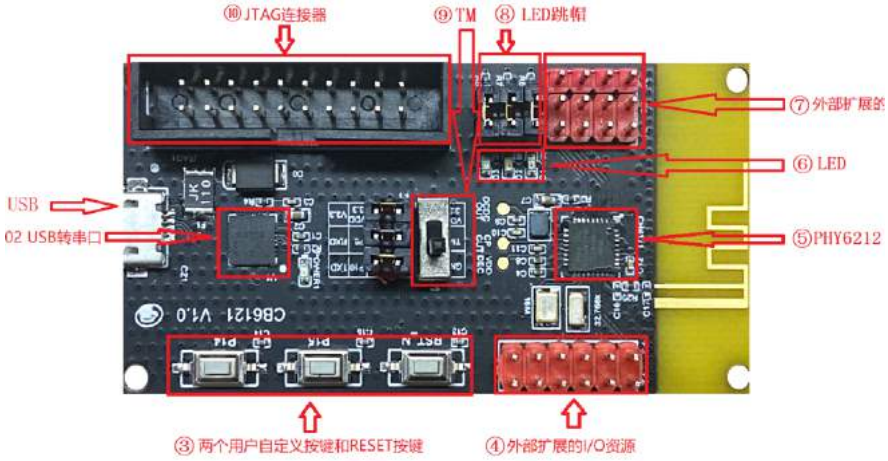
- 双击 JLink_Windows_V620f.exe。
- 点击下一步直到安装完成。
- 打开 JLink GDB Server, 选择 Target Device 为 Cortex-M0。


Connection to J-Link
USB
Serial No.
TCP/IP
连接方式
Target device
Cortex-M0
CPU 类型
...
Little endian
大小端选择
Target interface
JTAG
Speed
Auto selection
Adaptive clocking
1000
kHz
Misc. settings
Init registers
Command line option
-select USB -device Cortex-M0 -if JTAG -speed 1000 -noir
OK
Cancel
Exit


SEGGER J-Link GDB Server V6.46d
File Help
GDB aiting for connection
J-Link Connected JTAG 100 kHz
Device Cortex-M0 (Halted) 3.32V little endian
Stay on top
Show log window
Generate logfile
Verify download
Clear Log
Read 4 bytes @ address 0x20000924 (Data = 0xF7FFB510)
GDB closed TCP/IP connection
Connected to 127.0.0.1
Reading all registers
ERROR: Can not read register 25 (BASEPRI) while CPU is running
ERROR: Can not read register 26 (FAULTMASK) while CPU is running
ERROR: Can not read register 27 (CONTROL) while CPU is running
Read 4 bytes @ address 0x1FFFEEC2 (Data = 0x680A4906)
Reading 64 bytes @ address 0x20008580
Read 4 bytes @ address 0x20000924 (Data = 0xF7FFB510)
Read 2 bytes @ address 0x110343A4 (Data = 0xFF55)
GDB closed TCP/IP connection
0 bytes downloaded
Connected to target

2.4 开发板准备

CB6121 开发板资源介绍



序号	资源	说明
1	USB	供电，串口输入输出
2	CP2102USB 转串口	USB-UART 转换芯片
3	按键	P14/P15 普通 GPIO 按键，RST_N 复位按键
4	外部扩展的 I/O 资源	GPIO 引脚，定义见开发板背部
5	PHY 6212 BLE SoC	按下该键可重新复位 PHY6212 芯片
6	LED	3 个 LED 灯
7	外部扩展的 I/O 资源	GPIO 引脚，定义见开发板背部
8	LED 跳帽	LED 灯跳帽，去掉后断开 LED 和 GPIO 连接
9	TM	烧录模式选择开关，正常模式拨至 GND，烧录模式拨至 VDD
10	JTAG 连接器	JTAG 调试口

3. 编译烧录及调试

3.1 编译

- 例程编译方法

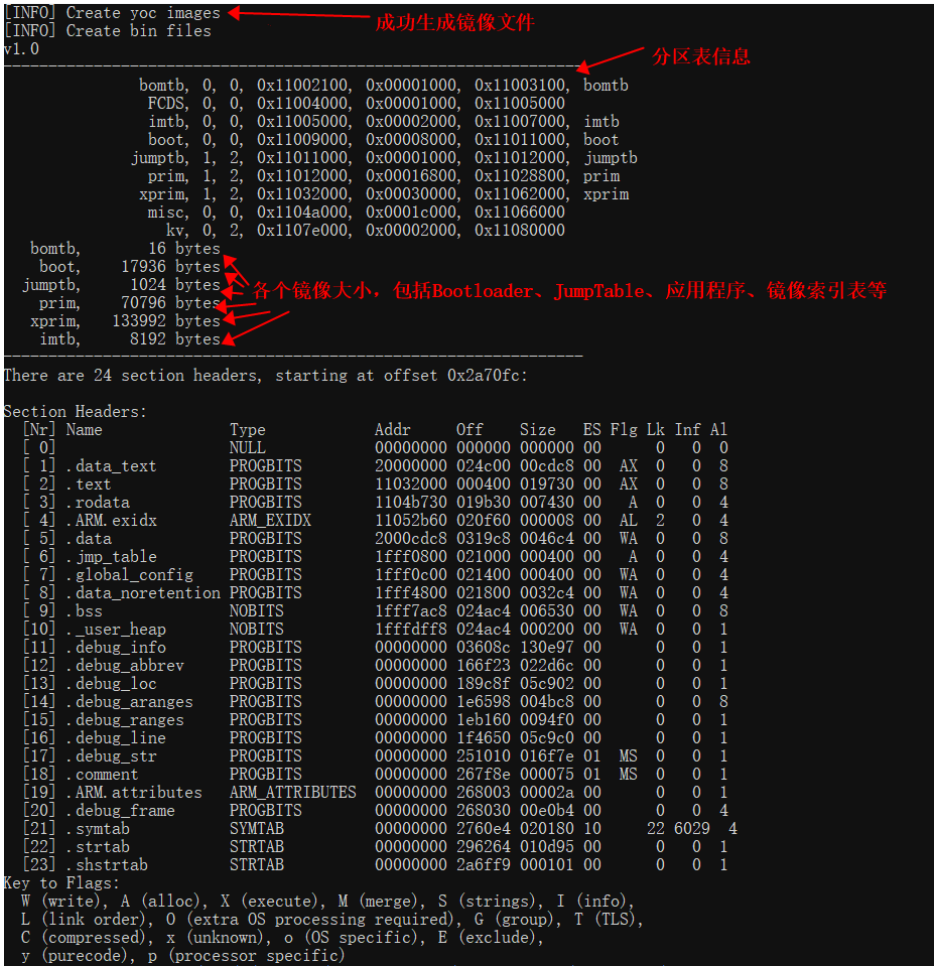
  使用 build.sh 脚本编译，命令格式为 ./build.sh <SDK 配置文件 > < 示例目录 > [多线程数]

```
$ make clean
$ ./build.sh defconfigs/defconfig_ch6121_evb_mesh_node applications/
bluetooth/mesh_light_node/ j64
```

- 编译后固件位于 applications/bluetooth/meshlightnode/generated

```
$ ll applications/bluetooth/mesh_light_node/generated
......
-rwxrwxrwx 1 xxx xxx 652920 Mar 13 19:03 total_image.hexf*
....
```

- 编译成功将输出如下图所示信息

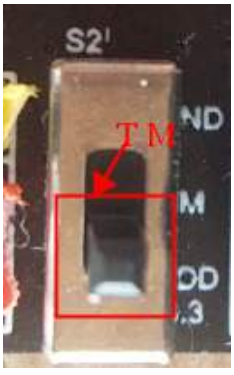


- 各个示例对应配置文件说明

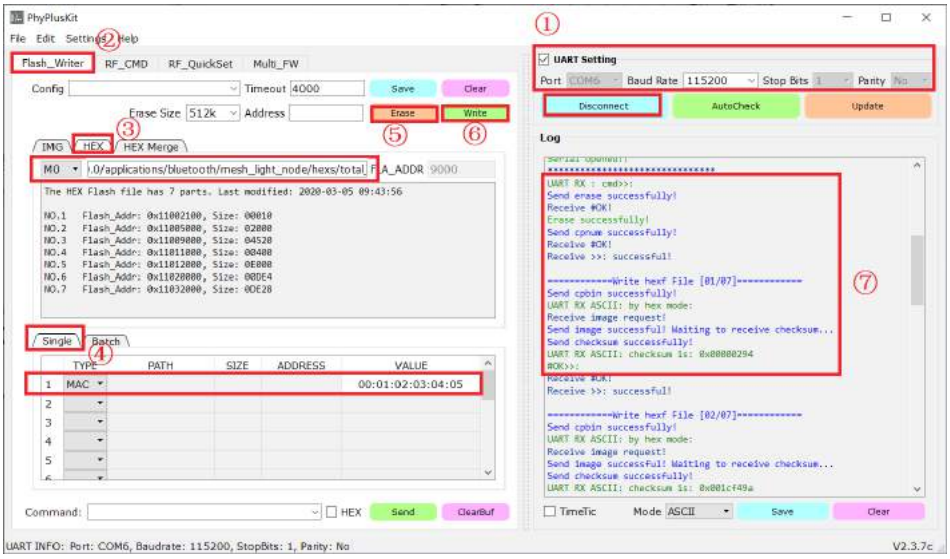
示例	SDK配置文件	示例目录
灯控示例	defconfigch6121evbmeshnode	applications/bluetooth/meshlightnode/
开关示例	defconfigch6121evbmeshnode	applications/bluetooth/meshswitchnode
AT Provisioner 示例	defconfigch6121evbmeshat_provisioner	applications/bluetooth/at-meshprovsioner
Provisioner 示例	defconfigch6121evbmeshat_provisioner	applications/bluetooth/mesh_provsioner
天猫精灵灯示例	defconfigch6121evbgeniemesh	applications/bluetooth/genie_light

3.2 镜像烧录

- 打开烧写工具 PhyPlusKit.exe。
- 勾选 UART Setting，选择开发板串口，串口配置为波特率：115200，停止位：1，校验：NO。
- 点击 Connect, 连接串口。
- 选择 Flash_writer 标签页。
- 选择 HEX 烧入方式标签页。
- 双击选择 applications/bluetooth/meshlightnode/generated/total_image.hexf。
- 下方选择 Single 标签，TYPE 选择 MAC，VALUE 填写 MAC 地址。
- 将拨码开关拨到 VDD TM 。



- 按开发板上的 RESET 按键，重启开发板，串口打印 UART RX : cmd>>。
- 点击 Erase 擦除。
- 点击 Write 烧写。
- LOG 区域显示烧录过程。



3.3 GDB 调试

- 打开 JLink GDB Server，连接开发板
- 编辑 Linux 环境中配置 GDB 环境变量并保存后退出 , 其中 2331 为默认端口号

```
$ cd applications/bluetooth/mesh_light_node
$ vi .gdbinit
target remote 127.0.0.1:2331
```

- 运行 GDB，开始调试，调试镜像为 yoc.elf

```
$ arm-none-eabi-gdb yoc.elf -x .gdbinit
```

- 若出现无法连接或长时间无响应，请确认

  工程目录下是否存在 .gdbinit 文件

JLink GDB Server 是否成功连接开发板

.gdbinit 文件中的 IP 地址是否正确

- 常用 GDB 命令

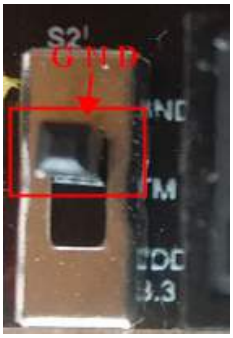
常用 GDB 命令	说明
hb	设置硬断点
watch	监测某个变量或内存地址的状态，发生变化时程序将暂停运行
next（n）	单步执行
stepin（s）	单步进入
continue（c）	继续执行
print（p）	查看变量
x /	查看内存地址中的值，比如 x /2wx 0x20000000, 查看内存地址为 0x20000000 的 2 WORD 的值，以十六进制输出
set	设置变量、内存以及寄存器值
mon reset	执行复位指令
d	清除所有断点
target remote IP:Port	连接调试器
file *.elf	装载符号表
i r	查看寄存器值

4. 例程运行

基于 CB6121 开发板的灯控示例工程

- 连接串口调试工具 , 配置为

  波特率 : 115200, 数据位 : 8, 校验位 : None, 停止位 : 1, 流控 : None



- 开发板拨码开关拨至 GND
- 按 RESET 按键，复位开发板
- MESH Light 启动 Log 如下

```
[     0.004000][I][init    ]Build:Mar 24 2020,21:52:51
[     0.017000][I][init    ]find 9 partitions
Welcome to CLI...
> [     0.738000][I][DEMO    ]Mesh light node demo
```

- 关于灯控配网请参考《蓝牙 Mesh 灯控开发指南》

蓝牙 Mesh 标准 Model 开发指南

1. 目标

PHY6212 蓝牙 Mesh SDK 的 MESH Models 组件，集成了 Generic OnOff Model、Generic Level Model、Generic Lightness Model、Light CTL Model、以及 Vendor Model。本文旨在指导用户使用 eneric OnOff Model 开发 MESH 灯控应用。

2. 步骤

使用 MESH Models 组件只需要三步即可完成智能灯设备的 MESH 网络接入及控制。

- 定义设备 Models，即定义设备的功能
- 设置设备 UUID、设备名称，并注册 Models 事件回调函数
- 处理相关 Models 事件
- LED 驱动

2.1 定义 Models

MESH 灯的主要功能为开关灯，我们需要配置好相关 Models。

文件路径：

```
applications\bluetooth\mesh_light_node\src\init\mesh_com_init.c

/* 定义设备的 Models */
static struct bt_mesh_model elem0_root_models[] = {
    /* Configuration Server Model, 必选 Model */
    BT_MESH_MODEL_CFG_SRV_NULL(),
    /* Health Server Model, 可选 Model */
```

```
    BT_MESH_MODEL_HEALTH_SRV_NULL(),
    /* Generic OnOff Server Model */
    BT_MESH_MODEL_GEN_ONOFF_SRV_NULL(),
};

/* 定义设备的 Composition data */
static const struct bt_mesh_comp mesh_comp = {
    .cid = Your_Company_ID,
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};

/* MESH Model 组件初始化 */
ret = ble_mesh_model_init(&mesh_comp);
```

2.2 设备参数设置及事件处理

设置设备的 UUID、名称，注册 Model 事件处理回调函数。

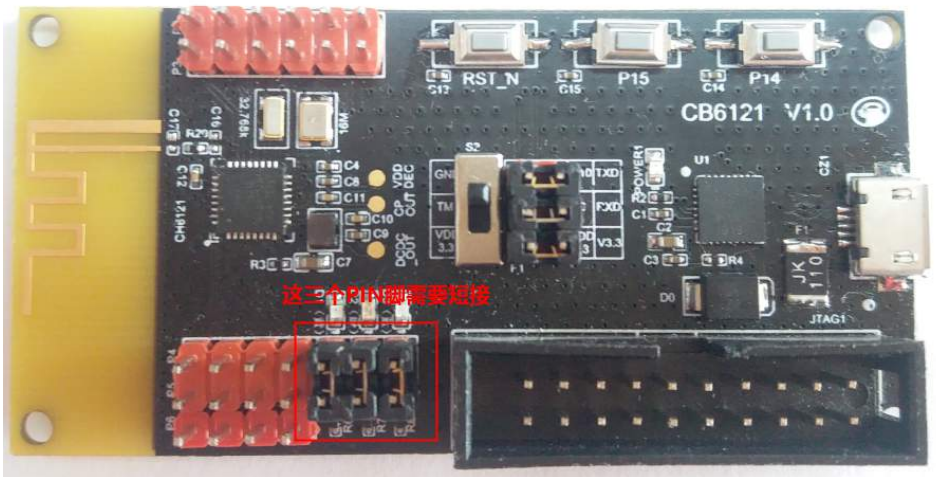
文件路径：

```
applications\bluetooth\mesh_light_node\src\app_main.c

#include "mesh_node.h"

/*
  设备 Models 定义完成后，将通过回调函数返回相关 Models 的事件，应用需要处理相关事件。
  以 Generic OnOff Server Modle 为例，
*/
void app_event_cb(uint16_t event, void *p_arg)
{
    switch (event) {
        /* Generic OnOff Server Model 的 SET 事件，
           说明 Generic OnOff Client 来设置开关状态 */
        case BT_MESH_MODEL_ONOFF_SET: {
            if (p_arg) {
                model_message message = *(model_message *)p_arg;
                S_ELEM_STATE *elem_state = (S_ELEM_STATE *)message.user_data;
                                /* 解析数据，记录开关状态，并控制 LED 灯的开和关 */
                if (elem_state->state.onoff[T_TAR]) {
                    /* 开灯 */
                    led_set_status(led2, true);
                } else {
                    /* 关灯 */
                    led_set_status(led2, false);
                }
            }
```

```
        }
        break;
        default:
            break;
    }
}

node_config_t g_node_param = {
    /* 定义设备为 Node，支持 provision 配网 */
    .role = NODE,
    /* 设置设备 UUID */
    .dev_uuid = LIGHT_DEV_UUID,
    /* 设置设备名称 */
    .dev_name = DEVICE_NAME,
    /* 注册 Modles 事件处理回调函数 */
    .user_model_cb = app_event_cb,
    /* 注册 provision 事件处理回调，默认为 NULL，Node 组件将处理相关事件 */
    .user_prov_cb = NULL,
};


/* MESH Node 组件初始化 */
ret = ble_mesh_node_init(&g_node_param);
```

2.3 LED 灯驱动

开发板的 LED 灯可以通过跳线控制，默认使用跳冒时，使用管脚 P23、P31、P32 控制。本示例中使用的是 P31 管脚控制开发板 D2 灯。



文件路径：

```
applications\bluetooth\mesh_light_node\src\board\ch6121_evb\board_ble.c

#include "pinmux.h"
#include "pin.h"
#include "pin_name.h"
#include "drv_gpio.h"

/* 定义 LED 灯的控制管脚 */
#define LED2 P31

/* 定义 LED 灯的控制管脚的句柄 */
static gpio_pin_handle_t led1, led2, led3;

void led_set_status(uint8_t led,uint8_t status)
{
    switch(led){
      ......
      /* 控制 GPIO 输出电平状态 */
      case 1:   csi_gpio_pin_write(led2, status);break;
        ......
      default :break;
    }
}

void board_ble_init(void)
{
    ......

    /* 控制管脚复用为 GPIO */
    drv_pinmux_config(LED2, PIN_FUNC_GPIO);
    /* 获取该管脚 GPIO 句柄 */
    led2 = csi_gpio_pin_initialize(LED2, NULL);
    /* GPIO 初始化为输出模式 */
    csi_gpio_pin_config(led2, GPIO_MODE_PUSH_PULL, GPIO_DIRECTION_OUTPUT);

    ......
}
```

蓝牙 Mesh 私有 Model 开发指南

1.1 目的

这里以 MESH Model 组件里已有的 vendor model 为例，介绍如何开发自己私有的 Mesh model。

1.2 基本概念

开发 MESH Model 前先了解一些基本的概念：

元素：元素（Element）是节点中可寻址的最小单元，分为主要元素和次要元素，简单的节点通常只包含一个主要元素，元素通常由一个或者多个模型（model）组成。

模型：mesh model 是蓝牙 Mesh 协议中基础业务单元，一个或者多个模型对应特定的业务，模型又分为服务模型 (Server Model) 以及终端模型（Client Model）;mesh 模型可分为基础模型（Foundation Model）、通用模型（Generic Model）、以及厂家自定义模型（Vendor Model），用户使用基础模型和通用模型通常能组合出大部分应用，也可以开发私有 Mesh Model 实现自定义的服务。

状态：state 描述的模型的属性，给其它模型提供属性进行操作的的是 Server Model；对其它模型状态进行操作的模型是 Client Model。

消息：蓝牙 Mesh 网络中所有的交互都是通过消息 (message) 完成的，定义消息的目的是为了完成对状态的操作。

1.3 vendor model 设计

这里以 MESH 组件已有 vendor model 中自动配网属性设计为例，介绍 vendor model 开发流程 .vendor model 设计主要包含消息回调函数设计以及消息 api 设计。

1.3.1 消息回调函数实现

mesh model 消息回调函数的设计主要基于 btmeshmodel_op（结构体），该结构体定义如下。

btmeshmodel_op（结构体）定义	
u32_t opcode	操作码，vendor model 使用 3 字节操作码，第一个字节为操作码，第二三字节为厂家码，可使用 BTMESHMODELOP3 宏定义该操作码，参见 BTMESHMODELOP3 宏定义
sizet minlen	消息最短长度
void (*func)(struct btmeshmodel *model,struct btmeshmsgctx *ctx,struct netbuf_simple *buf)	消息回调函数
BTMESHMODELOP3（b0,cid）（宏定义）	
uint8_t b0	操作码
uint16_t cid	厂家码

以下为 MESH 组件已有 vendor srv model 部分消息回调函数实现

```
const struct bt_mesh_model_op g_vendor_srv_model_alibaba_op[VENDOR_SRV_MODEL_
OPC_NUM] = {
    { BT_MESH_MODEL_OP_3(VENDOR_OP_ATTR_MESH_AUTOCONFIG, CONFIG_MESH_VENDOR_
COMPANY_ID), 1, _vendor_model_autoconfig},/*_vendor_model_autoconfig 为接收到
VENDOR_OP_ATTR_MESH_AUTOCONFIG 消息的回调函数 */
    { BT_MESH_MODEL_OP_3(VENDOR_OP_ATTR_MESH_AUTOCONFIG_GET, CONFIG_MESH_
VENDOR_COMPANY_ID), 0, _vendor_model_autoconfig_get},/*_vendor_model_
autoconfig_get 为接收到 VENDOR_OP_ATTR_MESH_AUTOCONFIG_GET 消息的回调函数 */
    BT_MESH_MODEL_OP_END,
};
```

