



REALTEK

UM0203

**Ameba Common BT Mesh Application User
Manual**

Abstract

This document describes the information of BT mesh stack and the Mesh example project.

REVISION HISTORY

Revision	Release Date	Summary
0.1	2019/07/19	First version
0.2	2020/02/27	Add GATT api and Mesh multiple profile
0.3	2020/04/03	Add datatrans/lighting/scene/time/scheduler models
0.4	2021/01/18	Add remote provision api

REVISION HISTORY.....	2
List of Table.....	5
List of Figure	6
Glossary	7
1 Overview.....	8
1.1 Device UUID	8
1.2 Mesh Address.....	8
1.3 APP Model.....	8
1.4 Security	9
1.5 Provisioning.....	9
1.6 Configuration	9
1.7 Proxy	9
2 Mesh	10
2.1 Overview.....	10
2.1.1 Model Layer.....	11
2.1.2 Access Layer	12
2.1.3 Transport Layer (Upper & Lower)	12
2.1.4 Network Layer	13
2.1.5 Bearer Layer	13
2.1.6 Provisioning	13
2.1.7 Friendship	14
2.1.8 Miscellaneous.....	15
2.2 Model Architecture	17
2.2.1 Message Callback	17
2.2.1 Message Definition.....	17
2.2.1 Message Processing	18
2.3 User Cmd	19
2.3.1 Overview	19
2.3.2 Structure.....	19
2.3.3 API.....	21

3 Mesh Sample Projects	73
3.1 Generic On Off Demo	73
3.1.1 Introduction.....	73
3.1.2 Project Overview	73
3.1.3 Source Code Overview	74
3.1.4 Test Procedure	81
4 Mesh Multiple Profile	85
4.1 Structure.....	85
4.2 Implementation Process	86
4.2.1 Mesh Peripheral	86
4.2.2 Mesh Central	86
4.2.3 Mesh Scatternet	87
5 How To Build And Run BT Example	88
6 Q&A	89
6.1 How To Use Mesh User Api In Timer Callback	89
6.2 How To Config Low Power Node Number	89
6.3 Sync Mode and Async Mode configuration	89
6.4 How to add a mesh node	89
6.5 How to delete a mesh node	89

List of Table

Table 1-1 Mesh Address	8
Table 2-1 Model Id	11
Table 2-2 Create Element/Model	12
Table 2-3 Access Opcode	12
Table 2-4 Transporting Ping	12
Table 2-5 Other Bearers	13
Table 2-6 Device Provisioning Setting	13
Table 2-7 Provisioning Callback Register	14
Table 2-8 pb-adv Establishment	14
Table 2-9 pb-adv Callback	14
Table 2-10 Friend Node Initiation	14
Table 2-11 Low Power Node Initiation	15
Table 2-12 Low Power Node Call Friendship	15
Table 2-13 Mesh Log Setting	16
Table 2-14 Device Information Setting	16
Table 2-15 Device UUID Setting	16
Table 2-16 Device Information Setting	16
Table 2-17 Model Structure	17
Table 2-18 Model Message Definition	17
Table 2-19 Model Message Processing	18
Table 3-1 Provisioner command line procedure using adv bearer	83

List of Figure

Figure 2-1 Mesh Lib.....	10
Figure 2-2 Mesh Framework	11
Figure 2-3 User Command Enqueue Flow	19
Figure 2-4 Filers Structure	20
Figure 3-1 Start Menu	82
Figure 3-2 Connect Device	82
Figure 3-3 Connect AP.....	82
Figure 3-4 Select Device.....	82
Figure 3-5 Provisioning.....	82
Figure 3-6 Configure Light State.....	82
Figure 3-7 GOOS Result.....	83

Glossary

Terms	Definitions
BLE	Bluetooth Low Energy
ACK	Acknowledgement
AD	Advertising Data
AES	Advanced Encryption Standard
AID	Application Key Identifier
AKF	Application Key Flag
ATT	Attribute Protocol
GAP	Generic Access Profile
GATT	Generic Attribute Profile
ID	Identifier
LED	Light Emitting Diode
UUID	Universally Unique Identifier
OOB	Out-of-band
SAR	Segmentation And Reassembly
TTL	Time to live
GOOG	Generic On Off get
GOOS	Generic On Off set

1 Overview

Bluetooth Low Energy Mesh is a mesh network solution based on BLE technology. Mesh network is separated to two parts: flooding-based mesh network and routing-based mesh network.

BLE contains two communication channels: advertising channel and data channel. Mesh works primarily on advertising channel, through passive scan and advertising included by BLE. Data channel is a backup for devices those don't support advertising.

The main concepts of mesh are introduced below. Please refer to Mesh Profile Specification published by The Bluetooth Special Interest Group (SIG).

1.1 Device UUID

Each device will be allocated a unique 16-byte UUID, named Device UUID, which is used to identify mesh device. During establishing pb-adv link, Device UUID is necessary for identifying. Once mesh device fetches mesh address successfully, it will use mesh address instead of UUID for identifier.

1.2 Mesh Address

Besides establishing LE link, mesh communication doesn't depend on Bluetooth device address. Mesh will define a 2-byte mesh address for life cycle of BT mesh.

Mesh Address contains unassigned address, unicast address, virtual address and group address, as listed in table 1-1.

Table 1-1 Mesh Address

Range	Address Type	Instructions
0x0000	Unassigned address	An unassigned address is an address in which the element of a node has not been configured yet or no address has been allocated
0x0001~0x7fff	Unicast address	A unicast address is a unique address allocated to each element
0x8000~0xbfff	Virtual address	A virtual address represents a set of destination addresses
0xc000~0xffff	Group address	A group address is an address that is programmed into zero or more elements

1.3 APP Model

There are two types of devices within mesh network: Provisioner and Device.

- 1) Provisioner is a node that is capable of adding a device to a mesh network.
- 2) Device is an entity that is capable of being provisioned into a mesh network.

Once the provisioner succeeds in provisioning the device, the device becomes a node belongs to this mesh network. After provisioning, provisioner can send and receive messages as a Configuration Client, when node acts as Server.

Mesh standardizes the operation of typical application scenarios. Applications on each mesh device are organized on a Model basis. And model defines a model id, an operation code (opcode) and a state for sending and receiving messages.

Finally, to support multiple identical models, mesh introduces the concept of element, each element will be allocated an element address. Then we can configure the specific model through related element address.

1.4 Security

All messages are encrypted and authenticated using two types of keys. One key type is for the network layer communication, such that all communication within a mesh network would use the same network key. The other key type is for application data. Separating the keys for networking and applications allows sensitive access messages (e.g., for access control to a building) to be separated from non-sensitive access messages (e.g., for lighting). There are no unencrypted or unauthenticated messages within a mesh network.

1.5 Provisioning

Device factory default has no address and key, which will be provisioned through provisioning by provisioner.

According to the configuration of provisioning, there are different patterns.

Interaction of public key is divided into OOB and no OOB;

Authentication data is divided into input OOB, output OOB, static OOB and no OOB.

Provisioning process works on advertising channel and data channel, corresponding to pb-adv and pb-gatt transport layer.

1.6 Configuration

Network parameter's management belongs to model layer, named as configuration models.

1.7 Proxy

Proxy feature is used to be compatible with some devices which are not equipped with advertising.

2 Mesh

BT Mesh is an optional Profile implemented by Bluetooth devices, which will define a managed-flood-based mesh network to transmit and relay messages through advertising channels.

There are two related Lib:

AmebaD:

- 1) `component/common/bluetooth/realtek/sdk/example/bt_mesh/lib/lib/amebad/btmesh_prov.a.`
- 2) `component/common/bluetooth/realtek/sdk/example/bt_mesh/lib/lib/amebad/btmesh_dev.a`

AmebaZ2:

- 1) `component/common/bluetooth/realtek/sdk/example/bt_mesh/lib/lib/amebaz2/btmesh_prov.a.`
- 2) `component/common/bluetooth/realtek/sdk/example/bt_mesh/lib/lib/amebaz2/btmesh_dev.a`

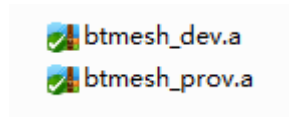


Figure 2-1 Mesh Lib

Header files are provided in SDK: `component\common\bluetooth\realtek\sdk\example\bt_mesh`.

BT Mesh will be introduced according to the following several parts:

- Mesh layer structure will be introduced in *Overview*.
- Layered architecture is described in detail in the following section.
- Standard Model framework and application will be introduced in *Model Architecture*.
- Usage of user application interfaces will be offered in *User Api*.

2.1 Overview

Mesh framework is listed is shown in figure 2-2, separated to bearer layer, network layer, lower transport layer, upper transport layer, access layer, foundation model layer and model layer.

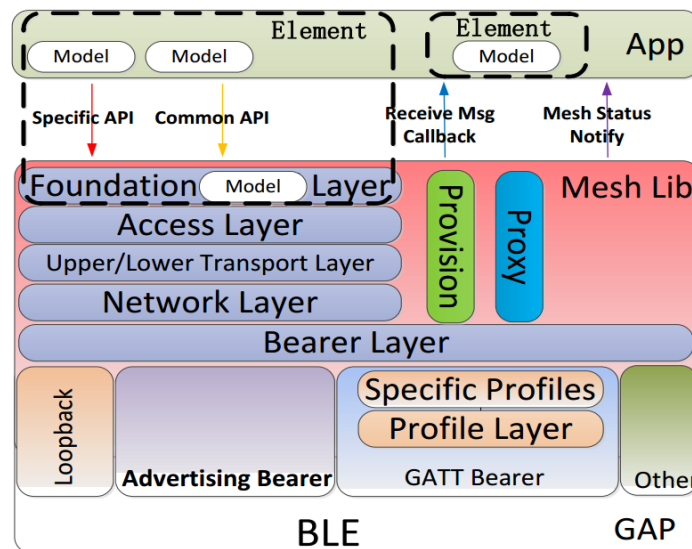


Figure 2-2 Mesh Framework

2.1.1 Model Layer

The Model layer defines models that are used to standardize the operation of typical user scenarios and are defined in the Bluetooth Mesh Model specification or other higher layer specifications. Examples of higher layer model specifications include models for lighting and sensors.

Model achieves the goal through model id, which is separated to 16-bits-model-id sig model and 32-bits-model-id vendor model. And 16-bits-model-id will be coded to 32-bits-model-id. For examples, model id for configuration server and client are 0x0000 and 0x0001, will be coded to 0x0000FFFF and 0x0001FFFF, listed in table 2-1.

Table 2-1 Model Id

label	32 bits model id
#define MESH_MODEL_CFG_SERVER	0x0000FFFF
#define MESH_MODEL_CFG_CLIENT	0x0001FFFF

Element is used to isolate the same model, even if user uses the model with the same access layer opcode. Although the models are same, we can differ them through different element address. Mesh stack will allocate element firstly, then register the model in each element. If we need to use repetitive model, different element should be created in advance. Mesh stack taking charge of registering the configuration server model is mandatory by specification. Health server model can be created by users.

Only primary element is requested by specification, other element can be registered by users.

After implement of Element and model, the content of composition data page 0 can be generated, according to the header, company id, product id, version id, replay protection table size and feature.

Example is listed in table 2-2.

Table 2-2 Create Element/Model

```
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);
```

2.1.2 Access Layer

The access layer defines how higher layer applications can use the upper transport layer. It defines the format of the application data; it defines and controls the application data encryption and decryption performed in the upper transport layer; and it checks whether the incoming application data has been received in the context of the right network and application keys before forwarding it to the higher layer.

Mesh Specification only predefine parts of opcode, leaving some opcodes for vendor-specific messages. These opcode is 3-byte, two most significant bits of the first octet should be set to 1, 6 least significant bits is vendor determined. The left two octets are company id, which is listed in table 2-3.

Table 2-3 Access Opcode

label	3 octets opcode
#define MESH_MSG_PING	0xC05D00

Messages of Access Layer will be encrypted by transport layer using AppKey, TransMIC check supports 4 octets and 8 octets. If obeying 4 octets, the largest length of access messages is 380 bytes, the other is 376 bytes.

