# RTL8762C Mesh SDK User Guide

V 1.0.4

2019/1/29

# Revision History

| Date | Version | Comments | Author | Reviewer |
|---|---|---|---|---|
| **2015/10/12** | Draft v0.1 | Draft version | bill | |
| **2015/11/26** | Draft v0.2 | Remove debug info, add new feature introduction | bill | |
| **2017/09/26** | Draft v0.9 | Follow Mesh Spec 1.0 | bill | |
| **2017/10/30** | Draft v0.9.1 | Add pb-adv link information callback, device information callback. Add the setting of device uuid, composition data page and subscribe list operation | bill | |
| **2017/11/09** | Draft v0.9.2 | Add the mechanism introduction of Mesh, and the flash/ram resource usage statistics of demo projects | bill | |
| **2018/01/17** | Draft v0.9.3 | Adjust the architecture, add the light application | bill | |
| **2018/04/23** | Draft v0.9.5 | Modify the API trans_ping. Modify the user cmd dus/pbadvcon/con, add dis, delete rembd. | bill | |
| **2018/09/21** | Draft v1.0.3 | Sync with Chinese edition | bill | |
| **2018/09/26** | Draft v1.0.4 | Reviewed | astor | |

# Contents

# Table List

# Figure List

# Glossary

| Terms | Definitions |
|---|---|
| | |

# 1 Mesh Introduction

Bluetooth Low Energy Mesh is a mesh networking solution based on BLE. There are two mine kinds of mesh network: the flooding-based mesh network and the routing-based mesh network. Both have pros and cons, and BLE Mesh tries to integrate the advantages of both. Up to now, BLE mesh only makes use of the flooding method.

BLE has two kinds of physical channel: advertising channel and data channel. Mesh mainly works on the advertising channel, it uses passive scan to receive mesh packets and uses advertising to transmit mesh packets. To be compatible with some devices which don’t support advertise, the mesh network uses LE link on data channel to send mesh packets to the mesh network, known as proxy.

This document will introduce the vital concepts of mesh in brief. Please refer to Mesh Profile Specification [1] for detailed information, or refer to the second chapter “mesh system architecture” in the specification for a basic understanding of the BLE mesh solution.

## 1.1 Device UUID

Each mesh device is assigned a 128-bit UUID as a unique identity, known as Device UUID. Thus, there is no need to rely on the Bluetooth address to identify each device. When establishing a PB-ADV link in provisioning procedure, the Device UUID of the target device is used to identify the device. After the mesh device attains the mesh address, the mesh address can be used to identify the device individually.

## 1.2 Mesh Address

Except creating a LE link, the communication of mesh devices doesn’t rely on the Bluetooth address, i.e. address of each node can be the same or varies randomly. However, the mesh network defines its own 16-bit mesh address, and the address is classified into four types: unassigned, unicast, virtual and group, as shown in Table 1-1.

The mesh address is not assigned by the manufacture but uniformly managed and assigned by the user. Devices will be allocated addresses during the provisioning procedure when configured to enter network. A device may have multiple mesh addresses to distinguish every duplicate model in one device, and each element of each node will be assigned one continuous unicast address. The Provisioner is responsible for allocating no duplicate unicast addresses to the mesh devices.

Table 1-1 Mesh Address

| Range | Address Type | Instruction |
|---|---|---|
| 0x0000 | unassigned address | The default address of device, can’t be used to send/receive mesh packets |
| 0x0001~0x7fff | unicast address | Each element in node will be assigned one unicast address.<br>The source address of mesh message must be the unicast address. |
| 0x8000~0xbfff | virtual address | Virtual address　is generated by the label UUID and has tremendous amount, so there is no need to apply centralized management |
| 0xc000~0xffff | group address | Group address, which can be divided into two types:<br>dynamic group address（0xc000~0xfeff）<br>fixed group address（0xff00~0xffff） |

## 1.3 Application Architecture

BLE link is one to one and connection-based between two devices, known as master and slave. However, the mesh network is a many-to-many communication architecture. As a result, there is a nature feature that nodes do not know the existence of each other. So we need a third party, usually the provisioner, to play the role of navigator to guide the connection of devices. For example, a generic switch device bought by a customer doesn’t know which bulb is controlled by itself. At this moment, the customer need use the provisioner to configure the publish address of the new mesh switch where to send the lighting control message, and add the address to the subscribe address list of the bulbs where to receive the lighting control message. If the publish address is an unicast address of a bulb, then the switch can control the specific bulb, but if the publish address is a group address, the switch can control a group of bulbs.

Mesh standardizes operations in typical application scenarios and the application on every mesh device is organized in the unit of Model. Each model defines a model ID, a set of opcodes and a group of states to stipulate which messages should be received and sent, and which states to be operated. Mesh model is similar to BLE GATT service, both of them defines a specific application scenario.

To support multiple duplicate models on one device, the concept of element is defined, and each element will be assigned one mesh address which is continuous. That is to say, duplicate models are dispersed on different elements so that the duplicate models can use different mesh addresses to send or receive mesh messages. The address of the first element, also called primary element, is the node address which is allocated in the provisioning procedure, and the address of the other element is increased by one in sequence. For example, there are two same bulbs on one mesh device and can be controlled individually A switch is used to control both bulbs and it needs to identify which bulb it is controlling. Put the corresponding models of both bulbs into 2 elements to make the switch able to control bulbs individually. Certainly, both bulbs can also be controlled at the same time with a group address if the group address is subscribed by both lighting models.

The element and model structure on a mesh device is described by the composition data page 0, the Provisioner can recognize the supported application type of the mesh device through composition data page 0.

## 1.4 Security

The mesh network is designed to protect the security and privacy from many attacks, such as passive eavesdrop, man-in-the-middle attack, replay attack, trashcan attack and brute-force crack.

All mesh messages are encrypted and authenticated to protect against eavesdropping and forge. There are two layers of secret keys: the network key (NetKey) and the application key (AppKey), and each layer has up to 4096 keys which is identified by a 12-bit global key index. The AppKey shall be bound to one and only one NetKey. Access layer messages will be encrypted and authenticated by the AppKey and then the NetKey successively. It can prevent the relay nodes eavesdropping or faking the relayed messages to use two layers of secret keys. For example, Node A transmits a message to Node C via Node B, Node A&B&C have the same NetKey, but only Node A&C have the same AppKey. Then Node B can only use the NetKey to relay the message at the network layer, so that Node A and Node C can communicate secretly at the application layer without worrying about Node B.

NetKey supports multiple keys to help separate Mesh range and isolate devices. The key with index 0 is primary net key and the rest are subnet keys. Only the nodes in primary network can participate in IV Update Procedure and propagate IV update information to other subnets. In other words, only nodes of primary network can update IV index network parameters, while the nodes of subnets can only accept IV index update passively. The unequal design of network key aims at restraining data transmission frequency of subnet nodes to prevent subnet nodes from abusing IV index and running out of IV index, in case of bringing security issues. Usually majority of nodes are located in primary network, while some nodes belong to both primary network and some subnet. A small amount of nodes that only belong to some subnet can communicate locally within subnet to limit their communication range. For instance, hotel customer can only control the light in his own room instead of others'.

A mesh network can use multiple AppKey to isolate nodes by application, and there is no difference between all the AppKey. For example, the light control and the door control make use of different AppKey. In addition, an application can use multiple AppKey for different users, such as one light support two AppKey, AppKey 1 is for user 1, and AppKey 2 is for user 2. If the AppKey 1 is deleted from the light, then user 1 can't control the light any more, but user 2 won't be influenced.

The NetKey and AppKey on a node is distributed and managed by provisioning procedure and configuration client. A provisioner is a network administrator which manages all keys and nodes and makes sure that devices that need to communicate with each other share the same keys for both network and access layers, e.g. light and the switch that controls the light share the same key. Mesh address and one and only one NetKey will be distributed during provision procedure, other network parameters will be managed through configuration afterwards, e.g. adding NetKey and AppKey.

=A special AppKey will also be generated in provision procedure randomly, known as DevKey, which is unique for each device. DevKey is only known by the device and provisioner. The DevKey is not shared with other devices, so the provisioner can achieve one to one secure communication with the target device. Only the device with DevKey can access the configuration server, so only provisioner can configure target device, e.g. switch can control the light but cannot group the light.

## 1.5 Provisioning

The mesh device does not contrain the mesh address and the network keys by default. The customer should provision the mesh device to a mesh node by assigning the mesh address and the network keys. This document and demo projects don't distinguish the two concept of mesh device and mesh node strictly. The provisioning procedure, which is similar to the secure connection pairing in BLE, makes use of ECDH algorithm to negotiate and distribute the keys. Besides, the provisioning procedure uses the authentication data to authenticate the identity of the mesh device to guarantee the security of the mesh network key NetKey, which protects devices against the eavesdropper, brute-force crack and MITM.

According to the different provision configuration of the mesh device, there will be different specific procedures. There are 2 interaction methods of public key, oob and no oob respectively. There are four kinds of interaction methods of the authentication data: input OOB, output OOB, static OOB and no OOB. Thus, there are up to 8 modes. Application layer of device may need to provide public/private key and authentication data to stack while that of provisioner may need to provide mode selection, public key of device and authentication data to stack.

The provisioning procedure can work on the advertising channel or the data channel, corresponding to the PB-ADV and PB-GATT bearer separately. The mesh device is mandatorily required to support PB-ADV, but optionally required to support PB-GATT. If the device supports both, then the provisioner can choose any bearer.

## 1.6 Configuration

The mesh network parameters of a mesh device are managed on the model layer, known as configuration models. The configuration model can configure many parameters, such as adding, modifying and deleting the NetKey and AppKey, binding model key, the publishing and subscription of message, acquiring composition page 0 ,default TTL of node, node features, network retransmit times and so on.

The configuration models are limited to use the DevKey for the purpose of security, i.e. only the provisioner can configure the network parameters of a mesh device.

## 1.7 Proxy

The mesh network mainly works on the advertising channel, and it defines the proxy function on data channel to support the Bluetooth devices that are not convenient to advertise. The proxy is constructed based on the GATT profile, and it uses GATT service to transmit mesh messages on the BLE link.