1.3.2 消息 api 实现

对于 mesh srv/cli 模型，大多数情况下是通过 cli 模型主动操作，srv model 通过消息回调函数被动响应的模式，因此设计主要是 cli model 的 api, 这里涉及 VENDORO-PATTRMESHAUTOCONFIG 以 及 VENDOROPATTRMESHAUTOCONFIG_GET 两条 api, 该类 api 的设计方法如下：

```
/bt_mesh_model_msg_init(p_msg, BT_MESH_MODEL_OP_3(opcode, cid));/* 消息头 */
net_buf_simple_add_u8(p_msg, model_msg->tid);/* 消息 tid, 用于 app 层防止重复报文,
```

```
可选 */
if (model_msg->len) {
  net_buf_simple_add_mem(p_msg, model_msg->data, model_msg->len);/* 其它待发送数
据，可选 */
}
ctx.app_idx = model_msg->appkey_idx;/* 消息发送使用的 appkey idx*/
ctx.net_idx = model_msg->netkey_idx;/* 消息发送使用的 netkey idx*/
ctx.addr = model_msg->dst_addr;/* 消息目标地址 */
ctx.send_ttl = bt_mesh_default_ttl_get();/* 消息 ttl，这里使用默认值，也可使用特定值
*/
err = bt_mesh_model_send(model_msg->model, &ctx, p_msg, NULL, NULL);/* 使用 bt_
mesh_model_send api 发送数据，参见 bt_mesh_model_send 相关 api 说明 */
```

1.3.3 model 结构体实现

实现消息回调函数及 mesh api 设计后，将消息回调函数结构体填充到 BTMESH-MODEL_VND(结构体)，参见该结构体定义。

BTMESHMODELVND(company, id, _op, _pub, _userdata) (宏) 定义	
_company	vendor model company id
_id	model id
_op	model 消息回调函数 , 参见 btmeshmodel_op(结构体) 定义
_pub	model pub 参数，参见 btmeshmodel_pub (结构体) 定义
userdata	用户数据
btmeshmodel_pub (结构体) 定义	
struct btmeshmodel *mod	mesh model, 参见 btmeshmodel(结构体) 定义
u16_t addr	pub 地址 (通常通过 cfg cli 配置)
u16_t key	pub 使用的 appkey idx (通常通过 cfg cli 配置)
u8_t ttl	pub 使用的 ttl (通常通过 cfg cli 配置)
u8_t period	pub 周期 (通常通过 cfg cli 配置)
struct netbufsimple *msg	pub 消息所用地址
int (*update)(struct btmeshmodel *mod)	使用的周期性 pub 函数，用户若使用周期性 pub 功能，需定义该函数

以下为 MESH 组件已有 vendor srv model pub 实现。

```
struct bt_mesh_model_pub g_vendor_srv_model_alibaba_pub = {
```

```
    .msg = NET_BUF_SIMPLE(3 + 377 + 4), // 这里只定义了 pub 消息所使用的地址
};
```

如下为 MESH 组件已有 vendor srv model 实现，至此就完成了一个私有 mesh model 的设计。

```
#define MESH_MODEL_VENDOR_SRV(_user_data) BT_MESH_MODEL_VND(BT_MESH_MODEL_
VND_COMPANY_ID, BT_MESH_MODEL_VND_MODEL_SRV,g_vendor_srv_model_alibaba_op,&g_
vendor_srv_model_alibaba_pub, _user_data)
```

蓝牙 Mesh 开关开发实例

1. 智能开关介绍

本文将使用蓝牙 Mesh SDK 和 CB6121 开发板，通过 Shell 命令和按键触发模拟智能开关，用来控制智能灯的开关状态。

2. 应用开发

2.1 应用初始化

应用入口函数 app_main()，主要实现如下功能：

- 板级初始化
- MESH 开关 Models 定义
- Mesh 节点初始化，注册 Model 事件回调处理函数

代码分析：

```
/* 定义设备名称，使用手机 APP 扫描时可以看到该名称 */
#define DEVICE_NAME "YoC Mesh Switch"

/* 定义设备 UUID */
#define SWITCH_DEV_UUID {0xcf, 0xa0, 0xea, 0x72, 0x17, 0xd9, 0x11, 0xe8,
0x86, 0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xdf, 0x00}

int app_main(int argc, char *argv[])
{
    /* 板级初始化，各业务模块初始化 */
    board_yoc_init();
      ......

      /* MESH 开关的 Models 定义和 Node 配置参数定义 */
    ret = mesh_dev_init();
      ......
```

```
    /* Mesh 开关 CLI 命令初始化 */
    cli_reg_cmd_switch_ctrl();
      ......

    /* 初始化开发板板载按键 */
    buttons_init();

    /* 信号量初始化 */
    aos_sem_new(&sync_sem, 0);

    while (1) {
        /* 等待按键触发 */
        aos_sem_wait(&switch_state.sync_sem, AOS_WAIT_FOREVER);

        if (switch_state.button1_press) {
            /* 按键 P14 触发 */
            switch_state.button1_press = 0;
            /* 向配置好的组播地址发送无需 ACK 的开或者关指令 */
            gen_onoff_set(switch_state.on_off, false);
            ......

            /* 记录开关状态 */
            prepare_onoff_state(!switch_state.on_off);
        }

        if (switch_state.button2_press) {
            /* 按键 P15 触发 */
            switch_state.button2_press = 0;
            /* 向配置好的组播地址发送需 ACK 的开或者关指令，等待 ACK 事件上报后，更新开关
状态 */
            gen_onoff_set(switch_state.on_off, true);
            ......
        }
    }
    return 0;
}
```

2.2 设备模型定义

MESH 解决方案中，模型用于定义设备节点的功能。MESH 开关可以定义为下列模型：

- Configuration Server Model

  这个模型用于存储节点的 MESH 网络配置。

- Health Server Model

  该模型主要用于 MESH 网络诊断。

- Generic OnOff Cient Model

  该模型用于获取、控制 Generic OnOff Server 设备的开关属性。

代码分析：

```
/* 定义记录开关状态的全局变量 */
static set_onoff_arg set_onoff_data;
static struct bt_mesh_model elem0_root_models[] = {
    /* 本设备节点的 Configuration Server 模型定义 */
    BT_MESH_MODEL_CFG_SRV_NULL(),
    /* 本设备节点的 Health Server 模型定义 */
    BT_MESH_MODEL_HEALTH_SRV_NULL(),
    /* 本设备节点的 Generic OnOff Client 模型定义，发布状态时，将记录的 state 发布出去
*/
    MESH_MODEL_GEN_ONOFF_CLI(&set_onoff_data),
};

/* 本设备节点的 Vendor Models 模型定义 */
static struct bt_mesh_model elem0_vnd_models[] = {
    BT_MESH_MODEL_VENDOR_SRV_NULL,
};

/* 本设备节点的 Elements 定义 */
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, elem0_root_models, elem0_vnd_models, 0),
};

/* Define device composition data */
static const struct bt_mesh_comp mesh_comp = {
    .cid = CONFIG_CID_TAOBAO,
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};

/* 本设备节点的 Composition Datas 定义，定义 CompanyID、Elements */
static const struct bt_mesh_comp mesh_comp = {
    /* 设置设备节点的 CompanyID */
    .cid = CONFIG_CID_TAOBAO,
     /* 设置设备节点的 Elements */
    .elem = elements,
    /* 设置设备节点 Elements 的个数 */
    .elem_count = ARRAY_SIZE(elements),
};
```

```
/* 节点参数定义 */
static node_config_t g_node_param = {
    /* 普通设备节点 */
    .role = NODE,
    /* 设备 UUID */
    .dev_uuid = SWITCH_DEV_UUID,
    /* 设备名称 */
    .dev_name = DEVICE_NAME,
    /* Models 事件回调函数注册 */
    .user_model_cb = app_models_event_cb,
    /* Provision 事件回调函数注册，为 NULL 时，Mesh Node 组件将处理 Provision 事件 */
    .user_prov_cb = app_prov_event_cb,
    /* Health Model 事件回调函数注册 */
    .health_cb = &g_app_health_cb,
};

int mesh_dev_init(void)
{
    int ret;
    memset(&set_onoff_data, 0, sizeof(set_onoff_arg));

    /* Models 定义 */
    ret = ble_mesh_model_init(&mesh_comp);
      ......

    /* 节点参数设置 */
    ret = ble_mesh_node_init(&g_node_param);
      ......
    return 0;
}
```

2.3 MESH Models 事件回调处理

MESH Node 组件将根据所定义的 Models，上报相关事件及数据。以 OnOff 属性为例，进行代码解析：

```
void app_models_event_cb(uint16_t event, void *p_arg)
{
    switch (event) {
        /* Generic OnOff Server 上报 OnOff 状态 */
        case BT_MESH_MODEL_ONOFF_STATUS: {
            if (p_arg) {
                model_message onoff_message = *(model_message *)p_arg;
                uint8_t onoff = onoff_message.status_data->data[0];
                ......
                /* 更新开关状态 */
                prepare_onoff_state(!onoff);
```

```
            }
        }
        break;
              ......
    }
}
```

注意：

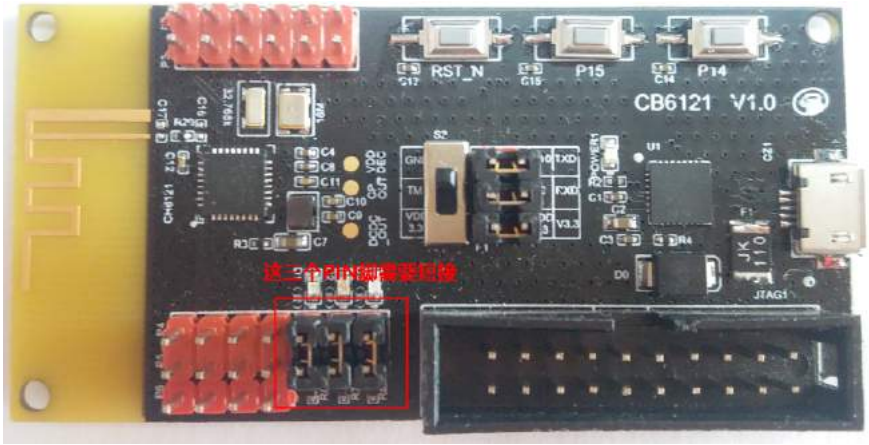
网络中需要存在多个节点时，需要更改代码中设备 UUID 后编译烧录该镜像，并烧录镜像时设置不同的 MAC 地址，避免组网出现问题 。

3. 例程运行

3.1 示例介绍

本示例的运行需要两块开发板，一块用于烧录 MESH 开关示例，一块用于烧录 MESH 灯示例。两个设备可同时通过手机 nRF Mesh APP 配置入网、绑定 APP Key、设置组地址。然后通过 MESH 开关设备的 Shell 命令或者按键触发，控制 MESH 灯的开关状态。实际运行效果可参看 SIG MESH 配网和控制演示视频。

3.2 开发板连线



3.3 编译

- 进入 SDK 根目录，编译 MESH 灯应用镜像

```
$ make clean
$ ./build.sh defconfigs/defconfig_ch6121_evb_mesh_switch_node applications/
bluetooth/mesh_switch_node/ j64
$ ll applications/bluetooth/mesh_switch_node/generated
......
-rwxrwxrwx 1 xxx xxx 652920 Mar 13 19:03 total_image.hexf*
....
```

3.4 运行

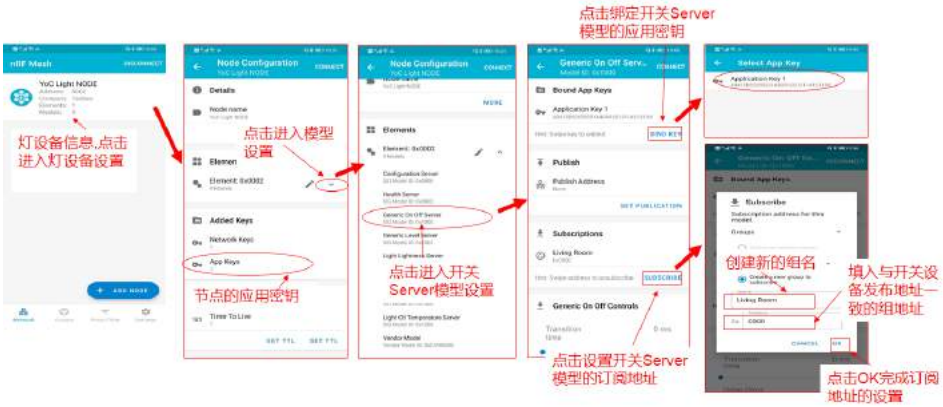
- 下载工程目录下的镜像文件 (generated/total_image.hexf) 至 CB6121 开发板，镜像下载可参考《CB6121 快速上手手册》
- 连接串口调试工具 , 配置为

  波特率 : 115200, 数据位 : 8, 校验位 : None, 停止位 : 1, 流控 : None
- 开发板拨码开关拨至 GND
- 按 RESET 按键，复位开发板
- 设备将开启 Unprovisoin Device Beacon 广播，串口上输出 'Mesh Switch node demo' 信息

```
[     0.004000][I][init    ]Build:Apr  8 2020,13:34:58
[     0.017000][I][init    ]find 9 partitions
// 硬复位计数，首次上电没有存储信息，3 秒内再次复位开发板计数将加 1，重复 5 次即可清除入网信息
[     0.176000][E][MESH_RESET]read size error
[     0.185000][I][MESH_RESET]reset_by_repeat_init, number = 1
Welcome to CLI...
> [     0.270000][I][DEMO    ]Mesh Switch node demo
```

- MESH 灯设备入网操作请参照智能灯应用开发实例 操作，发布地址的操作如下图所示：

注意： MESH 灯设备的 Generic OnOff Server Model 应用密钥需要设置与 MESH 开关设备一致，SUB 地址与 MESH 开关的 PUB 地址一致

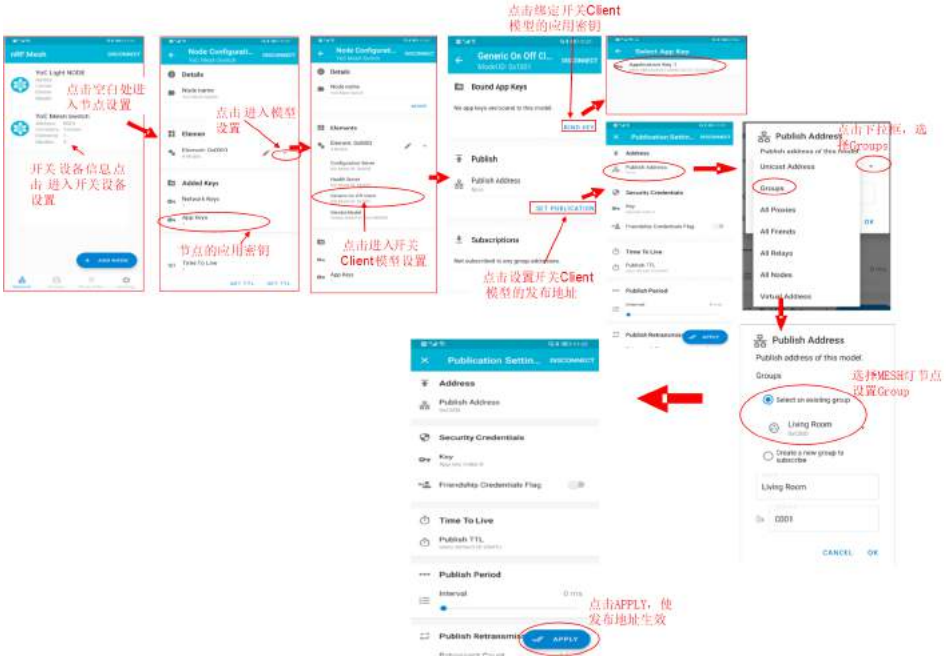


- MESH 开关设备配网可参照下图，通过手机 nRF Mesh APP 操作，绑定应用密钥的索引为 1

- 手机操作设备入网时，将在串口输出如下信息

```
/* 手机 APP 扫描到灯节点后，操作 IDENTITY -> PROVISION */
> [    75.233000][I][BT_MESH_NODE]Provisioning link opened on PB-GATT

[    90.943000][I][BT_MESH_NODE]Provisioning link closed on PB-GATT

[    90.973000][I][BT_MESH_NODE]provisioning complete netid 0, primary
element address 0x3
/* 入网成功，节点 unicast address 为 0x0003 */
[    90.990000][I][Mesh CTRL]prov complete 0003
```

- 设置 MESH 开关设备中的 Generic OnOff Client Model 的应用密钥，绑定应用密钥的索引为 1；并设置 Publication 地址为 0xC000



- 通过 Shell 命令控制智能灯的开关状态，设备将在串口上输出如下信息

```
// 控制网络中 0xC000 组中的 Mesh 灯开，因此会收到 Models BT_MESH_MODEL_ONOFF_STATUS 事件
> meshonoff 1 1

// 控制网络中 0xC000 组中的 Mesh 灯关，无 Models 事件上报
> meshonoff 0 0
```

- 还可以通过开发板上 P14 按键，发送无需 ACK 的开关控制操作；P15 按键，发送需要 ACK 的开关控制操作

```
[   207.266000][I][DEMO    ]send unack msg LED OFF
```

- 通过开发板上 P15 按键，发送需要 ACK 的开关控制操作

```
[   210.852000][I][DEMO    ]send ack msg LED ON
```

3.5 Shell 命令

```
// 开关控制命令
meshonoff <onoff 0,1> <ack 0,1>
onoff：开关状态   1 - 开；0 - 关；
ack：设置该请求是否需要 ACK 回应   1 - 需要 ACK，ACK 将通过 Models 事件上报；0 - 无需 ACK

// 节点网络配置复位
meshrst
```

蓝牙 Mesh 灯控开发实例

1. 智能灯介绍

蓝牙 MESH 智能灯，是智能家居系统中最基础的设施。通过设置智能灯的模型属性，能够实现轻松、高效地控制灯的状态。本文将使用蓝牙 Mesh SDK 和 CB6121 开发板，以智能灯的开关应用开发为例指导读者进行 MESH 开发。

2. 应用开发

2.1 应用初始化

应用入口函数 app_main()，主要实现如下功能：

- 板级初始化
- MESH 灯 Model 定义
- Mesh 节点初始化，注册 Model 事件回调处理函数

代码分析：

```
/* 定义设备名称，使用手机 APP 扫描时可以看到该名称 */
#define DEVICE_NAME "YoC Light"

/* 定义设备 UUID */
#define LIGHT_DEV_UUID {0xcf, 0xa0, 0xe3, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86,
0xd1, 0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0x02}

node_config_t g_node_param = {
    /* 设置当前为设备节点 */
    .role = NODE,
    /* 设置节点的 UUID */
    .dev_uuid = LIGHT_DEV_UUID,
    /* 设置节点的 Device Name */
    .dev_name = DEVICE_NAME,
    /* 注册 Model 的事件处理回调函数 */
    .user_model_cb = app_event_cb,
```

```
    /* 注册 Provision 的事件处理回调函数 */
    .user_prov_cb = NULL,
    /* RSSI 上报功能使能标识，默认关闭 */
    .rssi_report_enable = 0,
};

int app_main(int argc, char *argv[])
{
    /* 板级初始化，各业务模块初始化 */
    board_yoc_init();
      ......

      /* MESH 灯 Model 定义，根据灯实际功能定义，如开关、亮度、色温控制等 */
    ret = app_mesh_composition_init();
      ......

    /* MESH 节点初始化，设置节点 UUID、Device Name 并注册 Model 事件回调函数 */
    ret = ble_mesh_node_init(&g_node_param);
      ......

    /* 信号量初始化 */
    aos_sem_new(&sync_sem, 0);

    while (1) {
        /* 等待信号量 */
        aos_sem_wait(&sync_sem, AOS_WAIT_FOREVER);
    }

    return 0;
}
```

2.2 设备模型定义

MESH 解决方案中，模型用于定义设备节点的功能。以 MESH 灯为例，可以定义下列模型：

- Configuration Server Model

  这个模型用于存储节点的 MESH 网络配置。

- Health Server Model

  该模型主要用于 MESH 网络诊断。

- Generic OnOff Server Model

  该模型用于获取、控制设备的开关属性。

- Generic Level Server Model

  该模型用于获取、控制设备的档位属性。

- Generic Lightness Server Model

  该模型用于获取、控制设备的亮度属性。

- Generic CTL Server Model / Generic CTL Setup Server Model / Generic CTL Temperature Server Model

  这三个模型均用于调节设备的色温属性。

代码分析：

```
static struct bt_mesh_model elem0_root_models[] = {
    /* 本设备节点的 Configuration Server 模型定义 */
    BT_MESH_MODEL_CFG_SRV_NULL(),
    /* 本设备节点的 Health Server 模型定义 */
    BT_MESH_MODEL_HEALTH_SRV_NULL(),
    /* 本设备节点的 Generic OnOff Server 模型定义 */
    BT_MESH_MODEL_GEN_ONOFF_SRV_NULL(),
    /* 本设备节点的 Generic Level Server 模型定义 */
    BT_MESH_MODEL_GEN_LEVEL_SRV_NULL(),
    /* 本设备节点的 Generic Lightness Server 模型定义 */
    BT_MESH_MODEL_LIGHTNESS_SRV_NULL(),
    /* 本设备节点的 Generic CTL Server 模型定义 */
    BT_MESH_MODEL_CTL_SRV_NULL(),
    /* 本设备节点的 Generic CTL Setup Server 模型定义 */
    BT_MESH_MODEL_CTL_SETUP_SRV_NULL(),
    /* 本设备节点的 Generic CTL Temperature Server 模型定义 */
    BT_MESH_MODEL_CTL_TEMP_SRV_NULL(),
};

/* 本设备节点的 Vendor Models 模型定义 */
static struct bt_mesh_model elem0_vnd_models[] = {
    BT_MESH_MODEL_VENDOR_SRV_NULL,
};

/* 本设备节点的 Elements 定义 */
static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, elem0_root_models, elem0_vnd_models, 0),
```

```
};

/* 本设备节点的 Composition Datas 定义，定义 CompanyID、Elements */
static const struct bt_mesh_comp mesh_comp = {
    /* 设置设备节点的 CompanyID */
    .cid = 0x01A8,
     /* 设置设备节点的 Elements */
    .elem = elements,
    /* 设置设备节点 Elements 的个数 */
    .elem_count = ARRAY_SIZE(elements),
};

int app_mesh_composition_init()
{
    int ret;
    /* Models 初始化 */
    ret = ble_mesh_model_init(&mesh_comp);
    return ret;
}
```