2.1.3 Transport Layer (Upper & Lower)

Transport layer is responsible for application layer encryption/decryption, friendship maintenance, heartbeat detection and segmentation & reassembly.

If Access messages using 4 bytes TransMIC without SAR, and length of messages(including opcode) is less than 11 bytes, then this message will be sent in a advertising packet(unsegment), requesting receive transport layer acknowledge.

To test transport layer, there is a ping/pong mechanism, carrying TTL information indicating numbers of hop. This message will use service of network layer, encrypted only by NetKey. After provisioning by provisioner, a node can use ping function of transport layer for testing network layer. Currently node hasn't fetched the AppKey.

Table 2-4 Transporting Ping

label	detail
Function	mesh_msg_send_cause_t trans_ping(uint16_t dst, uint8_t ttl, uint16_t net_key_index, uint16_t

	pong_max_delay);
Brief	ping specific device through transport layer
Parameters	dst: destination address
	ttl: initial ttl
	net_key_index: NetKey index
	pong_max_delay: max pong reply delay(ms)

2.1.4 Network Layer

The network layer defines how transport messages are addressed towards one or more elements. It defines the network message format that allows Transport PDUs to be transported by the bearer layer. The network layer decides whether to relay/forward messages, accept them for further processing, or reject them. It also defines how a network message is encrypted and authenticated.

2.1.5 Bearer Layer

Bearer Layer contains of loopback bearer, advertising bearer, gatt bearer and other bearer. Loopback bear is send-receive channel used within model. Other bear is used to expand mesh network. For example, Gateway can connect Mesh network and Ethernet network with other bearer, making mesh network access Internet. Interface of other bearer is listed in table 2-5.

Table 2-5 Other Bearers

```
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);
```

2.1.6 Provisioning

Device need to fetch Address, Network Key and other information during provisioning. Firstly, devices need to configure provisioning capabilities, listed 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));
```

Device and Provisioner should both register callback function as table 2-7.

Table 2-7 Provisioning Callback Register

```
bool prov_params_set(prov_params_t params, void *pvalue, uint8_t len);
```

Provisioner and Device should establish pb-adv or pb-gatt channel to start provisioning.

2.1.6.1 PB-ADV

Provisioner will initiate pb-adv through device's UUID.

Table 2-8 pb-adv Establishment

label	detail
Function	void pb_adv_link_open(uint8_t ctx_index, uint8_t dev_uuid[16]);
Brief	Establish pb-adv bearer
Parameters	ctx_index: the ctx idx
	dev_uuid: remote device uuid

The result of pb_adv_link_open will be handled within prov_cb call back function, listed in table 2-9.

Table 2-9 pb-adv 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;
```

2.1.6.2 PB-GATT

Provisioner should create BLE connection with device firstly, searching provisioning service. Finally enable related CCCD.

2.1.7 Friendship

Friendship is used by Low Power nodes to limit the amount of time that they need to listen. If a node cannot receive continuously, then it is possible that it will not receive mesh messages that it should be processing. This includes security updates required for maintaining the security of the network as well as the normal mesh messages.

2.1.7.1 Friend Node

Friendship mode should be initialized before working.

Table 2-10 Friend Node Initiation

label	detail
Function	bool fn_init(uint8_t lpn_num, fn_params_t *pfn_params, pf_fn_cb_t pf_fn_cb);
Brief	Initialize friend node
Parameters	lpn_num: the maxium supported low power node number
	fn_params_t: friend node parameters
	pf_fn_cb: friend node friendship info callback

2.1.7.2 Low Power Node

Low Power Node should be initialized before working.

Table 2-11 Low Power Node Initiation

label	detail
Function	bool lpn_init(uint8_t fn_num, pf_lpn_cb_t pf_lpn_cb);
Brief	Initialize low power node
Parameters	lpn_num: the maxium supported fn number
	pf_lpn_cb: low power node friendship info callback

Establishment of friendship is list following.

Table 2-12 Low Power Node Call Friendship

label	detail
Function	lpn_req_reason_t lpn_req(uint8_t frnd_index, uint16_t net_key_index, lpn_req_params_t *p_req_params);
Brief	start to establish the friendship on the desired Netkey
Parameters	frnd_index: the friend node index
	net_key_index: the net key index to establish
	p_req_params: the friendship requirements

Low Power Node can clear the friendship.

Table 2-13 Low Power Node Clear Friendship

label	detail
Function	bool lpn_clear(uint8_t fn_index);
Brief	clear the friendship
Parameters	frnd_index: the friend node index

Low Power Node can add or remove sublist when the friendship exists.

Table 2-14 Low Power Node add or remove sublist

label	detail
Function	void frnd_sub_list_add_rm(uint8_t fn_index, uint16_t addr[], uint8_t addr_num, bool add_rm);
Brief	add or remove the subscribe list when the friendship exists
Parameters	frnd_index: the friend node index
	addr: the subscribe address list
	addr_num: the number of addresses in the list
	add_rm: add or remove flag

2.1.8 Miscellaneous

2.1.8.1 Mesh Log Configuration

Mesh stack contains a large number of components, each of them has unique log degree. These logs can be turned on/off, default on.

There are four degree of log: LEVEL_ERROR, LEVEL_WARN, LEVEL_INFO and LEVEL_TRACE. Cause bsp rate limitation, when log FIFO is full, some will be dropped. To guarantee most significant log, something unconsidered log can be disabled. We can refer to list 2-13.

Table 2-15 Mesh Log Setting

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

2.1.8.2 Mesh Log Configuration

Application Layer will call gap_sched_params_set to set up device name and appearance. Without this step, device is set to rtk_mesh, appearance to unknown.

Table 2-16 Device Information 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));
```

2.1.8.3 Device UUID Configuration

Mesh nodes may have same bt address or random address, then device UUID is needed for identifying mesh node.

Table 2-17 Device UUID Setting

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

2.1.8.4 Other Features Configuration

The following step can configure other features.

Table 2-18 Device Information 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);
```

2.2 Model Architecture

The Bluetooth Special Interest Group (SIG) defines lots of standard models, Realtek SDK has included most of models, and has passed the BQB certification.

To convince the development steps from application layer, SDK has encapsulated the messages between Model Layer and Access Layer. So we offer vendor numbers of services just like predefines of some measures, which makes app developers focusing on setup of business logic.

2.2.1 Message Callback

Before using a specific Model, a structure should be defined as table 2-17:

model_receive function is used to handle messages between Model Layer and Access Layer; model_data_cb is callback function for Model service.

Currently, model_receive function is realized within Realtek SDK, which will call model_data_cb according Access messages.

Table 2-19 Model Structure

```
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;
```

2.2.1 Message Definition

Each header file of Model will contain predefined service message, which has a specific structure, listed in table 2-18.

Table 2-20 Model 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;
```

2.2.1 Message Processing

Table 2-19 shows a simple demo about on/off light event.

Table 2-21 Model Message Processing

```
/** 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);
}

```

2.3 User Cmd

2.3.1 Overview

Realtek SDK has provided vendor some APIs for more efficient and simple development.

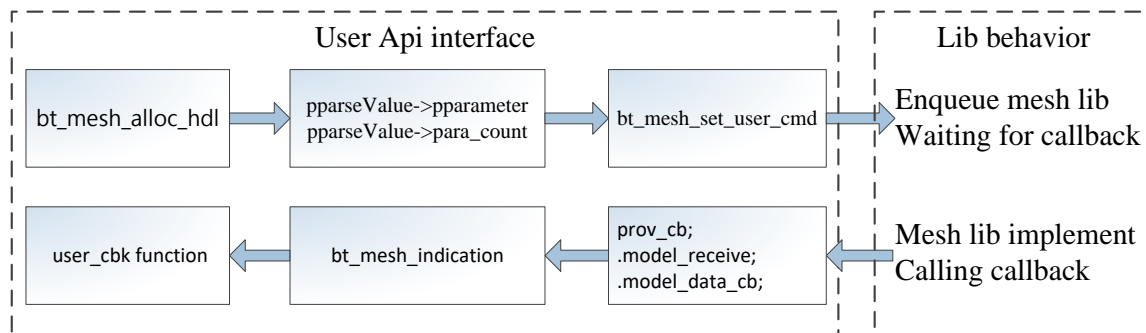


Figure 2-3 User Command Enqueue Flow

There are three steps before enqueueing commands to mesh lib, allocating hdl, filling parameters, and realizing user callback function.

2.3.2 Structure

2.3.2.1 Files Structure

The structure of user apis and user commands related files in Realtek SDK is shown as bellow;

/component/common/bluetooth/realtek/sdk/example/bt_mesh:

```

.
├── api
└── bt_mesh_cmd_types.h

```

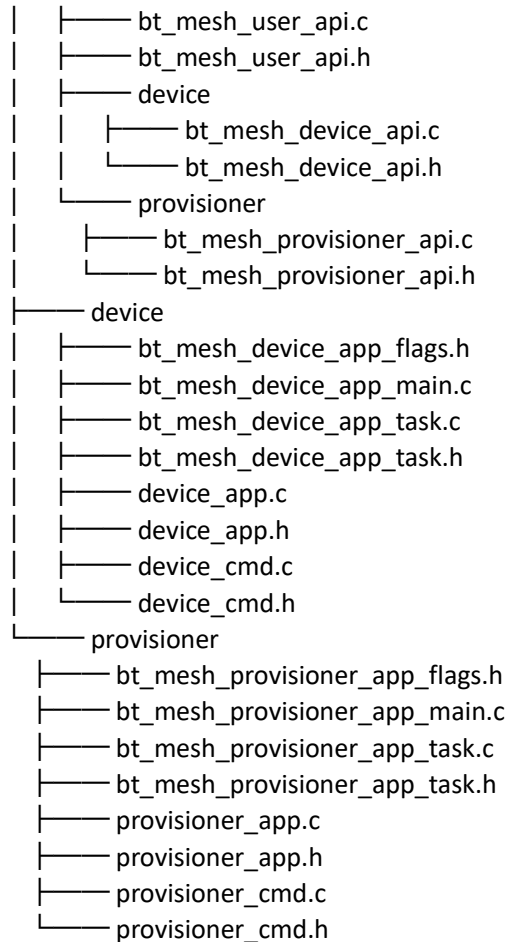



Figure 2-4 Filers Structure

Where

bt_mesh_user_api.c contains the apis functions as **bt_mesh_alloc_hdl**, **bt_mesh_free_hdl**, **bt_mesh_set_user_cmd**, and **bt_mesh_indication**.

bt_mesh_provisioner_api.c contains init and deinit flow for provisioner mode.

bt_mesh_provisioner_api.h contains **bt_mesh_provisioner_cmd** table.

bt_mesh_provisioner_app_main.c configures the initiation measure.

provisioner_cmd.c contains **provisionercmds** table, where finally cmds to mesh lib will be called.

bt_mesh_device_api.c contains init and deinit flow for device mode.

bt_mesh_device_api.h contains **bt_mesh_device_cmd** table.

bt_mesh_device_app_main.c configures the initiation measure.

device_cmd.c contains **devicecmds** table, where finally commands will be called to mesh lib.

2.3.2.2 Data Structure

a) The struct meshUserItem in **bt_mesh_user_api.h** is declared following:

```

struct meshUserItem
{
    uint8_t          userCmdResult;    //!< indicate results of command to upper layer.
}
  
```



```

uint8_t      userApiMode;      //!< 2: cmd line mode 1: sync mode 0: async mode.
_sema        userSema;         //!< sema for sync mode.
_sema        cmdLineSema;      //!< sema for command line mode.
user_cmd_parse_value_t *pparseValue; //!< pointer of parameter array.
void         *userParam;       //!< pointer of user parameter.
};

```

This struct is declared and used in `bt_mesh_user_api.c`, when developers call `bt_mesh_alloc_hdl`, it will allocate a memory size of `meshUserItem`, and pass the pointer to the member of `meshCmdItemSet`.

b) The struct `meshCmdItemSet` in `bt_mesh_user_api.h` is declared following:

```

typedef struct meshCmdItemSet
{
    struct list_head    list;          //!< list of .pmeshCmdItem.
    uint8_t             semaDownTimeOut; //!< wait time out for sync mode.
    uint8_t             userApiMode;    //!< 2: cmd line mode 1: sync mode 0: async mode.
    CMD_ITEM            *pmeshCmdItem;  //!< pointer of meshCmdItem.
}

```

This struct is declared and used in `bt_mesh_user_api.c`, when developers call `bt_mesh_alloc_hdl`, it will allocate a memory size of `meshCmdItemSet`, and return the pointer.

c) The struct `meshCmdItem` in `bt_mesh_user_api.h` is declared as follows:

```

typedef struct meshCmdItem
{
    uint16_t           meshCmdCode;    //!< user command code index.
    user_cmd_parse_value_t *pparseValue; //!< pointer of parameter array.
    bt_mesh_func       meshFunc;       //!< func will be called in cmd thread according to current meshMode.
    user_cmd_cbk       userCb;         //!< user callback will be involved in bt_mesh_indication if not NULL.
    void               *userData;      //!< pointer of user private data struct.
} CMD_ITEM;

```

This struct is declared and used in `bt_mesh_user_api.c`, when developers call `bt_mesh_cmdreg`, it will allocate a memory size of `meshCmdItem`, and pass the user mesh code index, parameter pointer, user callback and user data pointer to related members.

2.3.3 API

The detailed introductions within `bt_mesh_user_api.c`, `bt_mesh_provisioner_api.c` and `bt_mesh_device_api.c` are listed following.

2.3.3.1 common api

Api included by `bt_mesh_user_api.c` is discussed following.

2.3.3.1.1 bt_mesh_alloc_hdl

This function tries to allocate a data size for user to configure specific commands .

Syntax

```
PUSER_ITEM bt_mesh_alloc_hdl(uint8_t synch_enable)
```

Parameters

< synch_enable > 1 for synchronization mode
 0 for asynchronization mode
 2 for user command line mode

Return value

The pointer to the memory allocated.

Remarks

synch_enable determines whether the returning action of *bt_mesh_set_user_cmd* should waiting for the completion of user callback function.

2.3.3.1.2 bt_mesh_free_hdl

This function tries to free previous memory allocated by *bt_mesh_alloc_hdl*.

Syntax

```
void bt_mesh_free_hdl(PUSER_ITEM puser_item)
```

Parameters

< puser_item > the pointer to the memory size of USER_ITEM

Return value

None.

Remarks

None.

2.3.3.1.3 bt_mesh_set_user_cmd

This function tries to enqueue user command index and parameters to command queue and act according to the flag *puserItem->synchMode*.

Syntax

```
user_api_parse_result_t bt_mesh_set_user_cmd(uint16_t mesh_code,  
user_cmd_parse_value_t *pparse_value, user_cmd_cbk cbk, void *user_data)
```

Parameters

< mesh_code > mesh code index;

< pparse_value > *pointer to user's structure of parameters;*
< cbk > *user command call back function;*
< user_data > *pointer to user data structure which will be passed to
Users' call back function;*

Return value

USER_API_RESULT_OK	<i>Enqueueing related command successfully;</i>
USER_API_RESULT_NOT_FOUND	<i>mesh_code is not correct;</i>
USER_API_RESULT_ERROR	<i>Something error happens;</i>
	<i>Off state when calling goo model in synch mode;</i>
USER_API_RESULT_TIMEOUT	<i>Time out happened in synch mode;</i>
USER_API_RESULT_ERROR_MESH_MODE	<i>Currently mesh mode is not correct;</i>
USER_API_RESULT_NOT_ENABLE	<i>User api has not been initialized;</i>

Remarks

If synchronization is enabled, it will not return values until the user callback function completing giving up the semaphore allocated within bt_mesh_alloc_hdl. There is also a timeout period, which will return USER_API_RESULT_TIMEOUT result.

2.3.3.1.4 bt_mesh_indication

This function tries to transfer the commands' result and parameters to mesh_provisioner_cmd_thread or mesh_device_cmd_thread. Also it will call user's callback function.

Syntax

```
user_api_parse_result_t bt_mesh_indication(uint16_t mesh_code, uint8_t state, void  
*pparam);
```

Parameters

< mesh_code > *mesh code index;*
< state > *indicate the result of command;*
< pparam > *pointer the user parameters;*

Return value

USER_API_RESULT_OK	<i>Success;</i>
USER_API_RESULT_INCORRECT_CODE	<i>Currently mesh code is not equal with the mesh code parameters inputted, cause some mesh code using the same callback entrance. <u>This result doesn't mean failure of this function.</u></i>

Remarks

If synchronization is enabled, it will not return values until the user callback function completing up the sema allocated within `bt_mesh_alloc_hdl`. There is also a timeout period, which will return `USER_API_RESULT_TIMEOUT` result.

2.3.3.2 provisioner api

Api included by `bt_mesh_provisioner_api.c` is discussed following.

2.3.3.2.1 `bt_mesh_provisioner_api_init`

This function tries to initialize the api resources used during processing mesh provisioner mode.

Syntax

```
void bt_mesh_provisioner_api_init(void)
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.2.2 `bt_mesh_provisioner_api_deinit`

This function tries to deintialize the api resources used during processing mesh provisioner mode.

Syntax

```
void bt_mesh_provisioner_api_deinit(void)
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.2.3 bt_mesh_provisioner_cmd

bt_mesh_provisioner_api.h has provided a provisioner command index table named *bt_mesh_provisioner_cmd*. Developer can enqueue the mesh provisioner command using corresponding index and parameters.

Syntax

```
enum bt_mesh_provisioner_cmd
{
    GEN_MESH_CODE(_pb_adv_con), /*0*/
    GEN_MESH_CODE(_prov),
    GEN_MESH_CODE(_app_key_add),
    GEN_MESH_CODE(_model_app_bind),
    GEN_MESH_CODE(_model_pub_set),
    GEN_MESH_CODE(_generic_on_off_set), /*5*/
    GEN_MESH_CODE(_generic_on_off_get),
    GEN_MESH_CODE(_node_reset),
    GEN_MESH_CODE(_model_sub_delete),
    GEN_MESH_CODE(_model_sub_add),
    GEN_MESH_CODE(_model_sub_get), /*10*/
    GEN_MESH_CODE(_prov_discover),
    GEN_MESH_CODE(_prov_cccd_operate),
    GEN_MESH_CODE(_proxy_discover),
    GEN_MESH_CODE(_proxy_cccd_operate),
    GEN_MESH_CODE(_datatrans_write), /*15*/
    GEN_MESH_CODE(_datatrans_read),
    GEN_MESH_CODE(_connect),
    GEN_MESH_CODE(_disconnect),
    GEN_MESH_CODE(_list),
    GEN_MESH_CODE(_dev_info_show),
    GEN_MESH_CODE(_light_lightness_get),
    GEN_MESH_CODE(_light_lightness_set),
    GEN_MESH_CODE(_light_lightness_linear_get),
    GEN_MESH_CODE(_light_lightness_linear_set),
    GEN_MESH_CODE(_light_lightness_last_get),
    GEN_MESH_CODE(_light_lightness_default_get),
    GEN_MESH_CODE(_light_lightness_default_set),
    GEN_MESH_CODE(_light_lightness_range_get),
    GEN_MESH_CODE(_light_lightness_range_set),
    GEN_MESH_CODE(_light_ctl_get),
    GEN_MESH_CODE(_light_ctl_set),
    GEN_MESH_CODE(_light_ctl_temperature_get),
    GEN_MESH_CODE(_light_ctl_temperature_set),
    GEN_MESH_CODE(_light_ctl_temperature_range_get),
    GEN_MESH_CODE(_light_ctl_temperature_range_set),
}
```



```
GEN_MESH_CODE(_light_ctl_default_get) ,
GEN_MESH_CODE(_light_ctl_default_set) ,
GEN_MESH_CODE(_light_hsl_get) ,
GEN_MESH_CODE(_light_hsl_set) ,
GEN_MESH_CODE(_light_hsl_target_get) ,
GEN_MESH_CODE(_light_hsl_hue_get) ,
GEN_MESH_CODE(_light_hsl_hue_set) ,
GEN_MESH_CODE(_light_hsl_saturation_get) ,
GEN_MESH_CODE(_light_hsl_saturation_set) ,
GEN_MESH_CODE(_light_hsl_default_get) ,
GEN_MESH_CODE(_light_hsl_default_set) ,
GEN_MESH_CODE(_light_hsl_range_get) ,
GEN_MESH_CODE(_light_hsl_range_set) ,
GEN_MESH_CODE(_light_xyl_get) ,
GEN_MESH_CODE(_light_xyl_set) ,
GEN_MESH_CODE(_light_xyl_target_get) ,
GEN_MESH_CODE(_light_xyl_default_get) ,
GEN_MESH_CODE(_light_xyl_default_set) ,
GEN_MESH_CODE(_light_xyl_range_get) ,
GEN_MESH_CODE(_light_xyl_range_set) ,

MAX_MESH_PROVISIONER_CMD
};
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.2.3.1 GEN_MESH_CODE(_pb_adv_con)

This command tries to establish pb-adv bearer to specific device according UUID.

Syntax

```
GEN_MESH_CODE(_pb_adv_con);
```

Parameters

< pparameter [0] > *device uuid;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

Data's type *pparameter* [0] pointing should be **String**.

If not, **user_cmd_array2string()** should be called firstly before transfer pointer to *pparameter* [0].

2.3.3.2.3.2 GEN_MESH_CODE(_prov)

This command tries to process the provisioning action in the open adv bearer.

Syntax

```
GEN_MESH_CODE(_prov);
```

Parameters

< dw_parameter[0] >	<i>attention duration;</i>
< dw_parameter[1] >	<i>prov_manual;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.3 GEN_MESH_CODE(_app_key_add)

This command tries to add corresponding app key (related app key index) to related target (mesh address), together binding which to net key (net key index)

Syntax

```
GEN_MESH_CODE(_app_key_add);
```

Parameters

< dw_parameter[0] >	<i>device mesh address;</i>
< dw_parameter[1] >	<i>net key index;</i>
< dw_parameter[2] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.4 GEN_MESH_CODE(_model_app_bind)

This command tries to bind app key (app key index) to the model (model id) contained in specific element (element index).

Syntax

```
GEN_MESH_CODE(_model_app_bind);
```

Parameters

< dw_parameter[0] >	<i>device mesh address;</i>
< dw_parameter[1] >	<i>element index;</i>
< dw_parameter[2] >	<i>model id;</i>
< dw_parameter[3] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.5 GEN_MESH_CODE(_model_pub_set)

This command is going to configure the model setting parameters.

Syntax

```
GEN_MESH_CODE(_model_pub_set);
```

Parameters

< dw_parameter[0] >	<i>device mesh address;</i>
< dw_parameter[1] >	<i>mesh address + mesh index;</i>
< dw_parameter[2] >	<i>judgement flag whether element address is string type;</i>
< dw_parameter[3] >	<i>element_addr;</i>
< dw_parameter[4] >	<i>app key index;</i>
< dw_parameter[5] >	<i>friend node flag;</i>
< dw_parameter[6] >	<i>publish ttl (time to live);</i>
< dw_parameter[7] >	<i>publish period;</i>
< dw_parameter[8] >	<i>publish retransfer count;</i>
< dw_parameter[9] >	<i>publish retransfer steps;</i>

< dw_parameter[10] > *model id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.6 GEN_MESH_CODE(_generic_on_off_set)

This command tries to set the light state to the configured generic on off model.

Syntax

```
GEN_MESH_CODE(_generic_on_off_set);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *light state;*
< dw_parameter[2] > *judgement flag whether an acknowledge is need ;*
< dw_parameter[3] > *app key index;*
< dw_parameter[4] > *generic transition steps;*
< dw_parameter[5] > *generic transition step resolution;*
< dw_parameter[6] > *delay time;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.7 GEN_MESH_CODE(_generic_on_off_get)

This command tries to get the light state of the configured generic on off model.

Syntax

```
GEN_MESH_CODE(_generic_on_off_get);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *key index;*

< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.8 GEN_MESH_CODE(_node_reset)

This command tries to reset the node according the mesh address.

