PRBMD02 Application Note

MESH development note

Disclaimer

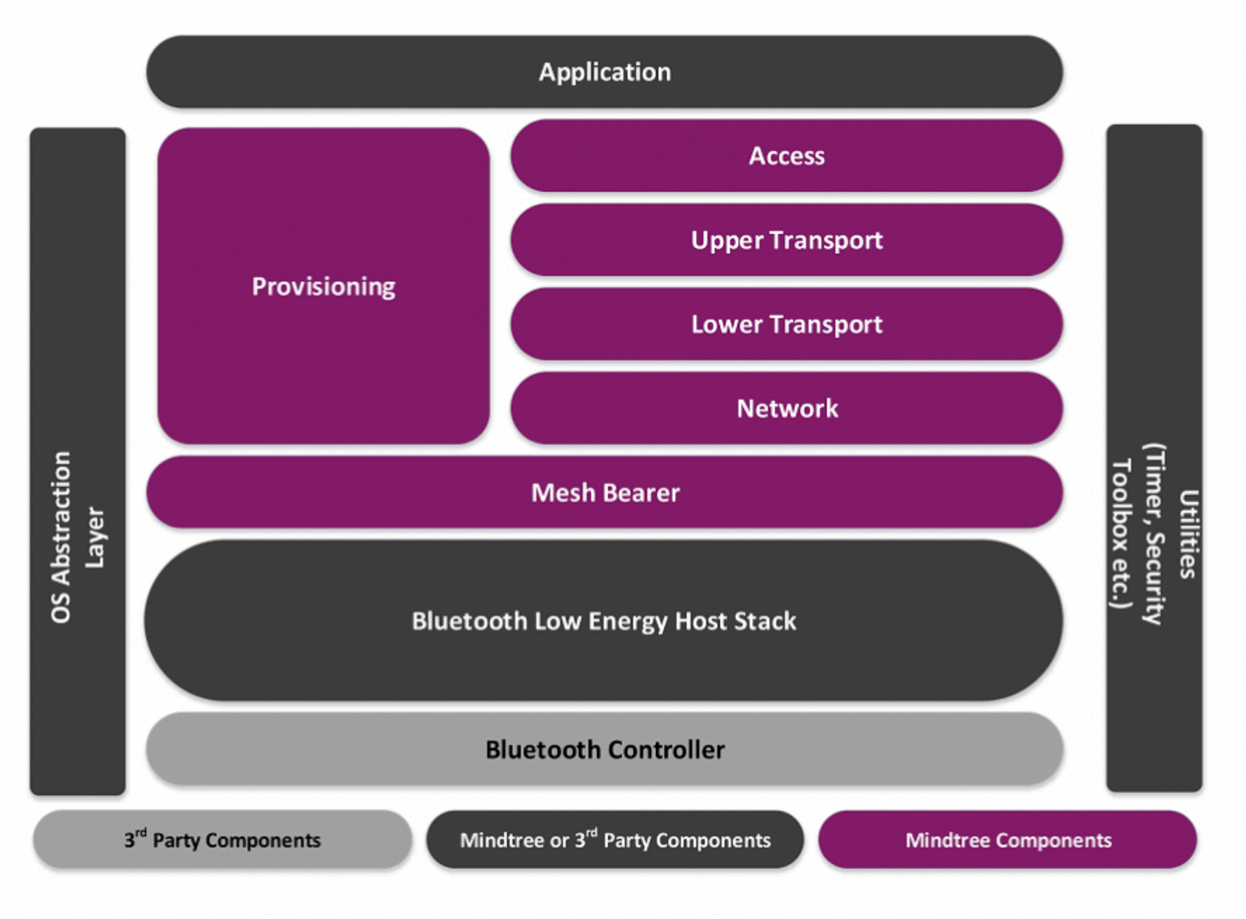
Liability Disclaimer
K-Solution Consulting Co. Ltd reserves the right to make changes without further notice to the product to improve reliability, function or design. K-Solution Consulting Co. Ltd does not assume any liability arising out of the application or use of any product or circuits described herein.

Life Support Applications
K-Solution Consulting Co. Ltd's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. K-Solution Consulting Co. Ltd customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify K-Solution Consulting Co. Ltd for any damages resulting from such improper use or sale.

The table of contents is empty because you aren't using the paragraph styles set to appear in it.

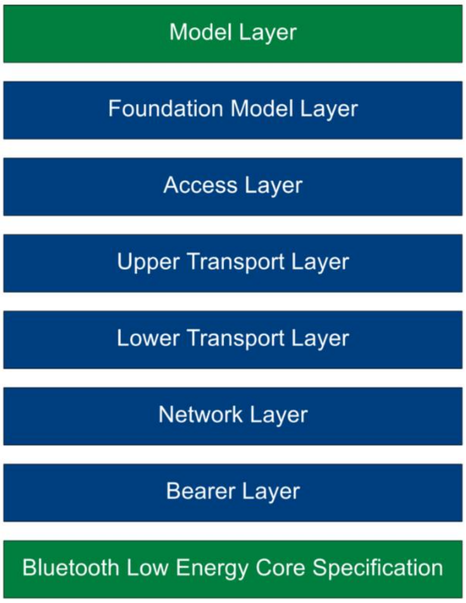
A. Introduction

This document is used for the introduction and usage of PHY622X Mesh. It helps you understand and understand the components provided by our company's Mesh, how to use the samples, and how to start BLE Mesh development from the samples provided.



