

RTL8762C Mesh SDK User Guide

V 1.0.3

2019/1/29

修订历史 (Revision History)

日期	版本	修改	作者	Reviewer
2015/10/12	Draft v0.1	初稿	bill	
2015/11/26	Draft v0.2	去除 mesh 调试部分，增加新功能说明	bill	
2017/09/26	Draft v0.9	根据 Mesh V1.0	bill	
2017/10/30	Draft v0.9.1	添加 pb-adv link 信息显示，device 信息显示，Device UUID 配置，composition data page 和 subscribe list 操作	bill	
2017/11/09	Draft v0.9.2	增加 mesh 原理介绍，增加 demo 工程资源情况介绍	bill	
2018/01/17	Draft v0.9.3	调整文档结构 增加 light 相关内容	bill	
2018/04/23	Draft v0.9.5	修改 trans_ping 接口 修改 dus/pbadvcon/con 等 user cmd 的接口，增加 dis 等 user cmd，删除 rembd 等 user cmd	bill	
2018/06/22	Draft v0.9.6	修改 model id 和 access opcode 定义 修改 lrs 灯控参数格式	bill	
2018/08/03	Draft v0.9.8	增加天猫精灵接入说明	bill	
2018/08/14	Draft v0.9.9	更新烧录说明	bill	
2018/08/30	Draft v1.0.0	更新 code 引用，完善文档	bill	
2018/08/31	Draft v1.0.1	增加 model 框架介绍	hector	
2018/09/05	Draft v1.0.2	增加 fsbl 烧录，修改 flash layout	bill	
2018/09/21	Draft v1.0.3	review,修改不同步的地方	bill	

目 录

修订历史 (Revision History)	2
目 录	3
表目录	5
图目录	7
词汇表	8
1 Mesh 简介	9
1.1 Device UUID	9
1.2 Mesh 地址	9
1.3 应用模型	10
1.4 安全性	10
1.5 Provisioning	11
1.6 Configuration	11
1.7 Proxy	12
2 Mesh 框架	13
2.1 Model Layer (Foundation)	13
2.2 Access Layer	14
2.3 Transport Layer (Upper & Lower)	15
2.3.1 Transport Ping	15
2.4 Network Layer	15
2.5 Bearer Layer	15
2.6 Provisioning	16
2.6.1 PB-ADV	16
2.6.2 PB-GATT	17
2.7 Friendship	17
2.7.1 Friend Node	17
2.7.2 Low Power Node	18
2.8 Miscellaneous	18

2.8.1 mesh log 配置.....	18
2.8.2 设备信息配置	19
2.8.3 Device UUID 设置.....	19
2.8.4 网络特性和参数配置	19
3 Model 框架.....	21
3.1 消息回调	21
3.2 消息定义	21
3.3 消息处理	22
4 烧录配置	24
4.1.1 Patch	24
4.1.2 烧录	24
5 Demo 工程	28
5.1.1 串口配置	29
5.1.2 命令	30
5.1.3 provisioning	30
5.1.4 friendship	31
5.1.5 model	32
5.1.6 Lighting Control	32
5.1.7 天猫精灵	36
6 FAQ	40
参考文献	42
附录	43

表目录

表 1-1 mesh address 范围.....	9
表 2-1 model id.....	13
表 2-2 element/model 创建注册	14
表 2-3 access opcode	14
表 2-4 transport ping.....	15
表 2-5 other bearer.....	15
表 2-6 device provisioning setting.....	16
表 2-7 provisioning callback register.....	16
表 2-8 pb-adv 创建.....	16
表 2-9 pb-adv link 处理.....	17
表 2-10 friend node 初始化.....	17
表 2-11 lpn 初始化.....	18
表 2-12 lpn 发起 friendship.....	18
表 2-13 mesh log setting.....	18
表 2-14 device info setting	19
表 2-15 Device UUID setting	19
表 2-16 networking parameters setting.....	19
表 3-1 model 信息结构体	21
表 3-2 model 消息定义	22
表 3-3 model 消息处理	22
表 5-1 provisioner setting	28
表 5-2 uart setting	29
表 5-3 通用命令	30
表 5-4 信息收集.....	30
表 5-5 pb-adv provisioning.....	30
表 5-6 pb-gatt provisioning	31
表 5-7 friendship.....	31
表 5-8 密钥管理.....	32
表 5-9 model 订阅配置	32
表 5-10 Light Models 配置	33
表 5-11 CWRGB Model.....	33
表 5-12 Light CWRGB 操作.....	34

表 5-13 Light HSL 操作.....	35
表 5-14 阿里 uuid 格式.....	36
表 5-15 配置阿里 uuid.....	37
表 5-16 prov capability 配置.....	37
表 5-17 authentication data 配置.....	38

Realtek Confidential

图目录

图 2-1 Mesh 框架	13
图 4-1 flash map tool	24
图 4-2 download	25
图 4-3 config file setting normal.....	26
图 4-4 config file setting layout.....	26
图 4-5 config file setting power mode.....	27
图 5-1 CLI 接口	29
图 5-2 上位机终端串口配置	29
图 5-3 Light HSL Server Models.....	35
图 6-1 log tool setting.....	40

词汇表

缩写	含义

Realtek Confidential

1 Mesh 简介

Bluetooth Low Energy Mesh 是基于低功耗蓝牙技术（BLE）的网状网络解决方案。网状网络主要分为两种：泛洪网状网络(flooding-based mesh network)和路由网状网络(routing-based mesh network)。这两种网络类型各有利弊，BLE Mesh 的技术路线是试图综合这两种网络的特点。目前，处于第一阶段，只采用 flooding 方式。

BLE 的通信信道有两类：^{广播信道} advertising channel 和 ^{数据信道} data channel。Mesh 主要工作在 advertising channel 上，通过 ^{被动扫描} passive scan 和 advertising，分别进行接收和发送。而 data channel 主要是为了兼容现有的不支持 advertising 设备，可以通过 LE link 方式进行通信。

下面介绍 mesh 的主要概念，Mesh 的详细内容请参考 SIG 的 Mesh Profile Specification [1]，初步了解可以参看 spec 的第二章 Mesh system architecture。

1.1 Device UUID

每个设备出厂时被分配一个唯一的 16 字节 UUID，称作 Device UUID，用于唯一标识一个 mesh 设备，不用依赖蓝牙地址来标识设备。在建立 pb-adv link 时，需要 Device UUID 字段来标识 ^{设备} device。然而，当 mesh device 获取 mesh 地址后，即可用 mesh address 来唯一标识 device。

1.2 Mesh 地址

除了建立 le link，mesh 通信并不依赖蓝牙地址，即节点的蓝牙地址可以一样，或者随机变化。mesh 定义了一套长度为 2 字节的 mesh 地址，分为 unassigned address、unicast address、virtual address 和 group address，地址范围如表 1-1 所示。

mesh 地址并不是出厂时设置的，而是由用户自己统一管理和分配的。用户在配置设备入网时，通过 ^{服务开通} Provisioning 流程给设备分配地址。^{服务提供者/规定者} Provisioner 需要确保给每个设备分配的地址是不重复的。mesh 设备可能不止一个 mesh 地址，设备内每个 ^{元素/成分} element 会被分配一个地址，且地址是连续的。多地址被设计用于区分 mesh 设备上重复的功能模块 ^{节点内模块} model。

蓝牙mesh中分三种model：Server、Client和控制。一个节点默认需要两个Server Model：Configuration Server（0x0000）和Health Server（0x0002）

表 1-1 mesh address 范围

Range	Address Type	说明
0x0000	unassigned address	未分配地址，device 默认地址，不能用于发送或接收 mesh 消息
0x0001~0x7fff	unicast address	单播地址，每个 element 会被分配一个单播地址 message 的源地址一定要是 unicast address
0x8000~0xbfff	virtual address	虚拟地址，由 label uuid 生成。数量多，可以不用集中管理。

0xc000~0xffff	group address	组地址，分为两类 0xc000~0xffff: 自由分配组地址 0xff00~0xffff: 固定组地址
---------------	---------------	--

1.3 应用模型

BLE 是 ^{主机} master 连接 ^{从机} slave 的一对一的通信，而 mesh 网络是多对多的通信。因而，mesh 网络存在一个天然特性，就是节点之间并不知道其他节点的存在。此时，需要一个第三方，通常是 Provisioner，来扮演月老的角色，将节点之间联系起来。例如一个通用的开关，出厂后是不知道自己要控制哪盏或者哪些灯，需要 provisioner 通过 configuration 配置开关发布 publish 消息（设置目的地址为单播、组播、广播地址）。如果是组播，则需同时配置相应的灯泡订阅 subscribe 消息（设置分组，即增加组播地址到订阅表）。然后，开关就可以控制这一盏灯、一组灯或者所有灯。

Mesh 将典型应用场景的操作进行了标准化，每个 mesh 设备上的应用是以 Model 为单位进行组织的。^{模型/型号} Model 定义了一个 model id、一套 ^{操作码} opcode 和一组状态，规定发送和接收哪些消息，分别操作哪些状态。Model 和 BLE 的 GATT service 是类似的，都用于定义一个特定的应用场景。

为了支持多个相同的 model，定义了 ^{元素/要素} element 的概念，每个 element 会单独分配一个 element address，且地址是连续的。第一个 element (^{主要/基本的} primary element) 的 element address 是在 provisioning 过程中分配的 node address，其他 element 的地址顺序往后排。例如一个 mesh 设备上有两盏完全一样的且可以独立控制的灯，开关设备去控制这个灯设备，需要区分控制哪盏灯。让这两盏灯对应的两个 Model 分开放在两个 element 中，这样每盏灯分别有一个 mesh address，就可以通过 mesh address 将两盏灯区分开来了，进行独立控制。当然，两盏灯的 model 也可以订阅同一个组地址，实现同时控制。这样既能独立控制，也能同时控制。

设备上 element、model 组成情况通过 composition data page 0 表达，Provisioner 可以通过获取设备的 composition data page 0 来辨识设备支持的应用。

node id
↓
model id
↓ 连续
element id

一个节点内包括多个 model (模块) 每个模块是单独 model id 区分。如果有重复的 model，例如多个相同的灯而且每个灯需要单独控制，就给每个灯分配单独的 element 地址，element 地址是由节点地址开始连续的地址

