



■ PHY622X

MESH 开发指南

Version 1.0

Author: PHY+ SW

Security: Public

Date: 2021.3

PhyPlus

Copyright © 2021 Phyplus Microelectronics Limited All rights reserved.
Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.



Revision History

Revision	Author	Date	Description
v1.0	PHY+ SW	2021.3	First Edition

目录

1	简介	1
1.1	mesh 协议栈	1
1.2	消息处理流程	2
1.3	mesh 配置	4
2	工程及 api 介绍	5
2.1	工程介绍	5
2.1.1	ethermind	5
2.1.2	mesh samples & application	6
2.2	公共模块定义	6
2.2.1	Mesh Sample 介绍	7
2.2.2	定义 model	7
2.3	api 介绍	8
2.3.1	UI_health_server_cb	8
2.3.2	UI_register_foundation_model_servers	8
2.3.3	UI_generic_onoff_model_states_initialization	8
2.3.4	UI_generic_onoff_model_state_get	9
2.3.5	API_RESULT MS_access_register_element	9
2.3.6	API_RESULT UI_register_foundation_model_servers	9
2.3.7	API_RESULT UI_register_generic_onoff_model_server	10
2.3.8	UI_generic_onoff_server_cb	10
2.3.9	UI_light_hsl_model_states_initialization	10
2.3.10	UI_light_hsl_model_state_get	11
2.3.11	UI_light_hsl_model_state_set	11
2.3.12	UI_light_hsl_server_cb	11
2.3.13	UI_light_ctl_model_states_initialization	12
2.3.14	UI_light_ctl_model_state_get	12
2.3.15	UI_light_ctl_model_state_set	12
2.3.16	UI_light_ctl_server_cb	13
2.3.17	UI_register_light_ctl_model_server	13
2.3.18	UI_vendor_defined_model_states_initialization	14
2.3.19	UI_vendor_example_model_state_get	14
2.3.20	UI_vendor_example_model_state_set	14
2.3.21	UI_phy_model_server_cb	15
2.3.22	UI_register_vendor_defined_model_server	15
2.3.23	UI_model_states_initialization	15
2.3.24	API_RESULT MS_access_cm_set_transmit_state	16
2.3.25	API_RESULT MS_access_cm_set_features_field	16
2.3.26	API_RESULT MS_access_bind_model_app	16
2.3.27	MS_proxy_server_adv_start	17
2.3.28	MS_prov_setup	17
2.3.29	API_RESULT MS_prov_bind	18
2.3.30	API_RESULT MS_prov_register	18

2.3.31	API_RESULT MS_proxy_server_adv_stop (void)	19
2.3.32	API_RESULT MS_proxy_register	19
2.3.33	API_RESULT MS_access_cm_get_primary_unicast_address	19
2.3.34	API_RESULT MS_access_reply.....	19
2.3.35	API_RESULT MS_scene_server_init.....	20
2.3.36	API_RESULT MS_scene_client_init.....	21
2.3.37	API_RESULT MS_light_hsl_server_init	21
2.3.38	API_RESULT MS_light_hsl_client_init.....	22
2.3.39	API_RESULT MS_light_ctl_server_init.....	22
2.3.40	API_RESULT MS_light_ctl_client_init	23
2.3.41	API_RESULT MS_generic_onoff_server_init	23
2.3.42	API_RESULT MS_health_server_init.....	23
2.3.43	API_RESULT MS_health_client_init.....	24
2.3.44	API_RESULT MS_access_register_model	24
2.3.45	UI_vendor_defined_model_states_initialization.....	25
2.3.46	UI_vendor_example_model_state_get.....	25
2.3.47	UI_vendor_example_model_state_set.....	25
2.3.48	UI_phy_model_server_cb.....	26
2.3.49	UI_register_vendor_defined_model_server.....	26
2.3.50	UI_model_states_initialization.....	26
2.4	Provision 接口	26
2.4.1	Unprovision beacon uuid	27
2.4.2	UI_provcfg_complete_timeout_handler.....	27
2.4.3	UI_prov_callback.....	28
2.4.4	UI_register_prov.....	28
2.4.5	UI_proxy_start_adv.....	28
2.4.6	UI_proxy_callback	29
2.4.7	UI_setup_prov.....	29
2.4.8	UI_register_proxy.....	29
2.4.9	UI_set_brr_scan_rsp_data	29
2.4.10	UI_gatt_iface_event_pl_cb	30
2.4.11	UI_sample_binding_app_key.....	30
2.4.12	vm_subscriptiong_binding_cb	30
2.4.13	vm_subscriptiong_add.....	30
2.4.14	vm_subscriptiong_delete.....	31
2.4.15	UI_app_config_server_callback	31
2.4.16	appl_mesh_sample	32
2.4.17	UI_sample_get_net_key	32
2.4.18	UI_sample_get_device_key	32
2.4.19	UI_sample_check_app.....	32
2.4.20	UI_sample_reinit.....	32
2.5	其他常用 api	33
2.5.1	MS_access_cm_get_primary_unicast_address	33
2.5.2	MS_access_get_element_handle	33
2.5.3	MS_access_get_model_handle.....	33

2.5.4	MS_access_get_model_handle.....	34
2.5.5	MS_access_get_element_handle	34
2.5.6	MS_access_get_model_handle.....	35
2.5.7	MS_access_get_model_handle.....	35
2.5.8	MS_access_get_element_handle	35
2.5.9	MS_access_get_model_handle.....	36
2.5.10	MS_access_get_model_handle	36
2.5.11	API_RESULT MS_config_client_send_reliable_pdu.....	37
2.5.12	API_RESULT MS_access_cm_set_model_publication	37
2.5.13	API_RESULT MS_access_send_pdu	37
2.5.14	MS_access_raw_data.....	38
2.5.15	API_RESULT MS_generic_onoff_client_send_reliable_pdu	38
2.5.16	API_RESULT MS_hsl_client_send_reliable_pdu.....	39
2.5.17	API_RESULT MS_access_publish	39
2.5.18	MS_common_reset.....	40
2.5.19	MS_access_ps_store_all_record	40
2.5.20	MS_access_ps_store_disable.....	40
2.5.21	开启/关闭 relay 功能	40
2.5.22	开启/关闭 proxy 功能	41
2.5.23	开启/关闭 friend 功能.....	41
3	应用实例	42
3.1	mesh 初始化	42
3.2	vendor model 状态上报.....	49
3.3	Generic onoff 状态上报	52

图表目录

图 1:	整体协议栈框架.....	1
图 2:	Mesh 协议栈框架	1
图 3:	Message Flow Diagram – Upper Layers.....	2
图 4:	Message Flow Diagram – Lower Layers.....	3
图 5:	配置协议框架	4
图 6:	工程目录内容	5
图 7:	ethermind 目录内容.....	5
图 8:	samples 和 application 目录内容.....	6
图 9:	mesh 初始化流程图.....	42