# 2 Mesh Architecture

The Mesh Profile specification is defined as a layered architecture, including bearer layer, network layer, lower transport layer, upper transport layer, access layer, foundation model layer and model layer, as shown in Figure 2-1.

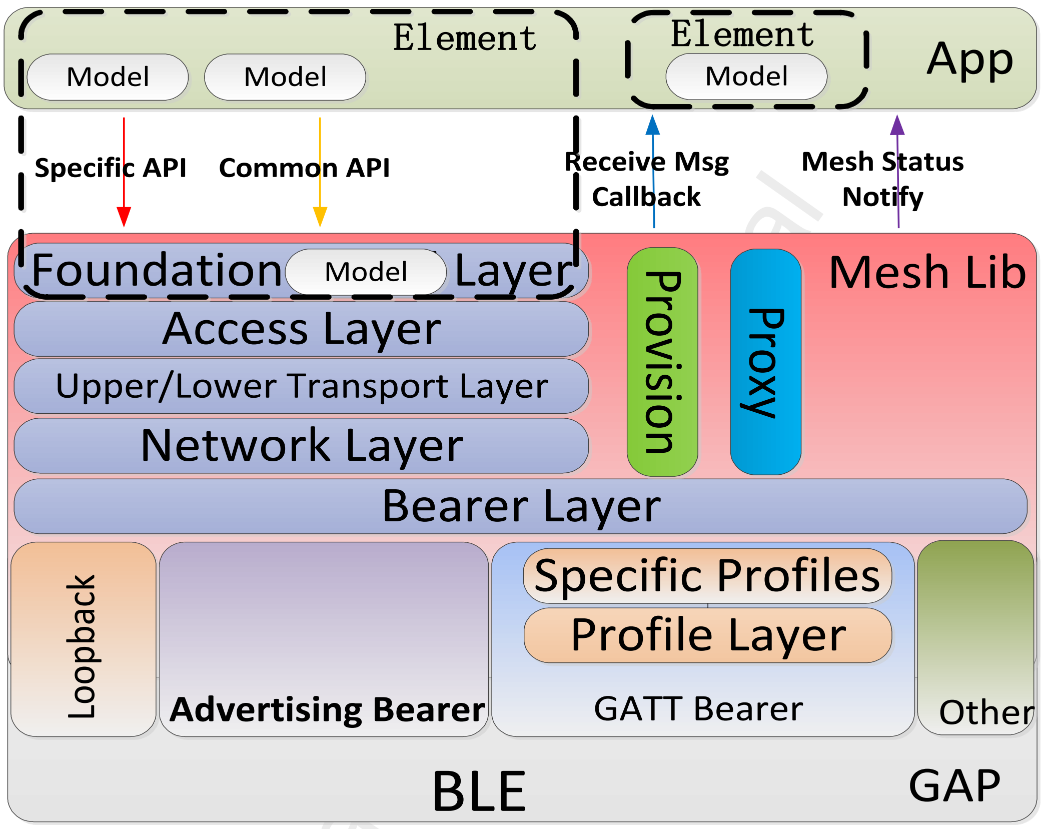


Figure 2-1 Mesh Architecture

## 2.1 Model Layer (Foundation)

Each model defines a group of opcodes and states, and it is identified by model ID. Models are divided to the SIG model and the vendor model, using 16-bit model ID and 32-bit model ID separately. In implementation, the 16-bit SIG model ID is coded to 32 bits model id. For example, the configuration server and client models ID are 0x0000 and 0x0001, but is coded to 0x0000FFFF and 0x0001FFFF in implementation, as listed in Table 2-1.

**Table 2-1 Model ID**

```
#define MESH_MODEL_CFG_SERVER                           0x0000FFFF
#define MESH_MODEL_CFG_CLIENT                          0x0001FFFF
```

The concept of element is used to isolate duplicate models, which use same access layer opcode. Because the duplicate models will use the same opcode, it can't be distinguished that who sends the the message if they use the same mesh address. So the element is used to separate duplicate models that use the same access layer opcode. Different element will be assigned different mesh addresses, and then models can be distinguished by source address or destination address of message. The mesh stack needs to create element firstly, and then register model under the appointed element. If to register a duplicate model, the application shall assign a new element to the duplicate model. Besides, the mesh stack is not responsible to detect duplicate models under the same element, but the application shall take charge.

The configuration server model is mandatory in the mesh specification, and is registered by the stack automatically.

The health server model, which is only mandatory in the primary element, is registered by the application.

After the creation of elements and models, the application can provide the company ID, product ID, version ID, replay list size and the supported features information to generate the composition data page 0.

The creation of elements, the register of models, and the generation of composition data page 0 is demonstrated in Table 2-2.

**Table 2-2 Element/Model Creation**

```
mesh_element_create(GATT_NS_DESC_UNKNOWN);
health_server_reg(0, &health_server_model);
ping_control_reg(ping_app_ping_cb, pong_receive);
compo_data_page0_header_t compo_data_page0_header = {COMPANY_ID, PRODUCT_ID,
VERSION_BUILD};
compo_data_page0_gen(&compo_data_page0_header);
```

## 2.2 Access Layer

The access layer defines the message format of upper layer, including the operation codes (access opcode). All mesh access messages shall use uniform opcodes. The mesh specification defines a part of opcodes, and leaves the space for vendors to define vendor opcodes. The manufacturer-specific opcode is a 3 bytes opcode that the highest two bits of the first byte are 1, and the last two bytes is the vendor company ID. That is to say, every company can only define 6-bit of the opcode with the maximum amount of 64. As shown in Table 2-3, 0x005D is the company ID of Realtek which shall be filled in the byte order, and 0xC05D00 is the first mesh vendor opcode of Realtek.

**Table 2-3 Access Opcode**

```
#define MESH_MSG_PING                                   0xC05D00
```

The access messages will be encrypted and authenticated by AppKey. TransMIC authentication has length 4 bytes and 8 bytes. If 4 bytes TransMIC is used, the maximum length of the access message is 380 bytes, otherwise the length is 376 bytes.

## 2.3 Transport Layer (Upper & Lower)

The transport layer is responsible for encrypting and decrypting the access message, friendship maintenance, heartbeat signaling, and the segmentation & reassemble of the upper transport layer messages.

Access layer message applies 4-byte TransMIC. If message length is no greater than 11 bytes (includes access opcode) and  SAR isn't requested, the message won't be segmented and will be sent in a single advertising packet. Otherwise, access message will be segmented and sent with one or more advertising packets, and the response of acknowledge of transport layer will be required from receiver.

### 2.3.1 Transport Ping

For convenience, the ping and pong messages at the transport layer is defined to test the connectivity of the network layer. The ping & pong messages contain the TTL value which can be used to calculate the hop times between the two devices. It directly uses service on network layer and will only be encrypted by NetKey. After being provisioned, the device can use the transport ping to test the mesh network without any AppKey.

**Table 2-4 Transport Ping**

| | |
|---|---|
| **Prototype** | **mesh_msg_send_cause_t trans_ping(uint16_t dst, uint8_t ttl, uint8_t net_key_index, uint16_t pong_max_delay)** |
| **Function** | send ping message to the designated address |
| **Parameters** | dst: destination address |

```
ttl: initial TTL value
net_key_index: NekKey index
pong_max_delay: Maxium pong delay with unit of 10ms
```

## 2.4 Network Layer

The network layer is responsible for encrypting, decrypting, obfuscating and relaying the network layer packets.

## 2.5 Bearer Layer

The bearer layer consists of loopback bearer, advertising bearer, GATT bearer and other bearer. The loopback bearer is an internal bearer for the communication between the models in the device. The other bearer is used to extend the range of the mesh network. For example, a gateway mesh device can use other bearer to connect the mesh network with the Internet. The interface of other bearer is listed in Table 2-5, which is used to register the callback to send and receive the mesh messages separately.

**Table 2-5 Other Bearer**

```
void bearer_other_reg(pf_bearer_other_send_t send);
void bearer_other_receive(bearer_pkt_type_t pkt_type, uint8_t *pbuffer, uint16_t len);
```

## 2.6 Provisioning

The device doesn’t have any network information in factory settings, and it need attain the mesh address and NetKey from the provisioner in the provisioning procedure. The device has to configure the provisioning capabilities, as shown in Table 2-6.

**Table 2-6 Device Provisioning Setting**

```
prov_capabilities_t prov_capabilities =
{
    .algorithm = PROV_CAP_ALGO_FIPS_P256_ELLIPTIC_CURVE,
    .public_key = 0,
    .static_oob = 0,
    .output_oob_size = 0,
    .output_oob_action = 0,
            .input_oob_size = 0,
            .input_oob_action = 0
        };
        prov_params_set(PROV_PARAMS_CAPABILITIES, &prov_capabilities,
sizeof(prov_capabilities_t));
```

Both the device and provisioner have to register the callback function to handle the provisioning process as shown in Table 2-7. According to the provisioning capability settings and the choice of the provisioner, the device and provisioner have to deal with different callback message.

**Table 2-7 Provisioning Callback Register**

```
prov_params_set(PROV_PARAMS_CALLBACK_FUN, prov_cb, sizeof(prov_cb_pf));
```

The provisioning procedure starts after the provisioning bearer is established. There are two kinds of bearer: PB-ADV and PB-GATT. The establishment of the bearer is introduced in the followed section.

### 2.6.1 PB-ADV

According to the device UUID, the provisioner launches the establishment procedure of the PB-ADV link using the API shown in Table 2-8.

**Table 2-8 PB-ADV Link Creation**

| Prototype | void pb_adv_link_open (uint8_t ctx_index, uint8_t device_uuid[16]) |
|---|---|
| Function | start to establish the PB-ADV bearer |
| Parameters | ctx_index： PB-ADV context<br>device_uuid： Device UUID of the unprovisioned device |

In the provisioning callback, the application can attain the PB-ADV link state information, as show in Table 2-9.