Syntax

```
GEN_MESH_CODE(_node_reset);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.9 GEN_MESH_CODE(_model_sub_delete)

This command tries to delete mesh address from group address previously add.

Syntax

```
GEN_MESH_CODE(_model_sub_delete);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *element index;*
< dw_parameter[2] > *model id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.10 GEN_MESH_CODE(_model_sub_add)

This command tries to add mesh address to group.

Syntax

```
GEN_MESH_CODE(_model_sub_add);
```

Parameters

< dw_parameter[0] >	<i>device mesh address;</i>
< dw_parameter[1] >	<i>element index;</i>
< dw_parameter[2] >	<i>model id;</i>
< dw_parameter[3] >	<i>group address;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.11 GEN_MESH_CODE(_model_sub_get)

This command tries to get subscribe of the mesh address and model id.

Syntax

```
GEN_MESH_CODE(_model_sub_get);
```

Parameters

< dw_parameter[0] >	<i>device mesh address;</i>
< dw_parameter[1] >	<i>element index;</i>
< dw_parameter[2] >	<i>model id;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.12 GEN_MESH_CODE(_prov_discover)

This command tries to start discovery provisioning service.

Syntax

```
GEN_MESH_CODE(_prov_discover );
```

Parameters

< dw_parameter[0] > *prov client connection id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.13 GEN_MESH_CODE(_prov_cccd_operate)

This command tries to configure CCCD(Client Characteristic Configuration Descriptor) to enable notify.

Syntax

```
GEN_MESH_CODE(_prov_cccd_operate);
```

Parameters

< dw_parameter[0] > *char CCCD;*
< dw_parameter[1] > *enable/disable;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.14 GEN_MESH_CODE(_proxy_discover)

This command tries to start discovery proxy service.

Syntax

```
GEN_MESH_CODE(_proxy_discover);
```

Parameters

< dw_parameter[0] > *proxy client connection id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.15 GEN_MESH_CODE(_proxy_cccd_operate)

This command tries to configure CCCD(Client Characteristic Configuration Descriptor) to enable notify for proxy.

Syntax

```
GEN_MESH_CODE(_proxy_cccd_operate);
```

Parameters

< dw_parameter[0] > *char CCCD;*
< dw_parameter[1] > *enable/disable;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.16 GEN_MESH_CODE(_datatrans_write)

This command tries to transmit data to other node.

Syntax

```
GEN_MESH_CODE(_datatrans_write);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *specific data;*

< dw_parameter[n] > *specific data.*
< dw_parameter[n+1] > *data len.*
< dw_parameter[n+2] > *app key index.*

< dw_parameter[n+3] > *ack.*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.17 GEN_MESH_CODE(_datatrans_read)

This command tries to read data from other node.

Syntax

```
GEN_MESH_CODE(_datatrans_read);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *data len;*
< dw_parameter[2] > *app key index.*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.18 GEN_MESH_CODE(_connect)

This command tries to connect to remote device.

Syntax

```
GEN_MESH_CODE(_connect);
```

Parameters

< dw_parameter[0] > *bt address;*
< dw_parameter[1] > *address type;*
 Bluetooth low energy public address.
 Bluetooth low energy random address.
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.19 GEN_MESH_CODE(_disconnect)

This command tries to disconnect to remote device.

Syntax

```
GEN_MESH_CODE(_disconnect);
```

Parameters

< dw_parameter[0] > *disconnect id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.20 GEN_MESH_CODE(_list)

This command tries to printf() provisioner's information.

Syntax

```
GEN_MESH_CODE(_list);
```

Parameters

< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.21 GEN_MESH_CODE(_dev_info_show)

This command tries to show device information.

Syntax


```
GEN_MESH_CODE(_dev_info_show);
```

Parameters

< dw_parameter[0] > *1 for on, 0 for off;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.22 GEN_MESH_CODE(_light_lightness_get)

This command tries to get light lightness.

Syntax

```
GEN_MESH_CODE(_light_lightness_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.23 GEN_MESH_CODE(_light_lightness_set)

This command tries to set light lightness.

Syntax

```
GEN_MESH_CODE(_light_lightness_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *light lightness;*
< dw_parameter[2] > *ack;*
< dw_parameter[3] > *app key index;*

< dw_parameter[4] >	<i>num steps;</i>
< dw_parameter[5] >	<i>step resolution;</i>
< dw_parameter[6] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.24 GEN_MESH_CODE(_light_lightness_linear_get)

This command tries to get light lightness linear.

Syntax

```
GEN_MESH_CODE(_light_lightness_linear_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.25 GEN_MESH_CODE(_light_lightness_linear_set)

This command tries to set light lightness linear.

Syntax

```
GEN_MESH_CODE(_light_lightness_linear_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>light lightness;</i>
< dw_parameter[2] >	<i>ack;</i>
< dw_parameter[3] >	<i>app key index;</i>
< dw_parameter[4] >	<i>num steps;</i>

< dw_parameter[5] > *step resolution;*
< dw_parameter[6] > *message execution delay;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.26 GEN_MESH_CODE(_light_lightness_last_get)

This command tries to get last light lightness.

Syntax

```
GEN_MESH_CODE(_light_lightness_last_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.27 GEN_MESH_CODE(_light_lightness_default_get)

This command tries to get default light lightness.

Syntax

```
GEN_MESH_CODE(_light_lightness_default_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.28 GEN_MESH_CODE(_light_lightness_default_set)

This command tries to set default light lightness.

Syntax

```
GEN_MESH_CODE(_light_lightness_default_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>light lightness;</i>
< dw_parameter[2] >	<i>ack;</i>
< dw_parameter[3] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.29 GEN_MESH_CODE(_light_lightness_range_get)

This command tries to get light lightness range.

Syntax

```
GEN_MESH_CODE(_light_lightness_range_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.30 GEN_MESH_CODE(_light_lightness_range_set)

This command tries to set light lightness range.

Syntax

```
GEN_MESH_CODE(_light_lightness_range_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>range min;</i>
< dw_parameter[2] >	<i>range max;</i>
< dw_parameter[3] >	<i>ack;</i>
< dw_parameter[4] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.31 GEN_MESH_CODE(_light_ctl_get)

This command tries to get light ctl.

Syntax

```
GEN_MESH_CODE(_light_ctl_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.32 GEN_MESH_CODE(_light_ctl_set)

This command tries to set light ctl.

Syntax


```
GEN_MESH_CODE(_light_ctl_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>light lightness;</i>
< dw_parameter[2] >	<i>temperature;</i>
< dw_parameter[3] >	<i>delta uv;</i>
< dw_parameter[4] >	<i>ack;</i>
< dw_parameter[5] >	<i>app key index;</i>
< dw_parameter[6] >	<i>num steps;</i>
< dw_parameter[7] >	<i>step resolution;</i>
< dw_parameter[8] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.33 GEN_MESH_CODE(_light_ctl_temperature_get)

This command tries to get light ctl temperature.

Syntax

```
GEN_MESH_CODE(_light_ctl_temperature_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.34 GEN_MESH_CODE(_light_ctl_temperature_set)

This command tries to set light ctl temperature.

Syntax

```
GEN_MESH_CODE(_light_ctl_temperature_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>temperature;</i>
< dw_parameter[2] >	<i>delta uv;</i>
< dw_parameter[3] >	<i>ack;</i>
< dw_parameter[4] >	<i>app key index;</i>
< dw_parameter[5] >	<i>num steps;</i>
< dw_parameter[6] >	<i>step resolution;</i>
< dw_parameter[7] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.35 GEN_MESH_CODE(_light_ctl_temperature_range_get)

This command tries to get light ctl temperature range.

Syntax

```
GEN_MESH_CODE(_light_ctl_temperature_range_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.36 GEN_MESH_CODE(_light_ctl_temperature_range_set)

This command tries to set light ctl temperature range.

Syntax

```
GEN_MESH_CODE(_light_ctl_temperature_range_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>range min;</i>
< dw_parameter[2] >	<i>range max;</i>
< dw_parameter[3] >	<i>ack;</i>
< dw_parameter[4] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.37 GEN_MESH_CODE(_light_ctl_default_get)

This command tries to get light default ctl.

Syntax

```
GEN_MESH_CODE(_light_ctl_default_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.38 GEN_MESH_CODE(_light_ctl_default_set)

This command tries to set light default ctl.

Syntax

```
GEN_MESH_CODE(_light_ctl_default_set);
```


Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>lightness;</i>
< dw_parameter[2] >	<i>temperature;</i>
< dw_parameter[3] >	<i>delta_uv;</i>
< dw_parameter[4] >	<i>ack;</i>
< dw_parameter[5] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.39 GEN_MESH_CODE(_light_hsl_get)

This command tries to get light hsl.

Syntax

```
GEN_MESH_CODE(_light_hsl_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.40 GEN_MESH_CODE(_light_hsl_set)

This command tries to set light hsl.

Syntax

```
GEN_MESH_CODE(_light_hsl_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>hue;</i>
< dw_parameter[2] >	<i>saturation;</i>
< dw_parameter[3] >	<i>lightness;</i>
< dw_parameter[4] >	<i>ack;</i>
< dw_parameter[5] >	<i>app key index;</i>
< dw_parameter[6] >	<i>num steps;</i>
< dw_parameter[7] >	<i>step resolution;</i>
< dw_parameter[8] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.41 GEN_MESH_CODE(_light_hsl_target_get)

This command tries to get light hsl target.

Syntax

```
GEN_MESH_CODE(_light_hsl_target_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.42 GEN_MESH_CODE(_light_hsl_hue_get)

This command tries to get light hsl hue.

Syntax

```
GEN_MESH_CODE(_light_hsl_hue_get);
```


Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.43 GEN_MESH_CODE(_light_hsl_hue_set)

This command tries to set light hsl hue.

Syntax

```
GEN_MESH_CODE(_light_hsl_hue_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>hue;</i>
< dw_parameter[2] >	<i>ack;</i>
< dw_parameter[3] >	<i>app key index;</i>
< dw_parameter[4] >	<i>num steps;</i>
< dw_parameter[5] >	<i>step resolution;</i>
< dw_parameter[6] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.44 GEN_MESH_CODE(_light_hsl_saturation_get)

This command tries to get light hsl saturation.

Syntax

```
GEN_MESH_CODE(_light_hsl_saturation_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.45 GEN_MESH_CODE(_light_hsl_saturation_set)

This command tries to set light hsl saturation.

Syntax

```
GEN_MESH_CODE(_light_hsl_saturation_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>saturation;</i>
< dw_parameter[2] >	<i>ack;</i>
< dw_parameter[3] >	<i>app key index;</i>
< dw_parameter[4] >	<i>num steps;</i>
< dw_parameter[5] >	<i>step resolution;</i>
< dw_parameter[6] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.46 GEN_MESH_CODE(_light_hsl_default_get)

This command tries to get default light hsl.

Syntax

```
GEN_MESH_CODE(_light_hsl_default_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
---------------------	----------------------------------

< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.47 GEN_MESH_CODE(_light_hsl_default_set)

This command tries to set default light hsl.

Syntax

```
GEN_MESH_CODE(_light_hsl_default_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *lightness;*
< dw_parameter[2] > *hue;*
< dw_parameter[3] > *saturation;*
< dw_parameter[4] > *ack;*
< dw_parameter[5] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.48 GEN_MESH_CODE(_light_hsl_range_get)

This command tries to get light hsl range.

Syntax

```
GEN_MESH_CODE(_light_hsl_range_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.49 GEN_MESH_CODE(_light_hsl_range_set)

This command tries to set light hsl range.

Syntax

```
GEN_MESH_CODE(_light_hsl_range_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>hue range min;</i>
< dw_parameter[2] >	<i>hue range max;</i>
< dw_parameter[3] >	<i>saturation range min;</i>
< dw_parameter[4] >	<i>saturation range max;</i>
< dw_parameter[5] >	<i>ack;</i>
< dw_parameter[6] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.50 GEN_MESH_CODE(_light_xyl_get)

This command tries to get light xyl.