1 简介

本文档用于 PHY622X Mesh 的介绍以及使用方法，它有助于您了解和理解本公司 Mesh 提供的组件，样例的使用方法，并且帮助您如何从提供的样例开始进行 BLE Mesh 的开发。

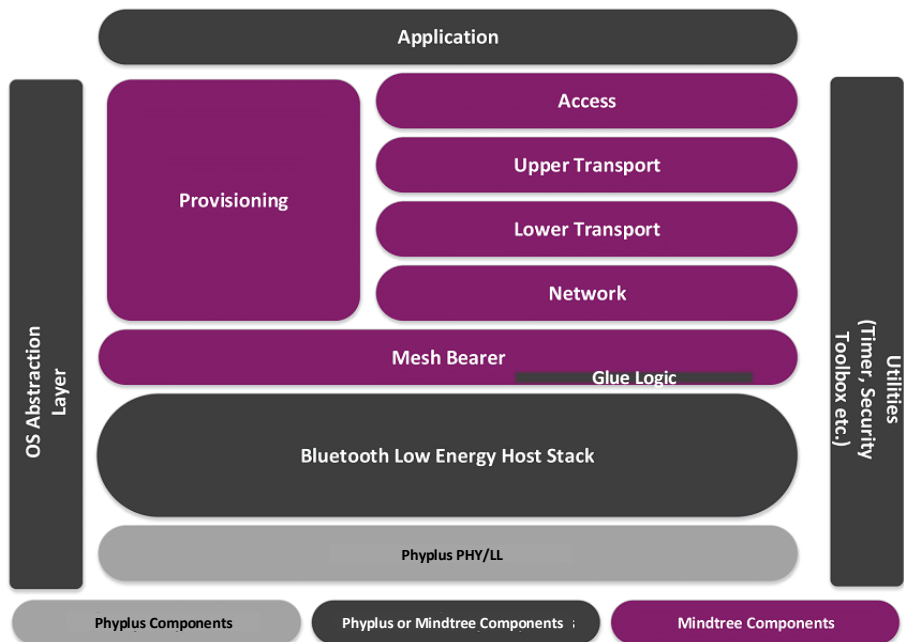


图 1：整体协议栈框架

1.1 mesh 协议栈

这一协议栈建立在低功耗蓝牙技术之上。下图描绘了协议栈的层级。

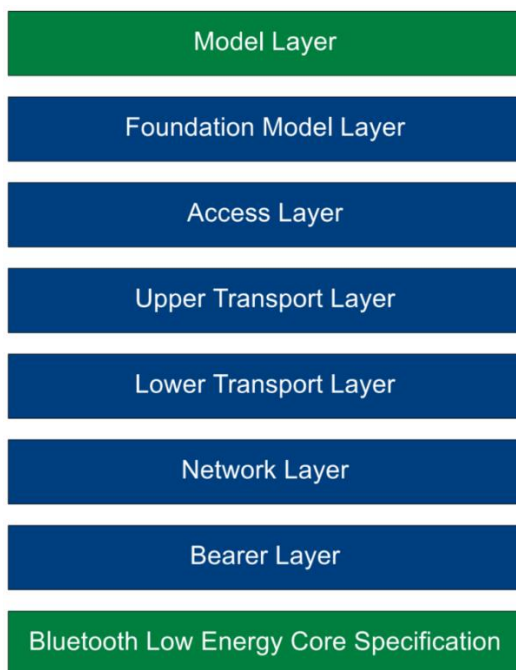


图 2：Mesh 协议栈框架

- 模型层（**Model Layer**）：模型层与模型等的实施、以及诸如行为、消息、状态等的实施有关。
- 基础模型层（**Foundation Model Layer**）：基础模型层负责实现与 mesh 网络配置和管理相关的模型。
- 访问层（**Access Layer**）：负责应用数据的格式、定义并控制上层传输层中执行的加密和解密过程，并在将数据转发到协议栈之前，验证接收到的数据是否适用于正确的网络和应用
- 上层传输层（**Upper Transport Layer**）：负责对接入层进出的应用数据进行加密、解密和认证。它还负责称为“传输控制消息”（**transport control messages**）这一特殊的消息，包括与“friendship”相关的心跳和消息。
- 底层传输层（**Lower Transport Layer**）：在需要之时，底层传输层能够处理 PDU 的分段和重组。
- 网络层（**Network Layer**）：网络层定义了各种消息地址类型和网络消息格式。中继和代理行为通过网络层实施。
- 承载层（**Bearer Layer**）：承载层定义了如何使用底层低功耗堆栈传输 PDU。目前定义了两个承载层：广播承载层（**Advertising Bearer**）和 GATT 承载层。