**Table 2-9 PB-ADV Link State Callback**

```
            case PROV_CB_TYPE_PB_ADV_LINK_STATE:
                switch(cb_data.pb_generic_cb_type)
                {
                    case PB_GENERIC_CB_LINK_OPENED:
                        data_uart_debug("Link Opened!\r\n>");
                        break;
                    case PB_GENERIC_CB_LINK_OPEN_FAILED:
                        data_uart_debug("Link Open Failed!\r\n>");
                        break;
                    case PB_GENERIC_CB_LINK_CLOSED:
                        data_uart_debug("Link Closed!\r\n>");
                        break;
                    default:
                        break;
                }
                break;
```

## 2.6.2 PB-GATT

To build a PB-GATT bearer, the provisioner needs to connect the device to create a LE link, inquire the provisioning GATT server, and then enable corresponding CCCD.

## 2.7 Friendship

To support low-power devices, the friendship between the friend node (FN) and the low power node (LPN) is created, then the FN will help to cache the message sent to the LPN so that the LPN can sleep to save energy.

The friendship communication will consume the space of internal NetKey in implement. The number of friendship that a FN can establish is less than the number of NetKey. The specification defines that a LPN can only establish only one friendship on a NetKey. So the number of friendship is less than half of the number of NetKeys.

### 2.7.1 Friend Node

To support the FN feature, the application need initialize the friendship setting using the API shown in Table 2-10.

**Table 2-10 FN Initiation**

| **Prototype** | **bool fn_init(uint8_t lpn_num, uint8_t queue_size, pf_fn_cb_t pf_fn_cb)** |
|---|---|
| Function | the initiation of FN |
| Parameters | lpn_num： the number of supported LPNs<br>queue_size： friend queue size<br>pf_fn_cb: friendship callback function pointer |

### 2.7.2 Low Power Node

To support the LPN feature, the application need initialize the friendship setting using the API shown in Table 2-11Table 2-10.

**Table 2-11 LPN Initiation**

| **Prototype** | **bool lpn_init(uint8_t fn_num, pf_lpn_cb_t pf_lpn_cb)** |
|---|---|
| Function | the initiation of LPN |
| Parameters | fn_num: the number of supported FNs<br>pf_lpn_cb: friendship callback function pointer |

The friendship is launched by LPN using the API shown in Table 2-12.

**Table 2-12 LPN Launches Friendship**

| **Prototype** | **lpn_req_reason_t lpn_req(uint8_t frnd_index, uint8_t net_key_index, lpn_req_params_t *p_req_params)** |
|---|---|
| Function | launch the friendship |
| Parameters | frnd_index：FN index<br>net_key_index：the network key on which to establish a friendship<br>p_req_params：the friendship requirements of LPN |

## 2.8 Miscellaneous

### 2.8.1 Mesh Log Setting

There are many modules in the mesh stack, the log of every module can be controlled on-off individually, and is opened by default. The log has four levels in the descending order: LEVEL_ERROR, LEVEL_WARN, LEVEL_INFO and LEVEL_TRACE. The lower level, the less important the log is. Due to the limitation of the rate of log UART, some of the log will be lost when the application prints too many logs. To make sure the vital logs are printed, some logs that are in low-level priority should be closed.

**Table 2-13 Mesh Log Setting**

```
uint32_t module_bitmap[MESH_LOG_LEVEL_SIZE] = {0};
diag_level_set(LEVEL_TRACE, module_bitmap);
```

### 2.8.2 Device Parameters Setting

The application layer can invoke gap_sched_params_set API to set the device name and appearance. The default device name is “rtk_mesh”, and the default appearance is unknown.

**Table 2-14 Device Parameters Setting**

```
char *dev_name = "Mesh Device";
uint16_t appearance = GAP_GATT_APPEARANCE_UNKNOWN;
gap_sched_params_set(GAP_SCHED_PARAMS_DEVICE_NAME, dev_name, GAP_DEVICE_NAME_LEN);
gap_sched_params_set(GAP_SCHED_PARAMS_APPEARANCE, &appearance, sizeof(appearance));
```

### 2.8.3 Device UUID setting

The Device UUID is used to distinguish the mesh device, and can be set by the API shown in Table 2-15.

**Table 2-15 Device UUID Setting**

```
uint8_t dev_uuid[] = MESH_DEVICE_UUID;
device_uuid_set(dev_uuid);
```

### 2.8.4 Network parameters and features setting

Network parameters and features need be configured according to the specific application, such as the supported features, the number of NetKeys and AppKeys, the number of virtual addresses, the size of subscription list, flash storage and advertising period etc., as shown in Table 2-16.

**Table 2-16 Networking Parameters & Features setting**

```
mesh_node_features_t features =
{
    .role = MESH_ROLE_DEVICE,
    .relay = 1,
    .proxy = 1,
    .fn = 0,
    .lpn = 0,
    .prov = 1,
    .udb = 1,
    .snb = 1,
    .bg_scan = 1,
    .flash = 1,
    .flash_rpl = 1
};
mesh_node_cfg_t node_cfg =
{
    .dev_key_num = 1,
    .net_key_num = 3,
    .app_key_num = 3,
    .vir_addr_num = 3,
    .rpl_num = 20,
    .sub_addr_num = 10,
    .proxy_num = 1
};
mesh_node_cfg(features, &node_cfg);
```

# 3 Model Architecture

The SIG has defined many standard models in the model spec [2]; most of them have been implemented by Realtek, and have passed in the BQB test.  To simplify the development of application, SDK has encapsulated the message handlers between Model layer and Access layer and provides a series of predefined service messages. Thus, the developer need only focus on development of service logic instead of handling and parsing the messages. For detailed information on Model refer to Mesh Model Specification 错误!未找到引用源。.

## 3.1 Message Callback

A structure of Model message needs to be defined before implementing certain model. The model_receive callback of model is to process the original access messages, and the model_data_cb callback is to process the service logic of model. That is to say, the model layer will implement the model_receive callback, while the application can just implement the simplified data callback model_data_cb.

**Table 3-1 Message Callback**

```
typedef struct _mesh_model_info_t
{
    uint32_t model_id;
    model_receive_pf model_receive;
    model_send_cb_pf model_send_cb; //!< indicates the msg transmitted state
    model_pub_cb_pf model_pub_cb; //!< indicates it is time to publishing
    model_data_cb_pf model_data_cb;
    /** point to the bound model, sharing the subscription list with the binding model */
    struct _mesh_model_info_t *pmodel_bound;
    /** configured by stack */
    uint8_t element_index;
    uint8_t model_index;
    void *pelement;
    void *pmodel;
    void *pargs;
} mesh_model_info_t;
```

## 3.2 Message Definition

In the callback model_data_cb, each model will define a group of callback data. For example, the generic on off server model defines the generic_on_off_server_get_t, generic_on_off_server_get_default_transition_time_t and generic_on_off_server_set_t three messages to interface with the application, as shown in Table 3-2.

**Table 3-2 Message Definition**

```
    #define GENERIC_ON_OFF_SERVER_GET                                      0
#define GENERIC_ON_OFF_SERVER_GET_DEFAULT_TRANSITION_TIME       1
#define GENRIC_ON_OFF_SERVER_SET                                        2

typedef struct
{
    generic_on_off_t on_off
} generic_on_off_server_get_t;

typedef struct
{
    generic_transition_time_t trans_time;
} generic_on_off_server_get_default_transition_time_t;

typedef struct
{
    generic_on_off_t on_off;
    generic_transition_time_t total_time;
    generic_transition_time_t remaining_time;
} generic_on_off_server_set_t;
```

## 3.3 Message Handler

Table 3-3 gives an example of the message handler of simple light control service logic. Define the structure of model first,   and then assign the API generic_on_off_server_data to the model_data_cb callback of generic on off server model. When the the server model receives the generic on off message,   generic_on_off_server_data function will be invoked and filled with the information correspond to the type of message. Application only needs to control the light based on the on/off value obtained from message, instead of focusing on handling the message between model layer and access layer.

**Table 3-3 Message Handler**

```
 /** generic on/off server model */
static mesh_model_info_t generic_on_off_server;

/** light on/off state */
generic_on_off_t current_on_off = GENERIC_OFF;
```

```

/** light on/off process callback */
static int32_t generic_on_off_server_data(const mesh_model_info_p pmodel_info, uint32_t type,
                                          void *pargs)
{
    int32_t ret = MODEL_SUCCESS;
    switch (type)
    {
     case GENERIC_ON_OFF_SERVER_GET:
        {
            generic_on_off_server_get_t *pdata = pargs;
            pdata->on_off = current_on_off;
        }
        break;
    case GENERIC_ON_OFF_SERVER_SET:
        {
            generic_on_off_server_set_t *pdata = pargs;
            current_on_off = pdata->on_off;
            ret = MODEL_STOP_TRANSITION;
        }
        break;
    default:
        break;
    }

    return ret;
}

 /** light on/off model initialize */
void light_on_off_server_models_init(void)
{
    /* register light on/off models */
    generic_on_off_server.model_data_cb = generic_on_off_server_data;
    generic_on_off_server_reg(0, &generic_on_off_server);
}
```

# 4 Configuration & Download

If the default flash layout etc. configuration is used, all related files are located at the SDK. \tool\download\ directory; otherwise, user has to configure and attain the image from the related tool.