1.4 安全性

mesh 中有很多保护网络安全和隐私的设计，能够抵挡被动监听、中间人攻击、重放攻击、垃圾桶攻击和暴力破解等常见的攻击。

mesh 网络中所有 mesh 消息都会被加密和校验，防止被窃听或篡改。mesh 网络中密钥分两层：NetKey 和 AppKey，每层最多可以有 4096 个密钥，通过 12 bit 的 index 标识。AppKey 必须绑定有且只能一个 NetKey。应用层发送消息会依次经过 AppKey 和 NetKey 两层加密和校验，接收消息会依次经过 NetKey 和 AppKey 两层解密和校验。采用两层密钥，是为了防止 ^{转播} relay 节点窃听或者篡改消息。例如节点 A 通过节点 B 转发给节点 C 发数据，A/B/C 有相同的 NetKey，A/C 有相同的 AppKey，而 B 没有该 AppKey。那么 A 和 C 间的应用层通信对 B 来说是保密的，B 只是使用 NetKey 在网络层帮忙转发，因为没有 AppKey 而不能进行窃听或者篡改应用层消息。

NetKey 支持多个密钥，多密钥可以用来划分网络范围，实现设备间的隔离。Key index 为 0 的是主网络密钥，其余的都是普通的其他子网络密钥。只有主网络中的节点才能参与 IV Update Procedure，并将 IV

第四更新过程

更新信息传递到其他子网中。也就是说，只有主网络节点才能更新 IV index 网络参数，其他子网的节点只能被动的接收 IV index 更新。^{第四指针}这样不平等的网络密钥设计的目的是约束子网络节点的数据发送频次，防止子网络节点滥用 IV index 更新而耗尽 IV index，从而导致网络安全问题。通常大部分节点在主网络中，部分节点同时处于主网络和某个子网络，少量节点只处于某个子网络，此时这些少量节点只能在子网络内进行局部通信，从而限定这些少量节点的通信范围，例如酒店顾客只能控制自己房间内的灯。

AppKey 也支持多密钥，密钥之间并无区别。不同的应用可以使用不同的密钥，实现应用间的隔离。例如灯控和门控使用不同的 AppKey。一个应用也可以使用多个 AppKey，例如一盏灯上支持 2 个 AppKey，AppKey0 给用户 0 用，AppKey1 给用户 1 用，将 AppKey0 删除，那么用户 0 就不能再控制灯，而用户 1 则不受影响。

Device 上的 NetKey 和 AppKey 是 ^{设备/终端}provisioner 通过 ^{服务提供者/规定者}provision 和 ^{规定}configuration 分发和管理的。^{配置}provisioner 是网络管理员，他管理着所有的 key，即管理网络中各个 device 各自可以使用哪些 key，而 device 间只有共享相同的密钥才能互相通信，例如灯和灯的开关使用相同的秘钥。provision 过程会分发 mesh address 和有且只有一个 NetKey，后续通过 configuration 来管理，例如通过 configuration 增加 NetKey 和 AppKey。

Provision 过程还会随机生成一种特殊的 AppKey，称作 DevKey。DevKey 只有 provisioner 和 device 两者知道，不和其他任何 device 共享，保证了 provisioner 可以单独和某一个 device 进行秘密的一对一通信。Configuration 配置被限制只能使用 DevKey，只有 provisioner 才知道 device 的 DevKey，所以只有 provisioner 才可以配置 device。举例而言，开关只能控制灯泡亮灭，而不能去配置灯泡的分组。

1.5 Provisioning

device 出厂默认是没有地址和密钥的，需要通过 ^{设备}provisioning 过程从 ^{服务开通}provisioner 获得。device 被 provisioning 后，就称作 ^{节点}node，本文和 ^{演示工程}demo project 并没有严格区分这两个概念。provisioning 过程类似于 ^{蓝牙配对}bt pairing 中的 ^{安全连接}secure connection，采用 ecdh 算法进行密钥协商和分发，通过 authentication data 进行身份鉴权，能够防止窃听、暴力破解和中间人攻击。

根据 provisioning 配置不同，会有不同的模式，^{公共密钥}public key 的交互分为 oob 和 no oob 两种，^{认证/验证数据}authentication data 的交互分为 input oob、output oob、static oob 和 no oob 四种，所以模式最多有 8 种。device 的应用层可能需要给 ^{堆栈}stack 提供 ^{公共/私有 密钥}public/private key 和 authentication data。provisioner 的应用层可能需要给 stack 提供模式选择、device 的 public key 和 authentication data。

provisioning 流程可以工作在 advertising channel 和 data channel 两种信道上，分别对应 pb-adv 和 pb-gatt 两种传输层。device 是被强制要求支持 pb-adv 的，如果同时支持 pb-gatt，被 provisioning 时可以任选一个。

1.6 Configuration

网络参数的管理是在 model 层实现的，称作 configuration models。可以配置的网络参数有很多，例如 NetKey 和 AppKey 增加、删除、修改等，model 的密钥绑定、消息发布、消息订阅等，节点应用结构 Composition data page 0 的获取，节点的默认 ttl、支持的 feature、网络重传次数等。

网络参数的配置被限定为只能使用 DevKey，也就是说只有 provisioner 才能配置节点的网络参数。

1.7 Proxy

mesh 主要工作在 advertising channel 上，为了兼容一些不能灵活自由的 advertising 的设备，mesh 定义了 proxy 特性。基于 BLE GATT profile，定义了 proxy service，让这些设备利用 ble link 的方式连接到支持 proxy 的节点上，从而接入到 mesh 网络。

Realtek Confidential

2 Mesh 框架

Mesh 框架如图 2-1 所示，分为 bearer layer、network layer、lower transport layer、upper transport layer、access layer、foundation model layer 以及 model layer。

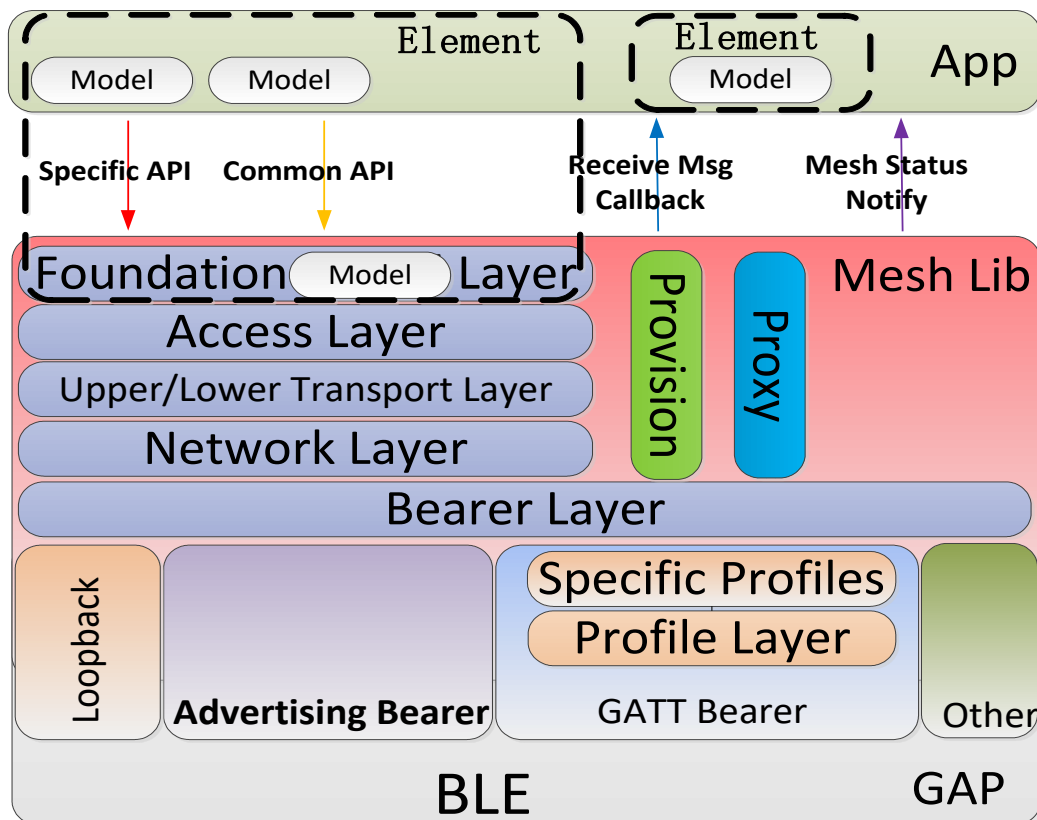


图 2-1 Mesh 框架

2.1 Model Layer (Foundation)

Model 定义了特定应用的一组状态和操作，通过 model id 来标识，分为 16 bits model id 的 sig model 和 32 bits model id 的 vendor model 两种。实现上，会对 sig model id 进行编码，16 bits sig model id 会被编码成 32 bits model id。例如 configuration server 和 client 的 model id 是 0x0000 和 0x0001，但实现上将 configuration model 编码成了 0x0000FFFF 和 0x0001FFFF，如表 2-1 所示。

表 2-1 model id

1. #define MESH_MODEL_CFG_SERVER	0x0000FFFF
----------------------------------	------------

2. #define MESH_MODEL_CFG_CLIENT

0x0001FFFF

Element 用于隔离相同的 model，即使用了相同 access layer opcode 的 model。重复的 model 会发送或接收相同的 opcode，此时无法区分是哪一个 model 在发送或者要接收这个 opcode 对应的 access msg。而 element 会被分配不同的 element address，这样可以通过 msg 的 source address 或者 destination address 来有效的区分 model。Mesh stack 采用先创建 element，再在指定 element 下注册 model 的方式。如果注册重复的 model，需要为该 model 指定不同的 element，stack 会去检查一个 element 下的 model 是否冲突。

Mesh Spec 中强制要求的 configuration server model 不需要 app 注册，由 stack 负责处理。

Health server model 由用户自己创建并注册。只有 primary element 是 spec 强制要求的，其他 element 用户可根据需求自行创建并注册。