1.2 消息处理流程

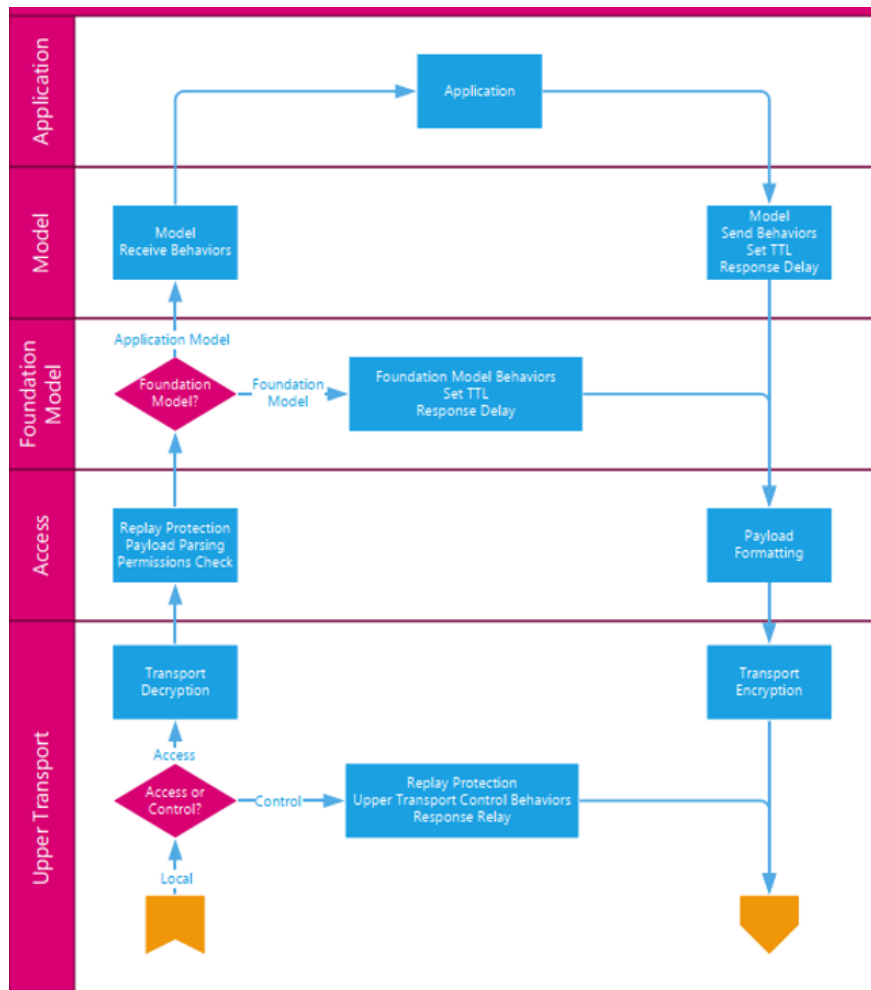


图 3: Message Flow Diagram – Upper Layers

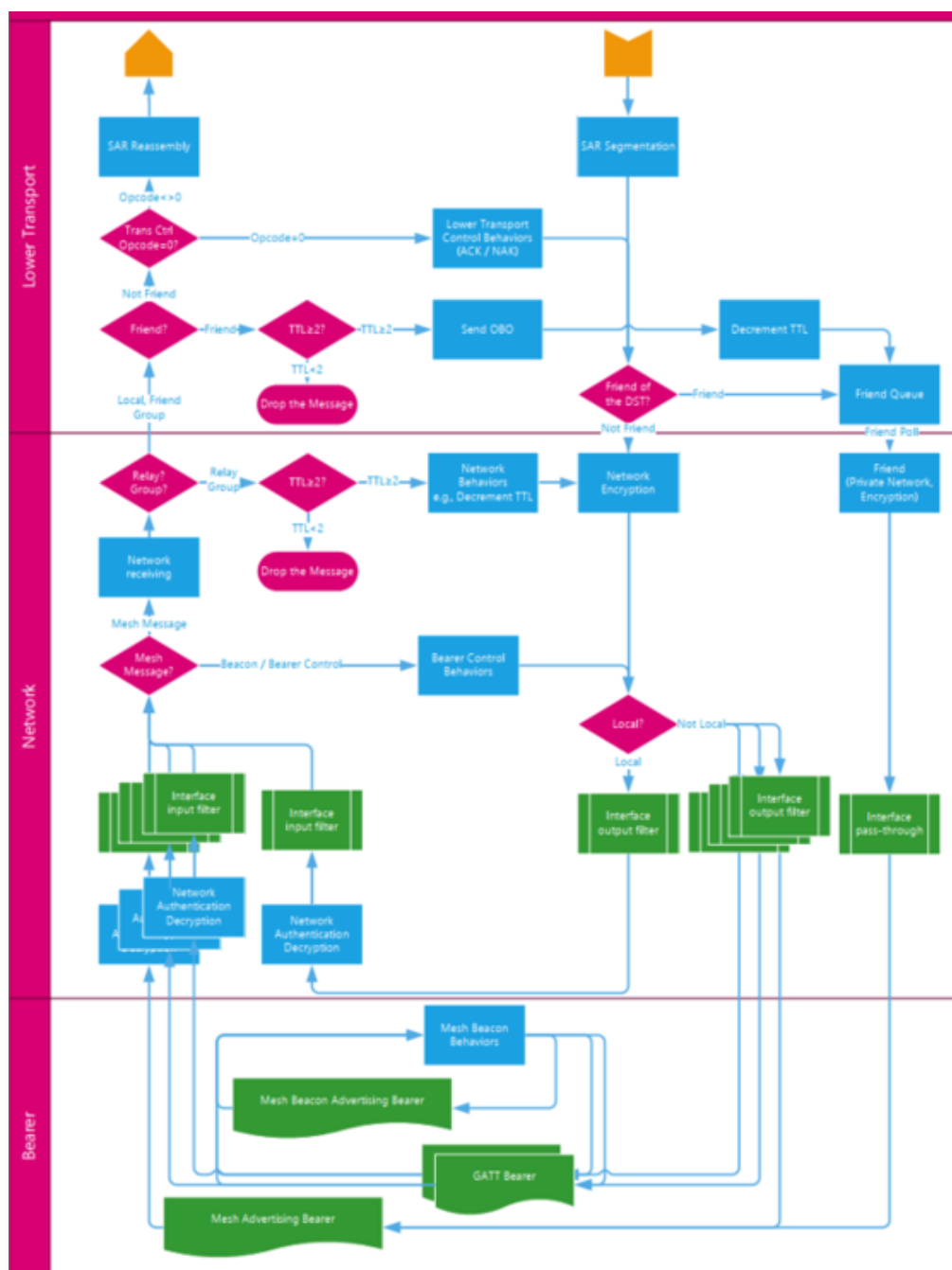


图 4: Message Flow Diagram – Lower Layers

Mesh 消息传输流程如上两图所示，mesh 消息从 bear 层通过 ADV/GATT 进入之后，在网络层解码通过接口输入过滤器之后，如果是中继或者代理消息则在网络层实现，非中继消息会进入底层传输层进行拆分和重组之后进入上层传输层进行传输解密，在访问层经过合法性检查发送给基础模型层，最后通过模型层的实施来实现。

发送消息时则通过模型层实例发送，经过访问层定义数据格式，在上层传输层加密，传入底层传输层进行拆包与组包，在网络层进行加密，通过接口输出过滤器进入承载层之后输出。

1.3 mesh 配置

配置是将未入网的设备添加到 mesh 网络的过程。由配网器为未配网设备提供配置数据使其入网。从而使其成为 mesh 节点。发放数据包括网络密钥、当前 IV 索引和单播每个元素的地址。

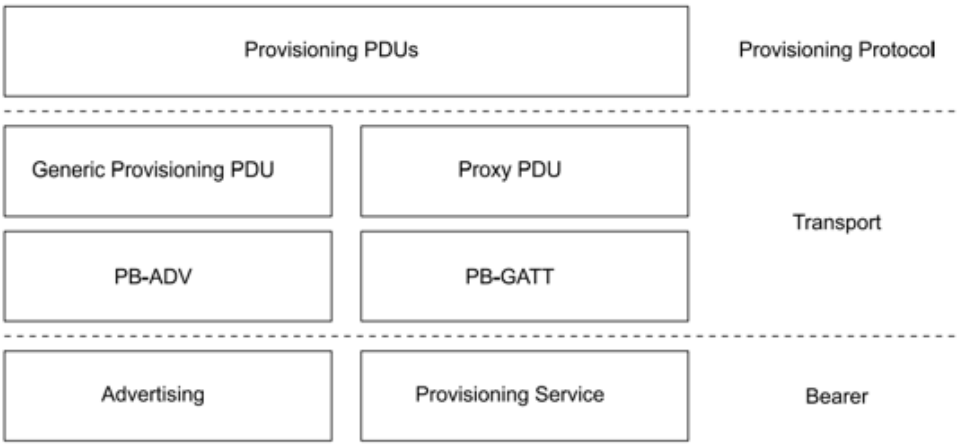


图 5：配置协议框架

设备的配置是使用发送配置 pdu 的配置协议完成的。配置 pdu 通过通用配置层传输到未配置的设备。这一层定义了如何将配置 pdu 作为可以分割和重新组装的事务进行处理。这些处理是通过一个配置承载层发送的。配置承载层定义如何建立会话，以便将来自通用配置层的事务交付到单个设备。最后，配置体系结构的底部是承载层。