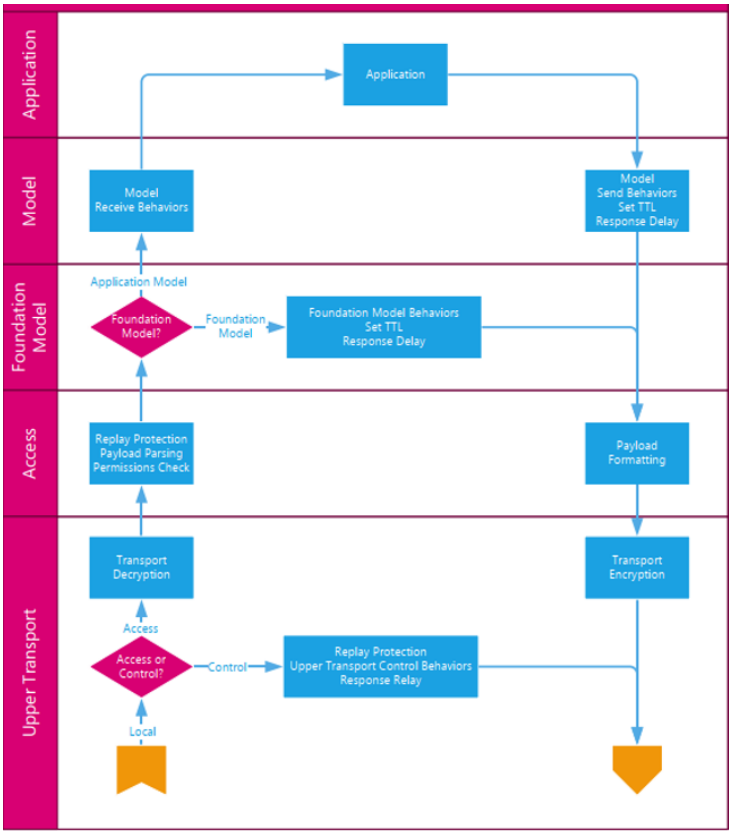
1. MESH protocol stack

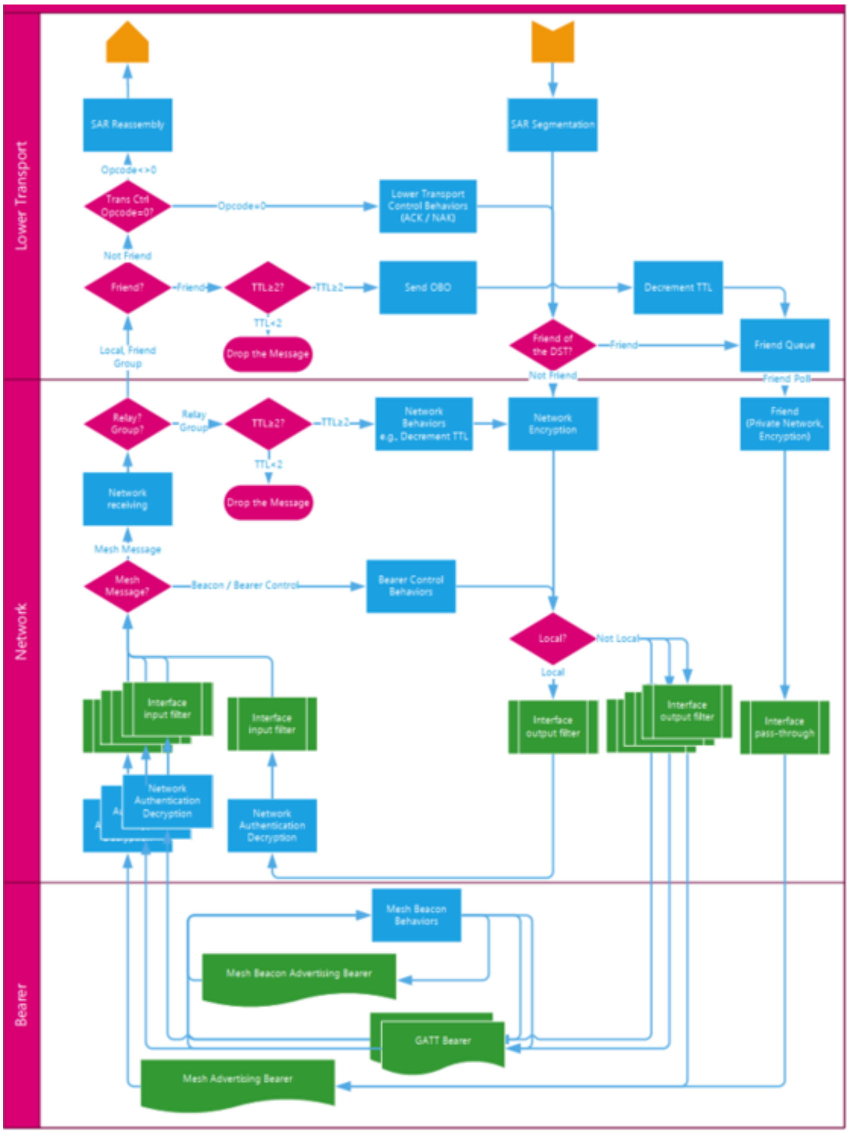
This protocol stack is built on Bluetooth low energy technology. The following diagram depicts the layers of the protocol stack.



- Model Layer: The model layer is related to the implementation of models, etc., and the implementation of behaviors, messages, states, etc.
- Foundation Model Layer: The Foundation Model Layer is responsible for implementing models related to mesh network configuration and management.
- Access Layer: Responsible for the format of application data, define and control the encryption and decryption process performed in the upper transport layer, and verify that the received data is suitable for the correct network and before forwarding the data to the protocol stack. application
- Upper Transport Layer: It is responsible for encrypting, decrypting and authenticating application data in and out of the access layer. It is also responsible for special messages called "transport control messages", including "friendship" related heartbeats and messages.
- Lower Transport Layer: The lower transport layer can handle the segmentation and reassembly of PDUs when needed.
- Network Layer: The network layer defines various message address types and network message formats. Relay and proxy behavior is implemented through the network layer.
- Bearer Layer: The bearer layer defines how to transmit PDUs using the underlying low-power stack. Two bearer layers are currently defined: the Advertising Bearer and the GATT bearer.

2. Message Flow

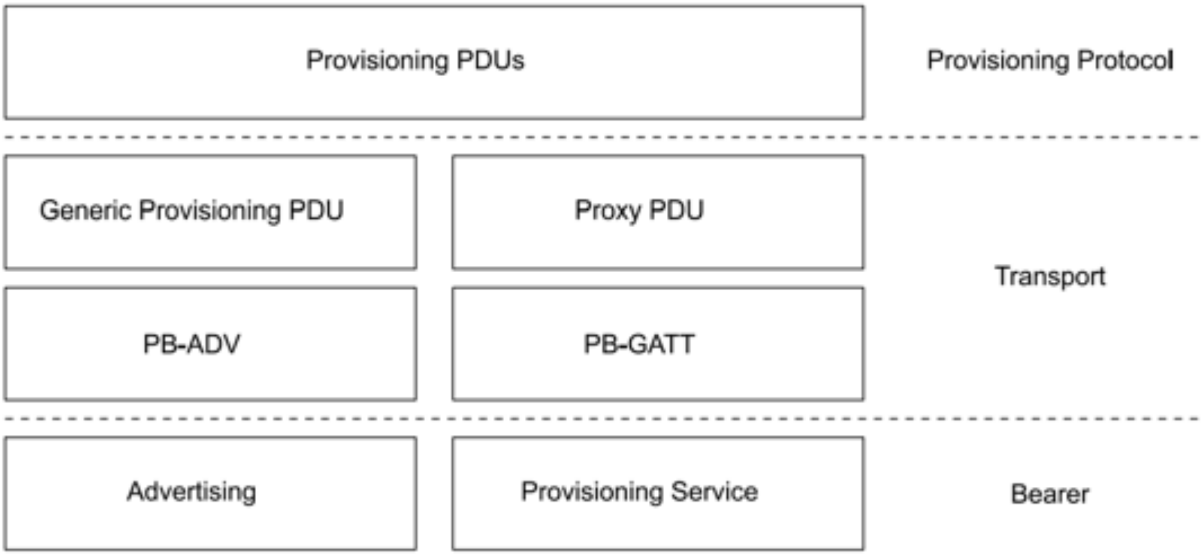




The Mesh message transmission process is shown in the above two figures. After the mesh message enters from the bear layer through ADV/GATT, after the network layer decodes the input filter through the interface, if it is a relay or proxy message, it is implemented at the network layer, and the non-relay message is implemented at the network layer. It will enter the bottom transport layer for splitting and reorganization, and then enter the upper transport layer for transmission and decryption. At the access layer, it will be sent to the basic model layer after legality check, and finally implemented through the implementation of the model layer.
When sending a message, it is sent through an instance of the model layer. The data format is defined by the access layer, encrypted at the upper transport layer, passed to the bottom transport layer for unpacking and grouping, encrypted at the network layer, and then entered into the bearer layer through the interface output filter. output.

3. MESH configuration

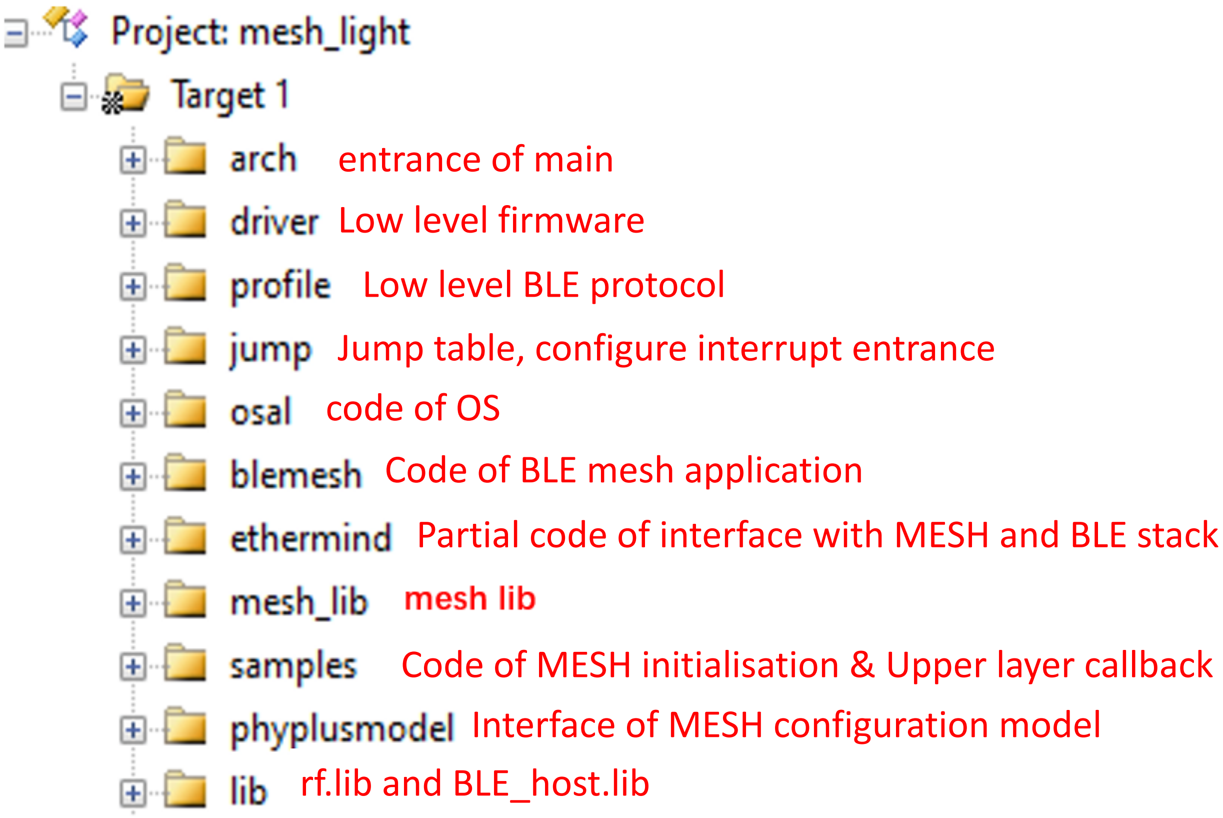
Provisioning is the process of adding unconnected devices to a mesh network. The network configuration device provides configuration data for the unconfigured devices to enable them to access the network. Thus making it a mesh node. The issuance data includes the network key, the current IV index, and the address of each element unicast.