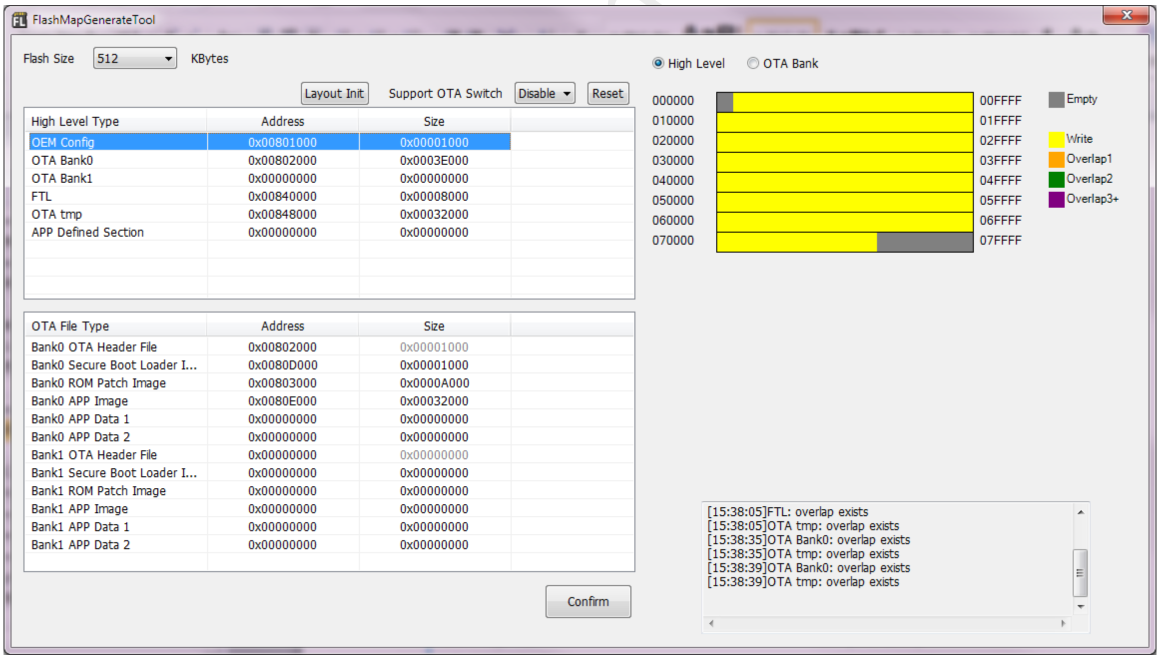
### 4.1.1 Patch

Keep in mind to use the mesh-specific patch image in Mesh SDK.

### 4.1.2 Download

#### 4.1.2.1 Flash layout

The mesh application is configured to use 512KB flash by default. The flash map tool setting is shown in Figure 4-1. The flash map tool will generate the flash map.ini for other tools and the flash_map.h for all mesh projects.



Flash Size 512 KBytes

Support OTA Switch Disable

| High Level Type | Address | Size |
|---|---|---|
| OEM Config | 0x00801000 | 0x00001000 |
| OTA Bank0 | 0x00802000 | 0x0003E000 |
| OTA Bank1 | 0x00000000 | 0x00000000 |
| FTL | 0x00840000 | 0x00008000 |
| OTA tmp | 0x00848000 | 0x00032000 |
| APP Defined Section | 0x00000000 | 0x00000000 |

| OTA File Type | Address | Size |
|---|---|---|
| Bank0 OTA Header File | 0x00802000 | 0x00001000 |
| Bank0 Secure Boot Loader I... | 0x0080D000 | 0x00001000 |
| Bank0 ROM Patch Image | 0x00803000 | 0x0000A000 |
| Bank0 APP Image | 0x0080E000 | 0x00032000 |
| Bank0 APP Data 1 | 0x00000000 | 0x00000000 |
| Bank0 APP Data 2 | 0x00000000 | 0x00000000 |
| Bank1 OTA Header File | 0x00000000 | 0x00000000 |
| Bank1 Secure Boot Loader I... | 0x00000000 | 0x00000000 |
| Bank1 ROM Patch Image | 0x00000000 | 0x00000000 |
| Bank1 APP Image | 0x00000000 | 0x00000000 |
| Bank1 APP Data 1 | 0x00000000 | 0x00000000 |
| Bank1 APP Data 2 | 0x00000000 | 0x00000000 |

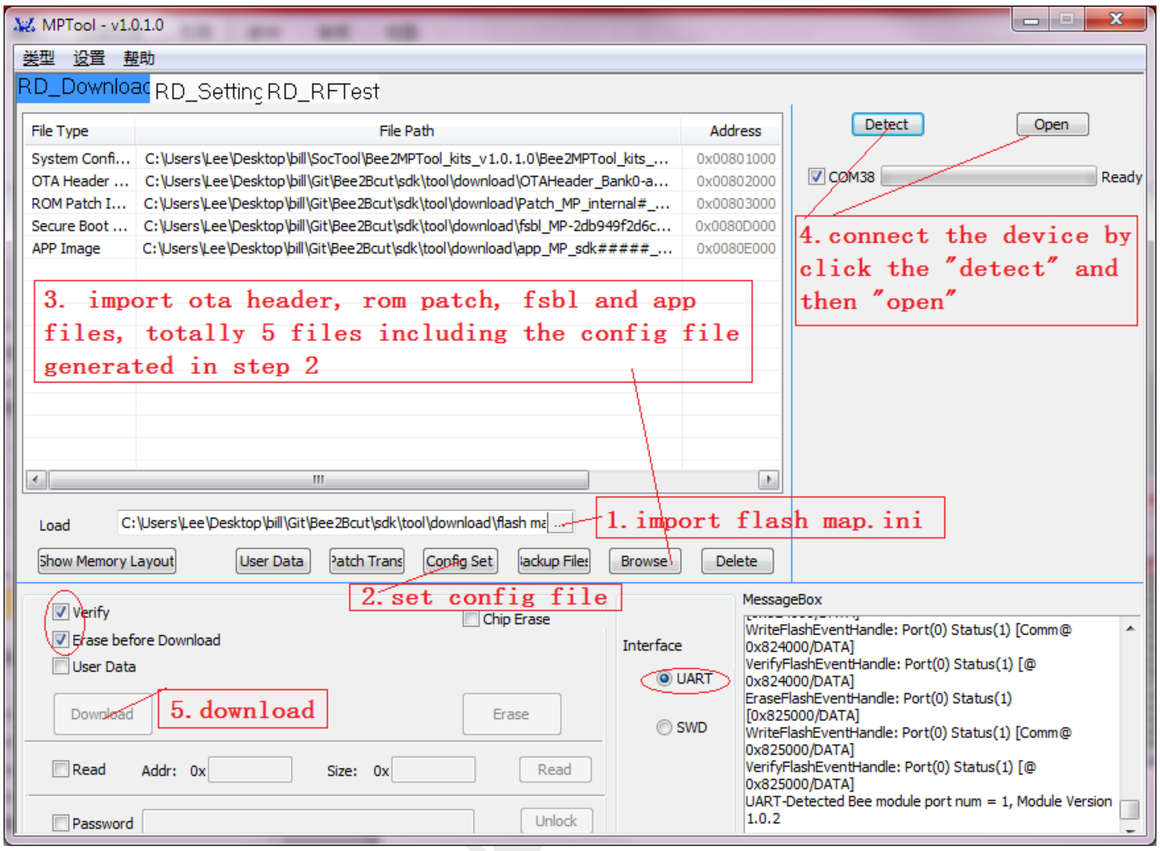
| | |
|---|---|
| 000000 | 00FFFF |
| 010000 | 01FFFF |
| 020000 | 02FFFF |
| 030000 | 03FFFF |
| 040000 | 04FFFF |
| 050000 | 05FFFF |
| 060000 | 06FFFF |
| 070000 | 07FFFF |

```
[15:38:05]FTL: overlap exists
[15:38:05]OTA tmp: overlap exists
[15:38:35]OTA Bank0: overlap exists
[15:38:35]OTA tmp: overlap exists
[15:38:39]OTA Bank0: overlap exists
[15:38:39]OTA tmp: overlap exists
```

Figure 4-1 Flash Map Tool

### 4.1.2.2 MP Tool

The download tool is named as MP tool; the operational approach is shown in Figure 4-2.



| File Type | File Path | Address |
|---|---|---|
| System Confi… | C:\Users\Lee\Desktop\bill\SocTool\Bee2MPTool_kits_v1.0.1.0\Bee2MPTool_kits_… | 0x00801000 |
| OTA Header … | C:\Users\Lee\Desktop\bill\Git\Bee2Bcut\sdk\tool\download\OTAHeader_Bank0-a… | 0x00802000 |
| ROM Patch I… | C:\Users\Lee\Desktop\bill\Git\Bee2Bcut\sdk\tool\download\Patch_MP_internal#_… | 0x00803000 |
| Secure Boot … | C:\Users\Lee\Desktop\bill\Git\Bee2Bcut\sdk\tool\download\fsbl_MP-2db949f2d6c… | 0x0080D000 |
| APP Image | C:\Users\Lee\Desktop\bill\Git\Bee2Bcut\sdk\tool\download\app_MP_sdk#####_… | 0x0080E000 |

```
WriteFlashEventHandle: Port(0) Status(1) [Comm@
0x824000/DATA]
VerifyFlashEventHandle: Port(0) Status(1) [@
0x824000/DATA]
EraseFlashEventHandle: Port(0) Status(1)
[0x825000/DATA]
WriteFlashEventHandle: Port(0) Status(1) [Comm@
0x825000/DATA]
VerifyFlashEventHandle: Port(0) Status(1) [@
0x825000/DATA]
UART-Detected Bee module port num = 1, Module Version
1.0.2
```

Figure 4-2 Download

1. Import the flash map.ini.

2. Click “Config Set” to generate the config file, the detail setting is described in 4.1.2.3.

3. Click “browse”, import four image including ota header, rom patch, fsbl and app.

   If above-mentioned configuration of flash layout is used, OTA Header file provided by Realtek can be applied. Or customers can use the OTA Header file generated with flash map tool and pack tool by themselves. Realtek provides rom patch and fsbl file. App file is compiled and generated with Keil, e.g. after compiling “mesh device” project, app file can be found in \sdk\board\evb\mesh_device\bin\ directory. Note that when use mp tool to download app files, the files containing “MP” in their name must be selected.

There are 5 files to be downloaded in total, includes config file generated in step 2.