2 工程及 api 介绍

2.1 工程介绍



图 6: 工程目录内容

2.1.1 ethermind

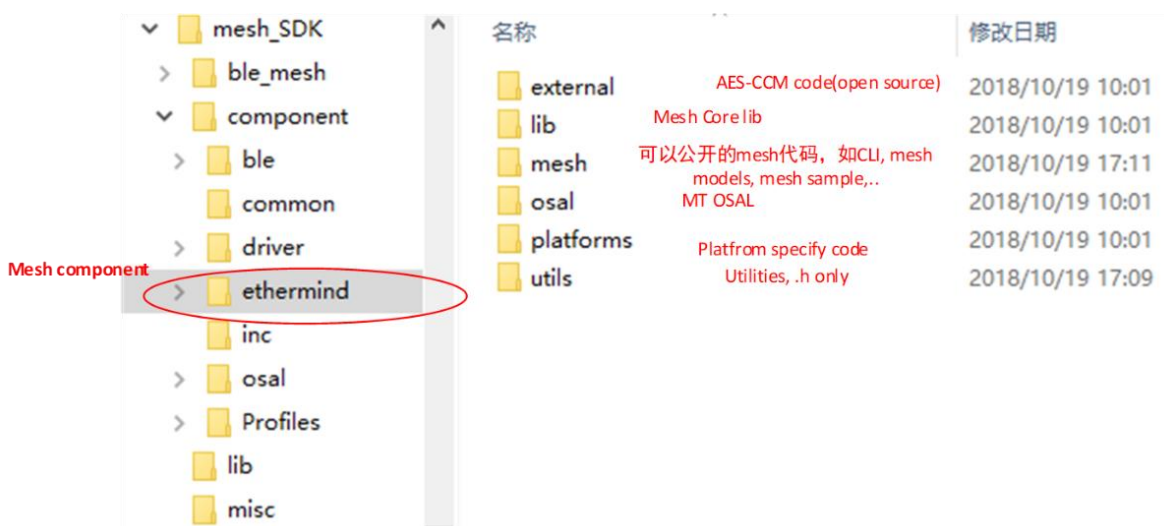


图 7: ethermind 目录内容

2.1.2 mesh samples & application

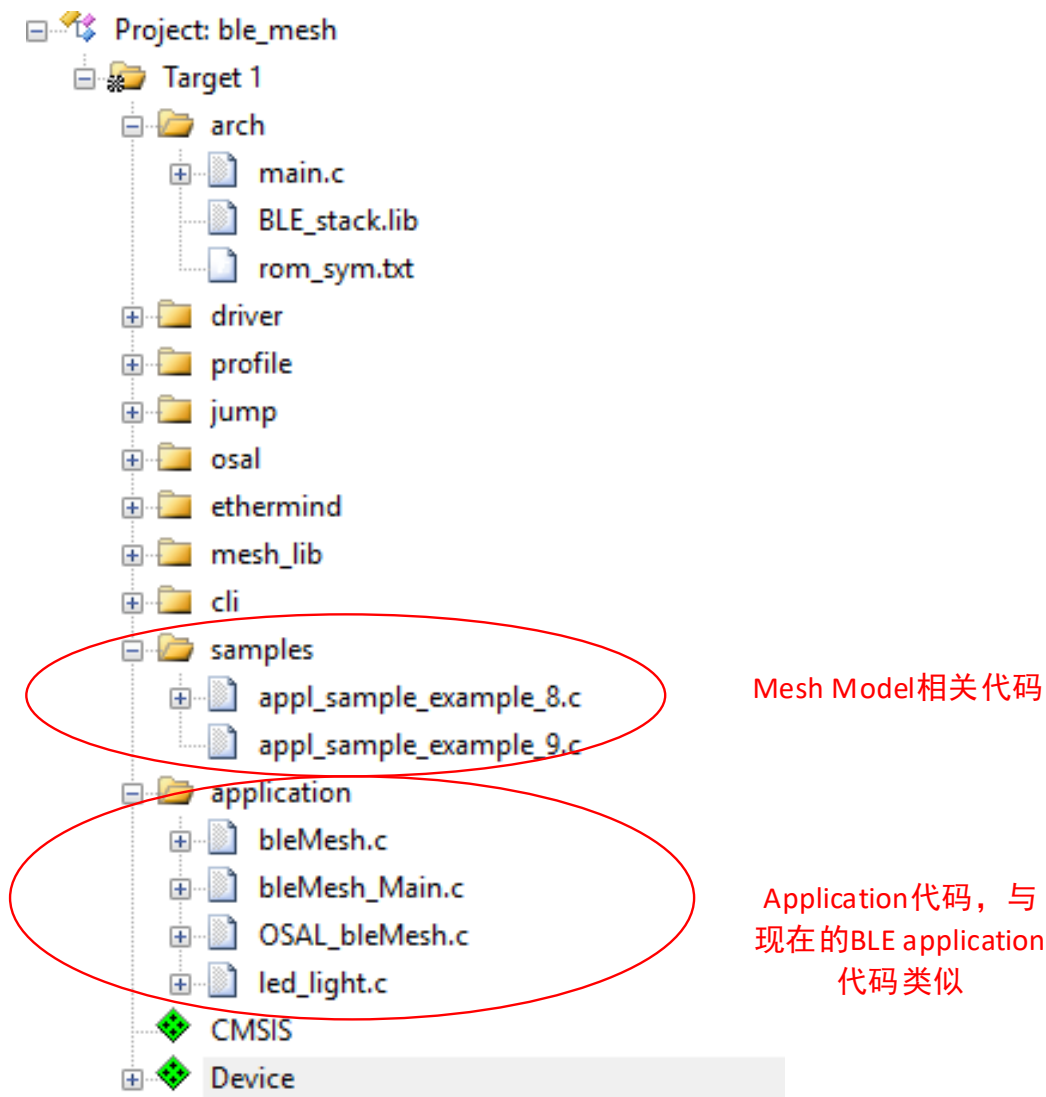


图 8: samples 和 application 目录内容

2.2 公共模块定义

一些公共的模块定义与其他 sdk demo 类似，以下是 mesh 方面的定义。

名称	值	说明
OSAL_CBTIMER_NUM_TASKS	1	使用内部 call back timer 的个数，目前只支持 1 个，且不能更改，mesh 里面配置使用的 timer 都是调用内部 callback timer
CFG_HEARTBEAT_MODE	0	关闭 heartbeat 功能
CFG_HEARTBEAT_MODE	1	打开 heartbeat 功能