2.3 MESH Models 事件回调处理

MESH Node 组件将根据所定义的 Models，上报相关事件及数据。以 OnOff 属性为例，进行代码解析：

```
void app_event_cb(uint16_t event, void *p_arg)
{
    switch (event) {
        /* 控制 OnOff 属性 */
        case BT_MESH_MODEL_ONOFF_SET: {
            if (p_arg) {
                model_message message = *(model_message *)p_arg;
                S_ELEM_STATE *elem_state = (S_ELEM_STATE *)message.user_data;
                                /* 根据数据包中 OnOff 值，控制 OnOff 属性 */
                if (elem_state->state.onoff[T_TAR]) {
                    /* 控制 LED 灯亮 */
                    led_set_status(led2, true);
                } else {
                    /* 控制 LED 灯灭 */
                    led_set_status(led2, false);
                }
                                ......
            }
        }
        break;
```

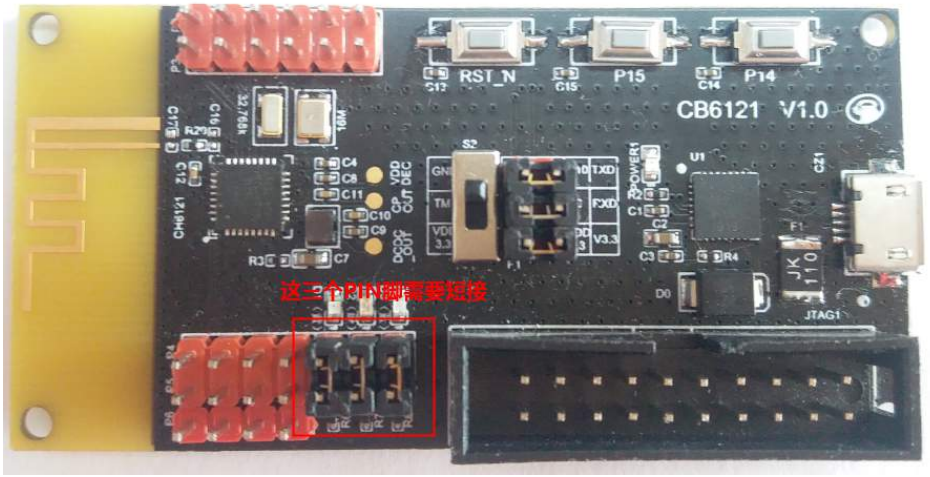
```
                ......

        /* 通过 Vendor Model，发送私有数据 */
        case BT_MESH_MODEL_VENDOR_MESSAGES: {
            if (p_arg) {
                model_message message = *(model_message *)p_arg;
                char data_out[80] = {0};
                ......
            }
        }
        break;

        default:
            break;
    }
}
```

注意：

网络中需要存在多个节点时，需要更改代码中设备 UUID 后编译烧录该镜像，并烧录镜像时设置不同的 MAC 地址，避免组网出现问题 。

3. 例程运行

3.1 开发板连线



3.2 编译

- 进入 SDK 根目录，编译 MESH 灯应用镜像

```
$ make clean
$ ./build.sh defconfigs/defconfig_ch6121_evb_mesh_node applications/
bluetooth/mesh_light_node/ j64
$ ll applications/bluetooth/mesh_light_node/generated
......
-rwxrwxrwx 1 xxx xxx 652920 Mar 13 19:03 total_image.hexf*
....
```

3.3 运行

- 下载工程目录下的镜像文件 (generated/total_image.hexf) 至 CB6121 开发板，镜像下载可参考《CB6121 快速上手手册》
- 连接串口调试工具 , 配置为

  波特率 : 115200, 数据位 : 8, 校验位 : None, 停止位 : 1, 流控 : None
- 开发板拨码开关拨至 GND
- 按 RESET 按键，复位开发板
- 设备将开启 Unprovisoin Device Beacon 广播，串口上输出 'Mesh light node demo' 信息

```
[     0.004000][I][init    ]Build:Apr  8 2020,13:34:01
[     0.017000][I][init    ]find 9 partitions
// 硬复位计数，首次上电没有存储信息，3 秒内再次复位开发板计数将加 1，重复 5 次即可清除入网信息
[     0.175000][E][MESH_RESET]read size error
[     0.184000][I][MESH_RESET]reset_by_repeat_init, number = 1
Welcome to CLI...
> [     0.206000][I][DEMO    ]Mesh light node demo
```

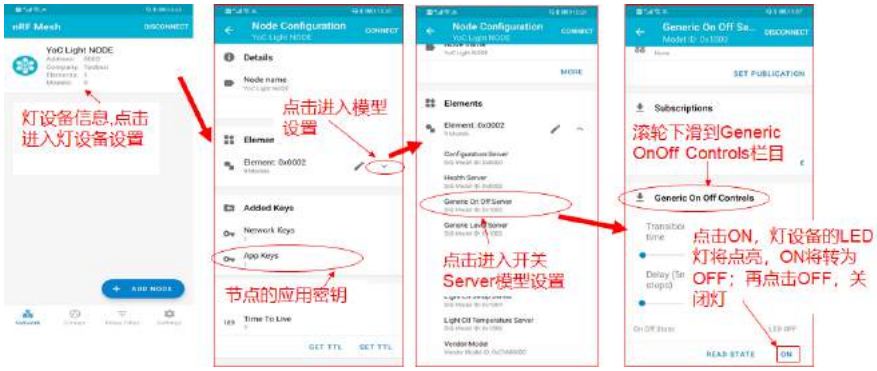
- MESH 灯设备配网可参照下图，通过手机 nRF Mesh APP 操作，绑定应用密钥的索引为 1

- 设置智能灯设备中的 Generic OnOff Server Model 的应用密钥，绑定应用密钥的索引为 1

- 手机操作设备成功入网后，开发板上 D2 绿灯将常亮，并在串口输出如下信息

```
/* 手机 APP 扫描到灯节点后，操作 IDENTITY -> PROVISION */
[    32.264000][I][BT_MESH_NODE]Provisioning link opened on PB-GATT

[    62.701000][I][BT_MESH_NODE]Provisioning link closed on PB-GATT

[    62.731000][I][BT_MESH_NODE]provisioning complete netid 0, primary
element address 0x2
/* 入网成功，节点 unicast address 为 0x0002 */
[    62.747000][I][DEMO    ]prov complete 0002
```

- 通过手机 nRF Mesh APP 绑定 Generic OnOff Server Model 的 APP Key 后，控制灯的开关状态



- 开发板 D3 绿灯将被控制，并在串口上输出如下信息

```
[   146.549000]<I>[DEMO    ]src:0x0001,led:ON
[   213.850000]<I>[DEMO    ]src:0x0001,led:OFF
[   225.002000]<I>[DEMO    ]src:0x0001,led:ON
[   287.153000]<I>[DEMO    ]src:0x0001,led:OFF
```

蓝牙 Mesh 配网模组用户手册（基于 AT 命令）

1. 概述

1.1 目的

本文主要介绍 Mesh 配网模块支持的 AT 指令集，为有 Mesh 配网需求的开发人员提供帮助。

开发者可以使用配网模块发现，添加未入网设备，对入网设备进行配置，管理。

1.2 名词解释

下面介绍本文中涉及的一些专有名词：

名词	说明
BLE	bluetooth Low energy，低功耗蓝牙
蓝牙 Mesh	基于 BLE 技术实现的多对多通信网络
MESH 配网模块	基于 AT 命令实现的 Provisioner
Provision	将一个未入网的设备加入蓝牙 Mesh 网络的过程， 期间会通过认证和加密的方式将密钥和地址分配给设备
Address	Mesh 节点的地址
RPL	replay protection list，重放攻击保护列表。
OOB	Out of band，带外数据，用于 Provision 过程中鉴权。
node	已入网设备
Composition Data	node 内部的可用信息
CTL	Color-tunable light，色温可调灯，支持色温和亮度控制

1.3 介绍

本文将会涉及到的如下几部分按章节介绍：

1. AT 命令语法
2. AT 命令

AT 命令将分为如下几个类别：

- 基础 AT 命令
- 包括通信检查，帮助，复位等功能的 AT 命令
- Provisioner AT 命令
- 配网相关功能 AT 命令，包括发现，添加，配置设备等功能
- Mesh Client models AT 命令
- 包括 Config Client model，Generic OnOff Client model，Generic Level Client model，Light Lightness Client model，Light CTL Client model，Vendor Client model 的 AT 命令

3. AT 事件

4. AT 使用指南

在 6.0 章节，以灯控为例，介绍了 Mesh 网络的构建和配置。

2. AT 指令语法

AT 命令基于 ASCII 码编码，以回车符 <\r> 作为结束符。

2.1 AT 请求消息格式

AT 请求消息格式：AT+[OP][para-1,para-2,……para-n]<\r>

AT 请求消息格式

域	说明
AT+	命令消息前缀
CMD	命令消息前缀
OP	指令操作符。可以是以下内容：“=”：表示写指令，用于参数设置。“?”：表示读指令，用于查询参数的当前值。“”：表示执行指令。“=?”：表示测试指令，查询指令使用方法。

para-ra-1,para-ra-2,……	表示设置的参数值，或者是指定要查询的参数
\r	回车结束符，ASCII 码为 0x0D

2.2 AT 响应消息格式

AT 响应消息为针对 AT 命令的响应。

AT 响应消息格式：<\r\n>[+CMD:][para-1,para-2,……para-n]<\r\n>

或者：<\r\n><\r\n>

或者上述两者都有。

AT 响应消息格式

域	说明
+	响应消息前缀
CMD	响应消息前缀
para-1,para-2,……	表示设置的参数值，或者是指定要查询的参数
\r\n	响应消息结束符，ASCII 码为 0x0D0A
STATUS	OK 或者 ERROR，表示成功或者失败

2.3 AT 事件消息格式

AT 事件是设备主动上报的消息

AT 事件格式：<\r\n>[+CMD:][para-1,para-2,……para-n]<\r\n>

域	说明
+	事件前缀
CMD	事件前缀
para-1,para-2,……	表示设置的参数值，或者是指定要查询的参数
\r\n	响应消息结束符，ASCII 码为 0x0D0A

3. AT 指令总览

3.1 AT 控制指令

命令	说明
AT	测试命令
AT+HELP	帮助命令
AT+IREBOOT	设备重启
AT+RST	设备恢复出厂设置
AT+MESHQUERYSTA	查询 Mesh model status
AT+MESHCLEARRPL	清空 RPL
AT+MESHPROVEN	Provisioner 配置及使能
AT+MESHPROVFILTERDEV	Provisioner 上报设备过滤
AT+MESHPROVSHOWDEV	Provisioner 上报查找到的未入网设备
AT+MESHADDDEV	Provisioner 添加未入网设备
AT+MESHDELDEV	Provisioner 删除未入网设备
AT+MESHOOB	Provisioner 添加 OOB 信息
AT+MESHAUTOCONFIG	Provisioner 对 node 节点进行自动配置
AT+MESHGETNODEINFO	Provisioner 查询 node 信息
AT+MESHGETCOMP	获取 node 节点组成信息
AT+MESHADDAPPKEY	添加 node app key
AT+MESHBINDAPPKEY	绑定 node model 和 app key
AT+MESHUNBINDAPPKEY	解绑 node model 和 app key
AT+MESHRELAY	配置 node relay 属性
AT+MESHPROXY	配置 node proxy 属性
AT+MESHFRINED	配置 node friend 属性
AT+MESHGETSUB	获取 node model 订阅地址
AT+MESHSETSUB	设置 node model 订阅地址
AT+MESHDELSUB	删除 node model 订阅地址
AT+MESHGETPUB	获取 node model 发布属性
AT+MESHSETPUB	设置 node model 发布属性
命令	**说明**
AT+MESHRST	清除 node 配网信息
AT+MESHONOFF	配置 node onoff 状态
AT+MESHLEVEL	配置 node generic level
AT+MESHLEVELMOVE	配置 node generic level move
AT+MESHLEVELDELTA	配置 node generic level delta
AT+MESHLIGHTNESSRANGE	配置 node lightness range
AT+MESHLIGHTNESSDEF	配置 node lightness default
AT+MESHLIGHTNESS	配置 node lightness
AT+ MESHLIGHTNESSLIN	配置 node lightness linear
AT+MESHCTLTEMPRANGE	配置 node CTL temperature range

AT+MESHCTLDEF	配置 node CTL default
AT+MESHCTL	配置 node CTL
AT+MESHCTLTEMP	配置 node CTL temperature
AT+MESHTRS	发送透传数据

3.2 AT 事件

事件	说明
+IREBOOT	设备启动信息
+MESHDEV	未入网设备信息上报
+MESHFOUNDDEVTIMEOUT	查找设备超时
+MESHPROVFAILD	Provisioner 添加设备失败
+MESHNODEADD	Provisioner 添加设备成功
+MESHNODEINFO	node 节点信息
+MESHOOBINPUT	Provisioner 输入 oob 信息
+MESHAPPKEYADD	node 添加 appkey 状态
+MESHAPPKEYBIND	node model 绑定 appkey 状态
+MESHFRIEND	node friend 属性
+MESHPROXY	node proxy 属性
+MESHRELAY	node relay 属性
事件	**说明**
+MESHRST	节点 RST 信息
+MESHCOMP	节点 Composition Data 信息
+MESHSUBLIST	node model sub 信息
+MESHSUBSET	node model sub 设置状态
+MESHPUB	node model pub 信息
+MESHPUBSET	node model pub 设置状态
+MESHONOFF	onff model 状态
+MESHLEVEL	Level model 状态
+MESHLIGHTNESS	Lightness model 状态
+MESHLIGHTNESSLIN	Lightness model linear 状态
+MESHLIGHTNESSRANGE	Lightness model range 状态
+MESHLIGHTNESSLAST	Lightness model last 状态
+MESHLIGHTCTL	LIGHT CTL model 状态

+MESHCTLTEMP	LIGHT CTL model temperature 状态
+MESHCTLTEMPRANGE	LIGHT CTL model range 状态
+MESHCTLDEF	LIGHT CTL model default 状态
+MESHAUTOCONFIG	node 节点自动配置状态

4. AT 指令说明

4.1 基础 AT 命令

主要完成 AT 状态测试、设备重启、AT 查询、MESH model 状态查询等功能

4.1.1 AT

基本功能：测试 AT 模块状态

4.1.2 AT+HELP

基本功能：测试 AT 支持地命令

4.1.3 AT+IREBOOT

基本功能：重启设备

4.1.4 AT+RST

基本功能：设备恢复出厂设置

4.1.5 AT+MESHQUERYSTA

基本功能：查询 model 的 status 状态

4.2 Provisioner AT 命令

4.2.1 AT+MESHPROVEN

基本功能：Provisioner 使能

4.2.2 AT+MESHPROVFILTERDEV

基本功能：Provisioner 设置上报过滤

4.2.3 AT+MESHPROVSHOWDEV

基本功能：Provisioner 设备上报功能开启

4.2.4 AT+MESHGETNODEINFO

基本功能：Provisioner 获取本地节点及入网节点信息

4.2.5 AT+MESHADDDEV

基本功能：Provisioner 添加待入网设备

4.2.6 AT+MESHAUTOCONFIG

基本功能：Provisioner 自动配置已入网节点

4.2.7 AT+MESHDELDEV

基本功能：Provisioner 删除待入网设备

4.2.8 AT+MESHOOB

基本功能：Provisioner OOB 输入

4.2.9 AT+MESHSETAPPKEY

基本功能：Provisioner 配置 app key

4.2.10 AT+MESHCLEARRPL

基本功能：清空 RPL 记录

4.3 Config Client model AT 命令

4.3.1 AT+MESHADDAPPKEY

基本功能：通过 CFG CLI 给 node 节点添加 app key

4.3.2 AT+MESHBINDAPPKEY

基本功能：通过 CFG CLI 给 node 节点绑定 app key

4.3.3 AT+MESHUNBINDAPPKEY

基本功能：通过 CFG CLI 给 node 节点解绑 app key

4.3.4 AT+MESHGETCOMP

基本功能：通过 CFG CLI 获取 node 节点信息

4.3.5 AT+MESHRELAY

基本功能：通过 CFG CLI 配置 node relay 属性

4.3.6 AT+MESHPROXY

基本功能：通过 CFG CLI 配置 node proxy 属性

4.3.7 AT+MESHFRIEND

基本功能：通过 CFG CLI 配置 node friend 属性

4.3.8 AT+MESHGETSUB

基本功能：通过 CFG CLI 获取 node model sub 属性

4.3.9 AT+MESHSETSUB

基本功能：通过 CFG CLI 配置 node model sub 属性

4.3.10 AT+MESHDELSUB

基本功能：通过 CFG CLI 删除 node model sub 地址

4.3.11 AT+MESHGETPUB

基本功能：通过 CFG CLI 获取 node model pub 属性

4.3.12 AT+MESHSETPUB

基本功能：通过 CFG CLI 配置 node model pub 属性

4.3.13 AT+MESHRST

基本功能：通过 CFG CLI 将 node 移除 Mesh 网络

4.4 Generic OnOff Client model AT 命令

4.4.1 AT+MESHONOFF

基本功能：通过 ONOFF CLI 发送消息

4.5 Generic Level Client model AT 命令

4.5.1 AT+MESHLEVEL

基本功能：通过 LEVEL CLI 设置 level

4.5.2 AT+MESHLEVELMOVE

基本功能：通过 LEVEL CLI 设置 level move

4.5.3 AT+MESHLEVELDELTA

基本功能：通过 LEVEL CLI 发送消息设置 level delta

4.6 Light Lightness Client model AT 命令

4.6.1 AT+MESHLIGHTNESSRANGE

基本功能：通过 LEVEL CLI 设置 lightness range

4.6.2 AT+MESHLIGHTNESSDEF

基本功能：通过 LEVEL CLI 设置 lightness default

4.6.3 AT+MESHLIGHTNESS

基本功能：通过 LEVEL CLI 设置 lightness

4.6.4 AT+MESHLIGHTNESSLIN

基本功能：通过 LEVEL CLI 设置 lightness linear

4.7 Light CTL Client model AT 命令

4.7.1 AT+MESHCTLTEMPRANGE

基本功能：通过 LIGHT CTL 设置 light ctl temperature range 状态

4.7.2 AT+MESHCTLDEF

基本功能：通过 LIGHT CTL 设置 light ctl default 状态

4.7.3 AT+MESHCTL

基本功能：通过 LIGHT CTL 设置 light ctl 状态

4.7.4 AT+MESHCTLTEMP

基本功能：通过 LIGHT CTL 设置 light ctl temperature 状态

4.8 Vendor Client model AT 命令

4.8.1 AT+MESHTRS

基本功能：通过 venodr model cli 透传数据

5 AT 事件

5.1 设备启动事件

5.2 provisioner 扫描设备信息上报事件

5.3 provisioner 扫描设备超时事件

5.4 Provisioner 设备入网失败事件

5.5 Provisioner 设备入网成功事件

5.6 OOB 输入提示事件

5.7 node 节点信息事件

5.8 Mesh Model 上报事件

5.8.1 config model status

5.8.1.1 appkey add status

5.8.1.2 appkey bind status

5.8.1.3 friend status

5.8.1.4 proxy status

5.8.1.5 relay status

6. AT 使用指南

6.1 准备工作

3 块开发板，其中 1 块作为 provisioner 节点，2 块作为 Mesh 节点。

6.2 获取未入网设备

```
provisioner 输入如下 AT 命令，其中 -> 代表命令响应或者上报事件
/*provisioner 设备上报打开 */
AT+MESHPROVSHOWDEV=0x01
->OK
/* 扫描到的未入网设备 */
->+MESHDEV:01:02:03:04:05:05,00,06050403020111e886d15f1ce28ade02,00,01
->+MESHDEV:01:02:03:04:05:06,00,06050403020111e886d15f1ce28ade02,00,01
/*provisioner 设备上报关闭 */
AT+MESHPROVSHOWDEV=0x00
```

6.3 自动配网

使用自动配网功能，Provisioner 将自动设置绑定所有 model 的 app key，配置订阅地址和发布地址。

6.3.1 单播地址自动配网

```
通过 AT+MESHAUTOCONFIG 命令对指定 node 进行配网
/* 添加未入网设备进行自动配网 */
AT+MESHADDDEV=01:02:03:04:05:05,00,06050403020111e886d15f1ce28ade02,00,01
->OK

/* 添加第二个未入网设备进行自动配网 */
AT+MESHADDDEV=01:02:03:04:05:06,00,06050403020111e886d15f1ce28ade02,00,01
->OK
/* 设备 1 入网成功 */
->+MESHnodeADD:0002,01,05050403020111e886d15f1ce28ade02
/* 设备 1  appkey 添加成功 */
->+MESHAPPKEYADD:0002,0
/* 设备 2 入网成功 */
->+MESHnodeADD:0003,01,06050403020111e886d15f1ce28ade02
/* 设备 2  appkey 添加成功 */
->+MESHAPPKEYADD:0003,0
```

```
/* 设备 1 自动配网 */
AT+MESHAUTOCONFIG=0x0002
->OK
/* 设备 1 自动配网完成 */
->+MESHAUTOCONFIG:0002,0

/* 设备 2 自动配网 */
AT+MESHAUTOCONFIG=0x0003
->OK
/* 设备 2 自动配网完成 */
->+MESHAUTOCONFIG:0003,0
```