Element 和 model 创建注册完毕后，就可以生成 composition data page 0 的内容，需要 app 提供具体的 header 信息，包含 company id、product id、version id、重放保护表大小以及所支持的 feature。

Element 创建、model 注册以及 composition data page 0 生成的具体步骤参考表 2-2 的示例。

表 2-2 element/model 创建注册

```
1. mesh_element_create(GATT_NS_DESC_UNKNOWN);
2. health_server_reg(0, &health_server_model);
3. ping_control_reg(ping_app_ping_cb, pong_receive);
4. compo_data_page0_header_t compo_data_page0_header = {COMPANY_ID, PRODUCT_ID,
    VERSION_BUILD};
5. compo_data_page0_gen(&compo_data_page0_header);
```

2.2 Access Layer

Access Layer 定义了上层应用的消息格式，其中包含 operation codes，即 access opcode，所有应用消息必须包含统一的 opcode。Mesh spec 定义了一部分 opcode，也预留了空间让厂商自定义 opcode。自定义 opcode 是 3 字节的，第一个字节的高两位为 1，低 6 位为厂商自定义的 opcode，后两个字节是 company id，即每个 company 只能自定义 64 个 opcode。如表 2-3 所示，0xC0 表示是 0 号 manufacturer-specific opcode，0x005D 对应 Realtek 的 company id，需要按字节序填入。

表 2-3 access opcode

1. #define MESH_MSG_PING

0xC05D00

access 层消息首先会被 transport layer 使用 AppKey 加密，TransMIC 校验长度支持 4 bytes 和 8 bytes。如果使用了 4 bytes TransMIC，access 层消息最大长度为 380 bytes，反之则最大长度为 376 bytes。

2.3 Transport Layer (Upper & Lower)

Transport layer 负责应用层加解密，friendship 维护，heartbeat 和 segmentation & reassembly。

Access layer 消息采用 4 bytes TransMIC，消息长度小于等于 11 bytes（包含 access opcode），且不主动要求使用 SAR，则这笔消息不会被分片 segmented，即以一笔 advertising packet 方式发送出去。反之，access 消息会被 segmented，根据消息长度不同被一笔或者多笔 advertising packet 发送出去，且会要求接收端回复 transport layer 的 acknowledge。

2.3.1 Transport Ping

为了方便测试网络层通信，在 transport layer 自定义了 ping/pong 机制，消息里面携带 TTL 信息用于计算 hop 次数。该消息直接利用 network layer 的服务，只会被 NetKey 加密。Device 被 provisioning 后，就可以使用 transport layer 的 ping 功能测试 network layer 是否 work，此时 device 还没有获得 AppKey。

表 2-4 transport ping

函数	mesh_msg_send_cause_t trans_ping(uint16_t dst, uint8_t ttl, uint8_t net_key_index, uint16_t pong_max_delay)
功能	在 transport layer ping 指定设备
参数	dst: 目的地址 ttl: 初始 ttl net_key_index: NetKey 索引 pong_max_delay: pong 回复最大 delay，单位 10ms

2.4 Network Layer

Network Layer 负责网络层加解密、加扰和 relay。

2.5 Bearer Layer

Bearer Layer 分为 loopback bearer、advertising bearer、gatt bearer 和 other bearer。Loopback bearer，即自发自收通道，用于 model 间通信。other bearer 是其他接口，用于在 bearer 层扩展 mesh 网络。例如 Gateway 可以通过 other bearer 连接 Mesh 网络和以太网，使得 Mesh 网络可以接入 Internet。Other bearer 的接口见表 2-5，分别用于在 other bearer 上发送消息，和从 other bearer 上接收消息。

表 2-5 other bearer

```
1. void bearer_other_reg(pf_bearer_other_send_t send);
```

```
2. void bearer_other_receive(bearer_pkt_type_t pkt_type, uint8_t *pbuffer, uint16_t len);
```

2.6 Provisioning

Device 出厂设置中是不含任何网络信息的，需要通过 provisioning 过程从 provisioner 获取 Address 和 Network Key 等网络信息。

Device 需要配置 provisioning capabilities，如表 2-6 所示。

表 2-6 device provisioning setting

```
1. prov_capabilities_t prov_capabilities =
2. {
3.     .algorithm = PROV_CAP_ALGO_FIPS_P256_ELLIPTIC_CURVE,
4.     .public_key = 0,
5.     .static_oob = 0,
6.     .output_oob_size = 0,
7.     .output_oob_action = 0,
8.     .input_oob_size = 0,
9.     .input_oob_action = 0
10. };
11. prov_params_set(PROV_PARAMS_CAPABILITIES, &prov_capabilities,
    sizeof(prov_capabilities_t));
```

Device 和 Provisioner 都需要注册 provisioning 的 callback 处理函数，如表 2-7 所示。根据 provisioning 具体流程的不同，需要在 provisioning callback 里完成不同的处理。

表 2-7 provisioning callback register

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

Provisioner 和 Device 间可以通过 pb-adv 或者 pb-gatt 两种通道完成 provisioning 流程。通道建立成功后，才可以开始 provisioning 流程。

2.6.1 PB-ADV

根据 device 的 Device UUID，provisioner 发起 pb-adv 链路建立流程。

表 2-8 pb-adv 创建

函数	void pb_adv_link_open (uint8_t ctx_index, uint8_t device_uuid[16])
----	--

功能	建立 pb-adv bearer
参数	ctx_index: 上下文索引 device_uuid: Device 的 Device UUID

在 prov_cb 回调函数里，会处理 pb-adv 链路建立情况，如表 2-9 所示。

表 2-9 pb-adv link 处理

```

1.      case PROV_CB_TYPE_PB_ADV_LINK_STATE:
2.          switch(cb_data.pb_generic_cb_type)
3.          {
4.              case PB_GENERIC_CB_LINK_OPENED:
5.                  data_uart_debug("Link Opened!\r\n>");
6.                  break;
7.              case PB_GENERIC_CB_LINK_OPEN_FAILED:
8.                  data_uart_debug("Link Open Failed!\r\n>");
9.                  break;
10.             case PB_GENERIC_CB_LINK_CLOSED:
11.                 data_uart_debug("Link Closed!\r\n>");
12.                 break;
13.             default:
14.                 break;
15.         }
16.         break;

```

2.6.2 PB-GATT

Provisioner 需要主动和 Device 建立 BLE 连接，查询 provisioning service，并使能相应的 CCCD。

2.7 Friendship

为了支持低功耗设备，需要通过 friendship 方式，让 friend node 帮助 low power node 缓存消息。

friendship 会额外消耗内部 NetKey 空间。FN 可以设置和建立最大 friendship 个数小于 NetKey 个数。LPN 在每个 NetKey 上只能建立一个 friendship，所以 friendship 个数小于等于总 NetKey 个数的一半。

2.7.1 Friend Node

如果要支持 friend node feature，需要先初始化。

表 2-10 friend node 初始化

函数	bool fn_init(uint8_t lpn_num, uint8_t queue_size, pf_fn_cb_t pf_fn_cb)
-----------	--

功能	fn 初始化
参数	lpn_num: 支持的 lpn 个数 queue_size: 支持的 friend queue size pf_fn_cb: 处理函数

2.7.2 Low Power Node

如果要支持 low power node feature，需要先初始化。

表 2-11 lpn 初始化

函数	bool lpn_init(uint8_t fn_num, pf_lpn_cb_t pf_lpn_cb)
功能	lpn 初始化
参数	fn_num: 支持的 fn 个数 pf_lpn_cb: 处理函数

发起 friendship 流程:

表 2-12 lpn 发起 friendship

函数	lpn_req_reason_t lpn_req(uint8_t frnd_index, uint8_t net_key_index, lpn_req_params_t *p_req_params)
功能	发起 friendship
参数	frnd_index: fn 索引 net_key_index: lpn 的一个 network key 只能建立一个 friendship p_req_params: lpn 的 friend 要求

2.8 Miscellaneous

2.8.1 mesh log 配置

Mesh stack 内部有多个模块，每个模块的各个级别 log 可以单独被开关，默认是都打开的。Log 级别共有四级，由高到低分别是 LEVEL_ERROR、LEVEL_WARN、LEVEL_INFO 和 LEVEL_TRACE，越低级别的 log 越不重要。由于 log 速率有限，当打印太多 log 时，log 会丢失。为了保证关键 log 正常打印，可以关掉一些不重要的 log。例如表 2-13，关闭所有 mesh 模块 LEVEL_TRACE 级别的 log。

表 2-13 mesh log setting

1. uint32_t module_bitmap[MESH_LOG_LEVEL_SIZE] = {0};

```
2. diag_level_set(LEVEL_TRACE, module_bitmap);
```

2.8.2 设备信息配置

应用层可以调用 `gap_sched_params_set` 接口配置 device name 和 appearance 等信息。如果不进行设置，设备默认 device name 为 `rtk_mesh`，appearance 为 `unknown`。

表 2-14 device info setting

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

2.8.3 Device UUID 设置

Mesh 节点可能拥有相同的蓝牙地址或者随机的蓝牙地址，需要通过 Device UUID 来唯一标识 mesh 节点。

表 2-15 Device UUID setting

```
1. uint8_t dev_uuid[] = MESH_DEVICE_UUID;
2. device_uuid_set(dev_uuid);
```

2.8.4 网络特性和参数配置

Node 支持的 features、密钥个数、virtual address 个数、subscription list 大小、flash 存储、advertising 周期等网络参数，需要 app 酌情配置。

表 2-16 networking parameters setting

```
1. mesh_node_features_t features =
2. {
3.     .role = MESH_ROLE_DEVICE,
4.     .relay = 1,
5.     .proxy = 1,
6.     .fn = 0,
```

```
7.         .lpn = 0,
8.         .prov = 1,
9.         .udb = 1,
10.        .snb = 1,
11.        .bg_scan = 1,
12.        .flash = 1,
13.        .flash_rpl = 1
14.    };
15.    mesh_node_cfg_t node_cfg =
16.    {
17.        .dev_key_num = 1,
18.        .net_key_num = 3,
19.        .app_key_num = 3,
20.        .vir_addr_num = 3,
21.        .rpl_num = 20,
22.        .sub_addr_num = 10,
23.        .proxy_num = 1
24.    };
25.    mesh_node_cfg(features, &node_cfg);
```