2.2.1 Mesh Sample 介绍

PHY62XX Mesh 内部接口都在 lib 库中，常用的 lib 为 libethermind_ecdh.lib、libethermind_mesh_core.lib、libethermind_mesh_models.lib 和 libethermind_utils.lib；功能如下：

1. libethermind_ecdh.lib：跟 ecdh 相关，目前 sdk 没有使用
2. libethermind_mesh_core.lib：mesh 协议栈相关；provision、config 和 message 处理都在这里进行
3. libethermind_mesh_models.lib：mesh model 相关；目前使用的 on/off 等 model 的实现都在这里处理
4. libethermind_utils.lib：mesh 存储相关

除了 lib 之外，用户接触和更改的文件一般在 appl_sample_mesh_XXX.c，本章节重点介绍 sample 相关的接口和定义（以常用的 mesh_light 为例）。

2.2.2 定义 model

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

Vendormodel 如果使能的话，会自动使能 easy bonding（目前我们使用的 sdk 是这样考虑的，配合 Phy mesh app 使用）。

2.3 api 介绍

2.3.1 UI_health_server_cb

- Health server 的回调函数

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	model handle
UINT8	event_type	event 的类型
UINT8*	event_param	event 的参数内容
UINT16	param_len	参数长度

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.2 UI_register_foundation_model_servers

- 注册 foundation model server

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.3 UI_generic_onoff_model_states_initialization

- Generic on/off model 状态初始化
无。
- 返回值
无。

2.3.4 UI_generic_onoff_model_state_get

- Generic on/off model 状态获取

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.5 API_RESULT MS_access_register_element

- 创建 Primary Element

类型	参数名	说明
MS_ACCESS_NODE_ID	node_id	节点 id, 初始值为 0
MS_ACCESS_ELEMENT_DESC*	element	指向需要注册到节点的元素描述符的元素指针
MS_ACCESS_ELEMENT_HANDLE*	element_handle	引用新注册元素的元素句柄标识符

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.6 API_RESULT UI_register_foundation_model_servers

- 创建 Foundation Model

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	引用新注册元素的元素句柄标识符

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.3.7 API_RESULT UI_register_generic_onoff_model_server

- 注册 Generic Onoff Model server

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	引用新注册元素的元素句柄标识符

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.3.8 UI_generic_onoff_server_cb

- Generic on/off model 回调函数

类型	参数名	说明
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	On/off 消息的上下文内容
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	原始消息
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	消息类型
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	消息内容
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	其他参数

- 返回值

API_SUCCESS	成功
-------------	----

API_FAILER 失败

2.3.9 UI_light_hsl_model_states_initialization

- Generic light hsl model 状态初始化

无。

- 返回值

无。

2.3.10 UI_light_hsl_model_state_get

- Generic light hsl model 状态获取

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.11 UI_light_hsl_model_state_set

- Generic light hsl model 状态设置

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.12 UI_light_hsl_server_cb

- Generic light hsl model 回调函数

类型	参数名	说明
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	On/off 消息的上下文内容
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	原始消息

MS_ACCESS_MODEL_REQ_MSG_T*	req_type	消息类型
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	消息内容
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	其他参数

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.13 UI_light_ctl_model_states_initialization

- Generic light ctl model 状态初始化
无。
- 返回值
无。

2.3.14 UI_light_ctl_model_state_get

- Generic light ctl model 状态获取

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.15 UI_light_ctl_model_state_set

- Generic light ctl model 状态设置

类型	参数名	说明
UINT16	state_t	State 类型

UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.16 UI_light_ctl_server_cb

- Generic light ctl model 回调函数

类型	参数名	说明
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	On/off 消息的上下文内容
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	原始消息
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	消息类型
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	消息内容
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	其他参数

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.17 UI_register_light_ctl_model_server

- 注册 generic light ctl model server

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE	handle	model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.18 UI_vendor_defined_model_states_initialization

- vendor model 状态初始化
无。
- 返回值
无。

2.3.19 UI_vendor_example_model_state_get

- vendor model 状态获取

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.20 UI_vendor_example_model_state_set

- Generic light ctl model 状态设置

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.21 UI_phy_model_server_cb

- vendor model 回调函数

类型	参数名	说明
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	On/off 消息的上下文内容
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	原始消息
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	消息类型
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	消息内容
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	其他参数

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.22 UI_register_vendor_defined_model_server

- 注册 vendor model server

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE	handle	model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.23 UI_model_states_initialization

- mesh model 所有状态初始化
无。
- 返回值
无。

2.3.24 API_RESULT MS_access_cm_set_transmit_state

- 设置网络/中继传输状态

类型	参数名	说明
UINT8	tx_state_type	传输状态类型(网络或中继)
UINT8	tx_state	复合状态(3 位 Tx 计数, 5 位 Tx 间隔步长)

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.25 API_RESULT MS_access_cm_set_features_field

- 使能或者禁用一个特性

类型	参数名	说明
UINT8	enable	使能或禁用
UINT8	tx_state	中继, 代理, 友谊, 低功耗四种特性

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.26 API_RESULT MS_access_bind_model_app

- 用 appkey 绑定模型

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE	model_handle	识别模型的模型句柄
UINT16	appkey_index	标识 appkey 的全局索引

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.27 MS_proxy_server_adv_start

- 使代理服务器开启可连接的无定向广告

类型	参数名	说明
MS_SUBNET_HANDLE	subnet_handle	代理服务器所在的子网句柄
UCHAR	proxy_adv_mode	代理广播的模式，两种模式 MS_PROXY_NET_ID_ADV_MODE / MS_PROXY_NODE_ID_ADV_MODE

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.28 MS_prov_setup

- 通过指定角色，承载和创建内容来配置设备，使其处于可供应状态

类型	参数名	说明
PROV_BRR	bearer	Device/Provisioner
PROV_ROLE	role	PB-ADV/ PB-GATT
PROV_DEVICE_S*	pdevice	指向设备的指针，仅在 role= PROV_ROLE_DEVICE 的情况下使用否 则忽略
UINT16	gatt_timeout	Gatt 应启动的时间
UINT16	adv_timeout	Adv 应启动的时间