6.3.2 组播地址自动配网

```
通过组播地址，对指定组的所有 node 进行配网，适用于大规模网络配置
/* 添加未入网设备进行自动配网 */
AT+MESHADDDEV=01:02:03:04:05:05,00,06050403020111e886d15f1ce28ade02,00,01
->OK
/* 添加第二个未入网设备进行自动配网 */
AT+MESHADDDEV=01:02:03:04:05:06,00,06050403020111e886d15f1ce28ade02,00,01
->OK

->+MESHnodeADD:0002,01,05050403020111e886d15f1ce28ade02/
->+MESHAPPKEYADD:0002,0
->+MESHnodeADD:0003,01,06050403020111e886d15f1ce28ade02
->+MESHAPPKEYADD:0003,0

/*node vendor model 默认订阅该地址，使用组播地址不返回执行状态 */
AT+MESHAUTOCONFIG=0xF000
->OK

/* 查找设备 1 自动配网状态 */
AT+MESHQUERYSTA=0x0002,0x0001,0xd701a8,0,0,0x01a8/* 手动查询 0x0002 号 node 自动配
网状态 */
->OK
/* 设备 1 自动配网完成 */
->+MESHAUTOCONFIG:0002,0

/* 查找设备 2 自动配网状态 */
AT+MESHQUERYSTA=0x0003,0x0001,0xd701a8,0,0,0x01a8/* 手动查询 0x0003 号 node 自动配
网状态 */
->OK
/* 设备 2 自动配网完成 */
->+MESHAUTOCONFIG:0003,0/* 自动配网完成 */
```

6.4 手动配网

通过单独 AT 命令，配置指定 node 的 app key，绑定对应 model 的 app key，设置订阅地址和发布地址。

```
/* 添加入网设备进行手动配网 */
AT+MESHADDDEV=01:02:03:04:05:05,00,05050403020111e886d15f1ce28ade02,00,01,00
->OK
->+MESHnodeADD:0002,01,05050403020111e886d15f1ce28ade02
/* 添加 appkey 0*/
AT+MESHADDAPPKEY=0x0002,0x0
->OK
/*appkey 添加成功 */
->+MESHAPPKEYADD:0002,0

/* 获取节点组成信息，方便对对端节点所有 model 进行绑定 */
AT+MESHGETCOMP=0x0002
->OK
->+MESHCOMPDATA:0002,01a8,0000,0000,000a,0007,0000,08,01,0000,0002,1000,1002,1300,
1303,1304,1306,01a8,0000

/* 手动绑定 appkey 0 和 0x1000 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1000
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x1002 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1002
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x1300 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1300
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x1303 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1303
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x1304 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1304
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x1306 model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x1306
->+MESHAPPKEYBIND:0002,0
/* 手动绑定 appkey 0 和 0x01a800 vendor model*/
AT+MESHBINDAPPKEY=0x0002,0x0000,0x0000,0x01a8
->+MESHAPPKEYBIND:0002,0
/* 手动设置 0x1000 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x1000,0xf000
/* 手动设置 0x1002 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x1002,0xf000
/* 手动设置 0x1300 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x1300,0xf000
/* 手动设置 0x1303 model 的订阅地址 0xf000*/
```

```
AT+MESHSETSUB=0x0002,0x1303,0xf000
/* 手动设置 0x1304 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x1304,0xf000
/* 手动设置 0x1306 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x1306,0xf000
/* 手动设置 0x01a800 model 的订阅地址 0xf000*/
AT+MESHSETSUB=0x0002,0x0000,0xf000,0x01a8
/* 手动设置 0x1000 model 的发布地址 0xf001*/
AT+MESHSETPUB=0x0002,0x1000,0xf001,3,0,0,0
```

6.5 AT App key 切换

```
使用 App key1 替换 app key 0
/*provisioner 本地添加 appkey 0*/
AT+MESHSETAPPKEY=0x0000
/*appkey 0 已存在，产生 appkey1*/
->+MESHSETAPPKEY:1
->OK

/* 发送新产生的 appkey 到 node 节点 */
AT+MESHADDAPPKEY=0x0002,0x0001
->OK
/*appkey 1 添加成功 */
->+MESHAPPKEYADD:0002,0
/* 解除 node 节点 onoff srv model 与 appkey 0 的绑定 */

AT+MESHUNBINDAPPKEY=0x0002,0x0000,0x1000
/*appkey 0 解绑成功 */
->+MESHAPPKEYBIND:0002,0
/* 绑定 node 节点 onoff srv model 与 appkey 1*/

AT+MESHBINDAPPKEY=0x0002,0x0001,0x1000
->OK
/*appkey 1 绑定成功 */
->+MESHAPPKEYBIND:0003,0

/* 发送新产生的 appkey 到 node 节点 */
AT+MESHADDAPPKEY=0x0001,0x0001
->OK
/*appkey 1 添加成功 */
->+MESHAPPKEYADD:0001,0
/* 解除 provisioner 节点 onoff cli model 与 appkey 0 的绑定 */
AT+MESHUNBINDAPPKEY=0x0001,0x0000,0x1001
->OK
/*appkey 0 解绑成功 */
->+MESHAPPKEYBIND:0001,0
/* 绑定 provisioner 节点 onoff cli model 与 appkey 1*/
AT+MESHBINDAPPKEY=0x0001,0x0001,0x1001
->OK
```

```
/* 使用 APPKEY 1 发送不带 ACK 开灯 */
AT+MESHONOFF=0x0002,0x01,0x00,0x0001
->OK

/* 使用 appkey1 查询 onoff status*/
AT+MESHQUERYSTA=0x0002,0x1001,0x8201,0x0000,0x0001
->OK
+MESHONOFF:0002,01

/* 使用 APPKEY 1 发送带 ACK 关灯 */
AT+MESHONOFF=0x0002,0x00,0x01,0x0001
->OK
+MESHONOFF:0002,00
```

6.6 SUB/PUB 地址配置

6.6.1 provisioner pub /node sub

```
/*node 节点 ONOFF srv model 增加订阅地址 0xC000*/
AT+MESHSETSUB=0x0002,0x1000,0xC000
->OK
/* 订阅成功 */
->+MESHSUBSET:0002,0
/* 向 0xC000 地址发送开灯命令 */
AT+MESHONOFF=0xC000,0x01,0x01
->OK
->+MESHONOFF:0002,01
```

6.6.2 provisioner sub /node pub

```
/* 查询 provisioner 节点 ONOFF cli model 已订阅地址 */
AT+MESHGETSUB=0x0001,0x1001
->OK
->+MESHSUBLIST:0001,ffff,1001,f001,c000
/*provisioner 节点 model 订阅地址最多为两个，若需增加新地址，需删除已有地址 */
AT+MESHDELSUB=0x0001,0x1001,0xC000
->OK
->+MESHSUBSET:0001,0
/*provisioner 节点 ONOFF cli model 增加订阅地址 0xC001*/
AT+MESHSETSUB=0x0001,0x1001,0xC001
->OK
/* 订阅成功 */
->+MESHSUBSET:0001,0
/* 设置 node 节点 pub 地址 0xC001 及 pub 周期 */
AT+MESHSETPUB=0x0002,0x1000,0xC001,0x03,0x44,0x00,0x00
->OK
```

```
/*provisioner 节点周期性地接收到 node 节点周期性 pub 的 onoff status*/
->+MESHPUBSET:0002,0
->+MESHONOFF:0002,01
->+MESHONOFF:0002,01
```

7. 附录

7.1 Mesh Model ID 查询表

model id	说明
config server model	0x0000
config client model	0x0001
health server model	0x0002
health client model	0x0003
generic onoff server model	0x1000
generic onoff client model	0x1001
generic level server model	0x1002
generic level client model	0x1003
light lightness server model	0x1300
lightlightness client model	0x1302
light CTL server model	0x1303
light CTL client model	0x1305

7.2 Mesh Model 状态查询表

status	query model id	query opcode	cid
friend status	0x0001	0x800f	NULL
proxy status	0x0001	0x8012	NULL
relay status	0x0001	0x8026	NULL
generic onoff status	0x1001	0x8201	NULL
generic level status	0x1003	0x8205	NULL
light lightness status	0x1302	0x824b	NULL
light lightness linear status	0x1302	0x824f	NULL
light lightness last status	0x1302	0x8253	NULL

light lightness default status	0x1302	0x8255	NULL
lightness range status	0x1302	0x8257	NULL
light CTL status	0x1305	0x825D	NULL
light CTL temperature status	0x1305	0x8261	NULL
light CTL default status	0x1305	0x8267	NULL
light CTL range status	0x1305	0x8262	NULL
vendor autoconfig status	0xd701a8	0x0001	0x01a8

蓝牙 Mesh 配网模组的二次开发指南

1. 简介

蓝牙 MESH 网络中，Provisioner 是不可缺失的角色，它可以将一个未配网的设备加入到 Mesh 网络中，为该节点分配网络密钥、IV 索引以及节点地址。同时 Provisioner 支持配置节点的各项参数，包括应用密钥，订阅和发布地址，开关 Relay/Friend/Proxy 特性等。

本文将介绍如何使用 Mesh Node 组件和 Mesh Model 组件实现 Provisioner 的功能，并且配置一个 Mesh 灯控节点，实现控制。

2. 应用开发

Provisioner 示例将实现如下几个功能：

- 发现未入网的灯控节点
- 自动对未入网的灯控节点入网
- 配置灯控节点的 APP KEY 和订阅地址
- 发送开发消息，控制亮灯 / 灭灯

注意点：

Provisioner 示例只是实现了一个简单的入网和配置过程，对于大规模的 Mesh 应用网络，开发者需要设计一个功能更加完善的 Mesh 节点管理方案，来处理并发和异常的情况。由于节点入网和配置均是异步过程，开发者可以使用状态机来维护这个过程，在本示例中，认为所有过程均是串行的，没有使用复杂的状态管理。

SDK 另外提供一个 AT Mesh Provisioner 解决方案，功能更加完善，开发者可以按

照模组的方式来使用 Provisioner 功能。

2.1 组件初始化

初始化 Mesh Model 和 Mesh Node 组件，处理相应的事件。

```
#define DEV_UUID {0xab, 0xa0, 0xe3, 0x7e, 0x17, 0xd9, 0x11, 0xe8, 0x86, 0xd1,
0x5f, 0x1c, 0xe2, 0x8a, 0xde, 0x02}

#define DEV_NAME "MESH_PROVISIONER"

/* 定义 Provisioner 支持的 SIG Models */
static struct bt_mesh_model elem0_root_models[] = {
    BT_MESH_MODEL_CFG_SRV_NULL(),
    BT_MESH_MODEL_CFG_CLI_NULL(),
    BT_MESH_MODEL_GEN_ONOFF_CLI_NULL(),
};

/* 定义 Provisioner 支持的 Vendor Models */
static struct bt_mesh_model elem0_vnd_models[] = {
    BT_MESH_MODEL_VENDOR_CLI_NULL(),
};

static struct bt_mesh_elem elements[] = {
    BT_MESH_ELEM(0, elem0_root_models, elem0_vnd_models, 0),
};

/* 定义 Provisioner 支持的 Compsition data */
static const struct bt_mesh_comp mesh_comp = {
    .cid = 0x01A8,
    .elem = elements,
    .elem_count = ARRAY_SIZE(elements),
};

/* 配置 Provisioner 参数 */
static provisioner_node provisioner_param = {
    .config.unicast_addr_local = 0x0001, //Provisioner 本地分配的 Unicast
Address
    .config.unicast_addr_start = 0x0002, //Provisioner 待分配的起始 Unicast
Address
    .config.attention_time = 5,  //Provisioning 过程中的 attention 事件，单位 s
    .config.cb = mesh_provisioner_cb, //Provision 事件回调
    .local_sub = 0xC001, //Provisioner 本地 Models 的订阅地址
    .local_pub = 0xC000, //Provisioner 本地 Models 的发布地址
};
o
/* Provisioner 节点配置参数 */
static node_config_t node_param = {
```

```
    .role = PROVISIONER,  // 节点角色为 Provisioner
    .provisioner_config = &provisioner_param, // 配置 Provisioner 参数
    .dev_uuid = DEV_UUID,  //Provisioner 的设备 UUID
    .dev_name = DEV_NAME,  //Provisioner 的名称
    .user_model_cb = mesh_model_cb, // 本地 Models 的事件回调
};

/* UUID 过滤，Provisioner 只识别 UUID 以 filter_uuid 结尾的节点，filter_uuid 为 SDK 中
Mesh Light Node demo 的 UUID 公共部分定义 */
static uint8_t filter_uuid[] = {0x11, 0xe8, 0x86, 0xd1, 0x5f, 0x1c, 0xe2,
0x8a, 0xde, 0x02};
static uuid_filter_t provisioner_filter = {
    .uuid = filter_uuid,
    .uuid_length = sizeof(filter_uuid),
    .filter_start = 6,
};

int app_main(int argc, char *argv[])
{
    int ret;
    // 板级初始化
    board_yoc_init();

    //Mesh Model 组件初始化
    ret = ble_mesh_model_init(&mesh_comp);

    if (ret) {
        LOGE(TAG, "model init fail\n");
        return ret;
    }
    //Mesh Node 组件初始化
    ret = ble_mesh_node_init(&node_param);

    if (ret < 0) {
        return ret;
    }

    // 打开 Unprovisoned 设备上报
    ble_mesh_provisioner_show_dev(1, 0);

    // 设置 Unprovisoned 设备过滤规则
    ble_mesh_provisioner_dev_filter(1, &provisioner_filter);

    ......

}
```

2.2 灯控节点入网

在 Provisioner 事件回调函数中，处理上报的未入网设备，进行入网。

```
static void mesh_provisioner_cb(mesh_provisioner_event_en event, void *p_arg)
{
    int ret;

    switch (event) {
        /* 未入网设备上报事件 */
        /* 此处上报的未入网设备是过滤后的设备，开发这可以根据需要在此处添加条件，判断是否
需要入网 */
        case BT_MESH_EVENT_RECV_UNPROV_DEV_ADV: {
            mesh_node_t *node = (mesh_node_t *)p_arg;

            /* 此处可以添加判断，此设备是否需要加入网络 */
            ......

            /* 将未入网设备加入入网列表，开始入网 */
            ble_mesh_provisioner_dev_add(node, 0);
        }
        break;

      /* 设备入网成功事件 */
        case BT_MESH_EVENT_PROV_COMP: {
            if (p_arg) {
                mesh_node_t *node = (mesh_node_t *)p_arg;

                LOGI(TAG, "provisioned, node
info:%04x,%02x,%02x,%02%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x"
\
                     , node->prim_unicast, node->element_num, node->net_idx,
\
                     node->uuid[0], node->uuid[1], node->uuid[2], node-
>uuid[3], node->uuid[4], node->uuid[5], node->uuid[6], node->uuid[7], \
                     node->uuid[8], node->uuid[9], node->uuid[10], node-
>uuid[11],
 node->uuid[12], node->uuid[13], node->uuid[14], node->uuid[15]);

                /* 其他操作 */
                ......
            }
        }
        break;

        default:
            break;
    }

}
```

2.3 灯控节点配置

```
在 Provisioner 事件回调函数中，处理灯控节点的入网成功事件，配置 APP Key
在 Mesh Model 事件回调函数中，配置灯控节点的 APP Key 绑定和 Model 的地址订阅
static void mesh_provisioner_cb(mesh_provisioner_event_en event, void *p_arg)
{
    int ret;

    switch (event) {
        ......
        case BT_MESH_EVENT_PROV_COMP: {
            if (p_arg) {
                mesh_node_t *node = (mesh_node_t *)p_arg;
                ......

                  /* 记录当前入网设备 */
                node_cfg = *node;

                /* 获取 Provisioner 本地的 APP Key */
                const uint8_t *app_key;
                extern const u8_t *bt_mesh_provisioner_local_app_key_get(u16_
t net_idx, u16_t app_idx);
                app_key = bt_mesh_provisioner_local_app_key_get(node->net_
idx, 0);

                if (NULL == app_key) {
                    LOGE(TAG, "get local appkey fail");
                    return;
                }

                       /* 配置灯控节点的 APP Key */
                ret = bt_mesh_cfg_app_key_add(node->net_idx, node->prim_
unicast, node->net_idx, 0, app_key, NULL);

                if (ret) {
                    LOGE(TAG, "add appkey fail, %d", ret);
                    return;
                }
            }
        }
        break;

        default:
            break;
    }

}

void mesh_model_cb(mesh_model_event_en event, void *p_arg)
{
```

```
    int16_t ret;

    switch (event) {
      /* APP Key 添加成功事件 */
        case BT_MESH_MODEL_CFG_APPKEY_STATUS: {
            model_message *message = (model_message *)p_arg;
            uint8_t status = message->status_data->data[0];

            if (!status) {

                LOGI(TAG, "Node[%04x] appkey add success", message->source_
addr);

                      /* 配置 APP Key 绑定 */
                ret = bt_mesh_cfg_mod_app_bind(node_cfg.net_idx, node_cfg.
prim_unicast, node_cfg.prim_unicast, 0, 0x1000, NULL);

                if (ret) {
                    LOGE(TAG, "mod bind fail %d", ret);
                }
            } else {
                LOGE(TAG, "appkey add fail %d", status);
            }
        }
        break;

      /* APP Key 绑定事件 */
        case BT_MESH_MODEL_CFG_APPKEY_BIND_STATUS: {

            model_message *message = (model_message *)p_arg;
            uint8_t status = message->status_data->data[0];

            if (!status) {
                      /* 配置订阅地址 0xC000 */
                LOGI(TAG, "Node[%04x] appkey bind success", message->source_
addr);
                ret = bt_mesh_cfg_mod_sub_add(node_cfg.net_idx, node_cfg.
prim_unicast, node_cfg.prim_unicast, 0xC000, 0x1000, NULL);

                if (ret) {
                    LOGE(TAG, "mod bind fail %d", ret);
                    return;
                }
            } else {
                LOGE(TAG, "appkey bind fail %d", status);
            }
        }

        break;
        /* 订阅事件 */
        case BT_MESH_MODEL_CFG_SUB_STATUS: {
```

```
            model_message *message = (model_message *)p_arg;
            uint8_t status = message->status_data->data[0];

            if (!status) {
                LOGI(TAG, "Node[%04x] mod sub success", message->source_
addr);

                /* 配置发布地址 0xC001 */
                struct bt_mesh_cfg_mod_pub pub = {0};
                pub.addr = 0xC001;
                pub.ttl = 7;
                ret = bt_mesh_cfg_mod_pub_set(node_cfg.net_idx, node_cfg.
prim_unicast, node_cfg.prim_unicast, 0x1000, &pub, NULL);

                if (ret) {
                    LOGE(TAG, "mod bind fail %d", ret);
                    return;
                }
            } else {
                LOGE(TAG, "appkey bind fail %d", status);
            }

        }

        break;
        /* 发布事件 */
        case BT_MESH_MODEL_CFG_PUB_STATUS: {

            model_message *message = (model_message *)p_arg;
            uint8_t status = message->status_data->data[0];

            if (!status) {
                LOGI(TAG, "Node[%04x] mod Pub success", message->source_
addr);
            } else {
                LOGE(TAG, "mod Pub fail %d", status);
            }

        }
        default:
            break;
    }

}
```

2.4 亮灯 / 灭灯控制

在应用任务中，每 5s 向组地址 0xC000 发送一次开关控制消息，并处理灯控节点返

回对的灯状态信息。

```
void mesh_model_cb(mesh_model_event_en event, void *p_arg)
{
    int16_t ret;

    switch (event) {
          ......
          /* 处理灯控节点返回的灯状态 */
        case BT_MESH_MODEL_ONOFF_STATUS: {

            model_message *message = (model_message *)p_arg;
            uint8_t status = message->status_data->data[0];

            LOGI(TAG, "Node[%04x] %s", status ? "ON" : "OFF");
        }

        break;

        default:
            break;
    }
}

int app_main(int argc, char *argv[])
{
    ......
    /* 灯控参数 */
    set_onoff_arg onoff_arg = {0};
    onoff_arg.onoff = 1;
    /* 每 5s 自动亮 / 灭灯一次 */
    while (1) {
        aos_msleep(5000);
        /* 发送需要返回 ACK 的灯控命令消息 */
        ble_mesh_generic_onoff_cli_publish(&elem0_root_models[2], &onoff_arg,
1);
        onoff_arg.onoff = !onoff_arg.onoff;
    }
    .....
}
```

3. 示例运行

3.1 编译

- 进入 SDK 根目录，编译 mesh provisioner 示例

```
  $ ./build.sh defconfigs/defconfig_ch6121_evb_mesh_at_provisioner
applications/bluetooth/mesh_provsioner/ j64
```

- 进入 SDK 根目录，编译 mesh light node 示例

```
  $ ./build.sh defconfigs/defconfig_ch6121_evb_mesh_node applications/
bluetooth/mesh_light_node j64
```

3.2 运行

- 下载 applications/bluetooth/meshprovsioner/generated/totalimage.hexf 至开发板 1
- 下载 applications/bluetooth/meshlightnode/generated/total_image.hexf 至开发板 2
- 镜像下载可参考《CB6121 快速上手手册》
- 连接串口调试工具 , 配置为
- 波特率 : 115200, 数据位 : 8, 校验位 : None, 停止位 : 1, 流控 : None
- 按 RESET 按键，复位开发板
- 成功启动后，开发板 1 串口将输出

```
Welcome to CLI...
// 扫描到的未入网灯控设备
[     7.435000][I][MESH_PROVISIONER]unprov
dev:044433229900,00,04443322990011e886d15f1ce28ade02,00,02
// 自动入网
[    21.292000][I][MESH_PROVISIONER]provisioned, node
info:0002,01,00,04443322990011e886d15f1ce28ade02
// 自动配置
[    24.076000][I][MESH_PROVISIONER]Node[0002] appkey add success

[    24.147000][I][MESH_PROVISIONER]Node[0002] appkey bind success

[    24.219000][I][MESH_PROVISIONER]Node[0002] mod sub success

[    24.500000][I][MESH_PROVISIONER]Node[0002] mod Pub success
// 接受到灯控节点灯状态信息
[    35.844000][I][MESH_PROVISIONER]Node[0002] OFF
[    40.845000][I][MESH_PROVISIONER]Node[0002] ON
```