3 Model 框架

蓝牙 SIG 组织定义了很多标准的 Model，Realtek 的 SDK 中已经包含了大部分常用 Model，并且都已经通过 BQB 认证。为了降低 App 开发难度，SDK 对 Model 层和 Access 层之间的消息处理进行了封装，对 APP 提供一系列预定义的业务消息，使得 App 只需关注于业务逻辑的开发。Model 详细内容请参考 SIG 的 Mesh Model Specification [2]。

3.1 消息回调

在使用一个具体的 Model 之前都会定义一个如表 3-1 所示的 Model 信息的结构体，其中 model_receive 函数是用来处理 Model 层和 Access 层之间消息的回调函数，model_data_cb 是 Model 业务逻辑处理回调函数，目前 Realtek 的 SDK 中已经实现了默认的 model_receive 函数，在 model_receive 函数中会根据具体的 Access 消息来调用 model_data_cb 函数来进行业务逻辑的处理，所以 App 只需要根据应用场景实现对应的 model_data_cb 函数即可。

表 3-1 model 信息结构体

```

1.  typedef struct _mesh_model_info_t
2.  {
3.      uint32_t model_id;
4.      model_receive_pf model_receive;
5.      model_send_cb_pf model_send_cb; //!< indicates the msg transmitted state
6.      model_pub_cb_pf model_pub_cb; //!< indicates it is time to publishing
7.      model_data_cb_pf model_data_cb;
8.      /** point to the bound model, sharing the subscription list with the binding model */
9.      struct _mesh_model_info_t *pmodel_bound;
10.     /** configured by stack */
11.     uint8_t element_index;
12.     uint8_t model_index;
13.     void *pelement;
14.     void *pmodel;
15.     void *pargs;
16. } mesh_model_info_t;

```

3.2 消息定义

在每个具体 Model 的头文件中都会有如表 3-2 中的预定义业务消息，每个消息对应一个具体的消息结构。

表 3-2 model 消息定义

1. #define GENERIC_ON_OFF_SERVER_GET	0
2. #define GENERIC_ON_OFF_SERVER_GET_DEFAULT_TRANSITION_TIME	1
3. #define GENRIC_ON_OFF_SERVER_SET	2
4.	
5. typedef struct	
6. {	
7. generic_on_off_t on_off	
8. } generic_on_off_server_get_t;	
9.	
10. typedef struct	
11. {	
12. generic_transition_time_t trans_time;	
13. } generic_on_off_server_get_default_transition_time_t;	
14.	
15. typedef struct	
16. {	
17. generic_on_off_t on_off;	
18. generic_transition_time_t total_time;	
19. generic_transition_time_t remaining_time;	
20. } generic_on_off_server_set_t;	

3.3 消息处理

如表 3-3 所示是一个简单的开关灯的业务逻辑的处理，先定义好 Model 的结构并给 model_data_cb 函数指定具体的处理函数，当远端发送开灯或者关灯的消息时，SDK 会自动调用到 generic_on_off_server_data 函数，并根据消息类型填上对应的信息，App 只需要在具体的消息下面取到开关灯的值之后进行灯的开关即可，完全不用关心 Model 层和 Access 层之间的消息处理，使用起来十分方便。

表 3-3 model 消息处理

```

1. /** generic on/off server model */
2. static mesh_model_info_t generic_on_off_server;
3.
4. /** light on/off state */
5. generic_on_off_t current_on_off = GENERIC_OFF;
6.
7. /** light on/off process callback */
8. static int32_t generic_on_off_server_data(const mesh_model_info_p pmodel_info, uint32_t type,
9.                                         void *pargs)
10. {

```

```
11.     int32_t ret = MODEL_SUCCESS;
12.     switch (type)
13.     {
14.         case GENERIC_ON_OFF_SERVER_GET:
15.             {
16.                 generic_on_off_server_get_t *pdata = pargs;
17.                 pdata->on_off = current_on_off;
18.             }
19.             break;
20.         case GENERIC_ON_OFF_SERVER_SET:
21.             {
22.                 generic_on_off_server_set_t *pdata = pargs;
23.                 current_on_off = pdata->on_off;
24.                 ret = MODEL_STOP_TRANSITION;
25.             }
26.             break;
27.         default:
28.             break;
29.     }
30.
31.     return ret;
32. }
33.
34. /** light on/off model initialize */
35. void light_on_off_server_models_init(void)
36. {
37.     /* register light on/off models */
38.     generic_on_off_server.model_data_cb = generic_on_off_server_data;
39.     generic_on_off_server_reg(0, &generic_on_off_server);
40. }
```

4 烧录配置

如果采用默认的 flash layout 等配置，烧录需要用的相关文件，目前都放在了 sdk 的. \tool\download\目录下。否则，需要用户根据实际情况，使用相关 tool 进行配置生成 image。

4.1.1 Patch

使用 Mesh SDK 中给出的 mesh 专用 patch。

4.1.2 烧录

4.1.2.1 flash layout

Mesh sdk 里默认按照 512KB flash 配置的。使用 flash map tool 配置如图 4-1 所示，tool 自动生成 flash map.ini 供其他 tool 使用，生成 flash_map.h 文件供各个 mesh project 使用。