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.29 API_RESULT MS_prov_bind

- 绑定指定设备

类型	参数名	说明
PROV_BRR	bearer	Device/Provisioner
PROV_DEVICE_S*	pdevice	指向设备的指针，仅在 role= PROV_ROLE_DEVICE 的情况下使用否则忽略
UCHAR	attention	设备超时的提醒时间
PROV_HANDLE*	Phandle1	引用配网内容的句柄

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.30 API_RESULT MS_prov_register

- 配网层注册

类型	参数名	说明
MS_SUBNET_HANDLE	subnet_handle	代理服务器所在的子网句柄
UCHAR	proxy_adv_mode	代理广播的模式，两种模式 MS_PROXY_NET_ID_ADV_MODE / MS_PROXY_NODE_ID_ADV_MODE

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.31 API_RESULT MS_proxy_server_adv_stop (void)

- 使代理服务器停止可连接广播无。

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.32 API_RESULT MS_proxy_register

- 向网络代理层注册接口

类型	参数名	说明
PROXY_NTF_CB	proxy_cb	上层通知回调

- 返回值

EM_SUCCESS	成功。
EM_FAILURE	失败

2.3.33 API_RESULT MS_access_cm_get_primary_unicast_address

- 取得主要的单播地址

类型	参数名	说明
MS_NET_ADDR*	addr	要填充主单播地址的内存位置

- 返回值

EM_SUCCESS	成功。
EM_FAILURE	失败

2.3.34 API_RESULT MS_access_reply

- 回应访问层消息

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	模型句柄
MS_NET_ADDR	saddr	源地址

MS_NET_ADDR	daddr	目的地址
MS_SUBNET_HANDLE	subnet_handle	子网句柄
MS_APPKEY_HANDLE	appkey_handle	Appkey 句柄
UINT8	ttl	Time to live
UINT32	opcode	操作码
UCHAR*	data_param	访问参数
UINT16	data_len	访问参数长度

- 返回值

EM_SUCCESS	成功。
EM_FAILURE	失败

2.3.35 API_RESULT MS_scene_server_init

- 初始化 scene 服务器模型并在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	scene_model_handle	与场景模型实例关联的模型标识符
MS_ACCESS_MODEL_HANDLE*	scene_setup_model_handle	与场景设置模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.36 API_RESULT MS_scene_client_init

- 初始化 scene 客户端模型并在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	scene_model_handle	与场景模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.37 API_RESULT MS_light_hsl_server_init

- 初始化 hsl 服务器模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	hsl_model_handle	与 hsl 模型实例关联的模型标识符
MS_ACCESS_MODEL_HANDLE*	hsl_setup_model_handle	与 hsl 设置模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.38 API_RESULT MS_light_hsl_client_init

- 初始化 hsl 客户端模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	hsl_model_handle	与 hsl 模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.39 API_RESULT MS_light_ctl_server_init

- 初始化 ctl 服务器模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	ctl_model_handle	与 ctl 模型实例关联的模型标识符
MS_ACCESS_MODEL_HANDLE*	ctl_setup_model_handle	与 ctl 设置模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.40 API_RESULT MS_light_ctl_client_init

- 初始化 ctl 客户端模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	ctl_model_handle	与 ctl 模型实例关联的模型标识符
MS_SCENE_SERVER_CB	appl_cb	应用程序

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.41 API_RESULT MS_generic_onoff_server_init

- 初始化 generic onoff 服务器模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	model_handle	与模型实例关联的模型标识符
MS_GENERIC_ONOFF_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.3.42 API_RESULT MS_health_server_init

- 初始化 health 服务器模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	model_handle	与模型实例关联的模型标识符
UINT16	company_id	公司标识符

MS_HEALTH_SERVER_SELF_TEST*	self_tests	一系列可运行的自我测试
UINT32	num_self_tests	可运行的自我测试数量
MS_HEALTH_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.3.43 API_RESULT MS_health_client_init

- 初始化 health 客户端模型并且在访问层注册

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	element_handle	与模型实例相关联的元素标识符
MS_ACCESS_MODEL_HANDLE*	model_handle	与模型实例关联的模型标识符
MS_GENERIC_ONOFF_SERVER_CB	appl_cb	应用程序回调

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.3.44 API_RESULT MS_access_register_model

- 在访问层注册模型

类型	参数名	说明
MS_ACCESS_NODE_ID	node_id	模型需要注册的节点，默认节点的值总是 0
MS_ACCESS_MODEL*	model	指向需要注册到节点的模型描述符的指针
MS_ACCESS_MODEL_HANDLE*	model_handle	成功注册时与模型实例关联的模型标识符

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.3.45 UI_vendor_defined_model_states_initialization

- vendor model 状态初始化
无。
- 返回值
无。

2.3.46 UI_vendor_example_model_state_get

- vendor model 状态获取

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.47 UI_vendor_example_model_state_set

- Generic light ctl model 状态设置

类型	参数名	说明
UINT16	state_t	State 类型
UINT16	state_inst	initial state
void*	param	state 参数
UINT8	direction	方向

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.48 UI_phy_model_server_cb

- vendor model 回调函数

类型	参数名	说明
MS_ACCESS_MODEL_REQ_MSG_CONTEXT*	ctx	On/off 消息的上下文内容
MS_ACCESS_MODEL_REQ_MSG_RAW*	msg_raw	原始消息
MS_ACCESS_MODEL_REQ_MSG_T*	req_type	消息类型
MS_ACCESS_MODEL_STATE_PARAMS*	state_params	消息内容
MS_ACCESS_MODEL_EXT_PARAMS*	ext_params	其他参数

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.49 UI_register_vendor_defined_model_server

- 注册 vendor model server

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE	handle	model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.3.50 UI_model_states_initialization

- mesh model 所有状态初始化
无。
- 返回值
无。

2.4 Provision 接口

Provision 具体的实现部分都封装在 lib 里面，上层能看到的是一些配置信息以及回调函数，本单元重点介绍 provision。

- 配网超时配置

PROC_CFG_COMPLETE_TIMEOUT

可配，单位为 s

2.4.1 Unprovision beacon uuid

Mesh unprovision beacon 的广播包中，device ID 可以识别 mesh 不同的设备，定义如下：

名称	Size	Notes
Company ID	2	公司 ID，设置为 0x0504
Product ID	2	蓝牙设备 ID 0x62 0x12 0x62 0x22 0x62 0x52
ProductID	4	蓝牙设备类型 0x00xx – MESH_LIGHT 0x01xx – MESH_CTRL 0x02xx – MESH_LPN 0x03xx – MESH_SENS 0x04xx – TO BE ADD
版本号	2	软硬件版本号
MAC 地址	6	
RFU	2	Reserved for future use