Configuration of the device is done using the configuration protocol that sends the configuration PDU. The configuration pdu is transferred to the unconfigured device through the common configuration layer. This layer defines how the configuration pdu is handled as a transaction that can be split and reassembled. These transactions are sent through a configuration bearer. The configuration bearer layer defines how sessions are established to deliver transactions from the common configuration layer to a single device. Finally, at the bottom of the configuration architecture is the bearer layer.

B. Introduction of project and API

1. Introduction of project

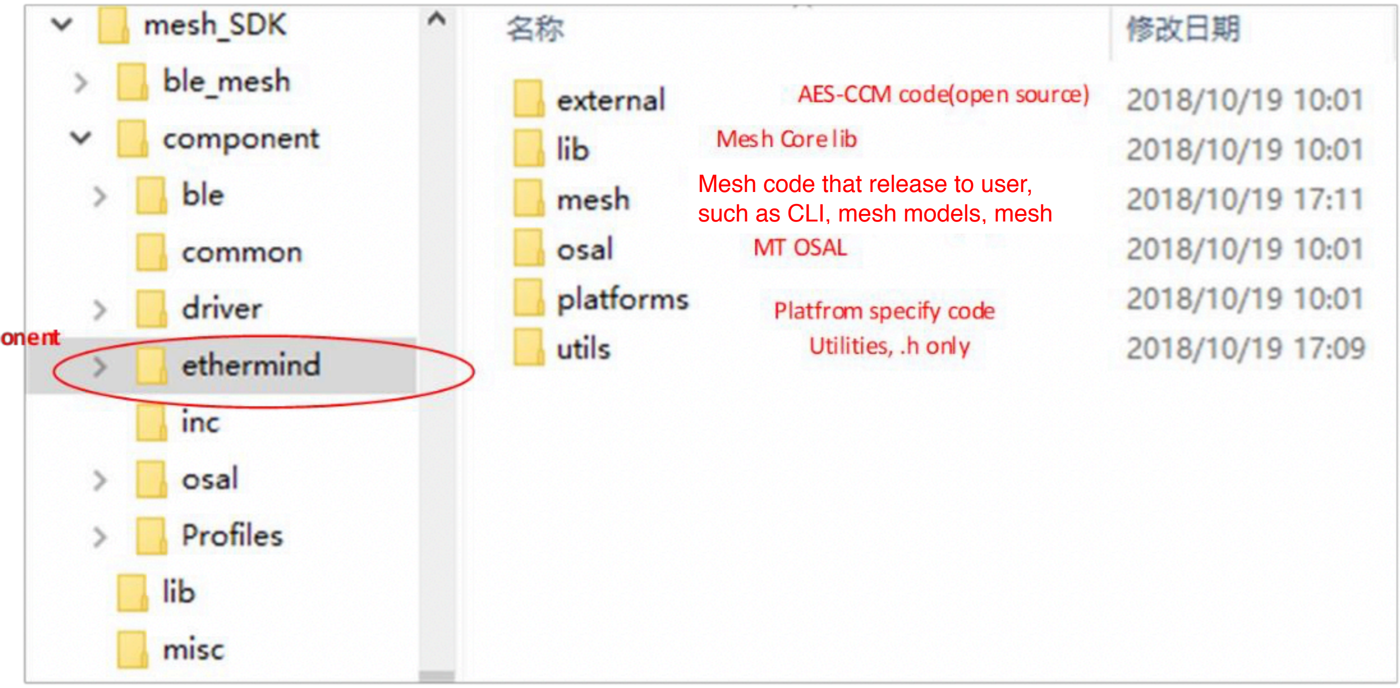


Project: mesh_light

Target 1

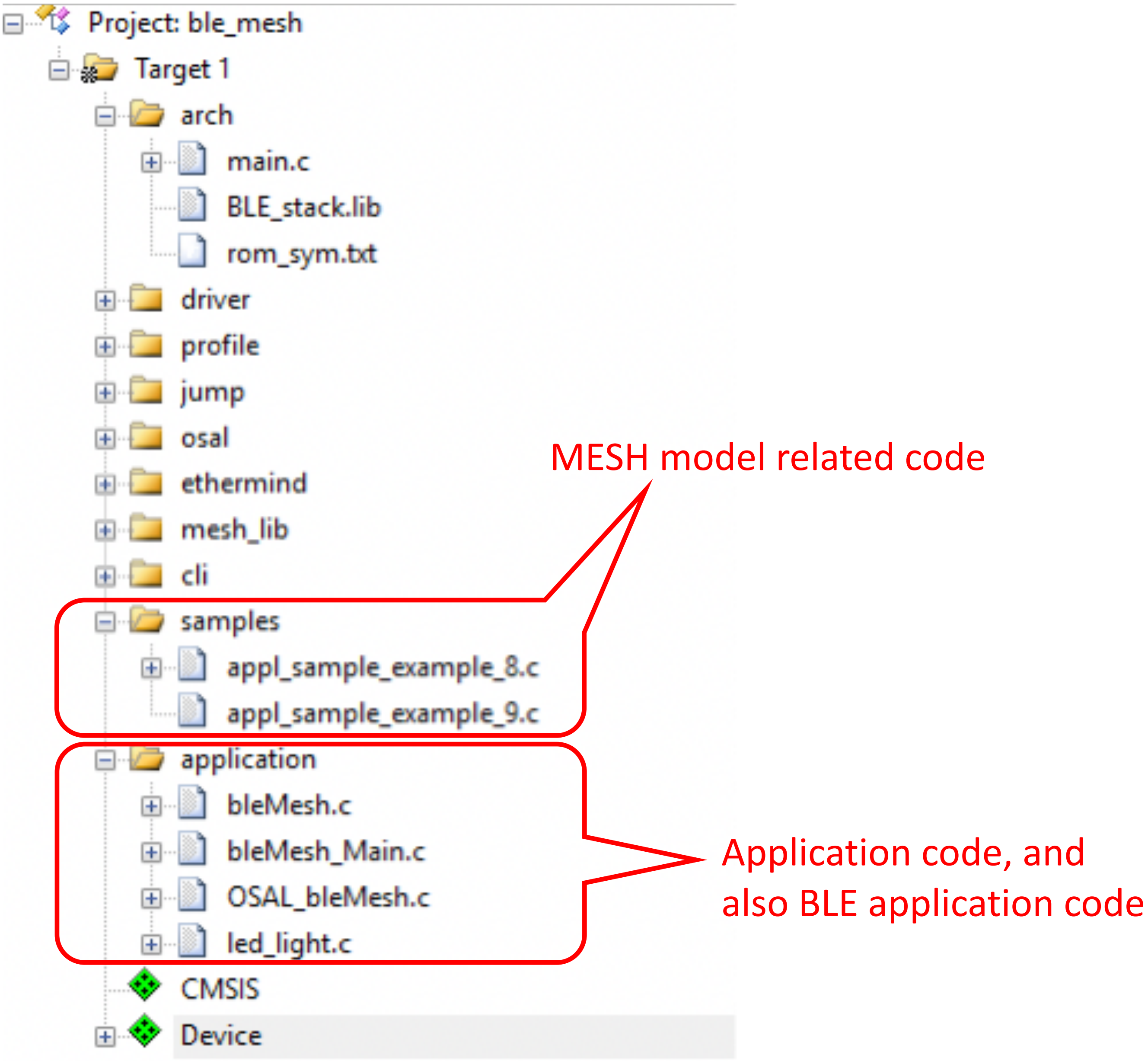
- arch entrance of main
- driver Low level firmware
- profile Low level BLE protocol
- jump Jump table, configure interrupt entrance
- osal code of OS
- blemesh Code of BLE mesh application
- ethermind Partial code of interface with MESH and BLE stack
- mesh_lib mesh lib
- samples Code of MESH initialisation & Upper layer callback
- phyplusmodel Interface of MESH configuration model
- lib rf.lib and BLE_host.lib

1.1.Ethermin



名称		修改日期
external	AES-CCM code(open source)	2018/10/19 10:01
lib	Mesh Core lib	2018/10/19 10:01
mesh	Mesh code that release to user, such as CLI, mesh models, mesh	2018/10/19 17:11
osal	MT OSAL	2018/10/19 10:01
platforms	Platfrom specify code	2018/10/19 10:01
utils	Utilities, .h only	2018/10/19 17:09

1.2.Mesh samples & application



```
Project: ble_mesh
  Target 1
    arch
      main.c
      BLE_stack.lib
      rom_sym.txt
    driver
    profile
    jump
    osal
    ethermind
    mesh_lib
    cli
    samples
      appl_sample_example_8.c
      appl_sample_example_9.c
    application
      bleMesh.c
      bleMesh_Main.c
      OSAL_bleMesh.c
      led_light.c
    CMSIS
    Device
```

MESH model related code

Application code, and also BLE application code

2. Common module definition

Some common module definitions are similar to other sdk demos, the following is the definition of mesh.