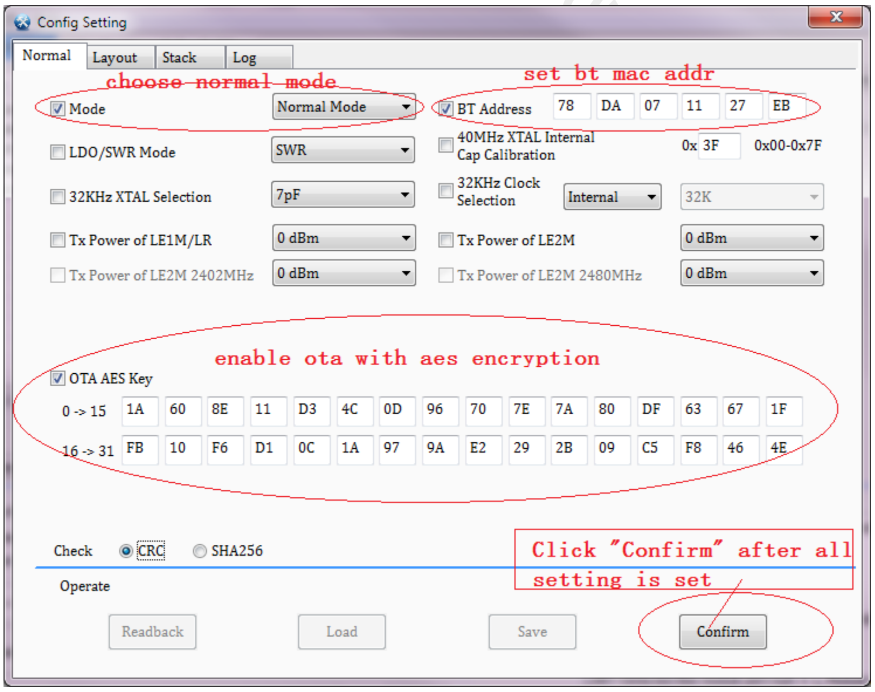
4. Connect UART transfer board to the UART Pin P3_0/P3_1（Tx/Rx）, connect the P0_3 to the GND, reboot the IC, then click “detect” and “open” in sequence to enable the downloading mode.

5. Click “download” to download files.

### 4.1.2.3 Config File

The MP tool will automatically generate the config file and add it to the download file list.

The setting is shown in Figure 4-3 and Figure 4-4.



| | | | |
|---|---|---|---|
| ✓ Mode | Normal Mode | ✓ BT Address | 78 DA 07 11 27 EB |
| LDO/SWR Mode | SWR | 40MHz XTAL Internal Cap Calibration | 0x 3F 0x00-0x7F |
| 32KHz XTAL Selection | 7pF | 32KHz Clock Selection | Internal 32K |
| Tx Power of LE1M/LR | 0 dBm | Tx Power of LE2M | 0 dBm |
| Tx Power of LE2M 2402MHz | 0 dBm | Tx Power of LE2M 2480MHz | 0 dBm |

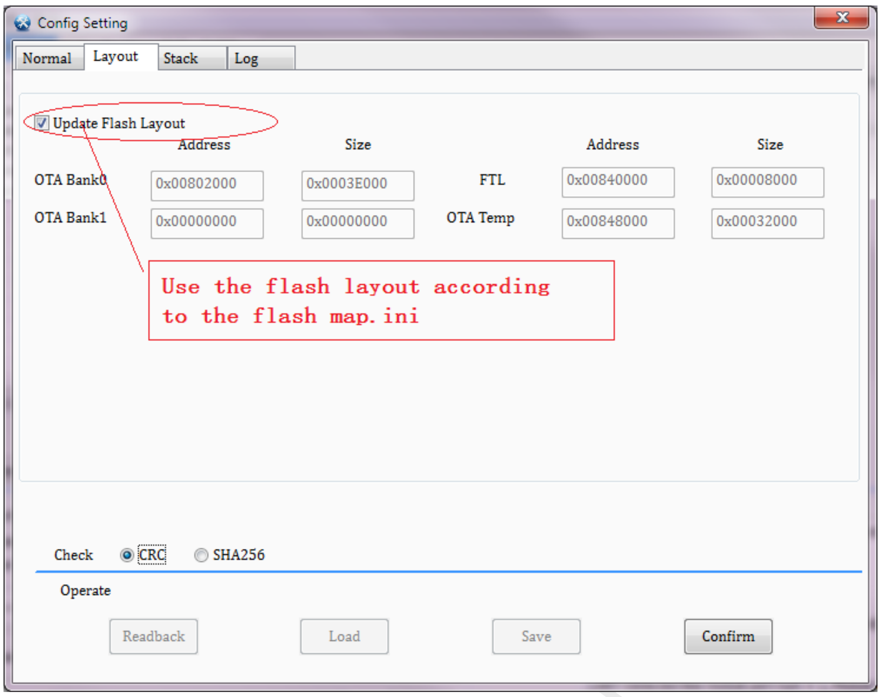
✓ OTA AES Key

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 -> 15 | 1A | 60 | 8E | 11 | D3 | 4C | 0D | 96 | 70 | 7E | 7A | 80 | DF | 63 | 67 | 1F |
| 16 -> 31 | FB | 10 | F6 | D1 | 0C | 1A | 97 | 9A | E2 | 29 | 2B | 09 | C5 | F8 | 46 | 4E |

Check [● CRC] ○ SHA256

Operate

Figure 4-3 config file setting normal



Config Setting

Normal | Layout | Stack | Log

[✓] Update Flash Layout

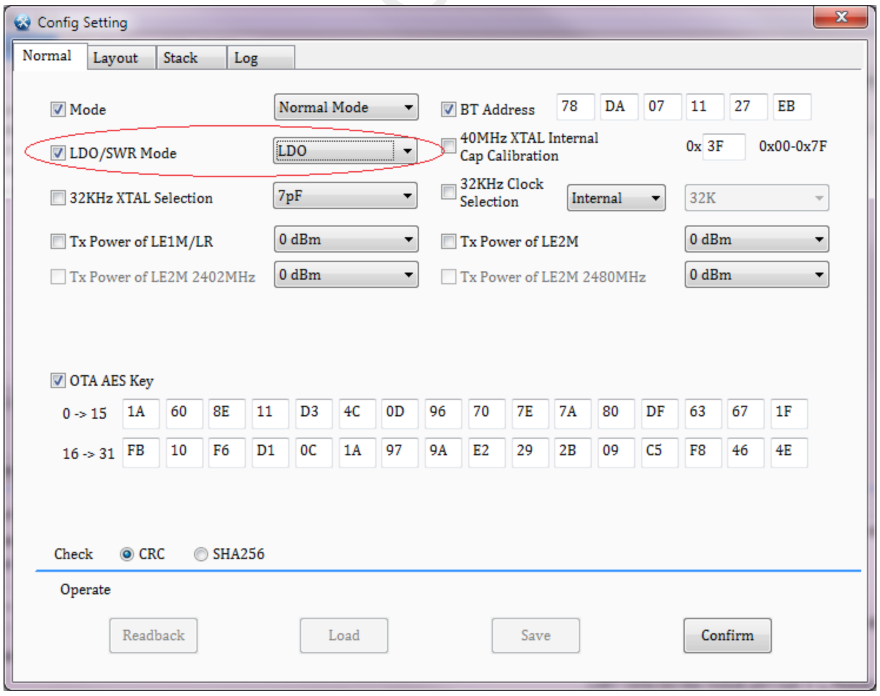
| | Address | Size | | Address | Size |
|---|---|---|---|---|---|
| OTA Bank0 | 0x00802000 | 0x0003E000 | FTL | 0x00840000 | 0x00008000 |
| OTA Bank1 | 0x00000000 | 0x00000000 | OTA Temp | 0x00848000 | 0x00032000 |

Use the flash layout according to the flash map.ini

Check [● CRC] ○ SHA256

Figure 4-4 config file setting layout

Attention: if the IC supports the LDO power mode only (e.g. inductor is not mounted on 8762CMF, only LDO mode can be used), the power mode shall be configured as shown in Figure 4-5.



Config Setting

Normal | Layout | Stack | Log

| | | | |
|---|---|---|---|
| [✓] Mode | Normal Mode | [✓] BT Address | 78 DA 07 11 27 EB |
| [✓] LDO/SWR Mode | LDO | [ ] 40MHz XTAL Internal Cap Calibration | 0x 3F 0x00-0x7F |
| [ ] 32KHz XTAL Selection | 7pF | [ ] 32KHz Clock Selection | Internal 32K |
| [ ] Tx Power of LE1M/LR | 0 dBm | [ ] Tx Power of LE2M | 0 dBm |
| [ ] Tx Power of LE2M 2402MHz | 0 dBm | [ ] Tx Power of LE2M 2480MHz | 0 dBm |

[✓] OTA AES Key

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 -> 15 | 1A | 60 | 8E | 11 | D3 | 4C | 0D | 96 | 70 | 7E | 7A | 80 | DF | 63 | 67 | 1F |
| 16 -> 31 | FB | 10 | F6 | D1 | 0C | 1A | 97 | 9A | E2 | 29 | 2B | 09 | C5 | F8 | 46 | 4E |

Check [● CRC] ○ SHA256

Figure 4-5 config file setting power mode

# 5 Demo Projects

There are four projects in the SDK currently: mesh_provisioner, mesh_device, mesh_light and mesh_ali_light. The mesh_provisioner and mesh_device demonstrate the provisioner and the device separately. The provisioner and device projects can be used together to test the basic function of Mesh. Based on the device, the light project implements the Light HSL Server Models defined in Mesh Model Specification [2]. The mesh_ali_light project demonstrates how to join up the Tmall Genie.

The provisioner is a network manager which is responsible to distribute the mesh address and mesh keys, and configure the mesh nodes. The provisioner will initiate the mesh address of the provisioner itself, and the NetKeys and AppKeys which can be used by the network, as shown in Table 5-1.