2.4.2 UI_provcfg_complete_timeout_handler

- 配网超时的回调函数

类型	参数名	说明
void*	args	callback 参数
UINT16	size	参数长度

- 返回值
无。

2.4.3 UI_prov_callback

- 配网回调函数

类型	参数名	说明
PROV_HANDLE*	phandle	Provision handler
UCHAR	event_type	event 类型
API_RESULT	event_result	本次 event 结果
void*	event_data	event 数据
UINT16	event_data_len	event 数据长度

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.4.4 UI_register_prov

- 注册 provision service
无。

- 返回值
无。

2.4.5 UI_proxy_start_adv

- 开启 proxy beacon 广播（可连接）

类型	参数名	说明
MS_SUBNET_HANDLE	subnet_handle	网络 handler
UCHAR	proxy_adv_mode	Proxy 广播模式
		NET ID
		NODE ID

- 返回值
无。

2.4.6 UI_proxy_callback

- proxy 回调函数

类型	参数名	说明
NETIF_HANDLE*	handle	网络 handler
UCHAR	p_evt	消息类型
UCHAR*	data_param	数据
UINT16	data_len	数据长度

- 返回值
无。

2.4.7 UI_setup_prov

- 设置 provision

类型	参数名	说明
UCHAR	role	Provision 角色
UCHAR	brr	Provision bear type

- 返回值
无。

2.4.8 UI_register_proxy

- 注册 proxy service
无。

- 返回值
无。

2.4.9 UI_set_brr_scan_rsp_data

- 设置 response data
无。

- 返回值
无。

2.4.10 UI_gatt_iface_event_pl_cb

- Gatt 事件 callback，包括 provision 事件和 proxy 事件

类型	参数名	说明
UCHAR	ev_name	Gatt event 名称
UCHAR	ev_param	Gatt model 名称

- 返回值
无。

2.4.11 UI_sample_binding_app_key

根据之前配置的 model 进行 key binding，只有 binding key 后消息才能进行有效传输，否则消息会解密失败。

- 参数
无。
- 返回值
无。

2.4.12 vm_subscriptiong_binding_cb

appkey add config 消息处理

- 参数
无。
- 返回值
无。

2.4.13 vm_subscriptiong_add

- subscription add 消息处理

类型	参数名	说明
MS_NET_ADDR	addr	加组的地址

- 返回值
无。

2.4.14 vm_subscriptiong_delete

- subscription add 消息处理

类型	参数名	说明
MS_NET_ADDR	addr	删组的地址

- 返回值
无。

2.4.15 UI_app_config_server_callback

- Config 消息回调

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	Model 的 handle
MS_NET_ADDR	saddr	Source address
MS_NET_ADDR	daddr	Destination address
MS_SUBNET_HANDLE	subnet_handle	网络的 handle
MS_APPKEY_HANDLE	appkey_handle	Appkey 的 handle
UINT32	opcode	消息的 opcode
UCHAR*	data_parm	消息的内容
UINT16	data_len	消息的长度
API_RESULT	retval	消息的结果
UINT32	response_opcode	需要 response 的 opcode
UCHAR*	response_buffer	需要 response 的消息内容
UINT16	response_buffer_len	需要 response 的消息长度

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.4.16 appl_mesh_sample

- mesh sample 初始化
无。
- 返回值
无。

2.4.17 UI_sample_get_net_key

- 获取 netkey
无。
- 返回值
无。

2.4.18 UI_sample_get_device_key

- 获取 devicekey
无。
- 返回值
无。

2.4.19 UI_sample_check_app

- 获取 appkey
无。
- 返回值
无。

2.4.20 UI_sample_reinit

- Mesh sample 初始化，与 appl_mesh_sample 不同的是，这个接口主要是用作准备发起 unprovision beacon、发起 proxy beacon 或者获取 key
无。
- 返回值
无。

2.5 其他常用 api

2.5.1 MS_access_cm_get_primary_unicast_address

- 获取 unicast 地址

类型	参数名	说明
MS_NET_ADDR	addr	输出的 unicast address

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.2 MS_access_get_element_handle

- 获取 element handle

类型	参数名	说明
MS_NET_ADDR	elem_addr	要查询节点的 unicast address
MS_ACCESS_ELEMENT_HANDLE*	handle	输出的 element handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.3 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.4 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.5 MS_access_get_element_handle

- 获取 element handle

类型	参数名	说明
MS_NET_ADDR	elem_addr	要查询节点的 unicast address
MS_ACCESS_ELEMENT_HANDLE*	handle	输出的 element handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.6 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.7 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.8 MS_access_get_element_handle

- 获取 element handle

类型	参数名	说明
MS_NET_ADDR	elem_addr	要查询节点的 unicast address
MS_ACCESS_ELEMENT_HANDLE*	handle	输出的 element handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.9 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.10 MS_access_get_model_handle

- 获取 model handle

类型	参数名	说明
MS_ACCESS_ELEMENT_HANDLE	elem_handle	要查询节点的 element handle
MS_ACCESS_MODEL_ID	model_id	要查询节点的 model ID
MS_ACCESS_MODEL_HANDLE*	handle	输出的 model handle

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.11 API_RESULT MS_config_client_send_reliable_pdu

- 发送需回复的命令

类型	参数名	说明
UINT32	req_opcode	请求操作码
void*	param	操作码关联参数
UINT32	rsp_opcode	回应操作码

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.5.12 API_RESULT MS_access_cm_set_model_publication

- 设置与模型关联的发布信息

2.5.13 API_RESULT MS_access_send_pdu

- 发送访问层 PDU

类型	参数名	说明
MS_NET_ADDR	src_addr	源地址
MS_NET_ADDR	dst_addr	与目的地址
MS_SUBNET_HANDLE	subnet_handle	子网句柄
MS_APPKEY_HANDLE	appkey_handle	Appkey 句柄
UINT8	ttl	Time to live
UINT32	opcode	操作码
UCHAR*	data_param	访问层参数
UINT16	data_length	数据长度
UINT8	reliable	如果需要底层传输层回复，reliable 为真

- 返回值

APL_SUCCESS	成功。
其他数值	参考<MS_error.h>

2.5.14 MS_access_raw_data

- 向指定的地址发送消息

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	节点的 model handle
UINT32	opcode	消息的 opcode
MS_NET_ADDR	dst_addr	发送的目标地址
MS_APPKEY_HANDLE	appKeyHandle	发送消息需要加密的 appkey handle
UCHAR*	data_param	发送的消息内容
UINT16	data_len	发送消息的长度
UINT8	reliable	是否需要可靠传输 true, 无论是否满足拆包条件, 统一进行拆包处理 false, 满足拆包条件, 才能进行拆包处理, 否则会按照不拆包的流程进行处理