Syntax

```
GEN_MESH_CODE(_light_xyl_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.51 GEN_MESH_CODE(_light_xyl_set)

This command tries to set light xyl.

Syntax

```
GEN_MESH_CODE(_light_xyl_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>xyl_lightness;</i>
< dw_parameter[2] >	<i>xyl_x;</i>
< dw_parameter[3] >	<i>xyl_y;</i>
< dw_parameter[4] >	<i>ack;</i>
< dw_parameter[5] >	<i>app key index;</i>
< dw_parameter[6] >	<i>num steps;</i>
< dw_parameter[7] >	<i>step resolution;</i>
< dw_parameter[8] >	<i>message execution delay;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.52 GEN_MESH_CODE(_light_xyl_target_get)

This command tries to get light xyl target.

Syntax

```
GEN_MESH_CODE(_light_xyl_target_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.53 GEN_MESH_CODE(_light_xyl_default_get)

This command tries to get default light xyl.

Syntax

```
GEN_MESH_CODE(_light_xyl_default_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.54 GEN_MESH_CODE(_light_xyl_default_set)

This command tries to set light default xyl.

Syntax

```
GEN_MESH_CODE(_light_xyl_default_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>lightness;</i>
< dw_parameter[2] >	<i>xyl_x;</i>
< dw_parameter[3] >	<i>xyl_y;</i>

< dw_parameter[4] > *ack;*
< dw_parameter[5] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.55 GEN_MESH_CODE(_light_xyl_range_get)

This command tries to get light xyl range.

Syntax

```
GEN_MESH_CODE(_light_xyl_range_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.56 GEN_MESH_CODE(_light_xyl_range_set)

This command tries to set light xyl range.

Syntax

```
GEN_MESH_CODE(_light_xyl_range_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *xyl_x range min;*
< dw_parameter[2] > *xyl_x range max;*
< dw_parameter[3] > *xyl_y range min;*

< dw_parameter[4] > *xyl_y range max;*
< dw_parameter[5] > *ack;*
< dw_parameter[6] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.57 GEN_MESH_CODE(_time_set)

This command tries to set time.

Syntax

`GEN_MESH_CODE(_time_set);`

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *tai_seconds[0];*
< dw_parameter[1] >> 8> *tai_seconds[1];*
< dw_parameter[1] >> 16> *tai_seconds[2];*
< dw_parameter[1] >> 24> *tai_seconds[3];*
< dw_parameter[2] > *tai_seconds[4];*
< dw_parameter[3] > *subsecond;*
< dw_parameter[4] > *uncertainty;*
< dw_parameter[5] > *time_authority;*
< dw_parameter[6] > *tai_utc_delta;*
< dw_parameter[7] > *time_zone_offset;*
< dw_parameter[8] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.58 GEN_MESH_CODE(_time_get)

This command tries to get time.

Syntax

```
GEN_MESH_CODE(_time_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.59 GEN_MESH_CODE(_time_zone_set)

This command tries to set time zone.

Syntax

```
GEN_MESH_CODE(_time_zone_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *time_zone_offset_new;*
< dw_parameter[2] > *change[0];*
< dw_parameter[2] >> 8> *change[1];*
< dw_parameter[2] >> 16> *change[2];*
< dw_parameter[2] >> 24> *change[3];*
< dw_parameter[3] > *change[4];*
< dw_parameter[4] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.60 GEN_MESH_CODE(_time_zone_get)

This command tries to get time zone.

Syntax

```
GEN_MESH_CODE(_time_zone_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.61 GEN_MESH_CODE(_time_tai_utc_delta_set)

This command tries to set time tai utc delta.

Syntax

```
GEN_MESH_CODE(_time_tai_utc_delta_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *tai_utc_delta_new;*
< dw_parameter[2] > *delta[0];*
< dw_parameter[2] >> 8> *delta[1];*
< dw_parameter[2] >> 16> *delta[2];*
< dw_parameter[2] >> 24> *delta[3];*
< dw_parameter[3] > *change[4];*
< dw_parameter[4] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.62 GEN_MESH_CODE(_time_tai_utc_delta_get)

This command tries to get time tai utc delta.

Syntax

```
GEN_MESH_CODE( set time tai utc delta);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.63 GEN_MESH_CODE(_time_role_set)

This command tries to set time role.

Syntax

```
GEN_MESH_CODE(_time_role_set);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *role;*
< dw_parameter[2] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.64 GEN_MESH_CODE(_time_role_get)

This command tries to get time role.

Syntax

```
GEN_MESH_CODE(_time_role_get);
```

Parameters

< dw_parameter[0] > *destination node address;*

< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.65 GEN_MESH_CODE(_scene_store)

This command tries to store scene.

Syntax

`GEN_MESH_CODE(_scene_store);`

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *scene number;*
< dw_parameter[2] > *ack;*
< dw_parameter[3] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.66 GEN_MESH_CODE(_scene_recall)

This command tries to recall scene.

Syntax

`GEN_MESH_CODE(_scene_recall);`

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *scene number;*
< dw_parameter[2] > *ack;*
< dw_parameter[3] > *app key index;*
< dw_parameter[4] > *num steps;*

< dw_parameter[5] > *step resolution;*
< dw_parameter[6] > *message execution delay;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.67 GEN_MESH_CODE(_scene_get)

This command tries to get scene.

Syntax

```
GEN_MESH_CODE(_scene_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.68 GEN_MESH_CODE(_scene_register_get)

This command tries to get scene register.

Syntax

```
GEN_MESH_CODE(_scene_register_get);
```

Parameters

< dw_parameter[0] > *destination node address;*
< dw_parameter[1] > *app key index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.2.3.69 GEN_MESH_CODE(_scene_delete)

This command tries to delete scene.

Syntax

```
GEN_MESH_CODE(_scene_delete);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>scene number;</i>
< dw_parameter[2] >	<i>ack;</i>
< dw_parameter[3] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.70 GEN_MESH_CODE(_scheduler_get)

This command tries to get scheduler.

Syntax

```
GEN_MESH_CODE(_scheduler_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.71 GEN_MESH_CODE(_scheduler_action_get)

This command tries to get scheduler action.

Syntax

```
GEN_MESH_CODE(_scheduler_action_get);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>app key index;</i>
< dw_parameter[2] >	<i>index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.72 GEN_MESH_CODE(_scheduler_action_set)

This command tries to set scheduler action.

Syntax

```
GEN_MESH_CODE(_scheduler_action_set);
```

Parameters

< dw_parameter[0] >	<i>destination node address;</i>
< dw_parameter[1] >	<i>index;</i>
< dw_parameter[2] >	<i>year;</i>
< dw_parameter[3] >	<i>month;</i>
< dw_parameter[4] >	<i>day;</i>
< dw_parameter[5] >	<i>hour;</i>
< dw_parameter[6] >	<i>minute;</i>
< dw_parameter[7] >	<i>second;</i>
< dw_parameter[8] >	<i>day_of_week;</i>
< dw_parameter[9] >	<i>action;</i>
< dw_parameter[10] >	<i>num_steps;</i>
< dw_parameter[11] >	<i>step_resolution;</i>
< dw_parameter[12] >	<i>scene number;</i>
< dw_parameter[13] >	<i>ack;</i>
< dw_parameter[14] >	<i>app key index;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.73 GEN_MESH_CODE(_fn_init)

This command tries to initialize friend node.

Syntax

```
GEN_MESH_CODE(_fn_init);
```

Parameters

< dw_parameter[0] >	<i>friend node number;</i>
< dw_parameter[1] >	<i>queue size;</i>
< dw_parameter[2] >	<i>receive window;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.74 GEN_MESH_CODE(_fn_deinit)

This command tries to deinitialize friend node.

Syntax

```
GEN_MESH_CODE(_fn_deinit);
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.2.3.75 GEN_MESH_CODE(_rmt_prov_client_link_open_prov)

This command tries to open the remote provision link layer through the relay node.

Syntax

```
GEN_MESH_CODE(_rmt_prov_client_link_open_prov);
```

Parameters

< dw_parameter[0] >	<i>dst addr//The relay node addr;</i>
< dw_parameter[1] >	<i>net key index;</i>
< dw_parameter[2] >	<i>UUID//unprovisioned device UUID;</i>
< dw_parameter[3] >	<i>link open timeout;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.76 GEN_MESH_CODE(_rmt_prov_client_close)

This command tries to close the open remote provision link bearer.

Syntax

```
GEN_MESH_CODE(_rmt_prov_client_close );
```

Parameters

< dw_parameter[0] >	<i>dst addr//The relay node addr;</i>
< dw_parameter[1] >	<i>net key index;</i>
< dw_parameter[2] >	<i>close reason;</i> <i>0x00//Success;</i> <i>0x01//Prohibited;</i> <i>0x02//Fail;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.2.3.77 GEN_MESH_CODE(_rmt_prov_client_scan_start)

This command tries to discover the unprovisioned device.

Syntax

```
GEN_MESH_CODE(_rmt_prov_client_scan_start );
```

Parameters

< dw_parameter[0] >	<i>dst addr//The relay node addr;</i>
< dw_parameter[1] >	<i>net key index;</i>
< dw_parameter[2] >	<i>scanned items_limit//limit the scan num;</i>
< dw_parameter[3] >	<i>scan timeout;</i>
< dw_parameter[4] >	<i>UUID// the specific remote device UUID;</i> <i>This parameter can be 0, for scanning all unprovisioned devices</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.3 device api

Api included by bt_mesh_device_api.c is discussed following.

2.3.3.3.1 bt_mesh_device_api_init

This function tries to initialize the api resources used during processing mesh device mode.

Syntax

```
void bt_mesh_device_api_init(void)
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.3.2 bt_mesh_device_api_deinit

This function tries to deinitialize the api resources used during processing mesh device mode.

Syntax

```
void bt_mesh_device_api_deinit(void)
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.3.3 bt_mesh_device_cmd

bt_mesh_device_api.h has provided a device command index table named bt_mesh_device_cmd, developer should enqueue the mesh device command using corresponding index and parameters.

Syntax

```
enum bt_mesh_device_cmd
{
    GEN_MESH_CODE(_node_reset) , /*0*/
    GEN_MESH_CODE(_prov_capa_set) ,
#ifdef MESH_LPN
    GEN_MESH_CODE(_lpn_init) ,
    GEN_MESH_CODE(_lpn_req) ,
    GEN_MESH_CODE(_lpn_sub) ,
    GEN_MESH_CODE(_lpn_clear) , /*5*/
#endif
    GEN_MESH_CODE(_data_transmission_notify) ,
    GEN_MESH_CODE(_generic_on_off_publish) ,
    GEN_MESH_HANDLER(_connect) ,
    GEN_MESH_HANDLER(_disconnect) ,
    GEN_MESH_CODE(_list) ,

    MAX_MESH_DEVICE_CMD
};
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.3.1 GEN_MESH_CODE(_node_reset)

This command tries to reset, clean, or restore mesh parameters.

Syntax

```
GEN_MESH_CODE(_node_reset);
```

Parameters

< dw_parameter[0] > *operation code(0 for reset, 1 for clean, 2 for restore);*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.2 GEN_MESH_CODE(_prov_capa_set)

This command tries to set the provision capability.

Syntax

```
GEN_MESH_CODE(_prov_capa_set);
```

Parameters

< dw_parameter[0] > *public key;*
< dw_parameter[1] > *static oob;*
< dw_parameter[2] > *static oob size;*
< dw_parameter[3] > *output oob action;*
< dw_parameter[4] > *input oob size;*
< dw_parameter[5] > *input oob action;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.3 GEN_MESH_CODE(_lpn_init)

This command tries to initialize the low power node.

Syntax

```
GEN_MESH_CODE(_lpn_init);
```

Parameters

< dw_parameter[0] > *maxium supported friend node;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.4 GEN_MESH_CODE(_lpn_deinit)

This command tries to deinitialize the low power node.

Syntax

```
GEN_MESH_CODE(_lpn_deinit);
```

Parameters

None.

Return value

None.

Remarks

None.

2.3.3.3.5 GEN_MESH_CODE(_lpn_req)

This command tries to start to establish the friendship on the desired Netkey.

Syntax

```
GEN_MESH_CODE(_lpn_req);
```


Parameters

< dw_parameter[0] >	<i>friend index;</i>
< dw_parameter[1] >	<i>the net key index to establish;</i>
< dw_parameter[2] >	<i>polling interval of the friendship requirements;</i>
< dw_parameter[3] >	<i>polling timeout of the friendship requirements;</i>
< dw_parameter[4] >	<i>receive delay of the friendship requirements;</i>
< dw_parameter[5] >	<i>receive window of the friendship requirements;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

dw_parameter[2]~ dw_parameter[5] are optional, not required.

2.3.3.3.6 GEN_MESH_CODE(_lpn_sub)

This command tries to add or remove the subscribe list when the friendship exists.

Syntax

```
GEN_MESH_CODE(_lpn_sub);
```

Parameters

< dw_parameter[0] >	<i>friend index;</i>
< dw_parameter[1] >	<i>the subscribe address list;</i>
< dw_parameter[2] >	<i>address or remove flag;</i>
< para_count >	<i>indicate parameters num;</i>

Return value

None.

Remarks

None.

2.3.3.3.7 GEN_MESH_CODE(_lpn_clear)

This command tries to clear the friendship.

Syntax

```
GEN_MESH_CODE(_lpn_clear);
```


Parameters

< dw_parameter[0] > *friend index;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.3.8 GEN_MESH_CODE(_data_transmission_notify)

This command tries to send characteristic value to peer device.

Syntax

```
GEN_MESH_CODE(_data_transmission_notify);
```

Parameters

< dw_parameter[0] > *connection id indicate which link is;*
< dw_parameter[1] > *service id;*
< dw_parameter[2] > *attribute index of characteristic;*
< dw_parameter[3] > *point to data to be sent;*
< dw_parameter[4] > *length of value to be sent;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.3.9 GEN_MESH_CODE(_datatrans_write)

This command tries to transmit data to other node.

Syntax

```
GEN_MESH_CODE(_datatrans_write);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *specific data;*
... ...