Name	value	description
OSAL_CBTIMER_NUM_TASKS	1	The number of internal call back timers used, currently only 1 is supported and cannot be changed. The timers configured in the mesh are all calling internal callback timers.
CFG_HEARTBEAT_MODE	0	Disable Heartbeat feature
CFG_HEARTBEAT_MODE	1	Enable Heart beat feature

2.1.Introduction to MESH sample

The internal interfaces of PHY62XX Mesh are all in the lib library. The commonly used libs are libethermind_ecdh.lib, libethermind_mesh_core.lib, libethermind_mesh_models.lib and libethermind_utils.lib; the functions are as follows:

- **libethermind_ecdh.lib**: related to ecdh, currently not used by sdk
- **libethermind_mesh_core.lib**: related to the mesh protocol stack; provision, config and message processing are all performed here
- **libethermind_mesh_models.lib**:mesh model is related; the implementation of models such as on/off currently used are all handled here
- **libethermind_utils.lib**:mesh storage related In addition to lib, the files that users touch and change are generally in appl_sample_mesh_XXX.c. This chapter focuses on sample related interfaces and definitions (take the commonly used mesh_light as an example).

2.2.Definition model

USE_HEALTH	#undef: disable health model #define: enable health model
USE_HSL	#undef: disable Light HSL model #define: enable Light HSL model
USE_LIGHTNESS	#undef: disable Light Lightness model #define: enable Light Lightness model
USE_CTL	#undef: disable Light CTL model #define: enable Light CTL model
USE_SCENE	#undef: disable Light Scene model #define: enable Light Scene model
USE_VENDORMODEL	#undef: disable vendormodel model #define: enable vendormodel model

If Vendormodel is enabled, it will automatically enable easy bonding (currently the sdk we use is considered this way, and it is used with the Phy mesh app).

2.3.API description

2.3.1.UI_health_server_cb

Health server Callback function

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	model handle
UINT8	event_type	event type
UINT8 *	event_param	event parameter content
UINT16	param_len	parameter length

Return value:

API_SUCCESS	Success
API_FAILER	Failure

2.3.2.UI_register_ foundation_model_servers

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	model handle

Return value:

API_SUCCESS	Success
API_FAILER	Failure

2.3.3.UI_generic_onoff_model_states_initialization

Generic on/off model status initialisation:

None

Return value:

None

2.3.4.UI_generic_onoff_model_state_get

Obtain Generic on/off model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	Direction

Return value:

API_SUCCESS	Success
API_FAILER	Failure

2.3.5.API_RESULT MS_access_register_element

Establish Primary element:

type	parameter	description
MS_ACCESS_NODE_ID	node_id	node ID, init value is 0
MS_ACCESS_ELEMENT_DESC*	element	initial Element pointer to the element descriptor that needs to be registered with the node
MS_ACCESS_ELEMENT_HANDLE*	element_handle	Element handle identifier referencing the newly registered element

Return value:

API_SUCCESS	Success
Other value	refers to <MS_error.h>

2.3.6.API_RESULT UI_register_foundation_model_servers

Establish Foundation element:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	Element handle identifier referencing the newly registered element

Return value:

API_SUCCESS	Success
Other value	refers to <MS_error.h>

2.3.7. API_RESULT UI_register_generic_onoff_model_server

Register Generic OnOff model server:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	Element handle identifier referencing the newly registered element

Return value:

API_SUCCESS	Success
Other value	refers to <MS_error.h>

2.3.8.UI_generic_onoff_server_cb

Generic on/off model callback:

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	Contextual content of on/off messages
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	raw message
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	message type
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	message content
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	other parameter

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.9.UI_ light_hsl_model_states_initialization

Generic light HSL model initialisation:

None

Return value:

None

2.3.10.UI_ light_hsl_model_state_get

Obtain generic light HSL model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.11.UI_light_hsl_model_state_set

Set generic light HSL model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.12.UI_light_hsl_server_cb

Generic light HSL model callback:

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	Contextual content of on/off messages

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	raw message
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	message type
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	message content
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	other parameter

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.13.UI_ light_ctl_model_states_initialization

Generic light ctl model initialisation:

None

Return value:

None

2.3.14.UI_ light_ctl_model_state_get

Obtain generic light ctl model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.15.UI_light_ctl_model_state_set

Set generic light ctl model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.16.UI_light_ctl_server_cb

Generic light ctl model callback:

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	Contextual content of on/off messages
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	raw message
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	message type
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	message content

type	parameter	description
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	other parameter

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.17.UI_register_light_ctl_model_server

Register generic light ctl model server:

type	parameter	description
MS_ACCESS_MODEL_HANDLE	handle	model handle

Return value:

API_SUCCESS	Success
API_Failer	failure

2.3.18.UI_vendor_defined_model_states_initialization

Vendor model status initialisation:

None

Return value:

None

2.3.19.UI_vendor_example_model_state_get

Obtain vendor model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.20.UI_vendor_example_model_state_set

Set vendor model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.21.UI_phy_model_server_cb

Vendor model callback:

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	Contextual content of on/off messages

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	raw message
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	message type
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	message content
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	other parameter

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.22.UI_register_vendor_defined_model_server

Register vendor model server:

type	parameter	description
MS_ACCESS_MODEL_HANDLE	handle	model handle

Return value:

API_SUCCESS	Success
API_Failer	failure

2.3.23.UI _model_states_initialization

Initialisation of Mesh model status:

None

Return value:

None

2.3.24.API_RESULT MS_access_cm_set_transmit_state

Set network/relay transfer status:

type	parameter	description
UNIT8	tx_state_type	Transmission state type (network or relay)
UNIT8	tx_state_type	Composite state (3-bit Tx count, 5-bit Tx interval step)

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.25.API_RESULT MS_access_cm_set_features_field

Enable/disable a feature:

type	parameter	description
UNIT8	enable	enable/disable
UNIT8	tx_state	Relay, proxy, friendship, low power consumption four characteristics

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.26.API_RESULT MS_access_bind_model_app

Bind the model with the appkey:

type	parameter	description
MS_ACCESS_MODEL_HANDLE	model_handle	A model handle that identifies the model

type	parameter	description
UNIT16	appkey_index	A global index that identifies the appkey

Return value:

API_SUCCESS	Success
other value	refers <MS_error.h>

2.3.27.MS_proxy_server_adv_start

Enable the proxy server to enable connectable non-targeted advertisement:

type	parameter	description
MS_SUBNET_HANDLE	subnet_handle	The subnet handle where the proxy server is located
UCHAR	proxy_adv_mode	Proxy broadcast mode, two modes: MS_PROXY_NET_ID_ADV_MODE / MS_PROXY_NODE_ID_ADV_MODE

Return value:

API_SUCCESS	Success
other value	refers <MS_error.h>

2.3.28.MS_prov_setup

Configure a device to be provisionable by assigning roles, hosting and creating content:

type	parameter	description
PROV_BRR	bearer	Device/Provisioner
PROV_ROLE	role	PB-ADV/ PB-GATT
PROV_DEVICE_S*	pdevice	pointer to device, only used if role=PROV_ROLE_DEVICE otherwise ignored
UNIT16	gatt_timeout	Gatt start up time
UNIT16	adv_timeout	Adv start up time

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.29.API_RESULT MS_prov_bind

Bind a specific device:

type	parameter	description
PROV_BRR	bearer	Device/Provisioner
PROV_DEVICE_S*	pdevice	pointer to device, only used if role=PROV_ROLE_DEVICE otherwise ignored
UCHAR	attention	Device overtime alert time
PROV_Handle*	Phandel1	A handle that references the content of the distribution network

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.30.API_RESULT MS_prov_register

Distribution layer registration:

type	parameter	description
MS_SUBNET_HANDLE	subnet_handle	The subnet handle where the proxy server is located
UCHAR	proxy_adv_mode	Proxy broadcast mode, two modes: MS_PROXY_NET_ID_ADV_MODE / MS_PROXY_NODE_ID_ADV_MODE

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.31.API_RESULT MS_proxy_server_adv_stop (void)

Make the proxy server stop connectable broadcasts:

None

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.32.API_RESULT MS_proxy_register

Register the interface with the network proxy layer:

type	parameter	description
PROXY_NTF_CB	proxy_cb	Upper layer notification callback

Return value:

EM_SUCCESS	Success
EM_FAILURE	failure

2.3.33.API_RESULT MS_access_cm_get_primary_unicast_address

Obtain primary unicast address:

type	parameter	description
MS_NET_ADDR*	add	The memory location to populate the primary unicast address

Return value:

EM_SUCCESS	Success
EM_FAILURE	failure

2.3.34.API_RESULT MS_access_reply

Response to access layer messages:

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	model handle
MS_NET_ADDR	saddr	source address
MS_NET_ADDR	daddr	destination address
MS_SUBNET_HANDLE	subnet_handle	sub-net handle
MS_APPKEY_HANDLE	appkey_handle	Apply handle
UNIT8	ttl	Time to live
UNIT32	opcode	Operation code
UCHAR*	data_param	access parameter
UNIT16	data_len	access parameter length

Return value:

EM_SUCCESS	Success
EM_FAILURE	failure

2.3.35.API_RESULT MS_scene_server_init

Initialize the scene server model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	scene_model_handle	The model identifier associated with the scene model instance
MS_ACCESS_MODEL_HANDLE*	scene_setup_model_handle	The model identifier associated with the scene setup model instance
MS_SCENE_SERVER_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.36.API_RESULT MS_scene_client_init

Initialize the scene client model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	scene_model_handle	The model identifier associated with the scene model instance
MS_SCENE_CLIENT_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.37.API_RESULT MS_light_hsl_server_init

Initialize the hsl server model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	hsl_model_handle	The model identifier associated with the hsl model instance
MS_ACCESS_MODEL_HANDLE*	hsl_setup_model_handle	The model identifier associated with the hsl setup model instance
MS_LIGHT_HSL_SERVER_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.38.API_RESULT MS_light_hsl_client_init

Initialize the hsl client model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	HSL_model_handle	The model identifier associated with the scene model instance
MS_LIGHT_HSL_CLIENT_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.39.API_RESULT MS_light_ctl_server_init

Initialize the ctl server model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	ctl_model_handle	The model identifier associated with the ctl model instance
MS_ACCESS_MODEL_HANDLE*	ctl_setup_model_handle	The model identifier associated with the ctl setup model instance
MS_LIGHT_CTL_SERVER_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.40.API_RESULT MS_light_ctl_client_init

Initialize the ctl client model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	ctl_model_handle	The model identifier associated with the ctl model instance
MS_LIGHT_CTL_CLIENT_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.41.API_RESULT MS_generic_onoff_server_init

Initialize the generic onoff server model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	model_handle	The model identifier associated with the ctl model instance
MS_GENERIC_ONOFF__SERVER_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.42.API_RESULT MS_health_server_init

Initialize the health server model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	model_handle	The model identifier associated with the ctl model instance
UINT6	company_id	company identifier
MS_HEALTH_SERVER_SELF_TEST*	self_tests	A series of runnable self-tests
Uint32	num_self_tests	Number of runnable self-tests
MS_HEALTH_SERVER_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.43.API_RESULT MS_health_client_init

Initialize the health client model and register it at the access layer:

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	element_handle	element tag associated with the model instance identifier
MS_ACCESS_MODEL_HANDLE*	model_handle	The model identifier associated with the ctl model instance
MS_HEALTH_CLIENT_CB	appl_cb	Application callback

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.44.API_RESULT MS_access_register_model

Register the model at the access layer:

type	parameter	description
MS_ACCESS_NODE_ID	node ID	The node that the model needs to register, the value of the default node is always 0
MS_ACCESS_MODEL*	model	pointer to the model descriptor that needs to be registered with the node
MS_ACCESS_MODEL_HANDLE*	model_handle	The model identifier associated with the model instance on successful registration

Return value:

API_SUCCESS	Success
Other value	refers <MS_error.h>

2.3.45.UI_vendor_defined_model_states_initialization

Vendor model status initialisation:

None.

Return value:

None.

2.3.46.UI_vendor_example_model_state_get

Obtain vendor model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.47.UI_vendor_example_model_state_set

Obtain generic light ctl model status:

type	parameter	description
UNIT16	state_t	State type
UNIT16	state_inst	initial state
void*	param	state parameter
UNIT8	direction	direction

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.48.UI_phy_model_server_cb

Vendor model call back:

type	parameter	description
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	Contextual content of on/off messages
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	raw message
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	message type
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	message content
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	other parameter

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.49.UI_register_vendor_defined_model_server

Register to Vendor model server:

type	parameter	description
MS_ACCESS_MODEL_HANDLE	handle	model handle

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.3.50.UI _model_states_initialization

All states of mesh model are initialized:

None

Return value:

None

2.4.Provision interface

The specific implementation part of Provision is encapsulated in lib. The upper layer can see some configuration information and callback functions. This unit focuses on provision.

PROCFG_COMPLETE_TIMEOUT:	configurable, unit is sec

2.4.1.Unprovision beacon uuid

In the broadcast packet of the mesh unprovision beacon, the device ID can identify different mesh devices, which are defined as follows:

Name	Size	Note
Company ID	2	company ID, default: 0x0504
Product ID	2	Bluetooth device ID 0x62 0x12 0x62 0x22 0x62 0x52
Product type	4	Bluetooth device type 0x00xx– MESH_LIGHT 0x01xx – MESH_CTRL 0x02xx – MESH_LPN 0x03xx– MESH_SENS 0x04xx – TO BE ADD
Version	2	version # of software and hardware
MAC address	6	
RFU	2	Reserved for future use

2.4.2.UI_provcfg_complete_timeout_handler

Callback function for network configuration timeout

type	parameter	description
void*	args	callback parameter
UINT16	size	parameter length

Return value:

None

2.4.3.UI_prov_callback

Distribution network callback function：

type	parameter	description
PROV_HANDLE*	phandle	Provision handler
UCHAR	event_type	event type
API_RESULT	event_result	result of this event
void*	event_data	event data
UINT16	event_datalen	event data length

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.4.4.UI_register_prov

Register provision service:

None

Return value:

None

2.4.5.UI_proxy_start_adv

enable proxy beacon board cast (able to connect)

type	parameter	description
MS_SUBNET_HANDLE	subnet_handle	Network handler
UCHAR	proxy_adv_mode	Proxy board cast mode NET ID NODE ID

Return value:

None

2.4.6.UI_proxy_callback

Proxy call back function

type	parameter	description
NETIF_HANDLE*	handle	network handler
UCHAR	p_evt	Message type
UCHAR*	data_param	data
UINT16	data_len	data length

Return value:

None

2.4.7.UI_setup_prov

Configure provision

type	parameter	description
UCHAR	role	Provision role
UCHAR	brr	Provision bear type

Return value:

None

2.4.8.UI_register_proxy

Register proxy service:

None

Return value:

None

2.4.9.UI_set_brr_scan_rsp_data

Configure response data:

None

Return value:

None

2.4.10.UI_gatt_iface_event_pl_cb

Proxy call back function

type	parameter	description
UCHAR	ev_name	GATT event name
UCHAR	ev_param	GATT model name

Return value:

None

2.4.11.UI_sample_binding_app_key

The key binding is performed according to the previously configured model, and the message can be effectively transmitted only after the key is bound, otherwise the message will fail to be decrypted.

Parameter:

None

Return value:

None

2.4.12.vm_subscriptiong_binding_cb

Handle appkey add config message.

Parameter:

None

Return value:

None

2.4.13.vm_subscriptiong_add

Handle subscription add message

type	parameter	description
MS_NET_ADDR	add	Address of add to group

Return value:

None

2.4.14.vm_subscriptiong_delete

Handle subscription delete message

type	parameter	description
MS_NET_ADDR	add	Address of remove from group

Return value:

None

2.4.15.UI_app_config_server_callback

Handle subscription add message

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	Model's handle
MS_NET_ADDR	saddr	Source address
MS_NET_ADDR	daddr	Dest. address
MS_SUBNET_HANDLE	subnet_handle	Netwrok's handle
MS_APPKEY_HANDLE	appkey_handle	Appkey's handle
UINT322	opcode	Message's operation code
UCHAR*	data_parm	Message's content
UINT16	data_len	Message's length
API_RESULT	retval	Message's result
UINT32	response_opcode	Opcode that needed to be response
UCHAR*	response_buffer	Content of response to opcode
UINT16	response_buffer_len	Length of response to opcode

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.4.16.appl_mesh_sample

Initialisation of MESH sample.

Parameter:

None

Return value:

None

2.4.17.UI_sample_get_net_key

Obtain netkey.

Parameter:

None

Return value:

None

2.4.18.UI_sample_get_device _key

Optain devicekey.

Parameter:

None

Return value:

None

2.4.19.UI_sample_check_app

Obtain appkey.