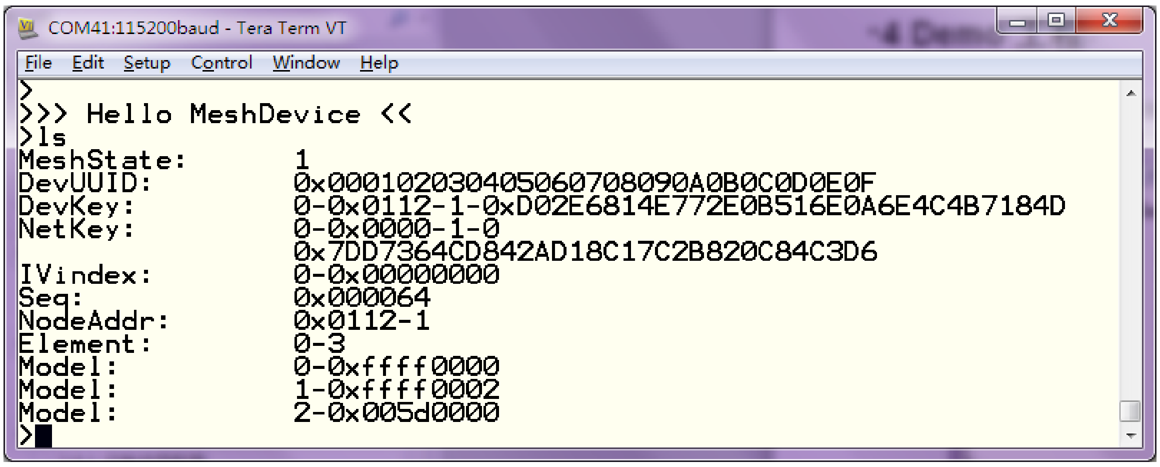
**Table 5-1 Provisioner Setting**

```
mesh_node.node_state = PROV_NODE;
mesh_node.iv_index = MESH_IV_INDEX;
mesh_node.ttl = MESH_TTL_DEFAULT;
mesh_node.unicast_addr = MESH_SRC;

const uint8_t net_key[] = MESH_NET_KEY;
const uint8_t net_key1[] = MESH_NET_KEY1;
const uint8_t app_key[] = MESH_APP_KEY;
const uint8_t app_key1[] = MESH_APP_KEY1;
uint8_t net_key_index = net_key_add(0, net_key);
app_key_add(net_key_index, 0, app_key);
uint8_t net_key_index1 = net_key_add(1, net_key1);
app_key_add(net_key_index1, 1, app_key1);
```

The mesh_provisioner and mesh_device demo projects use the command line user interface (CLI) to send the user command via UART to operate the mesh provisioner and the mesh device, as shown in Figure 5-1. In practice, the user interface may be more user-friendly, for example the smart phone can be used as a provisioner to configure the network. But the CLI user interface can be used to help developers understand the mesh network easily for developers.

Note: There is no CLI interface in the mesh_light project, but it still can be configured by the mesh_provisioner project.


COM41:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
>
>>> Hello MeshDevice <<
>ls
MeshState:      1
DevUUID:        0x000102030405060708090A0B0C0D0E0F
DevKey:         0-0x0112-1-0xD02E6814E772E0B516E0A6E4C4B7184D
NetKey:         0-0x0000-1-0
                0x7DD7364CD842AD18C17C2B820C84C3D6
IVindex:        0-0x00000000
Seq:            0x000064
NodeAddr:       0x0112-1
Element:        0-3
Model:          0-0xffff0000
Model:          1-0xffff0002
Model:          2-0x005d0000
>
```

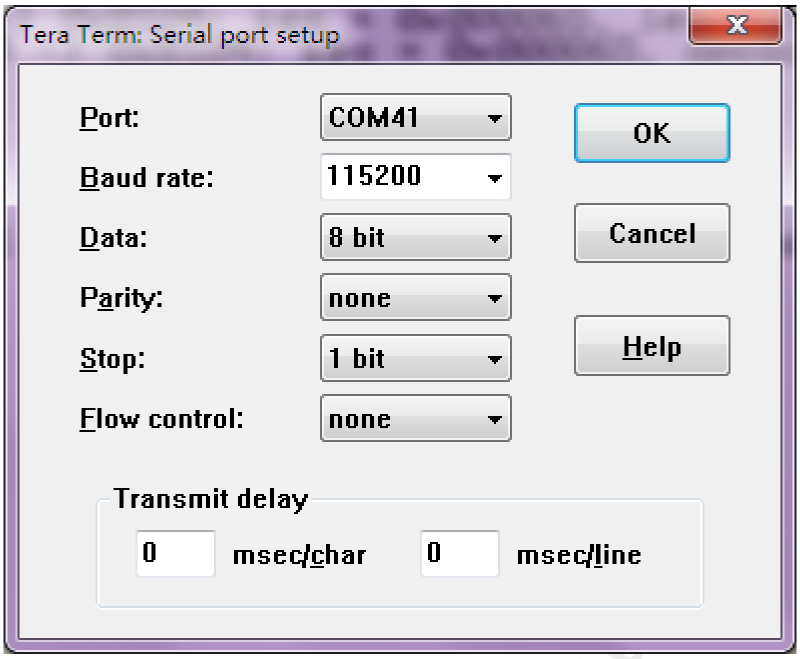
**Figure 5-1 CLI User Interface**

### 5.1.1 UART Setting

The CLI need connect the UART of RTL8762A EVB and the PC. The TX and RX pin of UART are P3_0 and P3_1 separately, as shown in Table 5-2. The UART setting of the PC is configured as shown in Figure 5-2.

**Table 5-2 UART Setting**

```
data_uart_init(P3_0, P3_1, app_send_uart_msg);
user_cmd_init("MeshDevice");
```



Port: COM41
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

**Figure 5-2 PC UART Setting**

### 5.1.2 Command

Some commands are universal for the device and the provisioner, please refer to mesh_cmd.c and mesh_cmd.h. Some of them are listed in Table 5-3.

**Table 5-3 Common Command**

| Cmd | Usage | Description |
|---|---|---|
| ? | ? | list the usage of all the commands |
| xxx ? | xxx ? | list the usage of the xxx command |
| ls | ls | list device information |
| nr | nr | factory reset |
| , | , | previous command |
| . | . | next command |
| [ | [ | cursor moves left |
| ] | ] | cursor moves right |
| / | / | cursor moves top of line |
| \ | \ | cursor moves end of line |
| backspace | backspace | delete one char |

The proper commands of the device are defined in device_cmd.c and device_cmd.h.

The proper commands of the provisioner are defined in provisioner_cmd.c and provisioner_cmd.h.

### 5.1.3 Provisioning

The provisioner can collect the information of the adjacent devices. To enable/disable the collection function, use the command dis as show in Table 5-4.

**Table 5-4 Device Information Collection**

| Step | Provisioner | Description |
|---|---|---|
| 1 | dis 1 | enable the device information show |
| 2 | dis 0 | disable the device information show after the collection of the desired mesh device |

Before provisioning, a PB-ADV bearer or a PB-GATT bearer should be created. The mesh device will get the NetKey and mesh address from the provisioning procedure. Then we can use the ping function at the transport layer to test the network. Besides, the provisioner can use configuration client to get the composition data page 0 from the device that has just been provisioned.

Device will obtain address and network keys during provisioning, after which can it join the network and perform Mesh communication. Ping function of transport layer can be used, i.e. tping command, to test network layer. The test confirms that such device can send/receive ping/pong message of transport layer to/from any node in the mesh network. Provisioner can also communicate with specified device through configuration, e.g. obtain composition data page 0.

Note: The command nr can be used to reset the provisioned device to unprovisioned state.

The provisioning procedure on the PB-ADV bearer is shown in Table 5-5.

**Table 5-5 PB-ADV Provisioning**

| Step | Provisioner | Device | Description |
|---|---|---|---|
| 1 | pbadvcon 000102030405060708090a0b0c0d0e0f | | create the PB-ADV bearer with the device of UUID 000102030405060708090a0b0c0d0e0f |
| 2 | prov | | start the provision procedure |
| 3 | | ls | list the NetKey and mesh address attained |

| | | | |
|---|---|---|---|
| 4 | | tping xffff | test the network communication |
| 5 | cdg x100 | | get the composition data page 0 of the device with address 0x100 |

The provisioning procedure on the PB-GATT bearer is shown in Table 5-6.

**Table 5-6 PB-GATT Provisioning**

| Step | Provisioner | Device | Description |
|---|---|---|---|
| 1 | con deadbeef1234 | | create a LE link with the device of 0xdeadbeef1234 |
| 2 | provdis | | inquire the provision GATT service |
| 3 | provcmd 0 1 | | enable the CCCD of the provision data out characteristic |
| 4 | prov | | start the provision procedure |
| 5 | | ls | list the NetKey and mesh address attained |
| 6 | | tping xffff | test the network communication |
| 7 | cdg x100 | | get the composition data page 0 of the device with address 0x100 |

### 5.1.4 Friendship

After being provisioned, a LPN can establish a friendship with a FN as shown in Table 5-7.

**Table 5-7 Friendship**

| Step | Provisioner | Device | Description |
|---|---|---|---|
| 1 | fninit 1 5 | lpninit 1 | Initiation of both |
| 2 | | lpnreq 0 0 | start a friendship |
| 3 | | tping xffff 3 | test friend cache of the network layer message |

### 5.1.5 Model

#### 5.1.5.1 Key Management

After being provisioned, the mesh node can be configured to add AppKey, bind the AppKey to a ping model. Then the ping model can send ping message of the model layer.

Table 5-8 Key Management

| Step | Provisioner | Device | Description |
|---|---|---|---|
| 1 | aka x100 0 0 | | add the AppKey 0 bound with the NetKey 0 to the device 0x100 |
| 2 | | ls | Check if app key 0 is successfully added to device |
| 3 | mab x100 0 x5d 0 | | bind the AppKey 0 to the ping model of model ID 0x0000005D of the first element of the device 0x100 |
| 4 | | ls | Check if app key 0 is successfully bound to model |
| 5 | | ping xffff 3 | test the ping control model |

### 5.1.5.2 Subscription Management

The provisioner can add a group address to the subscription list of a provisioned mesh node. Then the device can receive the message sent to the group address.

Table 5-9 Subscription Management

| Step | Provisioner | Device | Description |
|---|---|---|---|
| 1 | msa x100 0 x5d xc000 | | add the group address 0xc000 to the ping model of model ID 0x0000005D of the first element of the device 0x100 |
| 2 | | ls | list the subscription list |
| 3 | ping xc000 3 | | test the group address 0xc000 |

## 5.1.6 Lighting Control

Before operating the light, please repeat the provisioning procedure, add the AppKey, bind the AppKey to the models, as done at the provisioner and the device.

Because the great amount of the models of the Light HSL Server, it takes many steps to configure the light, as shown in Table 5-10. Although many models are binded to others, they only share a subscription list, but not the AppKey. So each model need be binded at least one AppKey. If not binded with any AppKey, the model can't send or receive mesh messages. In addition, to configure the model through the configuration client, it need assign the element index of the model, e.g. the element index of the Light HSL Server model, Light HSL Hue Server model and Light HSL Saturation Server model is 0, 1, and 2 separately.

After the provisioning and the configuration, power on and power off rapidly 3 times to restore the light to factory setting: power on and wait for 3~8 seconds, then power off and wait the light to exhausts the energy. At last, when power on the light for the fourth time, the light will be factory restored.

**Table 5-10 Light Models Configuration**

| Step | Provisioner | Description |
|---|---|---|
| 1 | mab x100 0 x1005d 0 | bind the AppKey 0 to the vendor light cwrgb model with model ID 0x0001005D of the 0th element of the light 0x100 |
| 2 | mab x100 0 x1307ffff 0 | bind the AppKey 0 to the sig light hsl model with model ID 0x1307 of the 0th element of the light 0x100 |
| 3 | mab x100 0 x1300ffff 0 | bind the AppKey 0 to the sig light lightness model with model ID 0x1300 of the 0th element of the light 0x100 |
| 4 | mab x100 0 x1000ffff 0 | bind the AppKey 0 to the sig generic on off model with model ID 0x1000 of the 0th element of the light 0x100 |
| 5 | mab x100 0 x1002ffff 0 | bind the AppKey 0 to the sig generic level model with model ID 0x1002 of the 0th element of the light 0x100 |
| 6 | mab x100 1 x130affff 0 | bind the AppKey 0 to the sig light hsl hue model with model ID 0x130a of the 1th element of the light 0x100 |
| 7 | mab x100 1 x1002ffff 0 | bind the AppKey 0 to the sig generic level model with model ID 0x1002 of the 1th element of the light 0x100 |
| 8 | mab x100 2 x130bffff 0 | bind the AppKey 0 to the sig light hsl saturation model with model ID 0x130b of the 1th element of the light 0x100 |
| 9 | mab x100 2 x1002ffff 0 | bind the AppKey 0 to the sig generic level model with model ID 0x1002 of the 2th element of the light 0x100 |

#### 5.1.6.1 Proprietary Lighting

In the light project, there are not only the SIG Lighting models, but also a proprietary lighting model, named as Light CWRGB model. The Light CWRGB model is simpler than the SIG defined light models for the sake of comparison. The CWRGB means cold, warn, red, green and blue. The CW channels are tunable white light that are used to illuminate and their color temperature can be adjusted, while the RGB channels are the color changing part of a light.

The Light CWRGB is a vendor model, and it defines four manufacturer-specific opcodes, as shown in Table 5-11.

Table 5-11 CWRGB Model