< dw_parameter[n] > *specific data.*
< dw_parameter[n+1] > *data len.*
< dw_parameter[n+2] > *app key index.*
< dw_parameter[n+3] > *ack.*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.10 GEN_MESH_CODE(_datatrans_read)

This command tries to read data from other node.

Syntax

```
GEN_MESH_CODE(_datatrans_read);
```

Parameters

< dw_parameter[0] > *device mesh address;*
< dw_parameter[1] > *data len;*
< dw_parameter[2] > *app key index.*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.11 GEN_MESH_CODE(_connect)

This command tries to connect to remote device.

Syntax

```
GEN_MESH_CODE(_connect);
```

Parameters

< dw_parameter[0] > *bt address;*
< dw_parameter[1] > *address type;*
 Bluetooth low energy public address.

Bluetooth low energy random address.
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.12 GEN_MESH_CODE(_disconnect)

This command tries to disconnect to remote device.

Syntax

```
GEN_MESH_CODE(_disconnect);
```

Parameters

< dw_parameter[0] > *disconnect id;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.13 GEN_MESH_CODE(_list)

This command tries to printf() device's information.

Syntax

```
GEN_MESH_CODE(_list);
```

Parameters

< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.14 GEN_MESH_CODE(_dev_info_show)

This command tries to show device information.

Syntax

```
GEN_MESH_CODE(_dev_info_show);
```

Parameters

< dw_parameter[0] > *1 for on, 0 for off;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.15 GEN_MESH_CODE(_fn_init)

This command tries to initialize friend node.

Syntax

```
GEN_MESH_CODE(_fn_init);
```

Parameters

< dw_parameter[0] > *friend node number;*
< dw_parameter[1] > *queue size;*
< dw_parameter[2] > *receive window;*
< para_count > *indicate parameters num;*

Return value

None.

Remarks

None.

2.3.3.3.16 GEN_MESH_CODE(_fn_deinit)

This command tries to deinitialize friend node.

Syntax

```
GEN_MESH_CODE(_fn_deinit);
```

Parameters

None

Return value

None.

Remarks

None.

3 Mesh Sample Projects

Only Generic On Off demo for reference is listed following.

1. Generic on off client

- Start provisioning to device acts as Generic On off server;
- Set light state to server device through generic on off model;

2. Generic on off server

- Turn on/off the light according the generic on off client's message;
- Give the acknowledge to generic on off client if acknowledgement enabled;

3.1 Generic On Off Demo

3.1.1 Introduction

The purpose of this chapter is to give an overview of the Generic On Off application. The generic-on-off project implements a simple provisioner acts as goo client, together with a device acting as server.

Provisioner will implement node configuration, element creation, generic-on-off model and user command table initiation. Then it will start provisioning to device, switching that from unprovisioned to a node. After implementation of provisioning, it will configure the app key and bind it to model. Then users can call GOOG and GOOS command to set or get light state. Device will implements the similar configuration of provisioner, except generic on off server initiation. After provisioned by provisioner, it can start receiving access messages from provisioner. If received access messages requiring acknowledgement, it will acknowledge this message.

There are two ways for testing generic-on-off model:

- 1) App Demo;
- 2) Command line;

3.1.2 Project Overview

This section describes project directory, project structure, and macro controlling the example. Reference content as follows:

- Project source code directory:
 - a) Provisioner: component\common\bluetooth\realtek\sdk\example\bt_mesh_provisioner_rtk_demo
 - b) Device: component\common\bluetooth\realtek\sdk\example\bt_mesh_device_r

tk_demo

- Controlled by macro: CONFIG_BT_MESH_PROVISIONER,
CONFIG_BT_MESH_PROVISIONER_RTK_DEMO,
CONFIG_BT_MESH_DEVICE,
CONFIG_BT_MESH_DEVICE_RTK_DEMO,
CONFIG_BT_MESH_USER_API.
- CONFIG_BT and CONFIG_BT_MESH_USER_API should be set to 1 if user command interfaces is needed.
- Whether CONFIG_BT_MESH_PROVISIONER_RTK_DEMO or CONFIG_BT_MESH_DEVICE_RTK_DEMO should be set to 1 together related mesh mode macro(CONFIG_BT_MESH_PROVISIONER or CONFIG_BT_MESH_DEVICE), if app demo is on schedule.
- If command line test procedure is needed, only CONFIG_BT_MESH_PROVISIONER or CONFIG_BT_MESH_DEVICE should be set to 1.

3.1.3 Source Code Overview

The following sections will describe parts about both provisioner and device.

3.1.3.1 Initialization

3.1.3.1.1 provisioner initialization

bt_mesh_cmd_start_provisioner_api () function is invoked when the httpd thread receives starting provisioner command.

```
uint8_t bt_mesh_cmd_start_provisioner_api(void)
{
    T_GAP_DEV_STATE new_state;

    /*Wait WIFI init complete*/
    while(!(wifi_is_up(RTW_STA_INTERFACE) || wifi_is_up(RTW_AP_INTERFACE))) {
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
    //judge BLE central is already on
    le_get_gap_param(GAP_PARAM_DEV_STATE, &new_state);
    if (new_state.gap_init_state == GAP_INIT_STATE_STACK_READY) {
        printf("[BT Mesh Provisioner]BT Stack already on\n\r");
        return 0;
    }
    else
        bt_mesh_provisioner_app_main();
}
```



```
bt_coex_init();

/*Wait BT init complete*/
do {
    vTaskDelay(100 / portTICK_RATE_MS);
    le_get_gap_param(GAP_PARAM_DEV_STATE , &new_state);
}while(new_state.gap_init_state != GAP_INIT_STATE_STACK_READY);

/*Start BT WIFI coexistence*/
wifi_btcoex_set_bt_on();

#if defined(CONFIG_BT_MESH_PROVISIONER_RTK_DEMO) &&
CONFIG_BT_MESH_PROVISIONER_RTK_DEMO
    bt_mesh_provisioner_api_init();
#endif

return 1;

}
```

bt_mesh_provisioner_app_main is listed following:

```
int bt_mesh_provisioner_app_main(void)
{
    bt_trace_init();
    //bt_mesh_provisioner_stack_config_init();
    bte_init();
    bt_mesh_provisioner_board_init();
    bt_mesh_provisioner_driver_init();
    le_gap_init(APP_MAX_LINKS);
    bt_mesh_provisioner_app_le_gap_init();
    bt_mesh_provisioner_app_le_profile_init();
    bt_mesh_provisioner_stack_init();
    bt_mesh_provisioner_pwr_mgr_init();
    bt_mesh_provisioner_task_init();
    return 0;
}
```

following initialization functions will be invoked:

- le_gap_init() - Initialize GAP and set link number
- bt_mesh_provisioner_app_le_gap_init () - Initialize gap related parameters
- bt_mesh_provisioner_app_le_profile_init () - Add GATT services, clients and register callbacks
- bt_mesh_provisioner_stack_init () - Initialize peripheral and gap bond manager related parameters
- bt_mesh_provisioner_task_init () – Start bt_mesh_provisioner_app_main_task

bt_mesh_provisioner_api_init function will initialize related resources of user api components.

```
void bt_mesh_provisioner_api_init(void)
{
    btMeshCmdPriv.meshMode = BT_MESH_PROVISIONER;
    rtw_init_listhead(&btMeshCmdPriv.meshCmdList);
    rtw_mutex_init(&btMeshCmdPriv.cmdMutex);
    rtw_init_sema(&btMeshCmdPriv.meshThreadSema, 0);
    BT_MESH_CMD_ID_PRIV_MOD(MAX_MESH_PROVISIONER_CMD, NULL);
    if (rtw_if_wifi_create_task(&meshProvisionerCmdThread,
        "mesh_provisioner_cmd_thread", 1024, TASK_PRORITY_MIDDLE,
        mesh_provisioner_cmd_thread, NULL)!= 1) {
        printf("[BT_MESH] %s(): create mesh_provisioner_cmd_thread fail !\r\n", __func__);
    }
}
```

3.1.3.1.2 device initialization

bt_mesh_example_init_thread() will be invoked during bt mesh device demo running.

```
void bt_mesh_example_init_thread(void *param)
{
    T_GAP_DEV_STATE new_state;

    /*Wait WIFI init complete*/
    while(!(wifi_is_up(RTW_STA_INTERFACE) || wifi_is_up(RTW_AP_INTERFACE))) {
        vTaskDelay(1000 / portTICK_RATE_MS);
    }
    /*Init BT mesh device*/
    bt_mesh_device_app_main();

    bt_coex_init();

    /*Wait BT init complete*/
    do {
        vTaskDelay(100 / portTICK_RATE_MS);
        le_get_gap_param(GAP_PARAM_DEV_STATE , &new_state);
    } while (new_state.gap_init_state != GAP_INIT_STATE_STACK_READY);

    /*Start BT WIFI coexistence*/
    wifi_btcoex_set_bt_on();

    bt_mesh_device_api_init();

    vTaskDelete(NULL);
}
```


bt_mesh_device_app_main is listed following:

```
int bt_mesh_device_app_main(void)
{
    bt_trace_init();
    //bt_mesh_device_stack_config_init();
    bte_init();
    bt_mesh_device_board_init();
    bt_mesh_device_driver_init();
    le_gap_init(APP_MAX_LINKS);
    bt_mesh_device_app_le_gap_init();
    bt_mesh_device_app_le_profile_init();
    bt_mesh_device_mesh_stack_init();
    bt_mesh_device_pwr_mgr_init();
    bt_mesh_device_task_init();

    return 0;
}
```

following initialization functions will be invoked:

- le_gap_init() - Initialize GAP and set link number
- bt_mesh_device_app_le_gap_init () - Initialize gap related parameters
- bt_mesh_device_app_le_profile_init () - Add GATT services, clients and register callbacks
- bt_mesh_device_stack_init () - Initialize peripheral and gap bond manager related parameters
- bt_mesh_device_task_init() – Start bt_mesh_device_app_main_task

bt_mesh_device_api_init function will initialize related resources of user api components.

```
void bt_mesh_device_api_init(void)
{
    btMeshCmdPriv.meshMode = BT_MESH_DEVICE;
    rtw_init_listhead(&btMeshCmdPriv.meshCmdList);
    rtw_mutex_init(&btMeshCmdPriv.cmdMutex);
    rtw_init_sema(&btMeshCmdPriv.meshThreadSema, 0);
    BT_MESH_CMD_ID_PRIV_MOD(MAX_MESH_DEVICE_CMD, NULL);
    if (rtw_if_wifi_create_task(&meshDeviceCmdThread, "mesh_device_cmd_thread", 1024,
        TASK_PRIORITY_MIDDLE, mesh_device_cmd_thread, NULL)!= 1) {
        printf("[BT_MESH] %s(): create mesh_device_cmd_thread fail !\r\n", __func__);
    }
}
```