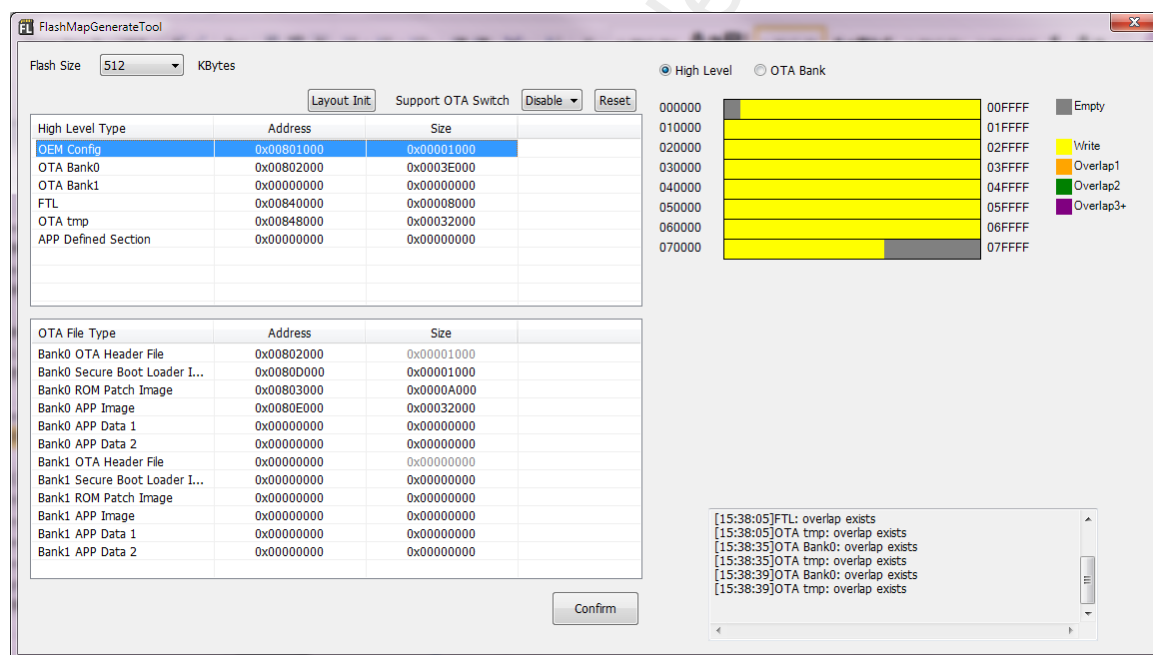


图 4-1 flash map tool

4.1.2.2 MP tool

烧录工具使用如图 4-2 所示。

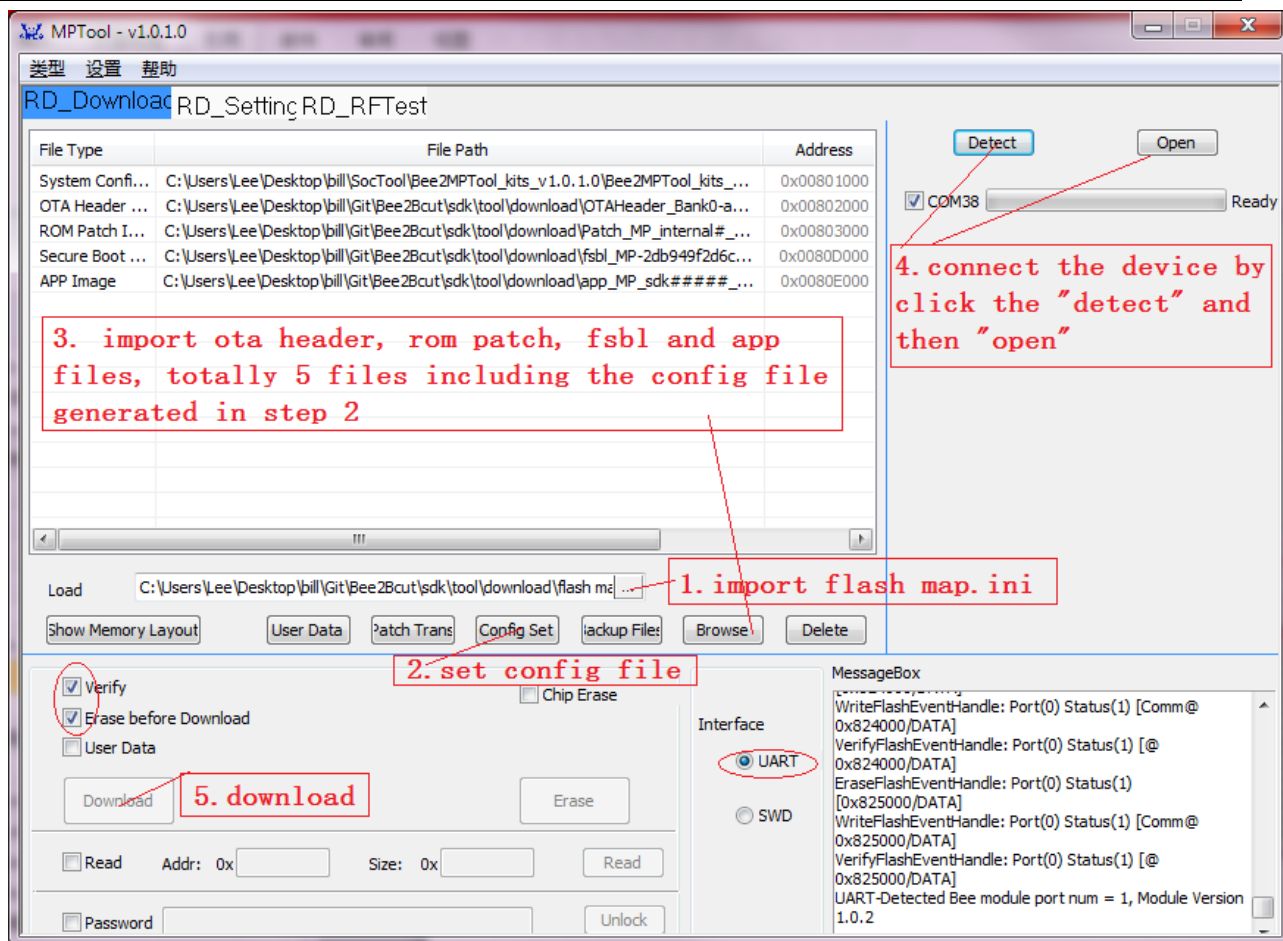


图 4-2 download

1. 导入 Flash layout 配置文件
2. 点击 config set 按钮，生成 config file，详细设定见 4.1.2.3
3. 点击 browse 按钮，导入 ota header、rom patch、fsbl、app 四个文件。

如果使用上述 flash layout 配置，可以使用 Realtek 提供的 ota header 文件。否则，由客户自行使用 flash map tool 和 pack tool 生成 ota header 文件。rom patch 和 fsbl 两个文件由 Realtek 提供。app 文件用 keil 编译生成，例如编译 mesh device 工程后，app 文件在\sdk\board\evb\mesh_device\bin\目录下。注意，使用 mp tool 下载 app 文件时，要选择文件名中包含“MP”字符的文件。

加上步骤 2 生成的 config file，一共需要烧录 5 个文件。

4. 连接设备的 Uart Pin P3_0/P3_1 (Tx/Rx)和 uart 转接板，短接 P0_3 和 GND，重启设备。然后，点击 detect 和 open 按钮，打开设备烧录模式。

（注：点击 open 会占用 uart，会影响其他 tool 使用该 uart，此时可以再次点击 detect，释放 uart 占用）

5. 点击 download 烧录

4.1.2.3 Config file 详细设定

MP tool 会自动在 tool 路径下生成 config file，并将其添加到烧录列表文件里。

Config file 详细设定见图 4-3 和图 4-4。

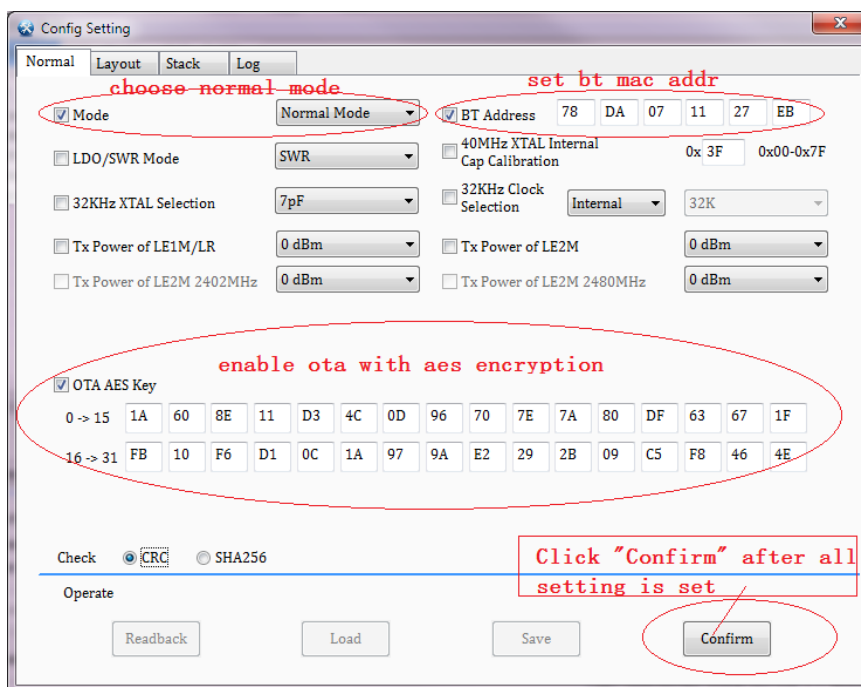


图 4-3 config file setting normal

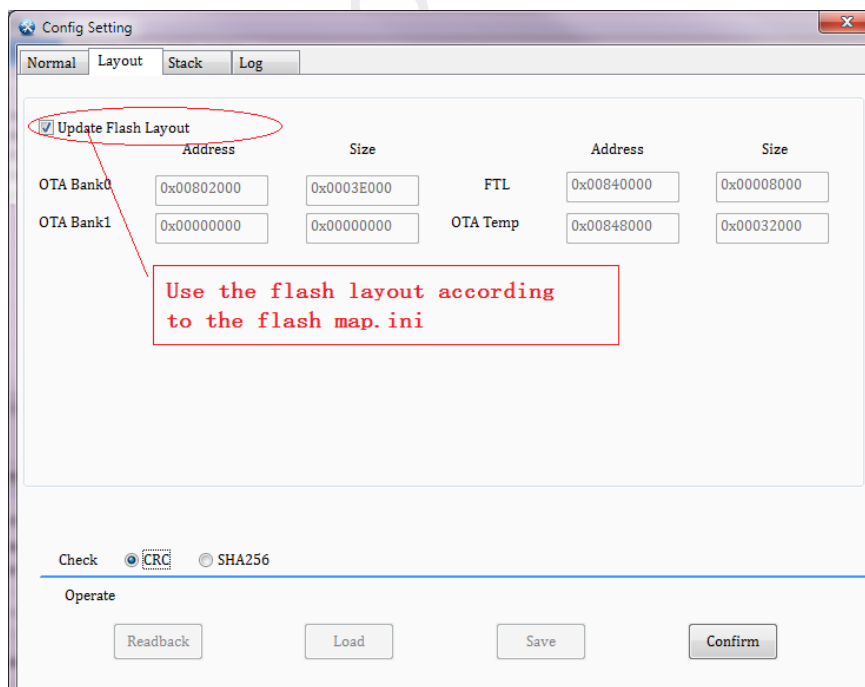
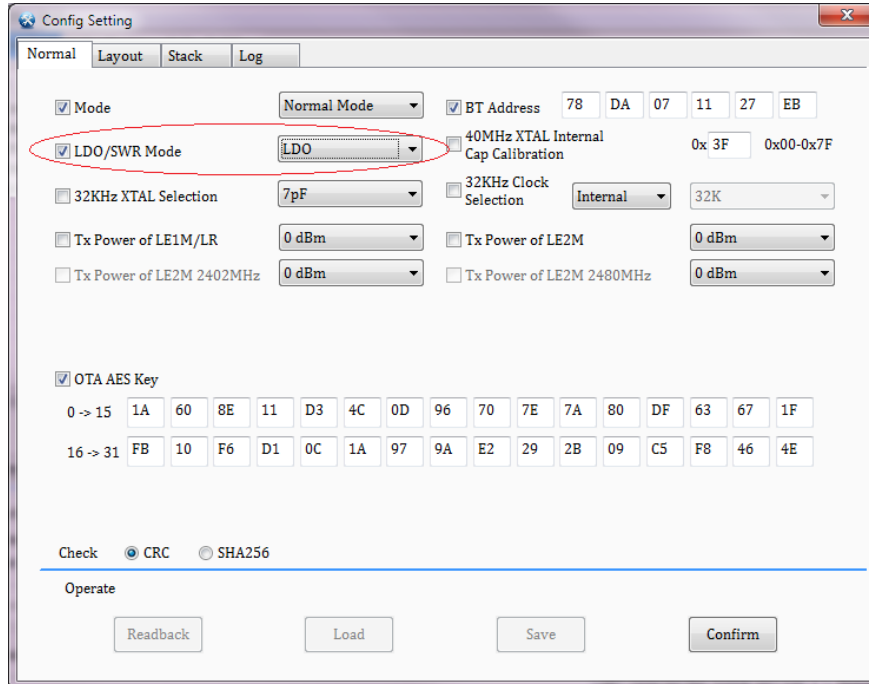


图 4-4 config file setting layout

注意，如果芯片只用 LDO mode（例如 8762CMF 没有上件电感，只能使用 LDO mode），需要在 config

file 配置电源模式，如图 4-5 所示。



The image shows a 'Config Setting' window with several tabs: 'Normal', 'Layout', 'Stack', and 'Log'. The 'Normal' tab is active. It contains various configuration options for the RTL8762C Mesh SDK. A red circle highlights the 'LDO/SWR Mode' dropdown menu, which is currently set to 'LDO'. Other visible settings include 'Mode' (Normal Mode), 'BT Address' (78 DA 07 11 27 EB), '40MHz XTAL Internal Cap Calibration' (0x 3F), '32KHz XTAL Selection' (7pF), '32KHz Clock Selection' (Internal), 'Tx Power of LE1M/LR' (0 dBm), 'Tx Power of LE2M' (0 dBm), 'Tx Power of LE2M 2402MHz' (0 dBm), 'Tx Power of LE2M 2480MHz' (0 dBm), and 'OTA AES Key' (0 -> 15: 1A 60 8E 11 D3 4C 0D 96 70 7E 7A 80 DF 63 67 1F; 16 -> 31: FB 10 F6 D1 0C 1A 97 9A E2 29 2B 09 C5 F8 46 4E). At the bottom, there are 'Check' options for CRC and SHA256, and 'Operate' buttons: Readback, Load, Save, and Confirm.