- 开发板 2 灯控节点串口输出

```
[     0.004000][I][init    ]Build:Apr  2 2020,23:38:27

[     0.017000][I][init    ]find 9 partitions

[     0.222000][I][MESH_RESET]reset_by_repeat_init, number = 1

Welcome to CLI...

> [     0.243000][I][DEMO    ]Mesh light node demo

// 自动配网
[   213.972000][I][BT_MESH_NODE]Provisioning link opened on PB-GATT
[   218.780000][I][BT_MESH_NODE]Provisioning link closed on PB-GATT
[   218.811000][I][BT_MESH_NODE]provisioning complete netid 0, primary
element address 0x2
[   218.828000][I][DEMO    ]prov complete 0002
// 灯控消息
[   222.074000][I][DEMO    ]src:0x0001,led:OFF
[   227.083000][I][DEMO    ]src:0x0001,led:ON
```

附录

附录一：蓝牙 Mesh SDK 快速上手演示视频

网址：https://occ.t-head.cn/community/course/detail?id=3770446941540913152&name=BLE%20Mesh%20SDK%E5%BF%AB%E9%80%9F%E4%B8%8A%E6%89%8B&inviteUserId=3769057297817612288

附录二：蓝牙 Mesh 配网及控制

https://occ.t-head.cn/community/course/detail?id=377044731547989196 8&name=SIG%20MESH%E9%85%8D%E7%BD%91%E5%92%8C%E6%8E%A7%E5%88%B6&inviteUserId=3769057297817612288

附录三：蓝牙 Mesh SDK API 手册

1. 概述

本文主要介绍 Mesh 协议栈和 Mesh 组件 API。

Mesh 协议栈部分将分章节介绍

- Access 层 API
- Config Model API
- Health Model API
- Provisioner API

Mesh 组件部分将分章节介绍

- Mesh Node 组件 API
- Mesh Models 组件 API
- Mesh Provisioner 组件 API

2. MESH 协议栈 API

2.1 Access API

本章节介绍了蓝牙 Mesh Access 接口，利用这些接口可以实现蓝牙 MESH 协议栈初始化、特性配置、消息发送等功能。

```
btmeshinit
```

- 函数原型

```
int bt_mesh_init(const struct bt_mesh_prov *prov,
                 const struct bt_mesh_comp *comp,
```

```
const struct bt_mesh_provisioner *provisioner)
```

- 功能描述

蓝牙 MESH 协议栈初始化

- 参数描述

IN/OUT	NAME	DESC
[in]	const struct bt*mesh*prov *prov	Provisioning 配置参数
[in]	const struct bt*mesh*comp *comp	Composition 配置参数
[in]	const struct bt*mesh*provisioner *provisioner	Provisioner 配置参数，设备无需支持配网器功能时，设置为 NULL
struct btmeshprov（结构体）定义		
const u8_t *uuid	待入网设备的通用唯一标识符	
const char *uri	待入网设备的统一资源标识符	
bt*mesh*prov*oob*info*t oob*info	OOB 交换方式，参见 bt*mesh*prov*oob*info_t 枚举定义	
const u8*t *static*val	静态 OOB 值	
u8*t static*val_len	静态 OOB 值的长度	
u8*t output*size	输出 OOB 长度	
u16*t output*actions	参见 bt*mesh*output*action*t 枚举定义	
u8*t input*size	输入 OOB 长度	
u16*t input*actions	参见 bt*mesh*input*action*t 枚举定义	
int (*output*number)(bt*mesh*output*action*t *act, u32*t num)	回调函数，用于 OOB 数字信息的显示	
int (*output_string)(const char *str)	回调函数，用于 OOB 字符串信息的显示	
int (*input)(bt*mesh*input*action*t act, u8_t size)	回调函数，用于 OOB 信息输入的通知	
void (*link*open)(bt*mesh*prov*bearer_t bearer)	回调函数，通知节点连接已开启	
void (*link*close)(bt*mesh*prov*bearer_t bearer)	回调函数，通知节点连接已关闭	

void (*complete)(u16*t net*idx, u16_t addr)	回调函数，通知节点配网成功
void (*reset)(void)	回调函数，通知节点配网信息重置成功，节点可以重新入网。

btmeshprovoobinfo_t（枚举）定义

BT*MESH*PROV*OOB*NONE = 0x00	无
BT*MESH*PROV*OOB*OTHER = BIT(0)	其他
BT*MESH*PROV*OOB*URI = BIT(1)	URI
BT*MESH*PROV*OOB*2D_CODE = BIT(2)	二维码
BT*MESH*PROV*OOB*BAR_CODE = BIT(3)	条形码
BT*MESH*PROV*OOB*NFC = BIT(4)	NFC
BT*MESH*PROV*OOB*NUMBER = BIT(5)	数字
BT*MESH*PROV*OOB*STRING = BIT(6)	字符串
BT*MESH*PROV*OOB*ON_BOX = BIT(11)	位于包装盒外
BT*MESH*PROV*OOB*IN_BOX = BIT(12)	位于包装盒内
BT*MESH*PROV*OOB*ON_PAPER = BIT(13)	位于纸上
BT*MESH*PROV*OOB*IN_MANUAL = BIT(14)	位于盒内说明书上
BT*MESH*PROV*OOB*ON_DEV = BIT(15)	位于设备上

btmeshoutputactiont（枚举）定义

BT*MESH*NO_OUTPUT = 0x00	无
BT*MESH*BLINK = BIT(0)	闪烁
BT*MESH*BEEP = BIT(1)	蜂鸣
BT*MESH*VIBRATE = BIT(2)	马达振动
BT*MESH*DISPLAY_NUMBER = BIT(3)	显示数字
BT*MESH*DISPLAY_STRING = BIT(4)	显示字符串

btmeshinputactiont（枚举）定义

BT*MESH*NO_INPUT = 0x00	无
BT*MESH*PUSH = BIT(0)	按压
BT*MESH*TWIST = BIT(1)	旋转
BT*MESH*ENTER_NUMBER = BIT(2)	输入数字
BT*MESH*ENTER_STRING = BIT(3)	输入字符串

struct btmeshcomp（结构体）定义	
uint16_t cid	Company ID, 由 Bluetooth SIG 分配
uint16_t pid	产品标识符
uint16_t vid	版本标识符
size*t elem*count	节点元素个数
bt*mesh*elem *elem	元素列表

btmeshelem（结构体）定义	
u16_t addr	入网成功后分配的 unicast 地址
u16*t grop*addr	组地址
u16_t loc	Location Descriptor (GATT Bluetooth Namespace Descriptors)
u8*t model*count	元素中 SIG Model 的个数
u8*t vnd*model_count	元素中 Vendor Model 的个数
struct bt*mesh*model * models	SIG Model 数组，可使用 BT_*MESH*MODEL 宏进行定义
struct bt*mesh*model * vnd_models	Vendor Model 数组，可使用 BT_*MESH*MODEL_VND 宏进行定义

btmeshmodel（结构体）定义	

union {
 u16*t id;
 struct {
 u16*t* company;
 u16_t id;
 } vnd;
 };
	SIG Model ID，占用 1-2 个字节；
	 Vendor Model ID，占用 2 个字节。

u8*t elem*idx	记录该 Model 所归属的节点元素的索引值
u8*t mod*idx	记录该 Model 的索引值
u16_t flags	记录该 Model 的状态
struct bt*mesh*model_pub * pub	Publication 的配置参数，无需支持消息发布时，可设置为 NULL
u16*t keys[CONFIG*BT*MESH*MODEL*KEY*COUNT]	APP Key 列表，记录该 Model 绑定的 APP key 列表
u16*t groups[CONFIG*BT*MESH*MODEL*GROUP*COUNT]	订阅列表，记录该 Model 订阅的组地址和虚拟地址
void *user_data	该 Model 关联的用户数据指针

struct btmeshmodel_pub（结构体）定义	
struct bt*mesh*model *mod	该消息发布配置所归属的 Model 指针
u16_t addr	发布的地址
u16_t key	APP Key 索引

u8_t ttl	消息的生存时间
u8_t retransmit	重传计数
u8_t period	发布周期
u8*t period*div:4, cred:1, fast_period:1 count:3	分频系数 FriendShip 建立凭证 使能快速分频 剩余重传次数
u32*t period*start	当前周期的起始时间
struct net*buf*simple *msg	发布的消息结构体指针
int (*update)(struct bt*mesh*model *mod)	回调函数，可实现周期性的状态更新；无需支持周期性发布时，可设置为 NULL
struct k*delayed*work timer	定时器，用来实现周期性的发布

btmeshmodel_op（结构体）定义

u32_t opcode	消息操作类型，使用 BT*MESH*-MODEL*OP** 宏进行定义 使用 BT*MESH*MODEL*OP*1 宏进行定义时，占用 1 个字节； 使用 BT*MESH*MODEL*OP*2 宏进行定义时，占用 2 个字节； 使用 BT*MESH*MODEL*OP*3 宏进行定义时，包含 Company ID，占用 4 个字节。
size*t min*len	消息的最小长度
void (*func)(struct bt*mesh*model *model, struct bt*mesh*msg*ctx *ctx, *struct net*buf_simple *buf)	消息处理回调函数 ctx 为发送消息的上下文（参见 bt*mesh*msg_ctx）， buf 为需要发送的消息
void (*func2)(struct bt*mesh*model *model, struct bt*mesh*msg*ctx *ctx, *struct net*buf*simple *buf, u32*t opcode)	消息处理回调函数 ctx 为发送消息的上下文（参见 bt*mesh*msg_ctx）， buf 为需要发送的消息

btmeshmsg_ctx（结构体）定义

u16*t net*idx	所属 MESH 网络的 Network Key 索引
u16*t app*idx	所属 MESH 网络的 APP Key 索引
u16_t addr	远端地址
u16*t recv*dst	接收消息的目的地址，发送消息时该值无效。
u8*t recv*ttl:7	接收消息的 TTL 值
u8*t send*rel:1	通过段确认发送可靠
u8*t send*ttl	发送消息的 TTL 值

btmeshprovisioner（结构体）定义

const u8*t *prov*uuid	配网器的 UUID
const u16*t prov*unicast_addr	配网器 Primary Element 的地址
u16*t prov*start_address	可分配的单播地址的起始地址
u8*t prov*attention	配网邀请阶段的提示定时器
u8*t prov*algorithm	配网算法
u8*t prov*pub*key*oob	配网器的公钥 OOB 信息
int (*prov*pub*key*oob*cb)(u8*t remote*pub_key[64])	回调函数，用于读取设备的公钥 OOB 信息。 返回值：0 – 成功；非 0 – 失败
u8*t *prov*static*oob*val	配网器的静态 OOB 信息
u8*t prov*static*oob*len	配网器的静态 OOB 信息长度
int (*prov*input*num)(bt*mesh*output*action*t act, u8_t size)	回调函数，用于输入 OOB 数字信息 act：设备的输出行为 size：设备的 OOB 信息长度
int (*prov*output*num)(bt*mesh*input*action*t act, u8_t size)	回调函数，用于输出的 OOB 数字信息 act：设备的输入行为 size：设备的 OOB 信息长度
u8_t flags	密钥更新和 IV 更新的标志位 BIT0：密钥更新标记 0：False 1：True BIT1：IV 更新标记 0: Normal Operation 1: IV Update active BIT2 ~ BIT7: RFU
u32*t iv*index	所属 MESH 网络的 IV 索引
void (*prov*link*open)(bt*mesh*prov*bearer*t bearer)	回调函数，通知配置连接打开事件
void (*prov*link*close)(bt*mesh*prov*bearer*t bearer, u8_t reason)	回调函数，通知配置连接关闭事件
void (*provcomplete)(int nodeidx， const u8t deviceuuid[16]，u16t unicastaddr， u8t elementnum， u16t netkeyidx， bool gatt_flag)	回调函数，通知配网器当前节点配网成功，且该节点已经被正确分配了 Network key 索引和首要元素地址。 nodeidx：在已配置节点队列中的节点索引。 deviceuuid：已配置节点的 uuid。 unicast_addr：已配置节点的单播地址。

- 返回值

返回值	
0	初始化成功
非 0	初始化失败

- 注意事项

  MESH 协议栈初始化成功后，需要调用 btmeshprov_enable ( ) 函数，开启 Unprovisioning Device Beacon 广播

```
btmeshreset
```

- 函数原型

```
void bt_mesh_reset(void)
```

- 功能描述

  将节点移除出当前的 MESH 网络

- 参数描述

  无

- 返回值

  无

- 注意事项

  节点被移除出当前 MESH 网络后，可以通过调用 btmeshprov_enable ( ) 函数，重新开启入网请求广播

```
btmeshsuspend
```

- 函数原型

```
int bt_mesh_suspend(void)
```

- 功能描述

  暂停该节点在 mesh 网络中的功能

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  节点长时间进入暂停状态后，配网器将认为该节点永久掉线。

```
btmeshresume
```

- 函数原型

```
int bt_mesh_resume(void)
```

- 功能描述

  恢复暂停节点的功能

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  在调用 btmeshsuspend() 后，可使用 btmeshresume() 函数，恢复节点的网络功能

```
btmeshprovision
```

- 函数原型

```
int bt_mesh_provision(const u8_t net_key[16], u16_t net_idx,
                      u8_t flags, u32_t iv_index, u16_t addr,
                      const u8_t dev_key[16])
```

- 功能描述

```
MESH device 收到prov data 后，保存net key 、unicast addr 、devkey 等信息
```

- 参数描述

IN/OUT	NAME	DESC
[in]	const u8*t netkey*[16]	Network key
[in]	u16*t netidx*	Network key 索引
[in]	u8_t flags	配置标志位
[in]	u32*t ivindex*	IV 索引
[in]	u16_t addr	Primary Element 地址
[in]	const u8*t devkey*[16]	设备秘钥

返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本接口用于获取到 provisioner 发送的 prov data 后，device 对相关信息进行保存

```
btmeshis_provisioned
```

- 函数原型

```
bool bt_mesh_is_provisioned(void)
```

- 功能描述

获取节点配网状态

- 参数描述

  无

- 返回值

返回值	
0	未配网
1	已成功配网

- 注意事项

  无

```
btmeshiv_update
```

- 函数原型

```
int bool bt_mesh_iv_update(void)
```

- 功能描述

更新网络中的 IV 值

- 参数描述

  无

- 返回值

返回值	
0	开启 IV 更新失败
1	开启 IV 更新成功

- 注意事项

  无

```
btmeshlpn_set
```

- 函数原型

```
int bt_mesh_lpn_set(bool enable)
```

- 功能描述

开启或关闭低功耗特性

- 参数描述

IN/OUT	NAME	DESC
[in]	bool enable	是否打开低功耗

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  该函数是一个具有实时性的函数，可以随时打开 / 关闭节点的低功耗特性，使用前节点需已完成 prov

```
btmeshlpn_poll
```

- 函数原型

```
int bt_mesh_lpn_poll(void)
```

- 功能描述

发送一个 friend poll 请求消息

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  当节点未建立 Friend 关系时，返回失败

```
btmeshlpnsetcb
```

- 函数原型

```
void bt_mesh_lpn_set_cb(void (*cb)(u16_t friend_addr, bool established))
```

- 功能描述

  回调函数注册，用于通知 Friendship 的改变，如建立或丢失

- 参数描述

IN/OUT	NAME	DESC
[in]	void (*cb)(u16t friendaddr, bool established)	注册的回调函数

- 返回值

  无

- 注意事项

  无

```
btmeshprovisioner_enable
```

- 函数原型

```
int bt_mesh_provisioner_enable(bt_mesh_prov_bearer_t bearers)
```

- 功能描述

开启配网器功能

- 参数描述

IN/OUT	NAME	DESC
[in]	btmeshprovbearert bearers	广播承载类型
btmeshprovbearert（枚举）定义		
BTMESHPROV_ADV = BIT(0)	ADV 承载	
BTMESHPROV_GATT = BIT(1)	GATT 承载	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需调用 btmeshinit() 进行协议栈初始化

```
btmeshprovisioner_disable
```

- 函数原型

```
int bt_mesh_provisioner_disable(bt_mesh_prov_bearer_t bearers)
```

- 功能描述

  关闭配网器功能

- 参数描述

IN/OUT	NAME	DESC
[in]	btmeshprovbearert bearers	广播承载类型

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  无

```
btmeshinput_string
```

- 函数原型

```
int bt_mesh_input_string(const char *str)
```

- 功能描述

配置 OOB 字符串信息

- 参数描述

IN/OUT	NAME	DESC
[in]	const char *str	OOB 字符串

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  prov 阶段使用，在收到配网器的 OOB 字符串输入请求时，可通过此函数传入设置的 OOB 信息

```
btmeshinput_number
```

- 函数原型

```
int bt_mesh_input_string(u32_t num)
```

- 功能描述

  配置 OOB 数字信息

- 参数描述

IN/OUT	NAME	DESC
[in]	u32_t num	OOB 数字

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  prov 阶段使用，在收到配网器的 OOB 数字输入请求时，可通过此函数传入设置的 OOB 信息

```
btmeshprov_enable
```

- 函数原型

```
int bt_mesh_prov_enable(bt_mesh_prov_bearer_t bearers)
```

- 功能描述

  开启配网功能，等待配网

- 参数描述

IN/OUT	NAME	DESC
[in]	btmeshprovbearert bearers	广播承载类型

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  无

```
btmeshprov_disable
```

- 函数原型

```
int bt_mesh_prov_disable(bt_mesh_prov_bearer_t bearers)
```

- 功能描述

  关闭配网功能

- 参数描述

IN/OUT	NAME	DESC
[in]	btmeshprovbearert bearers	广播承载类型

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  无

```
btmeshmodel_send
```

- 函数原型

```
int bt_mesh_model_send(struct bt_mesh_model *model,
                       struct bt_mesh_msg_ctx *ctx,
                       struct net_buf_simple *msg,
                       const struct bt_mesh_send_cb *cb,
                       void *cb_data)
```

- 功能描述

  消息发送

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshmodel *model	发送消息的 Model 指针
[in]	struct btmeshmsg_ctx *ctx	消息的上下文信息，包括
[in]	struct netbufsimple *msg	消息结构体指针
[in]	const struct btmeshsend_cb *cb	回调函数，通知消息发送的开始和结束事件
[in]	void *cb_data	回调函数的数据指针
struct btmeshsend_cb（结构体）定义		
void (*start)(u16t duration, int err, void *cbdata)	回调函数，通知消息发送开始事件	
void (*end)(int err, void *cb_data)	回调函数，通知消息发送结束事件	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  调用该接口前，节点应完成 prov, 并获取 appkey

```
btmeshmodel_publish
```

- 函数原型

```
int bt_mesh_model_publish(struct bt_mesh_model *model)
```

- 功能描述

  消息发布，消息将被发送到配置的组地址或者虚拟地址中

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshmodel *model	发送消息的 Model

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  在调用该函数之前，应该确保 Model 中发布的消息（btmeshmodelpub.msg）包含一个正确的信息。此 API 只是用来发布非周期性的消息。如果需要发送周期性消息，只需要保证当 btmeshmodelpub.update 函数回调时，btmeshmodel_pub.msg 有合理的值。

```
btmeshmodel_elem
```

- 函数原型

```
struct bt_mesh_elem *bt_mesh_model_elem(struct bt_mesh_model *mod);
```

- 功能描述

  获取 Model 所在 elem 指针

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshmodel *model	查询的 Model 指针

- 返回值

  返回 Model 的元素列表指针

- 注意事项

  无

2.2 Config Model API

本章节介绍了蓝牙 Mesh Config Model 接口，利用这些接口可以实现蓝牙 MESH 的配置信息的设置和读取。

```
btmeshcfgcompdata_get
```

- 函数原型

```
int bt_mesh_cfg_comp_data_get(u16_t net_idx,
                              u16_t addr, u8_t page,
                              u8_t *status,
                              struct net_buf_simple *comp)
```

- 功能描述

  获取节点的 Composition 数据

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8_t page	指定页，默认为 0
[out]	u8_t *status	节点的 Composition 数据的状态 0：正常 非 0：无效

[out]	struct netbufsimple *comp	获取到的 Composition 数据指针

- 返回值

返回值	
0	获取成功
非 0	获取失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgbeaconget
```

- 函数原型

```
int bt_mesh_cfg_beacon_get(u16_t net_idx, u16_t addr, u8_t *status)
```

- 功能描述

  获取 Beacon 状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[out]	u8_t *status	节点的 Beacon 状态 0：关闭 1：开启

- 返回值

返回值	
0	获取成功
非 0	获取失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgbeaconset
```

- 函数原型

```
int bt_mesh_cfg_beacon_set(u16_t net_idx, u16_t addr, u8_t val, u8_t *status)
```

- 功能描述

  设置节点的 Beacon 状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8_t val	设置状态值 1：开启 0：关闭
[out]	u8_t *status	节点的 Beacon 状态 0：关闭 1：开启

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgttlget
```

- 函数原型

```
int bt_mesh_cfg_ttl_get(u16_t net_idx, u16_t addr, u8_t *ttl)
```

- 功能描述

  获取节点数据包的默认 TTL 值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[out]	u8_t *ttl	网络数据包的 TTL 值

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgttlset
```

- 函数原型

```
int bt_mesh_cfg_ttl_set(u16_t net_idx, u16_t addr, u8_t val, u8_t *ttl)
```

- 功能描述

  设置对端设备网络数据包默认 TTL 值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8_t val	设置的 TTL 值
[out]	u8_t *ttl	返回实际的 TTL 值

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgfriendget
```

- 函数原型

```
int bt_mesh_cfg_friend_get(u16_t net_idx, u16_t addr, u8_t *status)
```

- 功能描述

  获取对端节点 friend 属性状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[out]	u8_t *status	freind 状态，0 为关闭状态，1 为开启状态

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgfriendset
```