- 返回值

API_SUCCESS	成功
-------------	----

2.5.15 API_RESULT MS_generic_onoff_client_send_reliable_pdu

- 发送需应答的 generic onoff 命令

类型	参数名	说明
UINT32	req_opcode	请求操作码
void*	param	与请求操作码相关的参数
UINT32	rsp_opcode	回应操作码

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.5.16 API_RESULT MS_hsl_client_send_reliable_pdu

- 发送需应答的 hsl 命令

类型	参数名	说明
UINT32	req_opcode	请求操作码
void*	param	与请求操作码相关的参数
UINT32	rsp_opcode	回应操作码

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.5.17 API_RESULT MS_access_publish

- 将访问层消息发布到与模型关联的发布地址

类型	参数名	说明
MS_ACCESS_MODEL_HANDLE*	handle	要发送的消息的访问模型句柄
UINT32	opcode	访问操作码
UCHAR*	data_param	数据包
UINT16	data_len	数据包长度
UINT8	reliable	MS_TRUE 为可靠消息， MS_FALSE 为其他

- 返回值

APL_SUCCESS	成功。
-------------	-----

其他数值 参考<MS_error.h>

2.5.18 MS_common_reset

- Mesh 协议栈重置，配网等信息都会被重置无。

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.19 MS_access_ps_store_all_record

- 存储 mesh 的配置消息到 flash 上无。

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.20 MS_access_ps_store_disable

- 打开或者关闭 mesh 消息存储功能

类型	参数名	说明
UINT8	enable	1: enable; 0: disable

- 返回值

API_SUCCESS	成功
API_FAILER	失败

2.5.21 开启/关闭 relay 功能

MS_DISABLE_RELAY_FEATURE	关闭 relay
MS_ENABLE_RELAY_FEATURE	开启 relay

2.5.22 开启/关闭 proxy 功能

MS_DISABLE_PROXY_FEATURE	关闭 proxy
MS_ENABLE_PROXY_FEATURE	开启 proxy

2.5.23 开启/关闭 friend 功能

MS_DISABLE_FRIEND_FEATURE	关闭 friend
MS_ENABLE_FRIEND_FEATURE	开启 friend

3 应用实例

3.1 mesh 初始化

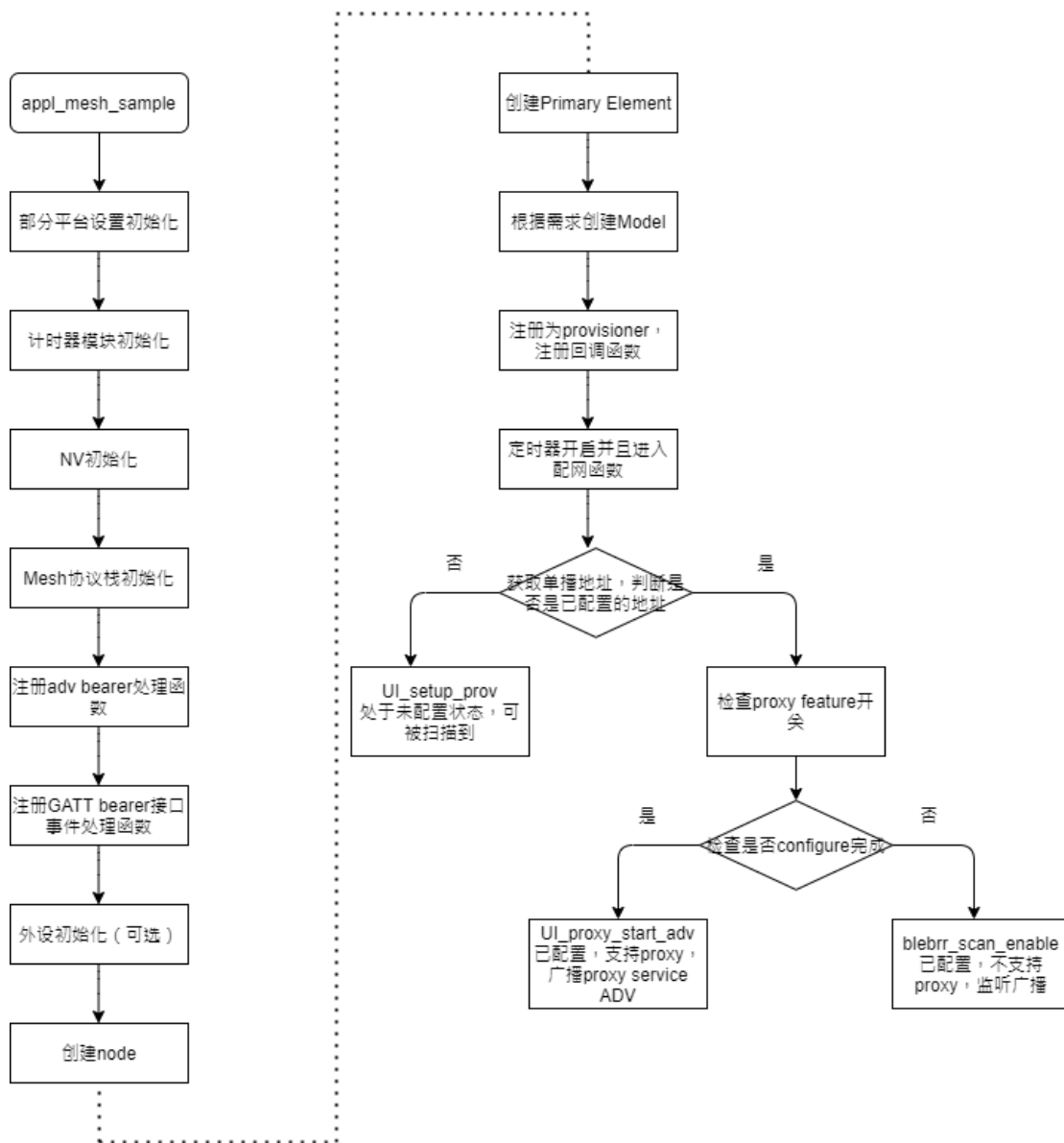


图 9：mesh 初始化流程图

```

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 that 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        is_prov_req;
    UCHAR        role, brr;
    UCHAR        state;
    retval        = API_SUCCESS;
    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 Scanning 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);
}
}

```

3.2 vendor model 状态上报

```
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; //请求内容
    req_context.saddr  = saddr;
    req_context.daddr  = daddr;
    req_context.subnet_handle = subnet_handle;
    req_context.appkey_handle = appkey_handle;

    req_raw.opcode = opcode; //请求参数
    req_raw.data_param = data_param;
    req_raw.data_len = data_len;
    state_params.phyplusmode_param = NULL;

    switch(opcode) //根据自定义的opcode执行相应功能
    {
    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 requested 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;
}

```

3.3 Generic onoff 状态上报

```

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;
}

```