图 4-5 config file setting power mode

5 Demo 工程

目前共有四个 Demo 工程：mesh_provisioner、mesh_device、mesh_light 和 mesh_ali_light。其中 mesh_provisioner 和 mesh_device 分别扮演 provisioner 和 device 两个角色，可以互相测试 mesh 的各项基本功能。而 mesh_light 工程则是在 device 基础上，实现了 mesh model specification [2]里的 Light HSL models 的具体灯控应用。mesh_ali_light 是对接天猫精灵的 sample。

Provisioner 是网络管理员，它负责在节点入网时分发地址和密钥，以及管理节点。在 provisioner 程序里，会初始化 provisioner 的地址以及网络可用的 NetKey 和 AppKey，如表 5-1 所示。

表 5-1 provisioner setting

```
1. mesh_node.node_state = PROV_NODE;
2. mesh_node.iv_index = MESH_IV_INDEX;
3. mesh_node.ttl = MESH_TTL_DEFAULT;
4. mesh_node.unicast_addr = MESH_SRC;
5.
6. const uint8_t net_key[] = MESH_NET_KEY;
7. const uint8_t net_key1[] = MESH_NET_KEY1;
8. const uint8_t app_key[] = MESH_APP_KEY;
9. const uint8_t app_key1[] = MESH_APP_KEY1;
10. uint8_t net_key_index = net_key_add(0, net_key);
11. app_key_add(net_key_index, 0, app_key);
12. uint8_t net_key_index1 = net_key_add(1, net_key1);
13. app_key_add(net_key_index1, 1, app_key1);
```

mesh_provisioner 和 mesh_device 两个 Demo 工程采用如图 5-1 所示的串口 CLI 命令行方式进行操作。现实应用中，provisioner 很可能是智能手机、平板电脑这样有着更加用户友好的交互方式的设备，由他们完成节点入网管理和配置。而 Demo 工程的 CLI 交互，通过敲命令的方式完成 mesh 操作，主要是为了让开发者能够直观地感受 mesh 应用涉及到哪些流程，便于前期的学习和后期的网络调试。

注意：尽管 light 等工程是没有 CLI 接口的，但还是可以被 mesh_provisioner 来配置管理。

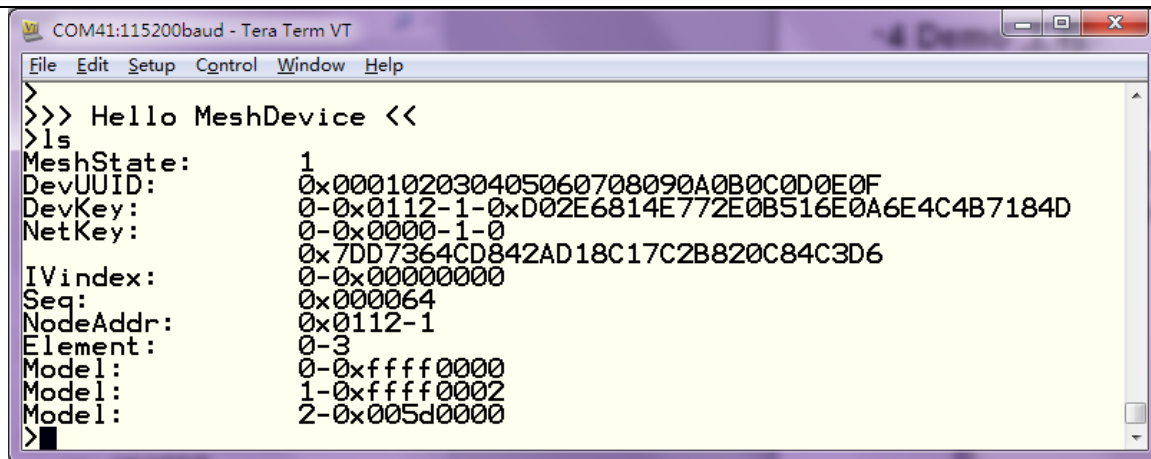


图 5-1 CLI 接口

5.1.1 串口配置

CLI 需要通过 Uart 连接 RTL8762C EVB 和 PC。RTL8762C 串口 pin 的配置: P3_0 为 TX, P3_1 为 RX, 如表 5-2。上位机终端工具的串口配置如图 5-2 所示。

表 5-2 uart setting

1. data_uart_init(P3_0, P3_1, app_send_uart_msg);
2. user_cmd_init("MeshDevice");

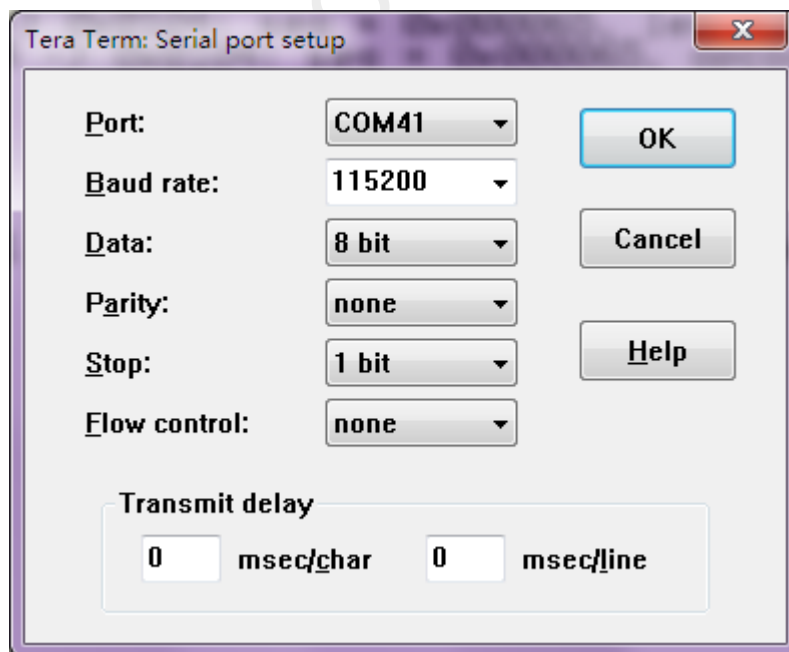


图 5-2 上位机终端串口配置

5.1.2 命令

Device 和 Provisioner 有一部分命令是通用的，参考 mesh_cmd.c 和 mesh_cmd.h。

表 5-3 通用命令

Cmd	用法	含义
?	?	列出所有 cmd 用法和注释
xxx ?	xxx ?	列出 xxx cmd 用法
ls	ls	显示设备各类信息
nr	nr	device 恢复出厂设置
,	,	上一条命令
.	.	下一条命令
[[光标左移
]]	光标右移
/	/	光标移动到行首
\	\	光标移动到行尾
backspace	backspace	删除字符

Device 特有的命令参看 device_cmd.c 和 device_cmd.h。

Provisioner 特有的命令参看 provisioner_cmd.c 和 provisioner_cmd.h。

5.1.3 provisioning

provisioner 可以收集附近节点信息并显示出来，通过 dis 命令开关。

表 5-4 信息收集

Step	Provisioner	说明
1	dis 1	显示周边 mesh 的 device
2	dis 0	搜集到目标 device 的信息后（device UUID 或者 bt address），关闭搜索

device 会在 provisioning 中获得地址和网络密钥，成功后即加入网络，可以进行 mesh 通信。使用 transport 层 ping 功能，即 tping 命令，进行网络层测试，可以与网络中的任意节点互相发送和接收 transport 层的 ping/pong 消息。provisioner 也可以使用 configuration 和指定的 device 进行通信，例如获取 composition data page 0。

注意，已经被 Provisioning 的 device，需要通过 nr 命令恢复 unprovisioned 状态。

pb-adv provisioning 流程如表 5-5 所列。

表 5-5 pb-adv provisioning

Step	Provisioner	Device	说明
1	pbadvcon 000102030405060708090a0b0c0d0e0f		发起对 000102030405060708090a0b0c0d0e0f 的连接，等待提示 pb-adv bearer 是否成功建立
2	prov		等待提示 prov 流程结束
3		ls	查看 device 是否成功获得地址和密钥
4		tping xfff	测试是否能互相进行网络层通信
5	cdg x100		获取 device 0x100 的 composition data page 0

pb-gatt provisioning 流程如表 5-6 所列。

表 5-6 pb-gatt provisioning

Step	Provisioner	Device	说明
1	con deadbeef1234		建立和对端蓝牙地址 0xdeadbeef1234 的 BLE link，等待提示 link 成功建立
2	provdiss		查找 provision service
3	provcmd 0 1		使能 provision cccd
4	prov		等待提示 prov 流程结束
5		ls	查看 device 是否成功获得地址和密钥
6		tping xfff	测试是否能互相通信
7	cdg x100		获取 device 0x100 的 composition data page 0

5.1.4 friendship

已经被 Provision 的 lpn 设备可以开始建立 friendship。

表 5-7 friendship

Step	Provisioner	Device	说明
1	fninit 1 5	lpninit 1	初始化
2		lpnreq 0 0	等待提示 friendship 建立结果
3		tping xfff 3	发现 tping 耗时会变长

5.1.5 model

5.1.5.1 密钥管理

已经被 Provision 的 node 可以被 configuration 配置，添加 application key、绑定 application key 到 ping model 上。然后，ping model 就可以发送 model 层的 ping 消息。

表 5-8 密钥管理

Step	Provisioner	Device	说明
1	aka x100 0 0		给目标节点 0x100 添加 app key 0，并绑定到 net key 0 上
2		ls	查看 device 是否成功添加 app key 0
3	mab x100 0 x5d 0		给目标节点 0x100 的第 0 个 element 的 model id 为 0x5d 的 ping model 绑定 app key 0
4		ls	查看 device 的 model 是否成功绑定 app key 0
5		ping xfff 3	测试 ping control model

5.1.5.2 订阅配置

provisioner 可以管理 device 的 model 订阅地址，即将其加入某个 group。

表 5-9 model 订阅配置

Step	Provisioner	Device	说明
1	msa x100 0 x5d xc000		给目标节点 0x100 的第 0 个 element 的 model id 为 0x5d 的 ping model 添加 subscribe address 0xc000
2		ls	查看 device 的 model 是否成功添加 subscribe address 0xc000
3	ping xc000 3		测试看 device 是否订阅 0xc000