3.1.3.3 Mesh Callback Handler

prov_cb() function is used to handle provisioning callback messages. Each callback type will invoke bt_mesh_indication() to call the user call back function, passing the result and parameters to which if necessary.

```
case PROV_CB_TYPE_COMPLETE:
{
    prov_data_p pprov_data = cb_data.pprov_data;
    data_uart_debug("Has provisioned device with addr 0x%04x in %dms!\r\n",
        pprov_data->unicast_address, plt_time_read_ms() - prov_start_time);
    /* the spec requires to disconnect, but you can remove it as you like! :) */
    prov_disconnect(PB_ADV_LINK_CLOSE_SUCCESS);
#ifdef MESH_USER_API_DEMO
    if (bt_mesh_indication(GEN_MESH_CODE(_prov), BT_MESH_USER_CMD_SUCCESS,
        (void *)&pprov_data->unicast_address) != USER_API_RESULT_OK) {
        data_uart_debug("[BT_MESH] %s(): user cmd %d fail!\r\n", __func__,
            GEN_MESH_CODE(_prov));
    }
#endif
}
```

cfg_client_receive() function is used to handle the callback messages of configure model. It acts just like prov_cb(), but for different events. Finally it will call bt_mesh_indication() to call user call back function.

```
case MESH_MSG_CFG_NODE_RESET_STAT:
{
    parse = TRUE;
    data_uart_debug("Node 0x%04x reseted!\r\n", pmesh_msg->src);
#ifdef MESH_USER_API_DEMO
    if (bt_mesh_indication(GEN_MESH_CODE(_node_reset),
        BT_MESH_USER_CMD_SUCCESS, &pmesh_msg->src) != USER_API_RESULT_OK) {
        data_uart_debug("[BT_MESH] %s(): user cmd %d fail!\r\n", __func__,
            GEN_MESH_CODE(_node_reset));
    }
#endif
}
break;
```

generic_on_off_client_data () function is used to handle the callback messages of goo model. It will call bt_mesh_indication() to call user call back function.

```
case GENERIC_ON_OFF_CLIENT_STATUS:
{
    generic_on_off_client_status_t *pdata = pargs;
```



```
        if (pdata->optional)
        {

        }
        else
        {
            data_uart_debug("goo client receive: src = %d, present = %d\r\n", pdata->src,
                pdata->present_on_off);
#ifdef MESH_USER_API_DEMO
            {
                uint8_t ret = USER_API_RESULT_ERROR;
                if (pdata->present_on_off) {
                    ret = bt_mesh_indication(GEN_MESH_CODE(_generic_on_off_get),
                        BT_MESH_GOO_ON, (void *)&pdata->src);
                } else {
                    ret = bt_mesh_indication(GEN_MESH_CODE(_generic_on_off_get),
                        BT_MESH_GOO_OFF, (void *)&pdata->src);
                }
            }
#endif
        }
    }
}
```

3.1.3.2 Mesh Message Handler

bt_mesh_send_io_msg() is used to send message to mesh stack, describing a request of an command. Event value of these messages from user command (bt_mesh_set_user_cmd) should be set to EVENT_USER_API_REQ. And io_msg.type should be the same mesh code index from bt_mesh_set_user_cmd.

```
void bt_mesh_send_io_msg(T_IO_MSG *p_io_msg)
{
    uint8_t event = EVENT_USER_API_REQ;
    if (os_msg_send(io_queue_handle, p_io_msg, 0) == false)
    {
        printf("[BT_MESH] Send io_msg to io_queue_handle fail!\r\n");
    }
    else if (os_msg_send(evt_queue_handle, &event, 0) == false)
    {
        printf("[BT_MESH] Send io_msg to evt_queue_handle fail!\r\n");
    }
}
```

Whenever mesh stack receives the corresponding messages, bt_mesh_io_msg_handler() will be invoked. Within bt_mesh_io_msg_handler(), io_msg.type will be judged firstly for illegally

access to `bt_mesh_cmd_hdl()`.

```
void bt_mesh_io_msg_handler(T_IO_MSG io_msg)
{
    uint8_t ret = 1;

    if (io_msg.type < MAX_MESH_PROVISIONER_CMD) {
        ret = bt_mesh_user_cmd_hdl(io_msg.type, (user_cmd_parse_value_t *)io_msg.u.buf);
        if (ret != 0) {
            printf("[BT_MESH] %s(): bt_mesh_io_msg_handler Fail!\r\n", __func__);
        }
    } else {
        printf("[BT_MESH] %s(): Error mesh code %d \r\n", __func__, io_msg.type);
    }
}
```

Finally, provisioner command table will be invoked in `bt_mesh_user_cmd_hdl()`, identifying with the mesh code index. Provisioner command table has be described in chapter ***bt_mesh_provisioner_cmd***. Device command table has be described in chapter ***bt_mesh_device_cmd***.

```
uint8_t bt_mesh_user_cmd_hdl(uint16_t mesh_code, user_cmd_parse_value_t
*pparse_value)
{
    uint8_t ret = 1;
    user_cmd_parse_result_t (*cmd_hdl)(user_cmd_parse_value_t *pparse_value);

    if (pparse_value == NULL) {
        printf("[BT_MESH] %s(): pparse_value is NULL!\r\n", __func__);
        goto exit;
    }
    cmd_hdl = provisionercmds[mesh_code].mesh_func;
    ret = cmd_hdl(pparse_value);
    if (ret != USER_CMD_RESULT_OK) {
        printf("[BT_MESH] %s(): provisioner cmd fail! (%d)\r\n", __func__, ret);
        goto exit;
    }

exit:
    return ret;
}
```

3.1.3.4 User Callback Handler

`bt_mesh_user_cmd_cbk()` is invoked by `bt_mesh_indication`, if it has been registered

bt_mesh_set_user_cmd(). bt_mesh_user_cmd_cbk() will act according the mesh_code index. If puserItem->synchMode is enabled, it will also up the userSema, which is requested by bt_mesh_set_user_cmd for synchronization mode. It will also invoke bt_mesh_free_hdl() to free resource allocated by bt_mesh_alloc_hdl() if the timeout event occurs.

```
case GEN_MESH_CODE(_pb_adv_con):
{
    if (puserItem->userCmdResult == BT_MESH_USER_CMD_SUCCESS) {
        bt_mesh_lib_priv.connect_device_flag = 1;
    }else {
        bt_mesh_lib_priv.connect_device_flag = 0;
    }
    if (puserItem->synchMode) {
        if (!puserItem->semaDownTimeOut) {
            rtw_up_sema(&puserItem->userSema);
        } else {
            bt_mesh_free_hdl(puserItem);
        }
    }else {
        printf("[BT_MESH] %s(): user cmd %d complete !\r\n", __func__, mesh_code);
        bt_mesh_free_hdl(puserItem);
    }
}
break;
```

3.1.4 Test Procedure

Firstly, please build and download the provisioner and device to separated Evolution Board. There are no other steps for Device, except powering on. But for Provsioner, currently, developer needs to install corresponding application in smart phone powered by android.

3.1.4.1 Test with Android Device

Supporting development app is named **BT_MESH_1.0.0.beta_05.apk**. After successfully installing, open it and switch to Join WIFI column, as Figure 3-1. Then tap the search, we can see “Scan BT Device” and then “Connect to Ameba” as Figure 3-2. Then it will try to scan AP details around. After successfully scanning the AP, developer should select the same AP currently connecting with your phone as Figure 3-3.

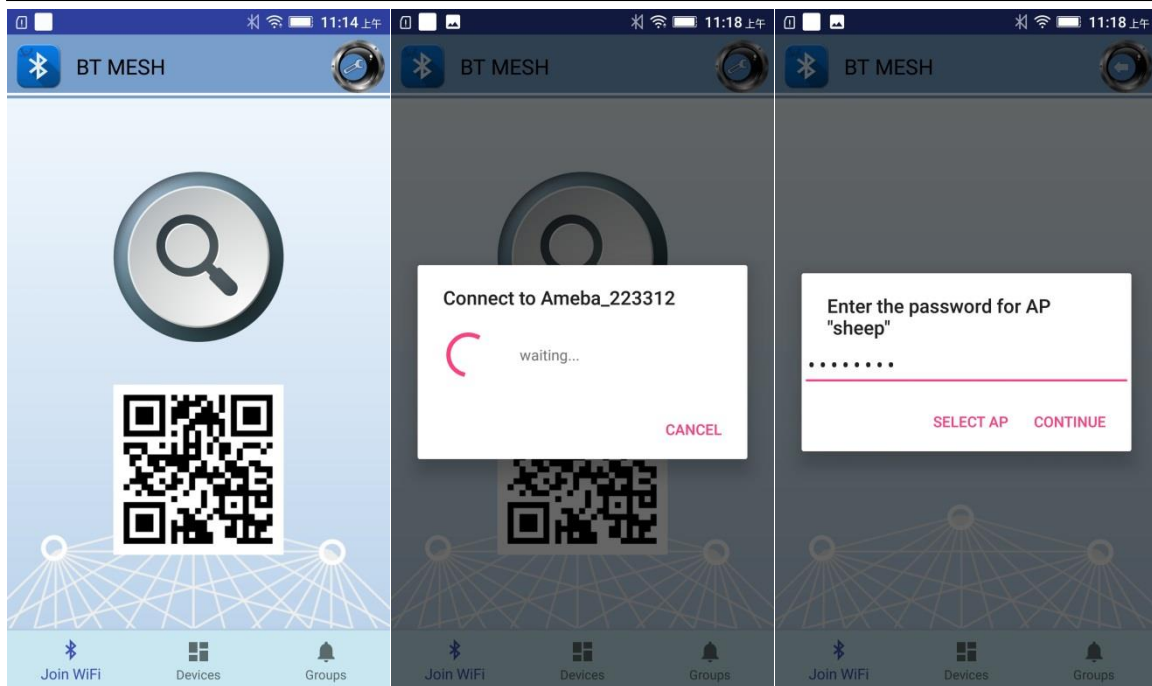


Figure 3-1 Start Menu

Figure 3-2 Connect Device

Figure 3-3 Connect AP

When provisioner successfully connects to AP, developer can change to Devices column as Figure 3-4. The devices or node around will be listed. Then we can start selecting devices waiting for provisioning, just like Figure 3-5. If provisioning process without errors, we can start to configure the device light state like Figure 3-6.