- 函数原型

```
int bt_mesh_cfg_friend_set(u16_t net_idx, u16_t addr, u8_t val, u8_t *status)
```

- 功能描述

  设置对端节点 friend 状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8_t val	设置的状态值，1：开启；0：关闭
[out]	u8_t *ttl	返回 friend 状态，1：开启；0：关闭

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

  friend 特性和 lpn 特性不能同时打开

```
btmeshcfggattproxy_get
```

- 函数原型

```
int bt_mesh_cfg_gatt_proxy_get(u16_t net_idx, u16_t addr, u8_t *status)
```

- 功能描述

  获取对端节点代理属性状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[out]	u8_t *status	代理节点状态 0：关闭 1：开启

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfggattproxy_set
```

- 函数原型

```
int bt_mesh_cfg_friend_set(u16_t net_idx, u16_t addr, u8_t val, u8_t *status)
```

- 功能描述

  设置对端节点代理属性状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8_t val	设置的状态值，1：开启；0：关闭
[out]	u8_t *status	返回代理节点状态，1：开启；0：关闭

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgrelayget
```

- 函数原型

```
int bt_mesh_cfg_relay_get(u16_t net_idx,
                          u16_t addr,
u8_t *status,
                                    u8_t *transmit)
```

- 功能描述

  获取对端节点中继属性状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[out]	u8_t *status	中继状态值，1：开启；0：关闭
[out]	u8_t *transmit	网络传输计数，用于控制来自节点的网络 PDU 的消息传输数量

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgrelayset
```

- 函数原型

```
int bt_mesh_cfg_relay_set(u16_t net_idx,
                          u16_t addr,
                          u8_t new_relay,
                          u8_t new_transmit,
                          u8_t *status,
                          u8_t *transmit)
```

- 功能描述

  设置对端节点中继属性状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u8t newrelay	需要设置的 Relay 状态 1：开启 0：关闭
[in]	u8t newtransmit	需要设置的网络传输计数值
[out]	u8_t *status	返回中继状态值
[out]	u8_t *transmit	返回网络传输计数

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgnetkey_add
```

- 函数原型

```
int bt_mesh_cfg_net_key_add(u16_t net_idx,
                            u16_t addr,
                            u16_t key_net_idx,
                            const u8_t net_key[16],
                            u8_t *status);
```

- 功能描述

  增加一个 netkey

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t keynet_idx	net key 的索引
[in]	const u8t netkey[16]	需要设置的 Netkey 的值
[out]	u8_t *status	返回状态； 0：成功 非 0：失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgappkey_add
```

- 函数原型

```
int bt_mesh_cfg_app_key_add(u16_t net_idx,
                            u16_t addr,
                            u16_t key_net_idx,
                            u16_t key_app_idx,
                            const u8_t app_key[16],
                            u8_t *status)
```

- 功能描述

  增加一个 APP Key

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t keynet_idx	net key 的索引
[in]	u16t keyapp_idx	APP Key 的索引
[in]	const u8t appkey[16]	需要设置的 APPkey 的值
[out]	u8_t *status	返回状态； 0：成功 非 0：失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodapp_bind
```

- 函数原型

```
int bt_mesh_cfg_mod_app_bind(u16_t net_idx,
                             u16_t addr,
```

```
                                u16_t elem_addr,
                                u16_t mod_app_idx,
                                u16_t mod_id,
                                u8_t *status)
```

- 功能描述

  绑定一个 SIG Model 的 APP Key

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t modapp_idx	APP Key 的索引
[in]	u16t modid	SIG Model 的 ID
[out]	u8_t *status	返回状态； 0：绑定成功 非 0：绑定失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodappbindvnd
```

- 函数原型

```
int bt_mesh_cfg_mod_app_bind_vnd(u16_t net_idx,
                                 u16_t addr,
                                 u16_t elem_addr,
                                 u16_t mod_app_idx,
                                 u16_t mod_id,
```

```
                                    u16_t cid,
                                    u8_t *status)
```

- 功能描述

  绑定一个 Vendor Model 的 APP Key

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t modapp_idx	APP Key 的索引
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[out]	u8_t *status	返回状态；0：绑定成功 非 0：绑定失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodpub_get
```

- 函数原型

```
int bt_mesh_cfg_mod_pub_get(u16_t net_idx, u16_t addr,
                            u16_t elem_addr,
                                      u16_t mod_id,
                            struct bt_mesh_cfg_mod_pub *pub,
                                                    u8_t *status)
```

- 功能描述

  获取 SIG Model 的发布配置参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t modid	SIG Model 的 ID
[out]	struct btmeshcfgmodpub *pub	发布配置参数，参见 struct bt*mesh*cfg*mod*pub（结构体）定义
[out]	u8_t *status	返回状态；0：获取成功 非 0：获取失败

struct btmeshcfgmodpub（结构体）定义

u16_t addr	发布地址
u16t appidx	APP Key 索引
bool cred_flag	安全机制 0：使用 Master 机制 1：使用 Friend 机制
u8_t ttl	消息的生存时间
u8_t period	发布周期
u8_t transmit	重传次数 = transmit & 0x0111 重传间隔 =(transmit / 8 +1)*10ms

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodpubgetvnd
```

- 函数原型

```
int bt_mesh_cfg_mod_pub_get_vnd(u16_t net_idx,
                                u16_t addr,
                                u16_t elem_addr,
                                u16_t mod_id,
                                u16_t cid,
                                struct bt_mesh_cfg_mod_pub *pub,
                                u8_t *status)
```

- 功能描述

  获取 Vendor Model 的发布配置参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[out]	struct btmeshcfgmodpub *pub	发布配置参数，参见 struct btmeshcfgmodpub（结构体）定义
[out]	u8_t *status	返回状态；0：获取成功 非 0：获取失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsub_del
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_del(u16_t net_idx,
                            u16_t addr,
                            u16_t elem_addr,
                            u16_t sub_addr,
                            u16_t mod_id,
                            u8_t *status)
```

- 功能描述

  删除 SIG Model 消息订阅地址

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t subaddr	订阅地址
[in]	u16t modid	SIG Model 的 ID
[out]	u8_t *status	返回状态；0：取消成功 非 0：取消失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubdelvnd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_del_vnd(u16_t net_idx,
                                u16_t addr,
```

```
                                    u16_t elem_addr,
                                    u16_t sub_addr,
                                    u16_t mod_id,
                                    u16_t cid,
                                    u8_t *status)
```

- 功能描述

  删除 vendor Model 的消息订阅地址

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t subaddr	订阅地址
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[out]	u8_t *status	返回状态；0：取消成功 非 0：取消失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsub_overwrite
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_overwrite(u16_t net_idx,
                                  u16_t addr,
                                  u16_t elem_addr,
```

```
                                    u16_t sub_addr,
                                    u16_t mod_id,
                                    u8_t *status)
```

- 功能描述

  擦除 SIG Model 所有订阅非虚拟地址，并增加 sub_addr 订阅地址

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t subaddr	订阅地址
[in]	u16t modid	SIG Model 的 ID
[out]	u8_t *status	返回状态；0：重写成功 非 0：重写失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsuboverwritevnd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_overwrite_vnd(u16_t net_idx,
                                    u16_t addr,
                                    u16_t elem_addr,
                                    u16_t sub_addr,
                                    u16_t mod_id,
                                    u16_t cid,
                                    u8_t *status)
```

- 功能描述

  擦除 Vendor Model 所有订阅非虚拟地址，并增加新的订阅地址 sub_addr

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	u16t subaddr	订阅地址
[in]	u16t modid	SIG Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[out]	u8_t *status	返回状态；0：重写成功 非 0：重写失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvaadd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_add(u16_t net_idx,
                               u16_t addr,
                               u16_t elem_addr,
                               const u8_t label[16],
                               u16_t mod_id,
                               u16_t *virt_addr,
                               u8_t *status)
```

- 功能描述

  针对 SIG Model 增加一个虚拟地址的消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	SIG Model 的 ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：订阅成功 非 0：订阅失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvaadd_vnd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_add_vnd(u16_t net_idx,
                                   u16_t addr,
                                   u16_t elem_addr,
                                   const u8_t label[16],
                                   u16_t mod_id,
                                   u16_t cid,
                                   u16_t *virt_addr,
                                   u8_t *status)
```

- 功能描述

  针对 Vendor Model 增加一个虚拟地址的消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：订阅成功 非 0：订阅失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvadel
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_del(u16_t net_idx,
                               u16_t addr,
                               u16_t elem_addr,
                               const u8_t label[16],
                               u16_t mod_id,
                               u16_t *virt_addr,
                               u8_t *status)
```

- 功能描述

  取消一个 SIG Model 的虚拟地址消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：取消订阅成功 非 0：取消订阅失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvadel_vnd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_del_vnd(u16_t net_idx,
                                   u16_t addr,
                                   u16_t elem_addr,
                                   const u8_t label[16],
                                   u16_t mod_id,
                                   u16_t cid,
                                   u16_t *virt_addr,
                                   u8_t *status)
```

- 功能描述

  取消一个 Vendor Model 的虚拟地址消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：取消订阅成功 非 0：取消订阅失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvaoverwrite
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_overwrite(u16_t net_idx,
                                     u16_t addr,
                                     u16_t elem_addr,
                                     const u8_t label[16],
                                     u16_t mod_id,
                                     u16_t *virt_addr,
                                     u8_t *status)
```

- 功能描述

  重写一个 SIG Model 的虚拟地址消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	SIG Model 的 ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：重写成功 非 0：重写失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfgmodsubvaoverwrite_vnd
```

- 函数原型

```
int bt_mesh_cfg_mod_sub_va_overwrite_vnd(u16_t net_idx,
                                         u16_t addr,
                                         u16_t elem_addr,
                                         const u8_t label[16],
                                         u16_t mod_id,
                                         u16_t cid,
                                         u16_t *virt_addr,
                                         u8_t *status)
```

- 功能描述

  重写一个 Vendor Model 的虚拟地址消息订阅

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t elemaddr	元素地址
[in]	const u8_t label[16]	Label UUID
[in]	u16t modid	Vendor Model 的 ID
[in]	u16_t cid	Vendor Model 的 Company ID
[in]	u16t *virtaddr	返回 Label UUID 对应的虚拟地址
[out]	u8_t *status	返回状态；0：重写成功 非 0：重写失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfghbpub_set
```

- 函数原型

```
int bt_mesh_cfg_hb_pub_set(u16_t net_idx,
                           u16_t addr,
                           const struct bt_mesh_cfg_hb_pub *pub,
                           u8_t *status)
```

- 功能描述

  设置 Heartbeat 消息发布参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	const struct btmeshcf-ghbpub *pub	HeartBeat 消息结构体指针，参见 struct btmeshcfghbpub（结构体）定义
[out]	u8_t *status	返回状态；0：设置成功 非 0：设置失败
struct btmeshcfghbpub（结构体）定义		
u16_t dst	心跳包的目标地址	
u8_t count	剩余的待发送的心跳包个数	
u8_t period	心跳包发送间隔	
u8_t ttl	心跳包的生存时长	
u8_t feat	心跳包发送时，改变的特征。参见 feat 参数说明	
u16t netidx	NetKey 索引值	
Feat 参数说明		
Relay = BIT(0)	0：未发生 Relay 变化；1：发生 Relay 变化	
Proxy = BIT(1)	0：未发生 Proxy 变化；1：发生 Proxy 变化	
Friend = BIT(2)	0：未发生 Friend 变化；1：发生 Friend 变化	
Low Power = BIT(3)	0：未发生 low power 变化；1：发生 low power 变化	
RFU = BIT(4) ~ BIT(15)	预留	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfghbpub_get
```

- 函数原型

```
int bt_mesh_cfg_hb_pub_get(u16_t net_idx,
                           u16_t addr,
                           struct bt_mesh_cfg_hb_pub *pub,
                           u8_t *status)
```

- 功能描述

获取 Heartbeat 消息发布参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	const struct btmeshcfghbpub *pub	HeartBeat 消息结构体指针，参见 struct btmeshcfghbpub（结构体）定义
[out]	u8_t *status	返回状态；0：获取成功 非 0：获取失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfghbsub_set
```

- 函数原型

```
int bt_mesh_cfg_hb_sub_set(u16_t net_idx,
                           u16_t addr,
                           struct bt_mesh_cfg_hb_sub *sub,
                           u8_t *status)
```

- 功能描述

  设置 Heartbeat 消息订阅参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	struct btmeshcfghbsub *sub	HeartBeat 消息结构体指针，参见 struct btmesh-cfghbsub（结构体）定义
[out]	u8_t *status	返回状态；0：设置成功 非 0：设置失败

struct btmeshcfghbsub（结构体）定义

u16_t src	心跳包的源地址
u16_t dst	心跳包的目的地址
u8_t period	心跳包发送间隔
u8_t count	已收到的心跳包个数
u8_t min	心跳包的最小 TTL 值，范围：0x00 ~ 0x7F
u8_t max	心跳包的最大 TTL 值，范围：0x00 ~ 0x7F

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

```
btmeshcfghbsub_get
```

- 函数原型

```
int bt_mesh_cfg_hb_sub_get(u16_t net_idx,
                           u16_t addr,
```

```
                                struct bt_mesh_cfg_hb_sub *sub,
                                u8_t *status)
```

- 功能描述

  获取 Heartbeat 消息订阅参数

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	struct btmeshcfghbsub *sub	HeartBeat 消息结构体指针，参见 struct bt-meshcfghbsub（结构体）定义
[out]	u8_t *status	返回状态；0：设置成功 非 0：设置失败

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口的前提是当前节点支持 cfg cli model，对端设备则支持 cfg srv model

2.3 Health Model API

本章节介绍了蓝牙 Mesh Health Model 接口，利用这些接口可以实现蓝牙 MESH 节点健康状态的操作。

```
btmeshhealthcliset
```

- 函数原型

```
int bt_mesh_health_cli_set(struct bt_mesh_model *model)
```

- 功能描述

  设置节点为 Health Model Client

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshmodel *model	Health Model Client 配置参数，参见 ACCESS 中 struct btmeshmodel（结构体）定义

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model

```
btmeshhealthfaultget
```

- 函数原型

```
int bt_mesh_health_fault_get(u16_t net_idx,
                             u16_t addr,
                             u16_t app_idx,
                             u16_t cid,
                             u8_t *test_id,
                             u8_t *faults,
                             size_t *fault_count)
```

- 功能描述

  获取对端节点注册的错误状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址

[in]	u16t appidx	APP Key 索引
[in]	cid	错误对应的 Company ID
[out]	u8t *testid	最后执行的测试 ID
[out]	u8_t *faults	错误列表
[out]	sizet *faultcount	错误个数

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthfaultclear
```

- 函数原型

```
int bt_mesh_health_fault_clear(u16_t net_idx,
                               u16_t addr,
                               u16_t app_idx,
                               u16_t cid,
                               u8_t *test_id,
                               u8_t *faults,
                               size_t *fault_count)
```

- 功能描述

  清除注册的错误状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引

[in]	cid	错误对应的 Company ID
[out]	u8t *testid	最后执行的测试 ID
[out]	u8_t *faults	错误列表
[out]	sizet *faultcount	错误个数

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthfaulttest
```

- 函数原型

```
int bt_mesh_health_fault_test(u16_t net_idx,
                              u16_t addr,
                              u16_t app_idx,
                              u16_t cid,
                              u8_t test_id,
                              u8_t *faults,
                              size_t *fault_count)
```

- 功能描述

  错误测试

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引
[in]	cid	Company ID
[in]	u8t testid	测试 ID

[out]	u8_t *faults	错误列表
[out]	sizet *faultcount	错误个数

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthperiodget
```

- 函数原型

```
int bt_mesh_health_period_get(u16_t net_idx,
                              u16_t addr,
                              u16_t app_idx,
                              u8_t *divisor)
```

- 功能描述

  获取 Health Model 消息发布的分频系数值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引
[out]	u8_t *divisor	分频系数（0 – 15）

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthperiodset
```

- 函数原型

```
int bt_mesh_health_period_set(u16_t net_idx,
                              u16_t addr,
                              u16_t app_idx,
                              u8_t divisor,
                              u8_t *updated_divisor)
```

- 功能描述

设置 Health Model 消息发布的分频系数值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引
[in]	u8_t divisor	分频系数值（0 – 15）
[out]	u8t *updateddivisor	更新后的分频系数

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthattentionget
```

- 函数原型

```
int bt_mesh_health_attention_get(u16_t net_idx,
                                 u16_t addr,
                                 u16_t app_idx,
                                 u8_t *attention)
```

- 功能描述

  获取 Health Model 的 Attention 定时器状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引
[out]	u8_t *attention	定时器状态 0x00: 关闭 0x01 ~ 0xFF：剩余时间（秒）

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthattentionset
```

- 函数原型

```
int bt_mesh_health_attention_set(u16_t net_idx,
                                 u16_t addr,
                                 u16_t app_idx,
                                 u8_t attention,
                                 u8_t *updated_attention)
```

- 功能描述

设置 Health Model 的 Attention 定时器状态

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16_t addr	节点的 unicast 地址
[in]	u16t appidx	APP Key 索引
[in]	u8_t *attention	定时器状态 0x00：关闭 0x01 ~ 0xFF：剩余时间（秒）
[out]	u8t *updatedattention	更新后的 attention 定时器状态

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

本节点需支持 health cli model，对端节点支持 health srv model

```
btmeshhealthclitimeout_get
```

- 函数原型

```
s32_t bt_mesh_health_cli_timeout_get(void)
```

- 功能描述

获取本 health cli 超时时间

- 参数描述

无

- 返回值

返回超时时间

- 注意事项

  本节点需支持 health cli model

```
btmeshhealthclitimeout_set
```

- 函数原型

```
void bt_mesh_health_cli_timeout_set(s32_t timeout)
```

- 功能描述

  设置本 helath cli 超时时间

- 参数描述

IN/OUT	NAME	DESC
[in]	s32_t timeout	超时时长

- 返回值

  无

- 注意事项

  本节点需支持 health cli model

2.4 Provisioner API

本章节介绍蓝牙 Mesh Provisioner 接口，利用这些接口可以实现蓝牙 MESH 的 Provision 功能。

```
btmeshprovisionerstorenode_info
```

- 函数原型

```
int bt_mesh_provisioner_store_node_info(struct bt_mesh_node_t *node_info)
```

- 功能描述

  存储被 prov devices 节点信息

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshnodet *nodeinfo	MESH 节点指针，参见 struct btmeshnode_t（结构体）定义
struct btmeshnode_t（结构体）定义		
char nodename[MESHNAME_SIZE]	节点名称	
u8t devuuid[16]	节点 UUID	
u16t oobinfo	节点 OOB 信息	
u16t unicastaddr	节点的 Primary Element 单播地址	
u8t elementnum	节点的 Element 个数	
u16t netidx	节点的 NetKey 索引值	
u8_t flags	节点的 device key 及 IV 更新标记	
u32t ivindex	IV 索引值	
u8t devkey[16]	节点的 Device Key	
bool node_active	节点激活标记 0：未激活 1：已激活	
u8t addrval[6]	节点的 MAC 地址	
u8t addrtype:4	节点 MAC 地址类型 0 – Public 1 – Random	
u8_t flag:4	节点信息是否已存在	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetallnodeunicast_addr
```

- 函数原型

```
int bt_mesh_provisioner_get_all_node_unicast_addr(struct net_buf_simple *buf)
```

- 功能描述

  获取配网器所有节点中 Primary Element 的单播地址

- 参数描述

IN/OUT	NAME	DESC
[out]	struct netbufsimple *buf	存储数据的 buf 指针

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionersetnode_name
```

- 函数原型

```
int bt_mesh_provisioner_set_node_name(int node_index, const char *name)
```

- 功能描述

  provisioner 设置节点名称

- 参数描述

IN/OUT	NAME	DESC
[in]	int node_index	节点索引
[in]	const char *name	节点名称

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetnode_name
```

- 函数原型

```
const char *bt_mesh_provisioner_get_node_name(int node_index)
```

- 功能描述

  获取索引值对应的节点的名称

- 参数描述

IN/OUT	NAME	DESC
[in]	int node_index	节点索引

- 返回值

  返回节点名称的字符串指针

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetnode_index
```

- 函数原型

```
int bt_mesh_provisioner_get_node_index(const char *name)
```

- 功能描述

  根据节点名称获取对应的索引值

- 参数描述

IN/OUT	NAME	DESC
[in]	const char *name	节点名称的字符串指针

- 返回值

返回值	
>= 0	索引值
< 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetnode_info
```

- 函数原型

```
struct bt_mesh_node_t *bt_mesh_provisioner_get_node_info(u16_t unicast_addr)
```

- 功能描述

  通过节点地址获取节点信息

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t unicastaddr	Network Key 索引

- 返回值

返回值	
NULL	失败
非 NULL	成功

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetnetkeycount
```

- 函数原型

```
u32_t bt_mesh_provisioner_get_net_key_count(void)
```

- 功能描述

  获取 MESH 网络中 Network Key 的数量

- 参数描述

  无

- 返回值

返回值	
0	无
> 0	网络中 netkey 数量

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetappkeycount
```

- 函数原型

```
u32_t bt_mesh_provisioner_get_app_key_count(void)
```

- 功能描述

  获取 MESH 网络中 APP Key 的数量

- 参数描述

  无

- 返回值

返回值	
0	无
> 0	网络中 APP Key 的数量

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalappkeyadd
```

- 函数原型

```
int bt_mesh_provisioner_local_app_key_add(const u8_t app_key[16], u16_t net_
idx, u16_t *app_idx)
```

- 功能描述

  添加配网器的 APP Key

- 参数描述

IN/OUT	NAME	DESC
[in]	const u8t appkey[16]	APP Key 值
[in]	u16t netidx	Network Key 索引值
[out]	u16t *appidx	APP Key 添加成功后返回的索引值