Parameter:

None

Return value:

None

2.4.20.UI_sample_reinit

Mesh sample initialization, different from appl_mesh_sample, this interface is mainly used to prepare to initiate unprovision beacon, initiate proxy beacon or obtain key

Parameter:

None

Return value:

None

2.5.Other APIs

2.5.1.MS_access_cm_get_primary_unicast_address

Obtain unicast address

type	parameter	description
MS_NET_ADDR	addr	Output of unicast address

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.2.MS_access_get_element_handle

Obtain element handle

type	parameter	description
MS_NET_ADDR	elem_addr	The unicast address of the node to query
MS_ACCESS_ELEMENT_HANDLE*	handle	element handle of output

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.3.MS_access_get_model_handle

Obtain model handle

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	elem_handle	Element handle of the node to query
MS_ACCESS_MODEL_ID	model_id	Model ID of the node to query
MS_ACCESS_MODEL_HANDLE*	handle	Model handle output

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.4.MS_access_get_model_handle

Obtain model handle

type	parameter	description
MS_ACCESS_ELEMENT_HANDLE	elem_handle	Element handle of the node to query
MS_ACCESS_MODEL_ID	model_id	Model ID of the node to query
MS_ACCESS_MODEL_HANDLE*	handle	Model handle output

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.5.MS_access_get_element_handle

Obtain element handle

type	parameter	description
MS_NET_ADDR	elem_addr	Unicast address of the node to query
MS_ACCESS_ELEMENT_HANDLE*	handle	Element handle output

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.6.MS_access_get_model_handle

Obtain model handle

type	parameter	description
MS_NET_ADDR	elem_addr	Unicast address of the node to query
MS_ACCESS_ELEMENT_HANDLE*	handle	Element handle output

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.7.MS_access_get_model_handle

2.5.8.MS_access_get_element_handle

2.5.9.MS_access_get_model_handle

2.5.10.MS_access_get_model_handle

2.5.11.API_RESULT MS_config_client_send_reliable_pdu

Send a command to reply

type	parameter	description
UINT32	req_opcode	request operation code
void*	param	Opcode associated parameters
UINT32	rsp_opcode	response op-code

Return value:

API_SUCCESS	Success
other value	refers<MS_error.h>

2.5.12.API_RESULT MS_access_cm_set_model_publication

Set the release information associated with the model

2.5.13.API_RESULT MS_access_send_pdu

Send access layer PDU

type	parameter	description
MS_NET_ADDR	src_addr	Source address
MS_NET_ADDR	dst_addr	Destination address
MS_SUBNET_HANDLE	subnet_handle	subnet handle
MS_APPKEY_HANDLE	appkey_handle	Appkey handle
UINT8	ttl	Time to live
UINT32	opcode	Operation code
UCHAR*	data_param	Access layer parameter

type	parameter	description
UINT16	data_length	data length
UINT8	reliable	reliable is true if a reply from the underlying transport layer is required

Return value:

API_SUCCESS	Success
other value	refers<MS_error.h>

2.5.14.MS_access_raw_data

Send a message to the specified address

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	Model handle of node
UINT32	opcode	Opcode of message
MS_NET_ADDR	dst_addr	destination address
MS_APPKEY_HANDLE	appKeyHandle	An encrypted appkey handle is required to send a message
UCHAR*	data_param	content of sent message
UINT16	data_len	Length of sent message
UINT8	reliable	Whether reliable transmission is required True: regardless of whether the unpacking conditions are met, the unpacking processing is performed uniformly False: only if the unpacking conditions are met, the unpacking process can be performed, otherwise it will be processed according to the process of not unpacking

Return value:

API_SUCCESS	Success

2.5.15.API_RESULT MS_generic_onoff_client_send_reliable_pdu

Send the generic onoff command to be answered

type	parameter	description
UINT32	req_opcode	request operation code
void*	param	Opcode associated parameters
UINT32	rsp_opcode	response op-code

Return value:

API_SUCCESS	Success
other value	refers<MS_error.h>

2.5.16.API_RESULT MS_hsl_client_send_reliable_pdu

Send the hsl command to be answered

type	parameter	description
UINT32	req_opcode	request operation code
void*	param	Opcode associated parameters
UINT32	rsp_opcode	response op-code

Return value:

API_SUCCESS	Success
other value	refers<MS_error.h>

2.5.17.API_RESULT MS_access_publish

Publish access layer messages to the publish address associated with the model

type	parameter	description
MS_ACCESS_MODEL_HANDLE*	handle	Access model handle of the message to send
UINT32	opcode	access opcode
UCHAR*	data_param	data packet
UINT16	data_len	length of data packet
UINT8	reliable	MS_TRUE for reliable messages; MS_FALSE for other

Return value:

API_SUCCESS	Success
other value	refers<MS_error.h>

2.5.18.MS_common_reset

Mesh protocol stack reset, network configuration and other information will be reset

Parameter: None

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.19.MS_access_ps_store_all_record

Save MESH configuration message to flash

Parameter: None

Return value:

API_SUCCESS	Success
API_FAILER	Fail

2.5.20.MS_access_ps_store_disable

Turn on or off the mesh message storage function

type	parameter	description
UINT8	enable	1:enable 0: disable

Return value:

API_SUCCESS	Success
API_FAILER	fail

2.5.21.Enable/Disable Relay feature

MS_DISABLE_RELAY_FEATURE	Disable relay
MS_ENABLE_RELAY_FEATURE	Enable relay

2.5.22.Enable/Disable Proxy feature

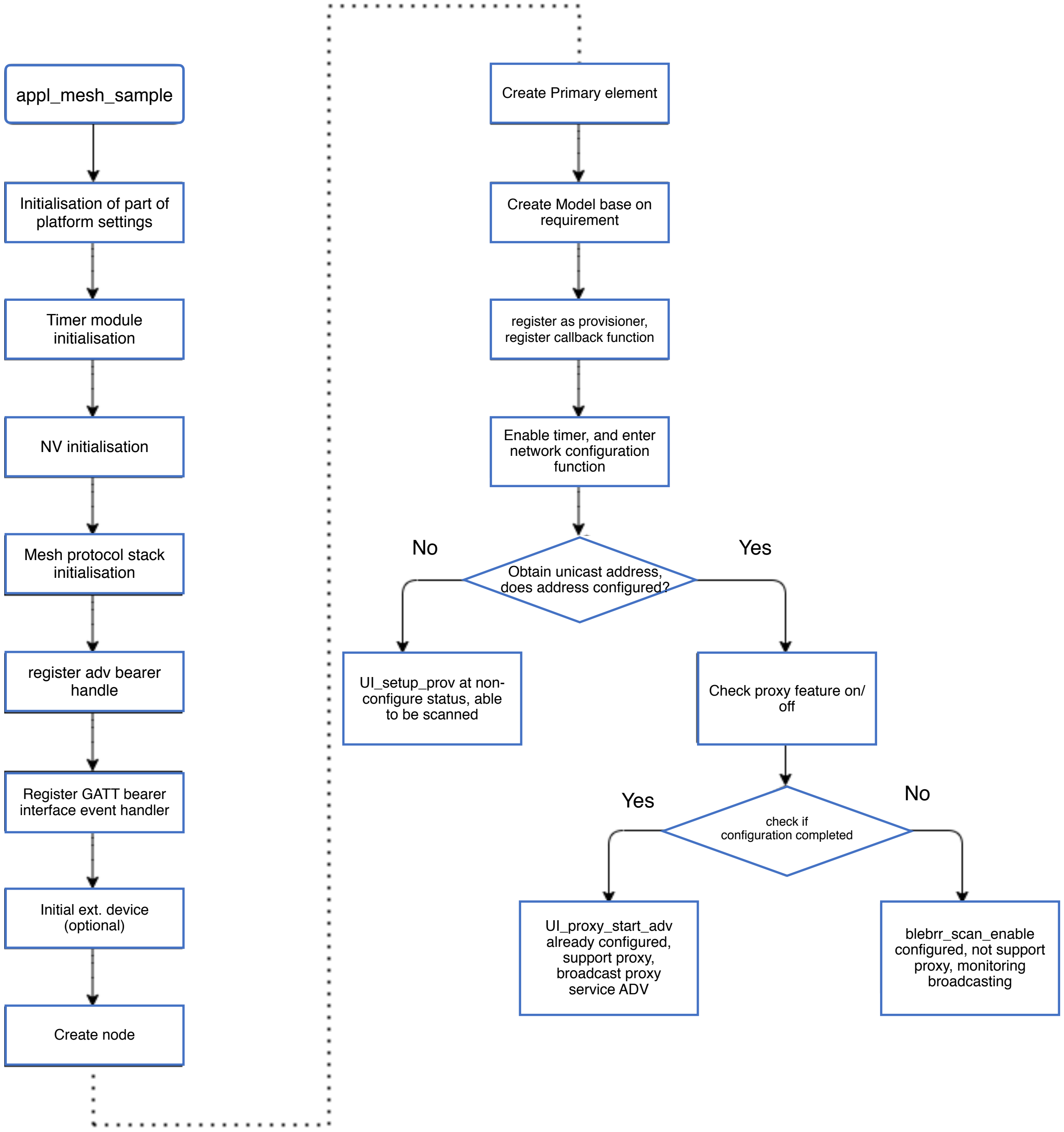
MS_DISABLE_PROXY_FEATURE	disable proxy
MS_ENABLE_PROXY_FEATURE	enable proxy