5.1.6 Lighting Control

在操作灯控应用 light 前，请按照 provisioner 配置 device 的操作方式，利用 provisioner 完成对 light 的 provisioning、AppKey 添加、model 绑定 AppKey 的系列操作，此处就不再重复介绍。

注意，因为 light 的 model 比较多，所以配置步骤也比较多，如表 5-10。虽然 model 之间有绑定关系，但只是共享 subscription list，并不共享 AppKey 绑定信息。所以，light 的每个 model 需要独立绑定 AppKey。

如果 model 没有绑定 AppKey，则不能接收和发送消息。另外，通过 configuration 配置 model 时，要指定 model 所在 element，例如 Light HSL Server model 是在 element index 0 上，Light HSL Hue Server model 是在 element index 1 上，而 Light HSL Saturation Server model 则是在 element index 2 上。

配置完的灯泡，要想恢复出厂设置需要通过 3 次快速给灯泡上下电：每次上电后等待 3~8s，然后关闭电源等待灯泡放电完毕。第 4 次上电时，灯泡会恢复出厂设置。

表 5-10 Light Models 配置

Step	Provisioner	说明
1	mab x100 0 x1005d 0	给 light 0x100 的第 0 个 element 的 model id 为 0x1005d 的 vendor light cwrngb model 绑定 app key 0
2	mab x100 0 x1307ffff 0	给 light 0x100 的第 0 个 element 的 model id 为 0x1307 的 sig light hsl model 绑定 app key 0
3	mab x100 0 x1300ffff 0	给 light 0x100 的第 0 个 element 的 model id 为 0x1300 的 sig light lightness model 绑定 app key 0
4	mab x100 0 x1000ffff 0	给 light 0x100 的第 0 个 element 的 model id 为 0x1000 的 sig generic on off model 绑定 app key 0
5	mab x100 0 x1002ffff 0	给 light 0x100 的第 0 个 element 的 model id 为 0x1002 的 sig generic level model 绑定 app key 0
6	mab x100 1 x130affff 0	给 light 0x100 的第 1 个 element 的 model id 为 0x130a 的 sig light hsl hue model 绑定 app key 0
7	mab x100 1 x1002ffff 0	给 light 0x100 的第 1 个 element 的 model id 为 0x1002 的 sig generic level model 绑定 app key 0
8	mab x100 2 x130bffff 0	给 light 0x100 的第 2 个 element 的 model id 为 0x130b 的 sig light hsl saturation model 绑定 app key 0
9	mab x100 2 x1002ffff 0	给 light 0x100 的第 2 个 element 的 model id 为 0x1002 的 sig generic level model 绑定 app key 0

5.1.6.1 私有灯控

Light 工程不仅有 SIG 定义的标准灯控 models，还有一个私有灯控 Light CWRGB model。同样是灯控，Light CWRGB model 非常简单，而 SIG 的 light models 结构和功能都比较复杂，形成对比，供用户参考。CWRGB 分别对应 cold、warn、red、green、blue 共五路 LED。其中 CW 都是白光，用来照明的，可以配置 C 和 W 两路的亮度来调节灯光的色温。而 RGB 则是三原色红绿蓝，可以用来搭配出任意色彩。

Light CWRGB model 是 vendor model，定义了四个 manufacturer-specific opcodes，如表 5-11 所示。

表 5-11 CWRGB Model

1. #define MESH_MSG_LIGHT_CWRGB_GET	0xC45D00
2. #define MESH_MSG_LIGHT_CWRGB_SET	0xC55D00

- | | |
|---|------------|
| 3. #define MESH_MSG_LIGHT_CWRGB_SET_UNACK | 0xC65D00 |
| 4. #define MESH_MSG_LIGHT_CWRGB_STAT | 0xC75D00 |
| 5. | |
| 6. #define MESH_MODEL_LIGHT_CWRGB_SERVER | 0x0001005D |
| 7. #define MESH_MODEL_LIGHT_CWRGB_CLIENT | 0x0002005D |

因为 manufacturer-specific opcode 占 3 字节，而一笔 unsegmented 的 mesh access 消息最长 11 字节，即只剩下 8 字节。所以，Light CWRGB model 的 set 消息中参数长度为 5 字节，CWRGB 各只用 1 字节表征，共 256 阶。

当 light 设备上的 Light CWRGB server model 被配置好后，provisioner 可以通过如表 5-12 所示命令来操作灯泡。

表 5-12 Light CWRGB 操作

Cmd	用法	含义
lrg	lrg 0x100	获取 light 0x100 的 cwrngb 状态
lrs	lrs 0x100 ff0000	完全点亮 red 灯

5.1.6.2 SIG 灯控

SIG 的 Light HSL model 是用色相、饱和度和亮度的方式来控制颜色可调的灯，等效上述 CWRGB 中的 RGB 控制。所以实现上，只要收到 HSL models 的灯控消息，都会将 CW 两路给关掉，将 HSL 值换算成 RGB 值后，操作 RGB 三路 LED。Light HSL Server Model 结构比较复杂，如 mesh model specification [2] 介绍的，它本身需要 3 个 element，每个 element 内又有多个 model，同时扩展了 Light Lightness Server model，而状态间的绑定如图 5-3 红线所示为 Hue、Saturation 和 Lightness 三个标量间的绑定。

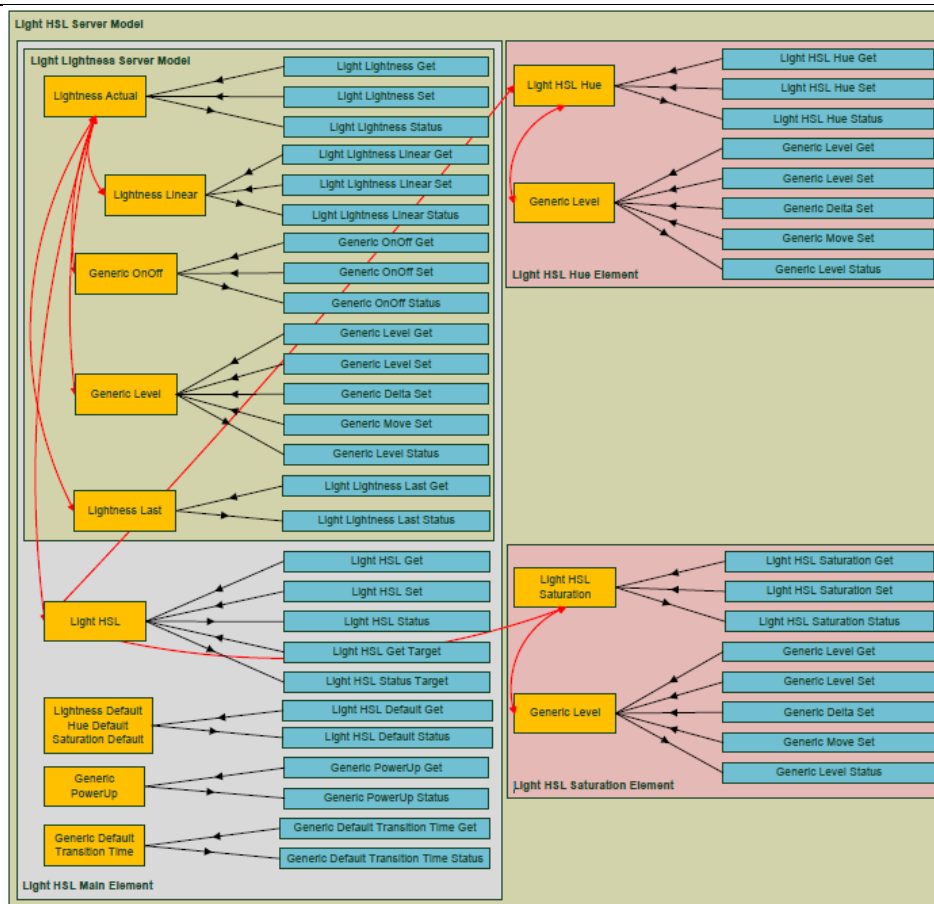


图 5-3 Light HSL Server Models

由于 Light HSL Server Models 很多，对应的消息也很多，不同的消息用不同的方式来控制灯，能达到不同的效果。例如 HSL 本身 models 定义的消息可以三路同时控制，也可以只控制一路。使用扩展的 generic models 可以控制某一路，还可以控制三路的开关。

表 5-13 列出了对应的 light 操作。其中值的注意的是，HSL model 里操作的 Lightness 都是 actual lightness，即感知的 lightness，和实际测量 linear lightness 有一定的差别。另外，单独操作 hue 和 saturation 时，目的地址不是 node address，而是 node address + 1 和 node address + 2，例如 node address 是 0x100，则 hue 对应的 element 操作地址为 0x101，而 saturation 对应的 element 操作地址为 0x102。

表 5-13 Light HSL 操作

Cmd	用法	含义
lhg	lhg 0x100	获取 light 0x100 的 hsl 状态
lhs	lhs 0x100 0 65535 46340	完全点亮 red 灯
lhhg	lhhg 0x101	获取 light 0x100 的色相
lhhs	lhhs 0x101 21845	修改色相为 green
lhsg	lhsg 0x102	获取 light 0x100 的饱和度
lhss	lhss 0x102 0	修改饱和度为 0，即白光

llg	llg 0x100	获取 light 0x100 的亮度
lls	lls 0x100 0	修改量度为 0，即灭灯
lllg	lllg 0x100	获取 light 0x100 的线性亮度
llls	llls 0x100 32767	修改线性亮度为 32767
goog	goog 0x100	获取 light 0x100 的量灭状态
goos	goos 0x100 0	关掉 light

5.1.7 天猫精灵

在阿里 mesh 生态中，天猫精灵当 provisioner，负责管理周边 mesh 设备。阿里定义了天猫精灵 mesh 接入规范，只有符合规范的设备才会被天猫精灵发现和配置，详细内容请参考阿里的 BLE Mesh 蓝牙模块软件规范[3]。

5.1.7.1 三元组

三元组包含 product id、bt address 和 secret 三块。天猫精灵周边设备要想接入天猫精灵，必须获得三元组，三元组由阿里分配。后面会介绍三元组如何合到设备端的 code 中。

sample:

product id = 0x293e2

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

secret = "atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"