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalappkeyget
```

- 函数原型

```
const u8_t *bt_mesh_provisioner_local_app_key_get(u16_t net_idx, u16_t app_
idx)
```

- 功能描述

  根据 Network Key 索引和 APP Key 索引查找 APP Key 值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16t appidx	APP Key 索引

- 返回值

返回值	
NULL	失败
非 NULL	16 字节的 APP Key 数组指针

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalappkeydelete
```

- 函数原型

```
int bt_mesh_provisioner_local_app_key_delete(u16_t net_idx, u16_t app_idx)
```

- 功能描述

  根据输入的 netkey 索引和 appkey 索引，删除 appkey

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引
[in]	u16t appidx	APP Key 索引

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalnetkeyadd
```

- 函数原型

```
int bt_mesh_provisioner_local_net_key_add(const u8_t net_key[16], u16_t *net_
idx)
```

- 功能描述

  配网器添加一个 netkey

- 参数描述

IN/OUT	NAME	DESC
[in]	const u8t netkey[16]	Network Key 值
[in]	u16t *netidx	添加成功后，返回 Network Key 的索引值

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalnetkeyget
```

- 函数原型

```
const u8_t *bt_mesh_provisioner_local_net_key_get(u16_t net_idx)
```

- 功能描述

  根据 netkey 索引获得 netkey 的值

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引

- 返回值

返回值	
NULL	失败
非 NULL	16 字节的 Network Key 数组指针

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerlocalnetkeydelete
```

- 函数原型

```
int bt_mesh_provisioner_local_net_key_delete(u16_t net_idx)
```

- 功能描述

  根据输入的 netkey 索引，删除 netkey

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t netidx	Network Key 索引

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetownunicastaddr
```

- 函数原型

```
int bt_mesh_provisioner_get_own_unicast_addr(u16_t *addr, u8_t *elem_num)
```

- 功能描述

  获取配网器的单播地址和元素数量

- 参数描述

IN/OUT	NAME	DESC
[out]	u16_t *addr	将返回配网器单播地址
[out]	u8t *elemnum	将返回配网器元素数量

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerbindlocalmodelapp_idx
```

- 函数原型

```
int bt_mesh_provisioner_bind_local_model_app_idx(u16_t elem_addr, u16_t mod_
id,
        u16_t cid, u16_t app_idx)
```

- 功能描述

  配网器本地 model 绑定 appkey

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t elemaddr	元素地址
[in]	u16t modid	Model ID
[in]	u16_t cid	该值为 0xFFFF 时，为 SIG Model； 非 0xFFFF 时，为 Vendor Model
[in]	u16t appidx	APP Key 索引值

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerprintlocalelementinfo
```

- 函数原型

```
int bt_mesh_provisioner_print_local_element_info(void)
```

- 功能描述

  打印配网器的所有元素信息，如：cid、pid、vid、元素数量以及元素内的所有信息

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionerprintnode_info
```

- 函数原型

```
int bt_mesh_provisioner_print_node_info(void)
```

- 功能描述

  打印入网节点的信息，如节点名称、UUID、MAC 地址、MAC 地址类型、单播地址、元素数量、Network Key 索引等

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshisprovisioneren
```

- 函数原型

```
bool bt_mesh_is_provisioner_en(void)
```

- 功能描述

  配网功能是否使能

- 参数描述

  无

- 返回值

返回值	
0	未使能
1	已使能

- 注意事项

  无

```
btmeshprovisionerpappkeyalloc
```

- 函数原型

```
struct bt_mesh_app_key  *bt_mesh_provisioner_p_app_key_alloc()
```

- 功能描述

申请 16 字节的 APP Key 内存空间

- 参数描述

  无

- 返回值

返回值	
NULL	APP Key 存储已满
非 NULL	申请成功

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionergetnodeinfoby_id
```

- 函数原型

```
struct bt_mesh_node_t *bt_mesh_provisioner_get_node_info_by_id(int node_
index)
```

- 功能描述

  根据节点索引获取节点信息

- 参数描述

IN/OUT	NAME	DESC
[in]	int node_index	节点索引

- 返回值

返回值	
NULL	成功
非 NULL	返回查询到的节点指针

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisioneraddunprov_dev
```

- 函数原型

```
int bt_mesh_provisioner_add_unprov_dev(struct bt_mesh_unprov_dev_add *add_
dev, u8_t flags)
```

- 功能描述

  将未入网设备信息加入未入网设备列表

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshun-provdevadd *add_dev	未入网设备列表
[in]	u8_t flags	BIT0：在设备成功入网后，从未入网设备列表中清除该设备的信息 BIT1: 当设备被加入到未入网设备列表后，立即进行配置 BIT2: 未入网设备列表已满时可以刷新设备
struct btmeshunprovdevadd（结构体）定义		
u8_t addr[6]	设备的 MAC 地址	
u8t addrtype	设备的 MAC 地址类型	
u8_t uuid[16]	设备的 UUID	
u16t oobinfo	设备的 OOB 信息	
u8_t bearer	设备的广播承载类型	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  如果加入的设备的 UUID、MAC 地址、MAC 地址类型已经存在于队列中，但是广播承载类型不一致，增加设备的操作仍然会成功，并且会更新广播承载类型的信息

```
btmeshprovisionerdeletedevice
```

- 函数原型

```
int bt_mesh_provisioner_delete_device(struct bt_mesh_device_delete *del_dev)
```

- 功能描述

  从队列中删除设备，重置当前的配置信息和节点信息

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshdevicedelete *deldev	即将删除的设备指针，参见 struct bt*mesh*-device_delete（结构体）定义

struct btmeshdevice_delete（结构体）定义

u8_t addr[6]	节点的 MAC 地址
u8t addrtype	节点的 MAC 地址类型
u8_t uuid[16]	节点的 UUID

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionersetdevuuidmatch
```

- 函数原型

```
int bt_mesh_provisioner_set_dev_uuid_match(u8_t offset,
                                           u8_t length,
                                           const u8_t *match,
                                           bool prov_flag)
```

- 功能描述

  在设备入网前，比较设备 UUID 的信息

- 参数描述

IN/OUT	NAME	DESC
[in]	u8_t offset	要比较的 uuid 的 offset 值
[in]	u8_t length	要比较的 uuid 的长度
[in]	const u8_t *match	比较的值
[in]	bool prov_flag	该标识指示接收到 uuid_match 的 adv 包时，设备需要立即进行 provision 还是给应用层上报。 0：给应用层上报；1：立即进行 provision

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
provadvpkt_cb
```

- 函数原型

```
typedef void (*prov_adv_pkt_cb)(const u8_t addr[6],
                                const u8_t addr_type,
                                const u8_t adv_type,
                                const u8_t dev_uuid[16],
                                u16_t oob_info,
                                bt_mesh_prov_bearer_t bearer)
```

- 功能描述

  定义了一个回调函数，当配网器收到未入网设备的 adv 包，且该设备不在 provisioner 的未配置设备队列中

- 参数描述

IN/OUT	NAME	DESC
[in]	const u8_t addr[6]	未入网设备的 MAC 地址
[in]	const u8t addrtype	未入网设备的 MAC 地址类型

[in]	const u8t advtype	广播类型 0x00: ADV*IND 0x01: ADV*DIRECT*IND 0x02: ADV*SCAN*IND 0x03: ADV*NONCONN*IND 0x04: ADV*DIRECT*IND*LOW_DUTY
[in]	const u8t devuuid[16]	未入网设备的 UUID
[in]	u16t oobinfo	未入网设备的 OOB 信息
[in]	btmeshprovbearert bearer	广播承载类型 0：ADV 1：GATT

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovadvpktcbregister
```

- 函数原型

```
int bt_mesh_prov_adv_pkt_cb_register(prov_adv_pkt_cb cb)
```

- 功能描述

  用来通知应用层收到了 mesh 组网过程中的 adv 包或者未入网设备发送的 beacon 包（设备不在未入网设备列表中）

- 参数描述

IN/OUT	NAME	DESC
[in]	provadvpkt_cb cb	参见 prov*adv*pkt_cb 回调函数定义

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovisionersetprovdatainfo
```

- 函数原型

```
int bt_mesh_provisioner_set_prov_data_info(struct bt_mesh_prov_data_info
*info)
```

- 功能描述

  设置配置信息中的 Network Key 索引或者 IV 索引

- 参数描述

IN/OUT	NAME	DESC
[in]	struct btmeshprovdatainfo *info	包含了 Network Key 索引或者 IV 索引的信息

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能

```
btmeshprovinputdata
```

- 函数原型

```
int bt_mesh_prov_input_data(u8_t *num, u8_t size, bool num_flag)
```

- 功能描述

  OOB 信息输入

- 参数描述

IN/OUT	NAME	DESC
[in]	u8_t *num	输入的数据内容
[in]	u8_t size	输入数据的大小
[in]	bool num_flag	输入的数据类型 0：字符串 1：数字

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能 ; prov 认证方式是 OUTPUT OOB 时使用

```
btmeshprovoutputdata
```

- 函数原型

```
int bt_mesh_prov_output_data(u8_t *num, u8_t size, bool num_flag)
```

- 功能描述

  OOB 信息输出

- 参数描述

IN/OUT	NAME	DESC
[in]	u8_t *num	输出的数据内容
[in]	u8_t size	输出数据的大小
[in]	bool num_flag	输出的数据类型 0：字符串 1：数字

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用前需使能 provisioner 功能 ; prov 认证方式是 INPUT OOB 时使用

3. MESH 组件 API

3.1 Mesh Models API

Mesh Models 组件中实现了 SIG 定义的 Generic Onoff Model,Generic Level Model, Light Lightness Model，Light CTL Model，开发者可以方便的组合各个 Model，实现不同的功能。

```
blemeshmodel_init
```

- 函数原型

```
int ble_mesh_model_init(const struct bt_mesh_comp *comp)
```

- 功能描述

MESH Models 组件初始化

- 参数描述

IN/OUT	NAME	DESC
[in]	const struct btmeshcomp *comp	MESH model 组件初始化结构体，参见 struct btmeshcomp (结构体) 定义

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshmodelgetcomp_data
```

- 函数原型

```
const struct bt_mesh_comp * ble_mesh_model_get_comp_data()
```

- 功能描述

  获取节点的 Composition data，包括节点信息、Models 的组成等

- 参数描述

IN/OUT	NAME	DESC
[out]	const struct btmeshcomp *	MESH model 组件初始化结构体，参见 struct btmeshcomp（结构体）定义

- 返回值

返回值	
非 NULL	成功
NULL	失败

- 注意事项

无

```
blemeshmodelsetcb
```

- 函数原型

```
int ble_mesh_model_set_cb(model_event_cb event_cb)
```

- 功能描述

  设置 MESH Models 事件回调函数

- 参数描述

IN/OUT	NAME	DESC
[in]	modeleventcb event_cb	model event 回调函数，参见 mod-eleventcb 回调函数定义返回值

- 返回值

返回值	
0	成功
< 0	失败

- 回调函数

```
void (*model_event_cb)(mesh_model_event_en event, void *p_arg)
```

- 参数描述

IN/OUT	NAME	DESC
[in]	meshmodelevent_en event	model model 事件，参见 meshmodelevent_en(枚举) 定义
[in]	void *p_arg	事件相关数据，具体数据类型参见 meshprovisioner-event_en (枚举) 定义

- 返回值

  无

meshmodelevent_en(枚举)定义

BTMESHMODELCF-GAPPKEYADD = 0x00	appkey 设置消息	parg 对应 appkeystatus 结构体，参见该结构体定义
BTMESHMODELCFG-COMPDATASTATUS = 0x02	comp data 消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-HEARTBEATPUBSTA-TUS = 0x06	heartbeat pub 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCF-GAPPKEY_STATUS = 0x8003	appkey 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-BEACON_STATUS = 0x800b	beacon 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFGT-TL_STATUS = 0x800e	TTL 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示

BTMESHMODELCF-GFRIEND_STATUS = 0x8011	FRIEND 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-PROXY_STATUS = 0x8014	PROXY 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFGN-ETKRPSTATUS = 0x8017	KRP 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-PUB_STATUS = 0x8019	PUB 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-SUB_STATUS = 0x801f	SUB 设置状态结果	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-SUB_LIST = 0x802a	SUB LIST 状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-SUBLISTVND = 0x802c	SUB VND 状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCF-GRELAY_STATUS = 0x8028	RELAY 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFG-HEARTBEATSUBSTA-TUS = 0x803c	heartbeat sub 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCF-GAPPKEYBINDSTATUS = 0x803e	appkey bind 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCF-GRST_STATUS = 0x804a	node rst 消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELCFGN-ETKEYSTATUS = 0x8044	netkey 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示
BTMESHMODELO-NOFFSET = 0x8202	onoff 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 status_data 表示

BTMESHMODELO-NOFFSTATUS = 0x8204	onoff 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLEVEL-SET = 0x8206	level 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLEVEL-MOVE_SET = 0x820B	level 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLEVEL-STATUS = 0x8208	level 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LEVELDELTA_SET= 0x8209	level delta 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSSET= 0x824C	lightness 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSSTATUS = 0x824E	lightness 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSLINEAR_SET = 0x8250	lightness linear 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSLINEAR_STATUS = 0x8252	lightness linear 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSLAST_STATUS =0x8254	lightness last 状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSDEF_STATUS =0x8256	lightess 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSRANGE_STATUS = 0x8258	lightness range 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELLIGHT-NESSDEF_SET = 0x8259	lightness default 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示

BTMESHMODELLIGHT-NESSRANGE_SET = 0x825B	lightness range 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTL_SET =0x825E	light ctl 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTL_STATUS =0x8260	light ctl 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLTEM-PRANGE_STATUS =0x8263	light temperature range 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLTEMPSET =0x8264	light temperature 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLTEMPSTATUS =0x8266	light temperature 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLDEFSTATUS =0x8268	light ctl default 设置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLDEFSET =0x8269	light ctl default 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODEL-LIGHTCTLRANGESET =0x826B	light ctl range 设置消息	parg 对应 modelmessage 结构体，参见该结构体定义，状态数据使用其中的 user_data 表示
BTMESHMODELVEN-DORMESSAGES =0xcf	透传消息	parg 对应 modelmessage 结构体，参见该结构体定义，数据使用其中的 ven_data 表示
BTMESHMODELVEN-DORMESH_AUTOCON-FIG = 0xD6	入网自动配置消息	parg 对应 modelmessage 结构体，参见该结构体定义，数据使用其中的 ven_data 表示
BTMESHMODELVEN-DORMESHAUTOCON-FIGSTATUS =0xD8	入网自动配置状态消息	parg 对应 modelmessage 结构体，参见该结构体定义，数据使用其中的 ven_data 表示

appkey_status（结构体）定义

uint8_t status	appkey 添加状态
uint16t netkeyidx	appkey 对应 netkey idx
uint16t appkeyidx	appkey idx

model_message（结构体）定义

uint16t sourceaddr	数据源地址
struct netbufsimple *status_data	标准 model 状态数据，用于 mesh 基础 model ,cfg model、health model
void *user_data	sig model 状态数据 , 用于 sig model 如 generic model、light model 等
vendordata vendata	厂家自定义数据，参见 vendor_data 结构体定义

vendor_data（结构体）定义

void *user_data	厂家自定义数据数据内容
uint16t datalen	厂家自定义数据数据长度

SELEMSTATE（结构体）定义

SMESHSTATE state	sig model 状态，参见 SMESHSTATE 结构体定义
SMESHPOWERUP powerup	sig moldel 默认状态，参见 SMESHPOWERUP 结构体定义

SMESHSTATE（结构体）定义

u8t onoff[TYPENUM]	onoff srv status, 其中 TYPENUM 参见 TYPENUM 枚举定义
s16t level[TYPENUM]	level srv status, 其中 TYPENUM 参见 TYPENUM 枚举定义
u16t lightnesslinear[TYPE_NUM]	lightness linear status, 其中 TYPENUM 参见 TYPENUM 枚举定义
u16t lightnessactual[TYPE_NUM]	lightness actual status, 其中 TYPENUM 参见 TYPENUM 枚举定义
s16t level[TYPENUM]	level status, 其中 TYPENUM 参见 TYPENUM 枚举定义
u16t lightnesslinear[TYPE_NUM]	lightness linear status, 其中 TYPENUM 参见 TYPENUM 枚举定义

u16t lightness[TYPENUM]	ctl lightness status, 其中 TYPENUM 参见 TYPENUM 枚举定义
u16t temp[TYPENUM]	ctl temperature status, 其中 TYPENUM 参见 TYPENUM 枚举定义
u16t UV[TYPENUM]	ctl UV status, 其中 TYPENUM 参见 TYPENUM 枚举定义

TYPE_NUM（枚举）定义

T_CUR = 0	当前状态
T_TAR	目标状态
TYPE_NUM	TYPE 种类

SMESHPOWERUP（结构体）定义

uint16t lightnessactual_def	lightness actual 默认值
u16t lightnesslast	lightness 上次值
RANGESTATUS lightnessrange	lightness 范围值，参见 RANGE_STATUS 结构体定义
uint16t lightnessdefault	ctl lightness 默认值
uint16t tempdefault	ctl temperature 默认值
uint16t UVdefault	ctl UV 默认值
RANGESTATUS ctltemp_range	ctl temperature 范围值，参见 RANGE_STATUS 结构体定义

RANGE_STATUS（结构体）定义

STATUSCODES code	range 设置结果状态码，参见 STATUSCODES 枚举定义
u16t rangemin	range 最小值
u16t rangemax	range 最大值

STATUSCODES（枚举）定义

SUCCESS = 0	设置成功
SETMINFAIL	设置最小值失败
SETMAXFAIL	设置最大值失败
RFU	保留

```
blemeshmodel_find
```

- 函数原型

```
struct bt_mesh_model *ble_mesh_model_find(uint16_t elem_idx, uint16_t mod_
idx, uint16_t CID)
```

- 功能描述

  获取指定 Model 结构体指针

- 参数描述

IN/OUT	NAME	
[in]	elem_idx	elem id 索引
[in]	uint16t modid	model id 标识 , 目前支持的 model id 如下表所示
[in]	uint16_t CID	vendor model company id 标识
[out]	struct btmeshmodel *	mesh model，参见 btmeshmodel(结构体) 定义

model id	说明
cfg srv	0x0000
cfg cli	0x0001
health srv	0x0002
health cli	0x0003
onoff srv	0x1000
onoff cli	0x1001
level srv	0x1002
level cli	0x1003
lightness srv	0x1300
lightness cli	0x1302
ctl srv	0x1303
ctl cli	0x1305

- 返回值

返回值	
NULL	失败
非 NULL	成功

- 注意事项

  无

```
blemeshmodelstatusget
```

- 函数原型

```
int ble_mesh_model_status_get(uint16_t netkey_idx, uint16_t appkey_idx,
uint16_t unicast_addr,struct bt_mesh_model *model,uint16_t op_code)
```

- 功能描述

  获取指定 Model 的状态值

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	uint16t opcode	status get 操作码

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  查询相应状态所对应的 model 及 opcode 对应如下表所示

model 状态查询表

status	query model id	query opcode
friend status	0x0001	0x800f
proxy status	0x0001	0x8012
relay status	0x0001	0x8026

onoff status	0x1001	0x8201
level status	0x1003	0x8205
lightness status	0x1302	0x824b
lightness linear status	0x1302	0x824f
lightness last status	0x1302	0x8253
lightness default status	0x1302	0x8255
lightness range status	0x1302	0x8257
ctl status	0x1305	0x825D
ctl temperature status	0x1305	0x8261
ctl default status	0x1305	0x8267
ctl range status	0x1305	0x8262

```
blemeshgeneriконoffget
```

- 函数原型

```
int ble_mesh_generic_onoff_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Generic OnOff Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgenericonoffset
```

- 函数原型

```
int ble_mesh_generic_onoff_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_onoff_arg *send_
arg, bool ack)
```

- 功能描述

  设置 Generic OnOff Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setonoffarg *send_arg	发送参数设置，参见 setonoffarg *send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）
setonoffarg（结构体）定义		
uint8_t onoff	onff 设置值	
uint8_t tid	数据 TID	
uint8t sendtrans	是否发送 trans	
uint8_t trans	trans 值	
uint8_t delay	状态转换延时值	

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgenericonoffcli_publish
```

- 函数原型

```
int ble_mesh_generic_onoff_cli_publish(struct bt_mesh_model *model, set_
onoff_arg *send_arg, bool ack)
```

- 功能描述

  设置 Generic OnOff Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setonoffarg *send_arg	发送参数设置，参见 setonoffarg *send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgenericlevelget
```

- 函数原型

```
int ble_mesh_generic_level_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Generic Level Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgeneriClevelset
```

- 函数原型

```
int ble_mesh_generic_level_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_level_arg *send_
arg, bool ack)
```

- 功能描述

  设置 Generic Level Server Model 的 level Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlevelarg *send_arg	发送参数设置，参见 setlevelarg *send_arg 结构体定义

[in]	bool ack	是否应答，0（不应答）/1（应答）

setlevelarg（结构体）定义

uint16_t level	level 设置值
uint16_t def	level 默认值
uint16_t move	level move 值
uint8_t tid	level tid 值
uint8t sendtrans	是否发送 trans
uint8_t trans	trans 值
uint8_t delay	状态转换延时值
s32_t delta	level delta 值

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgenericlevelmove_set
```

- 函数原型

```
int ble_mesh_generic_level_move_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_level_arg *send_
arg, bool ack)
```

- 功能描述

  设置 Generic Level Server Model 的 level Move 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引

[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlevelarg *send_arg	发送参数设置，参见 setlevelarg *send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshgenericleveldelta_set
```

- 函数原型

```
int ble_mesh_generic_level_delta_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_level_arg *send_
arg, bool ack)
```

- 功能描述

  设置 Generic Level Server Model 的 Delta 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlevelarg *send_arg	发送参数设置，参见 setlevelarg *send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightlightnessget
```

- 函数原型

```
int ble_mesh_light_lightness_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light Lightness Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnessset
```