2.5.23.Enable/Disable Friend feature

MS_DISABLE_FRIEND_FEATURE	disable friend
MS_ENABLE_FRIEND_FEATURE	enable friend

C. Application example

1. Mesh initialisation


appl_mesh_sample
Initialisation of part of platform settings
Timer module initialisation
NV initialisation
Mesh protocol stack initialisation
register adv bearer handle
Register GATT bearer interface event handler
Initial ext. device (optional)
Create node
Create Primary element
Create Model base on requirement
register as provisioner, register callback function
Enable timer, and enter network configuration function
No
Yes
Obtain unicast address, does address configured?
UI_setup_prov at non-configure status, able to be scanned
Check proxy feature on/off
Yes
No
check if configuration completed
UI_proxy_start_adv already configured, support proxy, broadcast proxy service ADV
blebrr_scan_enable configured, not support proxy, monitoring broadcasting

```
void appl_mesh_sample (void)
{
            MS_ACCESS_NODE_ID node_id;
            MS_ACCESS_ELEMENT_DESC element;
            MS_ACCESS_ELEMENT_HANDLE element_handle;
            API_RESULT retval;
            MS_CONFIG* config_ptr;
            #ifdef MS_HAVE_DYNAMIC_CONFIG
            MS_CONFIG config;
            /* Initialize dynamic configuration */
            MS_INIT_CONFIG(config);
            config_ptr = &config;
            #else
```

```
config_ptr = NULL;
#endif /* MS_HAVE_DYNAMIC_CONFIG */
/* Initialize OSAL */
EM_os_init();
/* Initialize Debug Module */
EM_debug_init();
/* Initialize Timer Module */
EM_timer_init();
timer_em_init();
#if defined ( EM_USE_EXT_TIMER )
EXT_cbtimer_init();
ext_cbtimer_em_init();
#endif
/* Initialize utilities */
nvsto_init(NVS_FLASH_BASE1,NVS_FLASH_BASE2);
/* Initialize Mesh Stack */
MS_init(config_ptr);
/* Register with underlying BLE stack */
blebrr_register();
/* Register GATT Bearer Connection/Disconnection Event Hook */
blebrr_register_gatt_iface_event_pl(UI_gatt_iface_event_pl_cb);
/* Enable LED Port */
/* Platform Abstraction Initializations of GPIOs/LEDs etc. */
mesh_model_platform_init_pl();
/* LED ON */
/* LED ON/OFF for BOOT UP Indication Abstraction Call */
mesh_model_device_bootup_ind_pl();
/* Create Node */
retval = MS_access_create_node(&node_id);
/* Register Element */
/**
        TBD: Define GATT Namespace Descriptions from
        https://www.bluetooth.com/specifications/assigned-numbers/gatt-namespace- descriptors
        Using 'main' (0x0106) as Location temporarily.
*/
element.loc = 0x0106;
retval = MS_access_register_element
(
        node_id,
        &element,
        &element_handle
);
if (API_SUCCESS == retval)
{
        /* Register foundation model servers */
        retval = UI_register_foundation_model_servers(element_handle);
}
if (API_SUCCESS == retval)
{
        /* Register Generic OnOff model server */
        retval = UI_register_generic_onoff_model_server(element_handle);
}
#ifdef USE_HSL
if (API_SUCCESS == retval)
{
        /* Register Light Lightness model server */
        retval = UI_register_light_hsl_model_server(element_handle);
}
```

```
    #endif
    #ifdef USE_CTL
    if (API_SUCCESS == retval)
    {
        /* Register Light Lightness model server */
        retval = UI_register_light_ctl_model_server(element_handle);
    }
    #endif
    #ifdef USE_SCENE
    if (API_SUCCESS == retval)
    {
        /* Register Light Scene model server */
        retval = UI_register_scene_model_server(element_handle);
    }
    #endif
    #ifdef USE_VENDORMODEL
    if (API_SUCCESS == retval)
     {
        /* Register Vendor Defined model server */
        retval = UI_register_vendor_defined_model_server(element_handle);
    }
    #endif
    if (API_SUCCESS == retval)
    {
        /* Initialize model states */
        UI_model_states_initialization();
    }
    /* Configure as provisionee/device */
    UI_register_prov();
    #if (CFG_HEARTBEAT_MODE)
    UI_register_heartbeat();
    #endif
    /**
    Set Scan Response Data Before Starting Provisioning.This is optional/additional set of Data t
    hat the device can set to enhance the User Experience. For Example, set a specific device
    name or URL as part of the Scan Response Data when awaiting connections over GATT bearer.
    */
    UI_set_brr_scan_rsp_data();
    APP_config_server_CB_init(UI_app_config_server_callback);
    uint32 address = VENDOR_PRODUCT_MAC_ADDR;
    hal_flash_read(address ++,&UI_lprov_device.uuid[10],1);
    hal_flash_read(address ++,&UI_lprov_device.uuid[11],1);
    hal_flash_read(address ++,&UI_lprov_device.uuid[12],1);
    hal_flash_read(address ++,&UI_lprov_device.uuid[13],1);
    hal_flash_read(address ++,&UI_lprov_device.uuid[8],1);
    hal_flash_read(address ++,&UI_lprov_device.uuid[9],1);
    EM_start_timer (&thandle, 3, timeout_cb, NULL, 0);
    return;
}

void timeout_cb (void* args, UINT16 size)
{
    thandle = EM_TIMER_HANDLE_INIT_VAL;
    UI_sample_reinit();
}
```

```
void UI_sample_reinit(void)
{
            API_RESULT retval;
            MS_NET_ADDR addr;
            UCHAR
            UCHAR
            UCHAR
            retval
            is_prov_req = MS_TRUE;
            retval = MS_access_cm_get_primary_unicast_address(&addr);
            if (API_SUCCESS == retval)
            {
                        if (MS_NET_ADDR_UNASSIGNED != addr)
                        {
                                /* Set Provisioning is not Required */
                                is_prov_req = MS_FALSE;
                        }
            }

            // MS_access_cm_set_transmit_state(MS_RELAY_TX_STATE, (8<<3)|2);
            MS_access_cm_set_transmit_state(MS_NETWORK_TX_STATE, (8<<3)|3);

            if (MS_TRUE == is_prov_req)
            {
               /* Start Provisioning over GATT here */
               /**
                        setup <role:[1 - Device, 2 - Provisioner]> <bearer:[1 - Adv, 2 - GATT]
               */
               role = PROV_ROLE_DEVICE;
               brr = PROV_BRR_GATT; PROV_BRR_ADV为ADV配网PROV_BRR_GATT则为GATT直连
               printf("Bearer type = 0x%02X(Bit0-adv, Bit1-GATT)\r\n", brr);
             //  UI_prov_brr_handle = brr;
              /**
                        Setting up an Unprovisioned Device over GATT
             */
              LIGHT_ONLY_RED_ON;
              blebrr_prov_started = MS_FALSE;
              UI_setup_prov(role, brr);
//            UI_prov_bind(brr, 0x00);
               //ms_access_ps_store(MS_PS_RECORD_SEQ_NUMBER);
              CONSOLE_OUT("\r\n Setting up as an Unprovisioned Device\r\n");
            }
            else
            {
               /* Fetch PROXY feature state */ MS_access_cm_get_features_field(&state,
                  MS_FEATURE_PROXY);
              /**
              Check if the Device is Configured. If not Configured, Start Proxy ADV. If it is Configured,
              Check if the Proxy Feature is Enabled. If not enabled, then Do Nothing!
              If it is, Start Proxy ADV.
              */
```

```
        if (API_SUCCESS == UI_sample_check_app_key())
        {
          UI_sample_get_device_key();
           if (MS_ENABLE == state) {
           light_blink_set(LIGHT_GREEN, LIGHT_BLINK_FAST,5);
           //for silab 2.0.0 app use NODE ID
          CONSOLE_OUT("\r\n Provisioned Device - Starting Proxy with NODE ID on Subnet 0x0000!\r\n");
           UI_proxy_start_adv(0x0000, MS_PROXY_NODE_ID_ADV_MODE);
           #if (CFG_HEARTBEAT_MODE)
           if(ms_provisioner_addr != 0)
           {
             printf("sub ms_provisioner_addr 0x%04X\n",ms_provisioner_addr);
             UI_trn_set_heartbeat_subscription(ms_provisioner_addr);
            }
           #endif
        }
        else
        {
          light_blink_set(LIGHT_GREEN, LIGHT_BLINK_SLOW,3);
          MS_brr_bcast_end(BRR_BCON_TYPE_PROXY_NODEID, BRR_BCON_ACTIVE);
         #if (CFG_HEARTBEAT_MODE)
         if(ms_provisioner_addr != 0)
         {
           printf("sub ms_provisioner_addr 0x%04X\n",ms_provisioner_addr);
           UI_trn_set_heartbeat_subscription(ms_provisioner_addr);
          }
          #endif
           CONSOLE_OUT("\r\n Provisioned Device!!!\r\n");
           /**
           Do Nothing!
           Already Scaning is Enabled at Start Up */
           blebrr_scan_enable();
           }
        }
        else
        {

        light_blink_set(LIGHT_BLUE, LIGHT_BLINK_FAST,5);
        //for silab 2.0.0 app use NODE ID
        if(UI_prov_brr_handle == PROV_BRR_GATT)
          {
             UI_proxy_start_adv(0x0000, MS_PROXY_NODE_ID_ADV_MODE);
           }
         }
        }
        if((ms_iv_index.iv_expire_time!=0)&&(ms_iv_index.iv_expire_time!=0xffffffff))
        {
           MS_net_start_iv_update_timer(ms_iv_index.iv_update_state,MS_TRUE);
        }
}
```