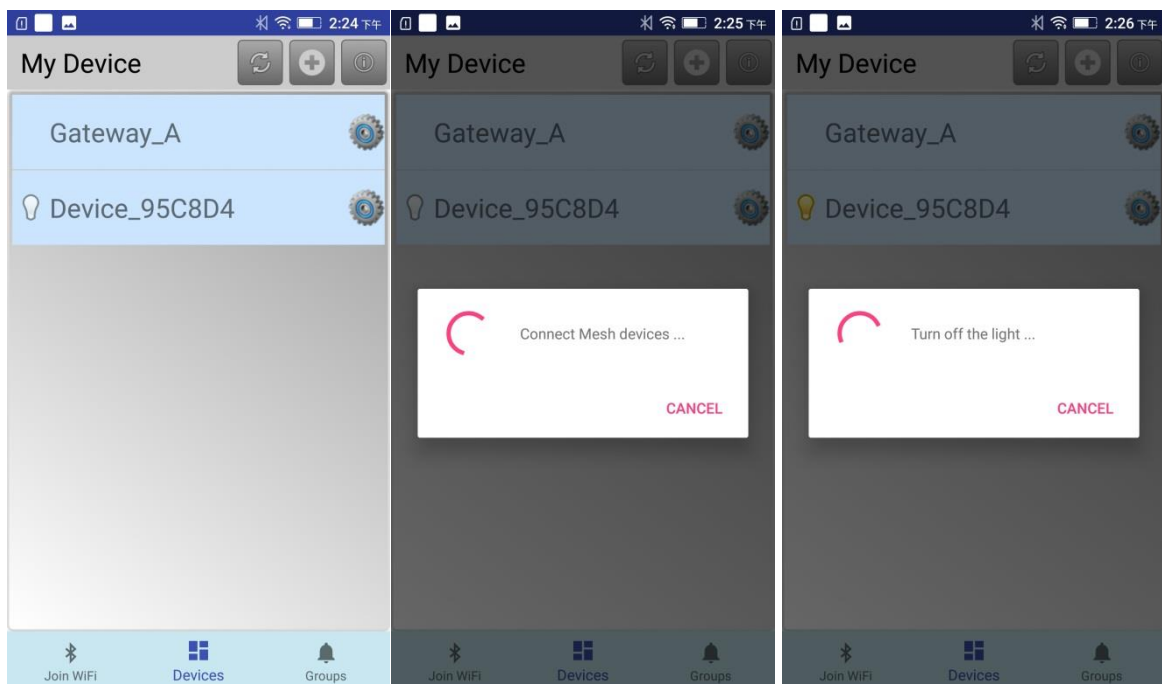


Figure 3-4 Select Device

Figure 3-5 Provisioning

Figure 3-6 Configure Light State

The results of the goos actions are shown in figure 3-7. The detailed movement after receiving goos from provisioner is configured in `generic_on_off_server_data()` function within device.


```
[rtw coex]: ble scan + sta unconnected
Provisioner turn light ON!
[rtw coex]: coex case use default
[rtw coex]: ble scan + sta unconnected
[rtw coex]: coex case use default
[rtw coex]: ble scan + sta unconnected
Provisioner turn light OFF!
[rtw coex]: coex case use default
[rtw coex]: ble scan + sta unconnected
Provisioner turn light ON!
[rtw coex]: coex case use default
[rtw coex]: ble scan + sta unconnected
[rtw coex]: coex case use default
[rtw coex]: ble scan + sta unconnected
```

Figure 3-7 GOOS Result

3.1.4.2 Test with Command Line Interface

Correctly Uart connection between Ameba board and pc is necessary. Then developer can tap command following the command table 3-1 for provisioner, using adv bearer. Table 3-2 using gatt bearer.

Table 3-1 Provisioner command line procedure using adv bearer

step	Provisioner	reference
1	ATBm=1	Start mesh provisioner mode
2	ATBM=pro,pbadvcon,x665544778822060708090A0B0 COD0E0F	Establish advertising connection with specific device through UUID
3	ATBM=pro,prov	Start provisioning via opened advertising connection, which will be closed after successfully processing. Unique node address will be printed through Uart(ex:0x100).
4	ATBM=pro,aka,x100,0,0	Add app key index 0 to node address 0x100, and bind which to net key index 0.
5	ATBM=pro,mab,x100,0,0x1000FFFF,0	Bind app key index 0 to model id 0x1000FFFF (Generic-On-Off Model) of the target node(0x100) with element index 0.
6	ATBM=pro,goos,x100,0,1	Set the light state to 0 of the node(0x100)
7	ATBM=pro,goos,x100,1,1	Set the light state to 1 of the node(0x100)
8	ATBM=pro,goog,x100,0	Get the light state of the node(0x100) with app index 0.
9	ATBM=pro,nr,x100	Delete node 0x100.

Table 3-2 Provisioner command line procedure using gatt bearer

step	Provisioner	reference
1	ATBm=1	Start mesh provisioner mode
2	ATBM=pro,con,deadbeef1234	Establish ble link with bt device, whose bt address is deadbeef1234

3	ATBM=pro,provdiss	Start searching provision service.
4	ATBM=pro,provcmd,0,1	Enable provision cccd.
5	ATBM=pro,prov	Start provisioning via opened advertising connection, which will be closed after successfully processing. Unique node address will be printed through Uart(ex:0x100).
6	ATBM=pro,con,deadbeef1234	Need to establish gatt link again, cause after provisioning, gatt link will be closed.
7	ATBM=pro,proxydis	Start searching proxy service.
8	ATBM=pro,proxycmd,0,1	Enable proxy ccd
9	ATBM=pro,aka,x100,0,0	Add app key index 0 to node address 0x100, and bind which to net key index 0.
10	ATBM=pro,mab,x100,0,0x1000FFFF,0	Bind app key index 0 to model id 0x1000FFFF (Generic-On-Off Model) of the target node(0x100) with element index 0.
11	ATBM=pro,goos,x100,0,1	Set the light state to 0 of the node(0x100)
12	ATBM=pro,goos,x100,1,1	Set the light state to 1 of the node(0x100)
13	ATBM=pro,goog,x100,0	Get the light state of the node(0x100) with app index 0.
14	ATBM=pro,nr,x100	Delete node 0x100.

Device command line procedure is listed in table 3-3.

Table 3-3 Device command line procedure

step	Provisioner	reference
1	ATBm=1	Start mesh device mode
2	ATBM=dev,ls	Print device information through Uart.
3	ATBM=dev,nr	Node Reset.

4 Mesh Multiple Profile

Mesh profile can coexist with other BLE profiles, such as central, peripheral and scatternet.

1. Mesh and Peripheral: Mesh basic function can work together with BLE peripheral features. Users can establish BLE link to device, which acts as provisioner or device role in mesh network.
2. Mesh and Central: Mesh basic function can work together with BLE central features. Device, which is a provisioner or node, can connect to peripheral using BLE link.
3. Mesh and scatternet: Mesh basic function coexists with BLE central and peripheral feature.

This chapter is going to describe the realization of mesh multiple profile introduced above.

4.1 Structure

The structure of mesh multiple profile related files in Realtek SDK is shown as below;
/component/common/Bluetooth/realtek/sdk/example/bt_mesh_multiple_profile:

```
.
├── device_multiple_profile
│   ├── bt_mesh_device_multiple_profile_app_flags.h
│   ├── bt_mesh_device_multiple_profile_app_main.c
│   ├── bt_mesh_device_multiple_profile_app_task.c
│   ├── bt_mesh_device_multiple_profile_app_task.h
│   ├── device_multiple_profile_app.c
│   ├── device_multiple_profile_app.h
│   ├── device_multiple_profile_cmd.c
│   └── device_multiple_profile_cmd.h
└── provisioner_multiple_profile
    ├── bt_mesh_provisioner_multiple_profile_app_flags.h
    ├── bt_mesh_provisioner_multiple_profile_app_main.c
    ├── bt_mesh_provisioner_multiple_profile_app_task.c
    ├── bt_mesh_provisioner_multiple_profile_app_task.h
    ├── provisioner_multiple_profile_app.c
    ├── provisioner_multiple_profile_app.h
    ├── provisioner_multiple_profile_cmd.c
    └── provisioner_multiple_profile_cmd.h
```

Where

bt_mesh_provisioner_multiple_profile_app_main.c configures the initiation measure.

provisioner_multiple_profile_cmd.c contains **provisionercmds** table, where finally commands to mesh lib will be called.

bt_mesh_device_multiple_profile_app_main.c configures the initiation measure.

device_multiple_profile_cmd.c contains **devicecmds** table, where finally commands will be called to mesh lib.

4.2 Implementation Process

Mesh multiple profile project is controlled by macro:

1) **CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE**

This macro should be set to 1 if provisioner role is needed.

2) **CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE**

This macro should be set to 1 if device role is needed.

3) **CONFIG_BT_MESH_PERIPHERAL**

This macro should be set to 1 if peripheral feature should coexist with bt mesh.

4) **CONFIG_BT_MESH_CENTRAL**

This macro should be set to 1 if central feature should coexist with bt mesh.

5) **CONFIG_BT_MESH_SCATTERNET**

This macro should be set to 1 if scatternet feature should coexist with bt mesh.

There are some rules should be obey for using these macro.

- **CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE** and **CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE** cannot be enabled simultaneously.
- **CONFIG_BT_MESH_PERIPHERAL**, **CONFIG_BT_MESH_CENTRAL** and **CONFIG_BT_MESH_SCATTERNET** cannot be enabled at the same time.
- **CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE** or **CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE** should be set before configuring **CONFIG_BT_MESH_PERIPHERAL**, **CONFIG_BT_MESH_CENTRAL** or **CONFIG_BT_MESH_SCATTERNET**.

4.2.1 Mesh Peripheral

CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE or **CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE** is in combination with **CONFIG_BT_MESH_PERIPHERAL**.

Mesh operation can refer to chapter 3.

BLE peripheral operation can refer to chapter 4.1 contained in document **UM0201 Ameba Common BT Application User Manual EN**

4.2.2 Mesh Central

CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE or **CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE** is in combination with **CONFIG_BT_MESH_CENTRAL**.

Mesh operation can refer to chapter 3.

BLE peripheral operation can refer to chapter 4.2 contained in document **UM0201 Ameba Common BT Application User Manual EN**

4.2.3 Mesh Scatternet

CONFIG_BT_MESH_PROVISIONER_MULTIPLE_PROFILE or
CONFIG_BT_MESH_DEVICE_MULTIPLE_PROFILE is in combination with
CONFIG_BT_MESH_SCATTERNET.

Mesh operation can refer to chapter 3.

BLE peripheral operation can refer to chapter 4.6 contained in document **UM0201 Ameba Common BT Application User Manual EN**

5 How To Build And Run BT Example

AmebaD:

Please refer to document **AN0400 Ameba-D Application Note v10.pdf**.

AmebaZ2:

Please refer to document **AN0500 Realtek Ameba-ZII application note.en.docx**.

6 Q&A

6.1 How To Use Mesh User Api In Timer Callback

If synchronization mode is needed, mesh user api will run in block mode waiting the response from mesh lib or timeout(2s). So we suggest not calling Mesh User Api within timer callback handler, cause this involving way may take relative long time every timer callback. Finally timer may be incorrectly.

Instead of calling mesh user api directly in timer callback, os queue message can be a properly way.

Cause the same reason above, mesh user api allocated in synchronization mode should be using prudently between mutex_claim and mutex_release.

6.2 How To Config Low Power Node Number

The parameter lpn_num when fn init represents the maximum number the friend node could establish friendship with. When config mesh node, net_key_num should be greater than lpn_num plus network key number.

6.3 Sync Mode and Async Mode configuration

Sync mode is just for mesh message with ack flag enabled.

If using sync mode api, bt_mesh_set_user_cmd will block until get the sema up by bt_mesh_indication or timeout. So bt_mesh_indication should be enable together with Sync mode.

In Async mode api, bt_mesh_set_user_cmd will return immediately without waiting for the remote device's ack(Sometimes there is no ack).

6.4 How to add a mesh node

1. table 3-1 has already support a way to provision a device using adv bearer;
2. table 3-2 has already support a way to provision a device using gatt;

6.5 How to delete a mesh node

1. Sync mode

```
puserItem = bt_mesh_alloc_hdl(USER_API_SYNCH);
```



```
if (!puserItem) {
    printf("[BT_MESH_DEMO] bt_mesh_alloc_hdl fail!\r\n");
    return 0;
}
puserItem->pparseValue->dw_parameter[0] = mesh_addr; //node
addr will be delete
puserItem->pparseValue->para_count = 1; //param number
ret = bt_mesh_set_user_cmd(GEN_MESH_CODE(_node_reset), puserI
tem->pparseValue, bt_mesh_user_cmd_cbk, puserItem);
if (ret != USER_API_RESULT_OK) {
    printf("[BT_MESH_DEMO] bt_mesh_set_user_cmd fail! %d\r\n"
, ret);
}
```

2. Async mode

```
puserItem = bt_mesh_alloc_hdl(USER_API_ASYNC);
if (!puserItem) {
    printf("[BT_MESH_DEMO] bt_mesh_alloc_hdl fail!\r\n");
    return 0;
}
puserItem->pparseValue->dw_parameter[0] = mesh_addr; //node
addr will be delete
puserItem->pparseValue->para_count = 1; //param number
ret = bt_mesh_set_user_cmd(GEN_MESH_CODE(_node_reset), puserI
tem->pparseValue, bt_mesh_user_cmd_cbk, puserItem);
```