- 函数原型

```
int ble_mesh_light_lightness_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_lightness_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light Lightness Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightnessarg*send_arg	发送参数设置，参见 setlight-nessarg*send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）
setlightnessarg（结构体）定义		
uint16_t lightness	lightness 设置值	
uint16t lightnesslinear	lightness linear 值	
uint16_t def	lightness 默认值	
uint16t rangemin	lightness 最小值	
uint16t rangemax	lightness 最大值	
uint8_t tid	lightness tid 值	
uint8t sendtrans	是否发送 trans	
uint8_t trans	trans 值	
uint8_t delay	状态转换延时值	

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnesslinear_get
```

- 函数原型

```
int ble_mesh_light_lightness_linear_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light Lightness Server Model 的 Lightness Linear 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnesslinear_set
```

- 函数原型

```
int ble_mesh_light_lightness_linear_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model,set_lightness_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light Lightness Server Model 的 Lightness Linear 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightnessarg*send_arg	发送参数设置，参见 setlight-nessarg*send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnessdef_get
```

- 函数原型

```
int ble_mesh_light_lightness_def_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light Lightness Server Model 的 Lightness Default 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引

[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnessdef_set
```

- 函数原型

```
int ble_mesh_light_lightness_def_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model,set_lightness_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light Lightness Server Model 的 Lightness Default 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightnessarg*send_arg	发送参数设置，参见 setlightnessarg*send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightlightnessrange_get
```

- 函数原型

```
int ble_mesh_light_lightness_range_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light Lightness Server Model 的 Lightness Range 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightlightnessrange_set
```

- 函数原型

```
int ble_mesh_light_lightness_range_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model,set_lightness_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light Lightness Server Model 的 Lightness Range 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightnessarg*send_arg	发送参数设置，参见 setlightnes-sarg*send_arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightlightnesslast_get
```

- 函数原型

```
int ble_mesh_light_lightness_last_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light Lightness Server Model 的 Lightness Last 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightctlget
```

- 函数原型

```
int ble_mesh_light_ctl_get(uint16_t netkey_idx, uint16_t appkey_idx,uint16_t
unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light CTL Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshlightctlset
```

- 函数原型

```
int ble_mesh_light_ctl_set(uint16_t netkey_idx, uint16_t appkey_idx,uint16_
t unicast_addr,struct bt_mesh_model *model, set_light_ctl_arg *send_arg, bool
ack)
```

- 功能描述

  设置 Light CTL Server Model 的 Status 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightctlarg*sendarg	发送参数设置，参见 set-lightctlarg*sendarg 结构体定义
[in]	bool ack	是否应答,0(不应答)/1(应答)
setlightnessarg(结构体)定义		
uint16_t lightness	lightness 设置值	
uint16_t tempature	light tempature 值	
uint16t deltauv	light delta_uv 默认值	
uint16t rangemin	light tempature 最小值	

uint16t rangemax	light tempature 最大值
uint8_t tid	lightness tid 值
uint8t sendtrans	是否发送 trans
uint8_t trans	trans 值
uint8_t delay	状态转换延时值

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctltemp_get
```

- 函数原型

```
int ble_mesh_light_ctl_temp_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light CTL Server Model 的 Temperature 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctltemp_set
```

- 函数原型

```
int ble_mesh_light_ctl_temp_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model,set_light_ctl_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light CTL Server Model 的 Temperature 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightctlarg*sendarg	发送参数设置，参见 set-*light*ctl*arg*send*arg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctldef_get
```

- 函数原型

```
int ble_mesh_light_ctl_def_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light CTL Server Model 的 Default 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctldef_set
```

- 函数原型

```
int ble_mesh_light_ctl_def_set(struct bt_mesh_model *model, uint16_t unicast_
addr,uint16_t netkey_idx, uint16_t appkey_idx, set_light_ctl_arg *send_arg,
bool ack)
```

- 功能描述

  设置 Light CTL Server Model 的 Default 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

[in]	setlightctlarg*sendarg	发送参数设置，参见 setlightctl*arg*sendarg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctltemprangeget
```

- 函数原型

```
int ble_mesh_light_ctl_temp_range_get(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model)
```

- 功能描述

  获取 Light CTL Server Model 的 Temperature Range 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

```
blemeshlightctltemprangeset
```

- 函数原型

```
int ble_mesh_light_ctl_temp_range_set(uint16_t netkey_idx, uint16_t appkey_
idx,uint16_t unicast_addr,struct bt_mesh_model *model, set_light_ctl_arg
*send_arg, bool ack)
```

- 功能描述

  设置 Light CTL Server Model 的 Temperature Range 信息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	struct btmeshmodel *model	发送 status get 使用的 model
[in]	setlightctlarg*sendarg	发送参数设置，参见 setlightctlarg*sendarg 结构体定义
[in]	bool ack	是否应答，0（不应答）/1（应答）

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshvendorclimodelmsgsend
```

- 函数原型

```
int ble_mesh_vendor_cli_model_msg_send(vnd_model_msg *model_msg)
```

- 功能描述

```
Vendor Client Model 的消息发送接口
```

- 参数描述

IN/OUT	NAME	
[in]	vndmodelmsg *model_msg	发送 vnd msg 参数 , 参见 vnd-modelmsg(结构体) 定义
vndmodelmsg (结构体) 定义		
struct btmeshmodel *model	发包使用的 model, 参见 btmeshmodel (结构体) 定义	
uint16t netkeyidx	发包使用的 netkeyid	
uint16t appkeyidx	发包使用的 appkey id	
uint16t dstaddr	发包的目标地址	
uint16_t len	发包的长度	
uint8_t retry	发包重试次数	
uint8t retryperiod	发包重试周期	
uint8_t opid	发包使用的 opcode	
uint8_t tid	发包使用的 tid	
uint8_t *data	发包数据	

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

```
blemeshvendorsrvmodelmsgsend
```

- 函数原型

```
int ble_mesh_vendor_srv_model_msg_send(vnd_model_msg *model_msg)
```

- 功能描述

```
Vendor Server Model 的消息发送接口
```

- 参数描述

IN/OUT	NAME	
[in]	vndmodelmsg *model_msg	发送 vnd msg 参数 , 参见 vndmodelmsg(结构体) 定义

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

3.2 Mesh Node API

Mesh Node 组件是对 Mesh 节点的一层抽象定义，使用 Mesh Node 组件开发者可以方便的实现节点的配置和开发。

```
blemeshnode_init
```

- 函数原型

```
int ble_mesh_node_init(node_config_t *param)
```

- 功能描述

mesh NODE 组件初始化

- 参数描述

IN/OUT	NAME	DESC
[in]	nodeconfigt *param	MESH NODE 组件初始化结构体，参见 struct nodeconfigt（结构体）定义
struct nodeconfigt（结构体）定义		
noderoleen role	NODE 节点角色，定义参见 noderoleen（枚举）定义	
provisionernode *provisioner-config	provisioner node 配置，见 provisioner_node 结构体定义	
uint8t devuuid[16]	NODE 节点设备 UUID	
uint8t devname[DEVICENAM-EMAX_LENGTH]	NODE 节点设备名称 ,DEVICE-NAMEMAX_LENGTH 为 28	
modeleventcb usermodelcb	NODE 节点 MODEL 消息回调函数，定义参见 MESH MODEL MODULE 文档 modeleventcb（函数指针定义）	
proveventcb userprovcb	NODE 节点入网过程回调函数，定义参见 proveventcb（函数指针）定义	
healthsrvcb *health_cb	health srv 回调函数，见 healthsrvcb 结构体定义	
oobmethod nodeoob	节点 OOB 信息，定义参见 oob_method（结构体）定义	

healthsrvcb（结构体）定义	
attncb atton	health srv attention on 回调函数，见 attn_cb（回调函数定义）
attncb attoff	health srv attention off 回调函数，见 attn_cb（回调函数定义）

- 回调函数原型

```
void (*attn_cb)(void)
```

- 功能描述

  用于 node 节点配网过程中 attention 信息回调

- 参数描述

  无

- 返回值

  无

- 注意事项

  只有 provisioner attention time 设置为非零值，node 节点才会在配网过程中回调该函数

provisioner_node（结构体）定义	
provisionerconfigt config	provisioner 配置，见 provisionerconfigt 结构体定义
uint16t localsub	provisioner node 本地订阅地址
uint16t localpub	provisioner node 本地发布地址
noderoleen（枚举）定义	
PROVISIONER	使能为 PROVISIONER NODE 节点
NODE	使能为普通 NODE 节点
struct oob_method(结构体)定义	
uint8t* staticoob_data	NODE 节点 static oob 信息
oobactionen input_action	NODE 节点 input OOB 方法，参见 oobactionen（枚举）定义
uint8t inputmax_size	NODE 节点 INPUT OOB 最大长度
oobactionen output_action	NODE 节点 output OOB 方法，参见 oobactionen（枚举）定义
uint8t outputmax_size	NODE 节点 OUTPUT OOB 最大长度
oobactionen(枚举)定义	
ACTION_NULL	无 OOB
ACTION_NUM	OOB NUM
ACTION_STR	OOB STRING

- 回调函数原型

```
void (*prov_event_cb)(mesh_prov_event_en event, void *p_arg)
```

- 功能描述

用于 MODEL 层向上层 APP 上报 prov 过程相关事件

- 参数描述

IN/OUT	NAME	DESC
[in]	meshprovevent_en	MESH PROV 事件，参见 meshprovevent_en（枚举）定义
[in]	void *p_arg	事件相关数据，具体数据类型参见 meshprovevent_en（枚举）定义
struct meshprovevent_en（枚举）定义		
BTMESHEVENTNODEREST	NODE REST 消息	NULL
BTMESHEVENTNODEPROV_COMP	NODE 设备入网成功	meshnodelocalt* node，参见 meshnodelocalt（结构体）定义
BTMESHEVENTNODEOOBINPUT-NUM	NODE 设备输入 NUM	uint8_t size, 输如 num 个数
BTMESHEVENTNODEOOBINPUT-STRING	NODE 设备输入 STR	uint8_t size, 输入 str 个数

- 返回值

  无

- 注意事项

  若需将 NODE 节点使能为 PROVISIONER 节点，需先使用 MESH PROVISIONER Module 相关 API，具体使用方法请参考该 API 说明文档

```
blemeshnodeOOBinput_num
```

- 函数原型

```
int ble_mesh_node_OOB_input_num(uint32_t num)
```

- 功能描述

```
NODE 节点 输入 OOB num
```

- 参数描述

IN/OUT	NAME	DESC
[in]	uint32_t num	oob num 个数

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

用户在接收到 BTMESHEVENTNODEOOBINPUTNUM 事件后，需调用该接口出入指定数量的 num，其它时间调用该接口无效

```
blemeshnodeOOBinput_string
```

- 函数原型

```
int ble_mesh_node_OOB_input_string(const char *str)
```

- 功能描述

```
provisioner 输入 OOB string
```

- 参数描述

IN/OUT	NAME	DESC
[in]	const char *str	oob string 字符

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  用户在接收到 BTMESHEVENTNODEOOBINPUTSTRING 事件后，需调用该接口出入指定数量的 string，其它时间调用该接口无效

```
blemeshnodeappkeyadd
```

- 函数原型

```
int ble_mesh_model_appkey_add(uint16_t netkey_idx,uint16_t appkey_idx,uint16_
t unicast_addr)
```

- 功能描述

  给对端节点添加 appkey

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	netkey 索引
[in]	uint16t appkeyidx	appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  该接口只适合 provisioner 节点使用

```
blemeshnodemodelautoconfig
```

- 函数原型

```
int ble_mesh_node_model_autoconfig(uint16_t netkey_idx,uint16_t appkey_idx,
uint16_t unicast_addr,model_auto_config_t config)
```

- 功能描述

  给 node 节点发送 model 自动配置消息

- 参数描述

IN/OUT	NAME	
[in]	uint16t netkeyidx	发送消息使用的 netkey 索引
[in]	uint16t appkeyidx	发送消息使用的 appkey 索引
[in]	uint16t unicastaddr	目标 unicast_addr
[in]	modelautoconfig_t config	自动配置信息，参见 modelautoconfig_t 定义
modelautoconfig_t（结构体）定义		
uint16t subaddr	自动绑定 sub addr	

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  无

3.3 Mesh Provisioner API

Mesh Provisioner 组件是对 Provisioner 节点的抽象定义，该组件实现了未入网设备的发现，入网，配置和管理功能。

```
blemeshprovisioner_init
```

- 函数原型

```
int ble_mesh_provisioner_init(provisioner_config_t *param)
```

- 功能描述

  mesh provisioner 组件初始化

- 参数描述

IN/OUT	NAME	DESC
[in]	provisionerconfigt *param	MESH Provisioner 组件初始化结构体，参见 struct provisionerconfigt（结构体）定义
struct provisionerconfigt（结构体）定义		
uint16t unicastaddr_local	Provisioner 节点本地地址	
uint16t unicastaddr_start	Provisioner 节点分配给 NODE 节点的起始地址	
uint8t attentiontime	Provisioner 配网过程 attention 时间	
provisioner_cb cb	Provisioner 节点回调函数，定义如下	

- 返回值

返回值	
=0	成功
非 0	失败

- 注意事项

  注意 unicastaddrstart – unicastaddrlocal 的值应大于 provisioner node 节点本地 elem 数目

- 回调函数原型

```
void (*provisioner_cb)(mesh_provisioner_event_en event, void *p_arg)
```

- 功能描述

  用于向上层应用上报 PROVISIONER 事件

- 参数描述

IN/OUT	NAME	DESC
[in]	meshprovisionerevent_en	MESH Provisioner 事件，参见 meshprovisioner-event_en（枚举）定义
[in]	void *p_arg	事件相关数据，具体数据类型参见 meshprovisioner-event_en（枚举）定义
struct meshprovisionerevent_en（枚举）定义		
BTMESHEVENTRECVUNPROVDEVADV	Provisioner 节点接收到未入网设备广播	meshnodet* node，参见 meshnodet（结构体）定义
BTMESHEVENTPROVCOMP	Provisioner 节点入网设备成功	meshnodet* node，参见 meshnodet（结构体）定义
BTMESHEVENTFOUNDDEV_TIMEOUT	Provisioner 节点查找设备超时	NULL
BTMESHEVENTPROVFAILD	Provisioner 节点入网设备失败	uint8_t* reason, 入网失败原因
BTMESHEVENTOOBINPUT_NUM	Provisioner 节点输入 OOB NUM 提示	uint8_t* size, 输入数字个数
BTMESHEVENTOOBINPUT_STRING	Provisioner 节点输入 OOB STR 提示	uint8_t* szie, 输入字符个数
BTMESHEVENTOOBINPUT-STATICOOB	Provisioner 节点输入 STATIC OOB 提示	NULL（static oob 默认输入 16 个字符）
struct meshnodet 结构体定义		
uint8_t uuid[16]	节点 UUID	
uint8t devaddr[6]	节点 MAC 地址	
uint8t addrtype	节点 MAC 地址类型	
uint16t primunicast	节点首要 element 地址	
uint16t oobinfo	节点 oob 信息	
uint8t elementnum	节点 elem 数量	
uint8_t bearer	节点入网 bearer	

uint8_t flags	节点 key 更新 /iv 更新标志
uint32t ivindex	节点 iv 索引
uint8t* nodename	节点名称

- 返回值

  无

- 注意事项

  无

```
blemeshprovisioner_enable
```

- 函数原型

```
int ble_mesh_provisioner_enable()
```

- 功能描述

  节点 provisioner 使能

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  无

```
blemeshprovisioner_disable
```

- 函数原型

```
int ble_mesh_provisioner_disable()
```

- 功能描述

  provisioner 节点禁用

- 参数描述

  无

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  无

```
blemeshprovisionerdevfilter
```

- 函数原型

```
int ble_mesh_provisioner_dev_filter(uint8_t enable, uuid_filter_t *filter)
```

- 功能描述

  provisioners 设备上报过滤

- 参数描述

IN/OUT	NAME	DESC
[in]	enable	关闭 0/ 开启 1
[in]	uuidfiltert *filter	uuidfiltert 过滤规则结构体，参见 uuidfiltert 结构体定义
struct uuidfiltert(结构体) 定义		
uint8_t *uuid	过滤 uuid 信息头指针	
uint8t uuidlength	过滤 uuid 信息长度	
uint8t filterstart	uuid 开始匹配位置	

- 返回值

返回值	
0	成功
非 0	失败

- 注意事项

  使用该接口，provisioner 将从上报的 UUID filterstart 处开始与过滤器输入的长度为 uuidlength 的 uuid 信息进行匹配，若匹配成功，则上报，否则不上报该信息。

```
blemeshprovisionershowdev
```

- 函数原型

```
int ble_mesh_provisioner_show_dev(uint8_t enable, uint32_t timeout)
```

- 功能描述

  设备上报使能

- 参数描述

IN/OUT	NAME	DESC
[in]	uint8_t enable	设备上报使能，0（关闭）/1（开启）
[in]	uint32_t timeout	设备上报超时，单位 s, 为 0 时默认无超时

- 返回值

返回值	
>= 0	成功
< 0	失败

- 注意事项

  使用前需初始化和使能 provisioner 功能

```
blemeshprovisionerdevadd
```

- 函数原型

```
int ble_mesh_provisioner_dev_add(mesh_node_t *node,uint8_t auto_add_appkey)
```

- 功能描述

  添加待入网 NODE 设备

- 参数描述

IN/OUT	NAME	DESC
[in]	meshnodet *node	meshnodet *node 入网节点信息，参见 struct mesh-nodet *node 定义
[in]	uint8t addappkey	是否添加 appkey，0（添加）/1（不添加）

- 返回值

返回值	
>= 0	成功
< 0	失败

- 注意事项

  使用前需初始化和使能 provisioner 功能

```
blemeshprovisionergetaddappkeyflag
```

- 函数原型

```
int ble_mesh_provisioner_get_add_appkey_flag(u16_t unicast_addr)
```

- 功能描述

  获取已入网节点是否需要添加 appkey 标志

- 参数描述

IN/OUT	NAME	DESC
[in]	unicast_addr	node 节点 unicast 地址

- 返回值

返回值	
>= 0	成功
< 0	失败

- 注意事项

  若使用 MESH NODE 组件开发，则用户一般不需要使用该接口

```
blemeshprovisionerdevdel
```

- 函数原型

```
int ble_mesh_provisioner_dev_del(uint8_t addr[6], uint8_t addr_type, uint8_t
uuid[16])
```

- 功能描述

  删除添加的待入网设备 / 正在入网设备节点

- 参数描述

IN/OUT	NAME	DESC
[in]	uint8_t addr[6]	设备 mac 地址
[in]	uint8t addrtype	设备地址类型，0（public）/1(random)
[in]	uint8_t uuid[16]	设备 uuid

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  使用前需初始化和使能 provisioner 功能

```
blemeshprovisionerOOBinput_num
```

- 函数原型

```
int ble_mesh_provisioner_OOB_input_num(uint32_t num)
```

- 功能描述

```
provisioner 输入 OOB num
```

- 参数描述

IN/OUT	NAME	DESC
[in]	uint32_t num	oob num 个数

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  用户在接收到 BTMESHEVENTOOBINPUT_NUM 事件后，需调用该接口出入指定数量的 num，其它时间调用该接口无效

```
blemeshprovisionerOOBinput_string
```

- 函数原型

```
int ble_mesh_provisioner_OOB_input_string(const char *str)
```

- 功能描述

```
provisioner 输入 OOB string
```

- 参数描述

IN/OUT	NAME	DESC
[in]	const char *str	oob string 字符

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  用户在接收到 BTMESHEVENTOOBINPUT_STRING 事件后，需调用该接口出入指定数量的 string，其它时间调用该接口无效

```
blemeshprovisionerstaticOOB_set
```

- 函数原型

```
int ble_mesh_provisioner_static_OOB_set(const uint8_t *oob, uint16_t oob_size)
```

- 功能描述

```
provisioner 输入 static OOB
```

- 参数描述

IN/OUT	NAME	DESC
[in]	const uint8_t *oob	oob 指针
[in]	uint16t oobsize	oob 长度

- 返回值

返回值	
0	成功
< 0	失败

- 注意事项

  用户在接收到 BTMESHEVENTOOBINPUTSTATICOOB 事件后，需调用该接口出入指定数量的 oob 字符，其它时间调用该接口无效

```
blemeshprovisionergetnode_info
```

- 函数原型

```
mesh_node_t * ble_mesh_provisioner_get_node_info(u16_t unicast_addr)
```

- 功能描述

```
provisioner 获取本身 NODE 节点以及入网 NODE 节点信息
```

- 参数描述

IN/OUT	NAME	DESC
[in]	u16t unicastaddr	NODE 节点 unicast_addr

- 返回值

返回值	
非 NULL	成功，返回参数参见 mesh*node*t 结构体定义
< 0	失败

- 注意事项

  用户输入的 unicast_addr 地址范围若是属于 Provisioner 本身地址，则返回 Provisioner NODE 节点信息，否则返回已入网设备节点地址信息

```
blemeshprovisionergetprovisioner_data
```

- 函数原型

```
const provisioner_comp *ble_mesh_provisioner_get_provisioner_data()
```

- 功能描述

  获取 provisioner comp

- 参数描述

  无

- 返回值

返回值	
非 NULL	provisioner comp ，参见 provisioner_comp 结构体定义
NULL	失败
struct provisioner_comp(结构体) 定义	
const struct btmeshprovisioner *	mesh provisioner 结构体
uint16t unicastaddr_local	mesh provisioner 本地起始地址
uint16t localsub	mesh provisioner 本地默认 sub 地址
uint16t localpub	mesh provisioner 本地默认 pub 地址

- 注意事项

  若使用 MESH NODE 组件开发，则用户一般不需要使用该接口

平头哥OCC钉钉交流群，
进群为你答疑解惑

平头哥芯片开放社区公众号，
扫码关注获取更多信息与资料

扫码注册平头哥OCC官网，
观看各类蓝牙视频及课程

阿里云开发者“藏经阁”
海量免费电子书下载