2. Vendor model status report

```
API_RESULT phyplusmodel_server_cb
(
        /* IN */ MS_ACCESS_MODEL_HANDLE*                handle,
        /* IN */ MS_NET_ADDR                            saddr,
        /* IN */ MS_NET_ADDR                            daddr,
        /* IN */ MS_SUBNET_HANDLE                       subnet_handle,
        /* IN */ MS_APPKEY_HANDLE                       appkey_handle,
        /* IN */ UINT32                                 opcode,
        /* IN */ UCHAR*                                 data_param,
        /* IN */ UINT16                                 data_ len
)
{
        MS_ACCESS_MODEL_REQ_MSG_CONTEXT             req_context;
        MS_ACCESS_MODEL_REQ_MSG_RAW                 req_raw;
        MS_ACCESS_MODEL_REQ_MSG_T                   req_type;
        MS_ACCESS_MODEL_EXT_PARAMS*                 ext_params_p
        MS_ACCESS_PHYPLUSMODEL_STATE_PARAMS         state_params
        UINT16 marker;
        API_RESULT retval;
        retval = API_SUCCESS;
        ext_params_p = NULL;
        marker = 0;

        req_context.handle = *handle;           // request content
        req_context.saddr = saddr;
        req_context.daddr = daddr;
        req_context.subnet_handle = subnet_handle;
        req_context.appkey_handle = appkey_handle;

        req_raw.opcode = opcode;                //request parameter
        req_raw.data_param = data_param;
        req_raw.data_len = data_len;
        state_params.phyplusmode_param = NULL;

        switch(opcode)          //Execute the corresponding function according to the customized opcode
        {
        case MS_ACCESS_PHYPLUSMODEL_GET_OPCODE:
        {
//              printf(
//              "MS_ACCESS_PHY_MODEL_GET_OPCODE\n");
                MODEL_OPCODE_HANDLER_CALL(vendor_example_get_handler);
                marker = 1;
                MS_UNPACK_LE_2_BYTE(&state_params.phyplusmode_type, data_param+marker);
                marker += 2;
                /* Get Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_GET;
                req_type.to_be_acked = 0x01;
                /* Assign reqeusted state type to the application */
        }
        break;
        case MS_ACCESS_PHYPLUSMODEL_SET_OPCODE:
        case MS_ACCESS_PHYPLUSMODEL_SET_UNACKNOWLEDGED_OPCODE:
         {
//              printf( “MS_ACCESS_PHY_MODEL_SET_OPCODE\n”);
                MODEL_OPCODE_HANDLER_CALL(vendor_example_set_handler);
                marker = 1;
                MS_UNPACK_LE_2_BYTE(&state_params.phyplusmode_type, data_param+marker);
                marker += 2;
                state_params.phyplusmode_param = &data_param[marker];
                /* Set Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_SET;
                if(MS_ACCESS_PHYPLUSMODEL_SET_OPCODE == opcode)
                {
                                req_type.to_be_acked = 0x01;
                }
                else
                {
                                req_type.to_be_acked = 0x00;
                }
        }
        break;

        case MS_ACCESS_PHYPLUSMODEL_STATUS_OPCODE:
        {
//              printf( "MS_ACCESS_PHY_MODEL_STATUS\n");
                MODEL_OPCODE_HANDLER_CALL(vendor_example_status_handler);
                 /* Set Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_OTHERS;
                req_type.to_be_acked = 0x00;
        }
        break;

        case MS_ACCESS_PHYPLUSMODEL_CONFIRMATION_OPCODE:
        {
//              printf( "MS_ACCESS_PHY_MODEL_CONFIRMATION\n");
                MODEL_OPCODE_HANDLER_CALL(vendor_example_confirmation_handler);
                 /* Set Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_OTHERS;
                req_type.to_be_acked = 0x00;
        }
        break;

        case MS_ACCESS_PHYPLUSMODEL_WRITECMD_OPCODE:
        {
                printf( "MS_ACCESS_PHY_MODEL_WRITECMD_OPCODE\n");
                marker = 1;
                MS_UNPACK_LE_2_BYTE(&state_params.phyplusmode_type, data_param+marker);
                marker += 2;
                state_params.phyplusmode_param = &data_param[marker];
                /* Set Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_OTHERS;
                req_type.to_be_acked = 0x00;
        }
        break;
        case MS_ACCESS_PHYPLUSMODEL_NOTIFY_OPCODE:
        {
//              printf( "MS_ACCESS_PHY_MODEL_NOTIFY_OPCODE\n");
                state_params.phyplusmode_type = MS_STATE_PHYPLUSMODEL_NOTIFY_T;
                marker = 1;
                state_params.phyplusmode_param = &data_param[marker];
                /* Set Request Type */
                req_type.type = MS_ACCESS_MODEL_REQ_MSG_T_OTHERS;
                req_type.to_be_acked = 0x00;
        }
        break;

        default:
                printf("MS_ACCESS_PHYPLUSMODEL_NONE_OPCODE\n");
                break;
        }


        /* Application callback */
        if (NULL != phyplusmodel_server_UI_cb)
        {
                phyplusmodel_server_UI_cb(&req_context, &req_raw, &req_type, &state_params, ext_params_p);
        }
        return retval;
}
```

3. Generic On/Off status report

```
static API_RESULT UI_generic_onoff_server_cb
(
        /* IN */ MS_ACCESS_MODEL_REQ_MSG_CONTEXT*       ctx,
        /* IN */ MS_ACCESS_MODEL_REQ_MSG_RAW*           msg_raw,
        /* IN */ MS_ACCESS_MODEL_REQ_MSG_T*             req_type,
        /* IN */ MS_ACCESS_MODEL_STATE_PARAMS*          state_params,
        /* IN */ MS_ACCESS_MODEL_EXT_PARAMS*            ext_params,
)
{
        MS_STATE_GENERIC_ONOFF_STRUCT                   param;
        MS_ACCESS_MODEL_STATE_PARAMS                    current_state_params;
        API_RESULT                                      retval;
        retval = API_SUCCESS;
        /* Check message type */
        if (MS_ACCESS_MODEL_REQ_MSG_T_GET == req_type->type)
        {
                CONSOLE_OUT("[GENERIC_ONOFF] GET Request.\n");
                UI_generic_onoff_model_state_get(state_params->state_type, 0, &param, 0);
                current_state_params.state_type = state_params->state_type;
                current_state_params.state = &param;
                /* Using same as target state and remaining time as 0 */
        }
        else if (MS_ACCESS_MODEL_REQ_MSG_T_SET == req_type->type)
        {
                CONSOLE_OUT("[GENERIC_ONOFF] SET Request.\n");
                retval = UI_generic_onoff_model_state_set(state_params->state_type, 0,
                (MS_STATE_GENERIC_ONOFF_STRUCT*)state_params->state, 0);
                                current_state_params.state_type = state_params->state_type;
                                current_state_params.state =
                                (MS_STATE_GENERIC_ONOFF_STRUCT*)state_params->state;
        }


        /* See if to be acknowledged */
        if (0x01 == req_type->to_be_acked)
         {
                CONSOLE_OUT("[GENERIC_ONOFF] Sending Response.\n");
                /* Parameters: Request Context, Current State, Target State (NULL: to be ignored),
Remaining Time (0: to be ignored), Additional Parameters (NULL: to be ignored) */
                retval = MS_generic_onoff_server_state_update(ctx, &current_state_params, NULL, 0,
NULL);
        }
        return retval;
}
```