5.1.7.2 UUID

mesh 设备会在 unprovisioned 状态下定时发送 UDB 和 provisioning adv，这些消息里携带有设备的 UUID。阿里自定义了 UUID 的格式，需要填充指定内容，天猫精灵才会识别。UUID 格式如表 5-14 所示，主要包含淘宝的 company id 0x01A8，三元组里的 product id 和设备的 bt address。

表 5-14 阿里 uuid 格式

```

1. typedef struct
2. {
3.     uint16_t cid;
4.     struct
5.     {
6.         uint8_t adv_ver: 4;
7.         uint8_t sec: 1;
8.         uint8_t ota: 1;
9.         uint8_t bt_ver: 2; //!< 0 bt4.0, 1 bt4.2, 2 bt5.0, 3 higher
10.    } pid;

```

```

11.     uint32_t product_id;
12.     uint8_t mac_addr[6];
13.     uint8_t rfu[3];
14. } _PACKED_ ali_uuid_t;

```

ali light 工程里，uuid 配置如表 5-15 所示。

表 5-15 配置阿里 uuid

```

1.     /** set device uuid */
2.     ali_uuid_t dev_uuid;
3.     dev_uuid.cid = 0x01A8; //!< taobao
4.     dev_uuid.pid.adv_ver = 1;
5.     dev_uuid.pid.sec = 1;
6.     dev_uuid.pid.ota = 0;
7.     dev_uuid.pid.bt_ver = 1;
8.     dev_uuid.product_id = ALI_PRODUCT_ID;
9.     uint8_t bt_addr[6];
10.    gap_get_param(GAP_PARAM_BD_ADDR, bt_addr);
11.    memcpy(dev_uuid.mac_addr, bt_addr, sizeof(bt_addr));
12.    memset(dev_uuid.rfu, 0, sizeof(dev_uuid.rfu));
13.    device_uuid_set((uint8_t *)&dev_uuid);

```

5.1.7.1 Provision 配置

阿里选择了 static oob authentication data 的 provision 方式。设备端需要支持这种 PROV_CAP_STATIC_OOB 方式，配置如下表 5-16:

表 5-16 prov capability 配置

```

1.     /** configure provisioning parameters */
2.     prov_capabilities_t prov_capabilities =
3.     {
4.         .algorithm = PROV_CAP_ALGO_FIPS_P256_ELLIPTIC_CURVE,
5.         .public_key = 0,
6.         .static_oob = PROV_CAP_STATIC_OOB,
7.         .output_oob_size = 0,
8.         .output_oob_action = 0,
9.         .input_oob_size = 0,
10.        .input_oob_action = 0
11.    };
12.    prov_params_set(PROV_PARAMS_CAPABILITIES, &prov_capabilities,
    sizeof(prov_capabilities_t));

```

具体的 authentication data 是由三元组里的 secret 生成的，如表 5-17 所示。

表 5-17 authentication data 配置

```

1.   case PROV_CB_TYPE_AUTH_DATA:
2.       {
3.           prov_start_p pprov_start = cb_data.pprov_start;
4.           char secret[32];
5.           uint32_t product_id;
6.           if (user_data_contains_ali_data())
7.               {
8.                   user_data_read_ali_secret_key((uint8_t *)secret);
9.                   product_id = user_data_read_ali_product_id();
10.                }
11.            else
12.                {
13.                    memcpy(secret, ALI_SECRET_KEY, 32);
14.                    product_id = ALI_PRODUCT_ID;
15.                }
16.            char data[8 + 1 + 12 + 1 + 32 + 1];
17.            uint8_t auth_data[SHA256_DIGEST_LENGTH]; // Only use 16 bytes
18.            sprintf(data, "%08x", product_id);
19.            data[8] = ',';
20.            uint8_t bt_addr[6];
21.            gap_get_param(GAP_PARAM_BD_ADDR, bt_addr);
22.            sprintf(data + 9, "%02x%02x%02x%02x%02x%02x", bt_addr[5], bt_addr[4],
                bt_addr[3], bt_addr[2],
23.                    bt_addr[1], bt_addr[0]);
24.            data[21] = ',';
25.            memcpy(data + 22, secret, 32);
26.            data[54] = 0; // for log print debug
27.            /*
28.             * sample: product id = 0x293e2, bt addr = AB:CD:F0:F1:F2:F3, secret =
                "atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"
29.             * input: "000293e2,abcdf0f1f2f3,atFY1tGDCo4MQSVCGVDqtti3PvBI5WXb"
30.             * ouput:
                8e-e2-17-bc-02-a5-ab-66-6d-d2-ce-39-5d-f7-20-55-85-4a-f2-7e-c5-c0-45-d9-2a-48-48-99-
                74-3a-dc-9f
31.             * authvalue: 8e-e2-17-bc-02-a5-ab-66-6d-d2-ce-39-5d-f7-20-55
32.             */
33.            printi("sha256 input: %s", TRACE_STRING(data));
34.            SHA256_CTX sha_ctx;
35.            SHA256_Init(&sha_ctx);

```

```
36.         SHA256_Update(&sha_ctx, data, sizeof(data) - 1);
37.         SHA256_Final(&sha_ctx, auth_data);
38.         printi("sha256 output: %b", TRACE_BINARY(SHA256_DIGEST_LENGTH,
auth_data));
39.         switch (pprov_start->auth_method)
40.         {
41.             case PROV_AUTH_METHOD_STATIC_OOB:
42.                 prov_auth_value_set(auth_data, 16);
43.                 APP_PRINT_ERROR1("prov_cb: Please exchange the oob data(%b) with the
provisioner", TRACE_BINARY(16,
44.                             auth_data));
45.                 break;
```

5.1.7.1 Model 配置

天猫精灵 provision 结束后，会根据 product 类型配置 model。目前，只有 model 配置完成了，天猫精灵才会认为配网成功。所以，要咨询阿里了解相关产品所需要实现的 model。

6 FAQ

1. MP tool 无法识别设备烧录 hci 串口，点击 detect/open 后，提示 fail。
 - 先 P0_3 接地，后重启，才能进入烧录模式。进入烧录模式后，不再需要短接 P0_3 和地。
 - 注意 P3_0/P3_1 和 uart 转接板连接顺序。
 - uart 转板是否支持 1M 波特率（强烈建议使用 FT232 芯片），否则 open 时更新 baudrate 时会失败。
 - 重新上电。
 - 重启 tool。
2. log tool 的 log 异常。
 - uart 转板是否支持 2M 波特率。
 - 注意 log pin P0_3 是否接对了，uart 转板是否和开发板一起共地。
 - 需要在 log tool setting 里指明 app.trace 位置如图 6-1 所示，才能正常解析 log。app.trace 在工程目录底下，例如 “.sdk\board\evb\mesh_ali_light\bin\App.trace”。

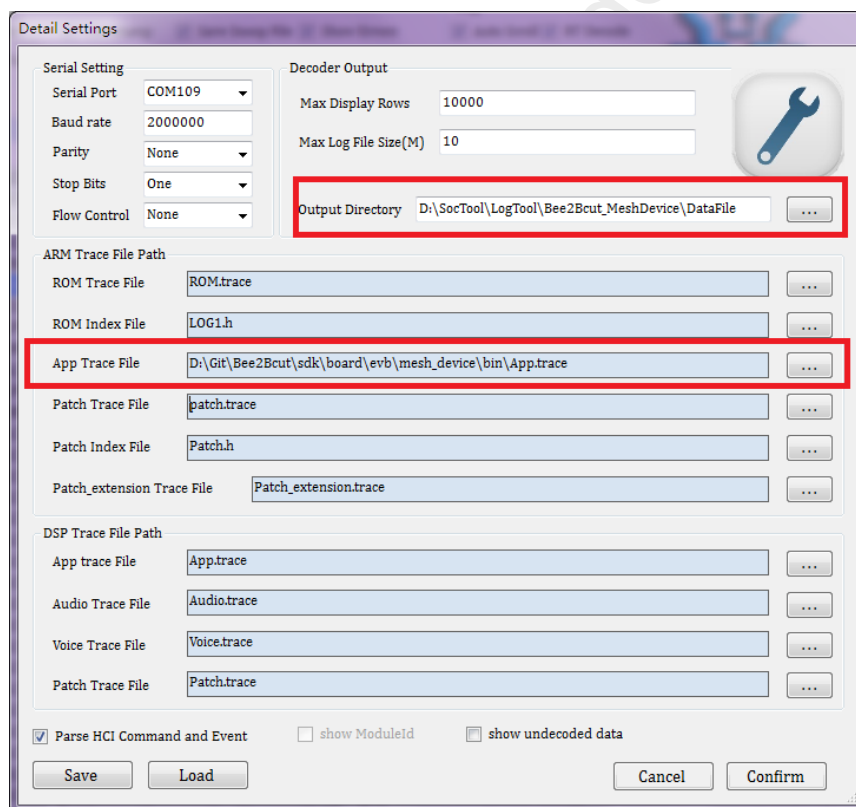


图 6-1 log tool setting

- app.trace 是否正确。如果改动了代码，app.trace 会伴随着 image 编译每次重新生成的。如果 app.trace 不对应，log tool 解析出来的 log 就会乱码。如果修改了 app.trace，即改动了 code，

需要在 Log tool 上 stop 再 start 一下，这样新 app.trace 才会重新加载。

- log 会自动分段保存在如图 6-1 所示 output directory 里。log 文件格式有两种：.log 明文格式和.bin 密文格式。.bin 格式 log 是 Realtek 内部 log 文件，需要搭配 app.trace 才能使用。如果需要 Realtek 分析 log 时，尽量提供.bin 格式 log，并且提供 app.trace 文件。

3. 程序启动异常。

- erase 整个芯片，重新烧录。
- 换板子，确认是否和硬件绑定，即个别现象。
- 收集 Log，确认复现手法，call for help。

Realtek Confidential

参考文献

- [1] [Mesh Profile Specification](#)
- [2] [Mesh Model Specification](#)
- [3] [阿里 BLE Mesh 蓝牙模块软件规范](#)

Realtek Confidential

附录

Realtek Confidential