```
#define MESH_MSG_LIGHT_CWRGB_GET                        0xC45D00
#define MESH_MSG_LIGHT_CWRGB_SET                        0xC55D00
#define MESH_MSG_LIGHT_CWRGB_SET_UNACK                    0xC65D00
#define MESH_MSG_LIGHT_CWRGB_STAT                       0xC75D00

#define MESH_MODEL_LIGHT_CWRGB_SERVER                     0x0001005D
#define MESH_MODEL_LIGHT_CWRGB_CLIENT                    0x0002005D
```

Because the manufacturer-specific opcode takes three bytes, and the length of an unsegmented access message is 11 bytes, then there are only 8 bytes left for the parameters. So the Light CWRGB Set and Light CWRGB Status messages have only 5 bytes to represent five channels of CWRGB, and each channel is 1 byte which means the total number of steps is 256.

After the configuration of the Light CWRGB server model, the provisioner can control the CWRGB state of the light using the commands list in Table 5-12.

Table 5-12 Light CWRGB Operation

| Cmd | Usage | Description |
|---|---|---|
| lrg | lrg 0x100 | attain the cwrgb state of light 0x100 |
| lrs | lrs 0x100 ff0000 | lighten the red channel |

#### 5.1.6.2 SIG Lighting

The SIG Light HSL models use the three dimensions (hue, saturation and lightness) to control the color which is equivalent to the RGB control. In practice, when received the HSL messages, the light will close the CW channels, and convert the HSL values to RGB values, and then control the LEDs as like the Light CWRGB model. The model structure of the Light HSL is complicated as defined in Mesh Model Specification [2]. The Light HSL Server has three elements, and each element has multiple models. The Light HSL Server models also extend the Light Lightness model and Generic Level models. The binding relation between states is shown in figure 5-3, where the red lines shows the relationship between the 3 scalars (Hue, Saturation and Lightness).

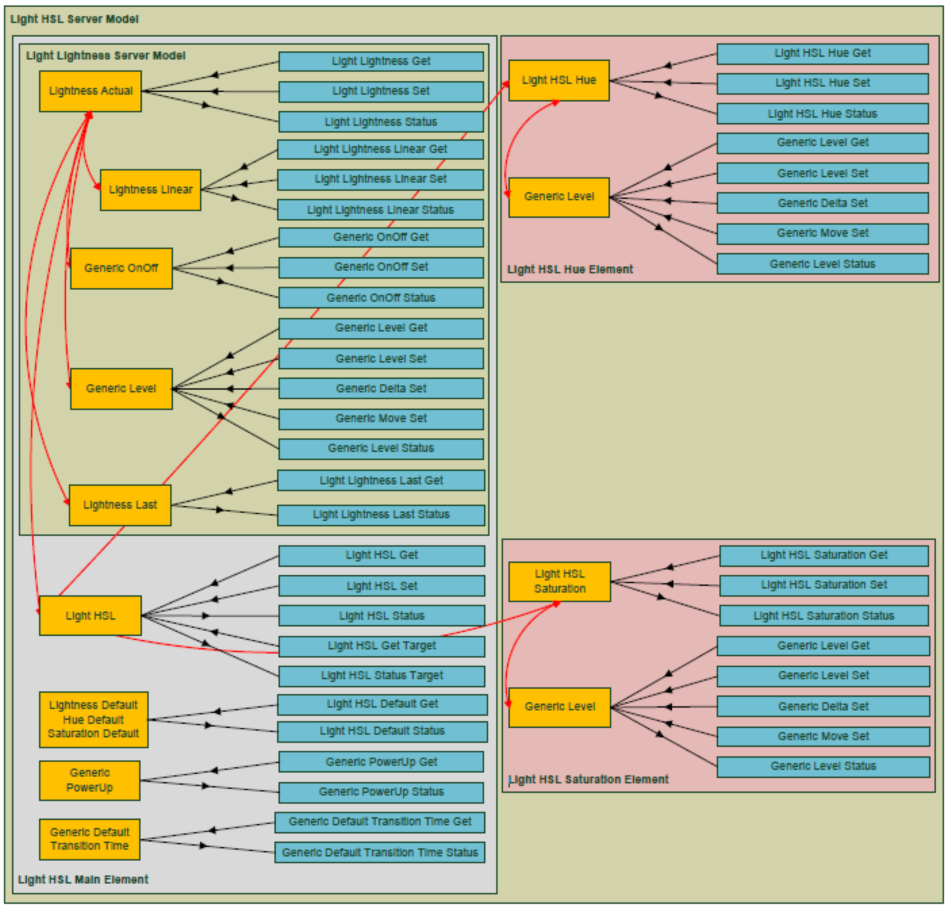


**Figure 5-3 Light HSL Server Models**

Since there are many models in Light HSL Server, many messages are defined to control the light in different ways. The Light HSL server itself defines the messages that can control the states of the HSL three dimensions at the same time or control each channel individually. However, the extended generic models can also control certain channel or the entire light.

Table 5-13 lists the commands to operate the Light HSL server models. Notice that the lightness parameter in the Light HSL model messages is the perceived actual lightness. The perceived lightness of a light is the square root of the measured light linear intensity. Besides, when operate the hue and saturation channels, the destination address is the element address of the Light HSL Hue model and the Light HSL Saturation model separately. For example, if the light attains a node address of 0x100 from the provisioning procedure, then using the address 0x101 to control the hue state, and using the address 0x102 to operate the saturation state.

**Table 5-13 Light HSL Operation**

| Cmd | Usage | Description |
|---|---|---|
| lhg | lhg 0x100 | attain the HSL state of the light 0x100 |

| | | |
|---|---|---|
| lhs | lhs 0x100 0 65535 46340 | lighten the red channel |
| lhhg | lhhg 0x101 | attain the hue state of the light 0x100 |
| lhhs | lhhs 0x101 21845 | change the hue state to green color |
| lhsg | lhsg 0x102 | attain the saturtaion state of the light 0x100 |
| lhss | lhss 0x102 0 | change the saturation to 0, i.e. white light |
| llg | llg 0x100 | attain the lightness of the light 0x100 |
| lls | lls 0x100 0 | change the lightness to 0, i.e. close the light |
| lllg | lllg 0x100 | attain the linear lightness of the light 0x100 |
| llls | llls 0x100 32767 | change the linear lightness of the light 0x100 |
| goog | goog 0x100 | attain the on/off state of the light 0x100 |
| goos | goos 0x100 0 | switch off the light 0x100 |

### 5.1.7 Tmall Genie

In the ecosystem of Alibaba mesh, the Tmall Genie plays the role of provisioner to manage the devices. Alibaba defines the specification of interface between the Tmall Genie and mesh devices. Only the devices that satisfy the requirement of Alibaba can be found and connected by the Tmall Genie. For detailed information refer to BLE Mesh Bluetooth Module Software Specification [1].

#### 5.1.7.1 Three Metadata

The three metadata contains the product id, bluetooth address and secret. The mesh devices must be allocated the three metadata to interface with the Tmall Genie. For example:

product id = 0x293e2

bt addr = AB:CD:F0:F1:F2:F3

secret = "atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"

#### 5.1.7.2 UUID

The unprovisioned mesh devices will broadcast the UDB beacon and provisioning adv which include the device UUID. Alibaba makes use of the customized UUID to recognize the compatible mesh devices. The UUID format is shown in Table 5-14, which includes company id 0x01A8, product id of three Metadata and BT address of device.

**Table 5-14 Alibaba Mesh UUID**

```
typedef struct
{
    uint16_t cid;
    struct
    {
        uint8_t adv_ver: 4;
        uint8_t sec: 1;
        uint8_t ota: 1;
        uint8_t bt_ver: 2; //!< 0 bt4.0, 1 bt4.2, 2 bt5.0, 3 higher
    } pid;
    uint32_t product_id;
    uint8_t mac_addr[6];
    uint8_t rfu[3];
} _PACKED_ ali_uuid_t;
```

In ali light project,    UUID configuration is shown in Table 5-15.

**Table 5-15 Alibaba Mesh UUID Setting**

```
    /** set device uuid */
    ali_uuid_t dev_uuid;
    dev_uuid.cid = 0x01A8; //!< taobao
    dev_uuid.pid.adv_ver = 1;
    dev_uuid.pid.sec = 1;
    dev_uuid.pid.ota = 0;
    dev_uuid.pid.bt_ver = 1;
    dev_uuid.product_id = ALI_PRODUCT_ID;
    uint8_t bt_addr[6];
    gap_get_param(GAP_PARAM_BD_ADDR, bt_addr);
    memcpy(dev_uuid.mac_addr, bt_addr, sizeof(bt_addr));
    memset(dev_uuid.rfu, 0, sizeof(dev_uuid.rfu));
    device_uuid_set((uint8_t *)&dev_uuid);
```

#### 5.1.7.3 Provisioning

Alibaba Mesh used the static OOB authentication data as provision method. The devices shall set the provisioning capability as shown in Table 5-16.

**Table 5-16 Provisioning Capability Setting**

```
/** configure provisioning parameters */
prov_capabilities_t prov_capabilities =
{
    .algorithm = PROV_CAP_ALGO_FIPS_P256_ELLIPTIC_CURVE,
    .public_key = 0,
    .static_oob = PROV_CAP_STATIC_OOB,
    .output_oob_size = 0,
    .output_oob_action = 0,
    .input_oob_size = 0,
    .input_oob_action = 0
};
prov_params_set(PROV_PARAMS_CAPABILITIES, &prov_capabilities,
sizeof(prov_capabilities_t));
```

In the provisioning procedure, the authentication data is calculated from the secret in the three metadata, as depicted in Table 5-17.

**Table 5-17 Authentication Data Calcultaion**

```
case PROV_CB_TYPE_AUTH_DATA:
    {
        prov_start_p pprov_start = cb_data.pprov_start;
        char secret[32];
        uint32_t product_id;
        if (user_data_contains_ali_data())
        {
            user_data_read_ali_secret_key((uint8_t *)secret);
            product_id = user_data_read_ali_product_id();
        }
        else
        {
            memcpy(secret, ALI_SECRET_KEY, 32);
            product_id = ALI_PRODUCT_ID;
        }
        char data[8 + 1 + 12 + 1 + 32 + 1];
        uint8_t auth_data[SHA256_DIGEST_LENGTH]; // Only use 16 bytes
        sprintf(data, "%08x", product_id);
        data[8] = ',';
        uint8_t bt_addr[6];
        gap_get_param(GAP_PARAM_BD_ADDR, bt_addr);
        sprintf(data + 9, "%02x%02x%02x%02x%02x%02x", bt_addr[5], bt_addr[4],
bt_addr[3], bt_addr[2],
                    bt_addr[1], bt_addr[0]);
            data[21] = ',';
            memcpy(data + 22, secret, 32);
            data[54] = 0; // for log print debug
            /*
             * sample: product id = 0x293e2, bt addr = AB:CD:F0:F1:F2:F3, secret =
"atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"
             * input: "000293e2,abcdf0f1f2f3,atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"
             * ouput:
8e-e2-17-bc-02-a5-ab-66-6d-d2-ce-39-5d-f7-20-55-85-4a-f2-7e-c5-c0-45-d9-2a-48-48-99-
74-3a-dc-9f
             * authvalue: 8e-e2-17-bc-02-a5-ab-66-6d-d2-ce-39-5d-f7-20-55
             */
            printi("sha256 input: %s", TRACE_STRING(data));
            SHA256_CTX sha_ctx;
            SHA256_Init(&sha_ctx);
            SHA256_Update(&sha_ctx, data, sizeof(data) - 1);
            SHA256_Final(&sha_ctx, auth_data);
            printi("sha256 output: %b", TRACE_BINARY(SHA256_DIGEST_LENGTH,
auth_data));
            switch (pprov_start->auth_method)
            {
            case PROV_AUTH_METHOD_STATIC_OOB:
                prov_auth_value_set(auth_data, 16);
                APP_PRINT_ERROR1("prov_cb: Please exchange the oob data(%b) with the
provisioner", TRACE_BINARY(16,
                                auth_data));
                break;
```

### 5.1.7.4 Model

After provisioning, the Tmall Genie will configure the model on the device according to the device product id. When the configuration of all required models is completed, the Tmall Genie regard as success. Please contact Alibaba for details.

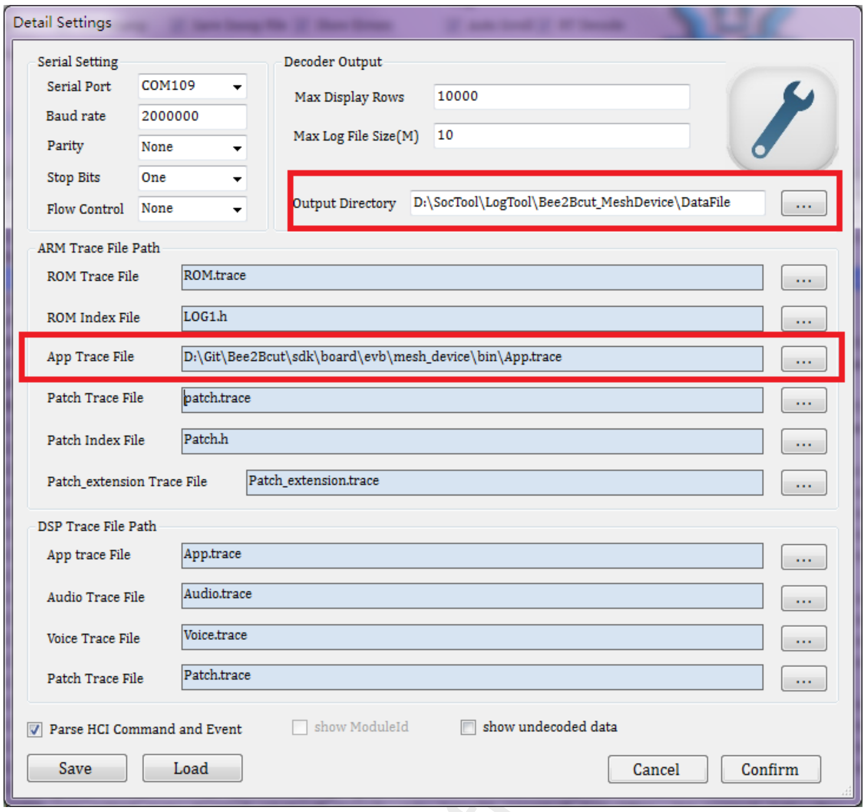
# 6 FAQ

**1. The MP tool “Detect/Open” fail, can’t enter the download mode.**

- P0_3 shall be connected to the GND before power on. After entering download mode, P0_3 and GND no longer need to be shorted.
- The UART Tx and Rx pin shall be connected appropiately.
- Whether the UART transfer board supports 1M Baud?
- Try to reboot the IC and the tool

**2. Log tool doesn’t work.**

- Confirm whether the UART transfer board supports 2M Baud rate.
- Confirm if log pin is P0_3; the GND shall be connected to the ground pin of transfer board.
- The trace file shall be allocated in the log tool as shown in Figure 6-1. App.trace is located in the root directory of project, e.g. “.\sdk\board\evb\mesh_ali_light\bin\App.trace”.



Detail Settings

Serial Setting
Serial Port: COM109
Baud rate: 2000000
Parity: None
Stop Bits: One
Flow Control: None

Decoder Output
Max Display Rows: 10000
Max Log File Size(M): 10
Output Directory: D:\SocTool\LogTool\Bee2Bcut_MeshDevice\DataFile

ARM Trace File Path
ROM Trace File: ROM.trace
ROM Index File: LOG1.h
App Trace File: D:\Git\Bee2Bcut\sdk\board\evb\mesh_device\bin\App.trace
Patch Trace File: patch.trace
Patch Index File: Patch.h
Patch_extension Trace File: Patch_extension.trace

DSP Trace File Path
App trace File: App.trace
Audio Trace File: Audio.trace
Voice Trace File: Voice.trace
Patch Trace File: Patch.trace

☑ Parse HCI Command and Event ☐ show ModuleId ☐ show undecoded data

**Figure 6-1 Log Tool Setting**

- The trace file will be refreshed when Keil project is rebuilt. If the trace file doesn't correspond to the current project, the parsed log will become messy code. Stop and then restart the tool to reload the new trace file.

- The log tool will segment the log file automatically. The log will be saved in two formats: ".log" and ".bin". The ".log" format is plaintext, while the ".bin" format is ciphertext for Realtek internal debug. If the log need be analyzed by Realtek, the app.trace shall be provided.

3. **Program boot exception.**

- Erase the whole IC, and reprogram the IC.

- Change another IC to test, to confirm that if the exception is related to hardware.

- Collect the log, keep the reproduction of exception and call for help.

# References

[1] Mesh Profile Specification

[2] Mesh Model Specification

[3] BLE Mesh Bluetooth Module Software Specification

# Appendix