



Telink

Telink SIG Mesh

SDK Developer Handbook

AN-17120400-E5

Ver14.0

2023.03.17

Keyword

SIG Mesh

Brief

This document is Telink SIG Mesh SDK Developer Handbook.

Published by
Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,
Zhangjiang Hi-Tech Park, Shanghai, China

© Telink Semiconductor
All Rights Reserved

Legal Disclaimer

This document is provided as-is. Telink Semiconductor reserves the right to make improvements without further notice to this document or any products herein. This document may contain technical inaccuracies or typographical errors. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein.

Copyright © 2023 Telink Semiconductor (Shanghai) Co., Ltd.

Information

For further information on the technology, product and business term, please contact Telink Semiconductor Company www.telink-semi.com

For sales or technical support, please send email to the address of:

telinksales@telink-semi.com

telinksupport@telink-semi.com

Revision History

Version	Changes
V1.0.0	Initial release.
V1.1.0	This is the second release, compare with last version, the following parts have been updated: 1. SDK Overview; 2. Global Configuration Introduction; 3. 8268 Mesh Project Introduction; 4. Provisioner(Gateway) Project Introduction; 5. SWITCH Project Introduction
V1.2.0	This is the third release, compare with last version, the following parts have be updated: 1 SDK Overview; 4 Debugging Tool Instruction; 11 Mesh LPN Project Introduction; 8 Global Configuration File Introduction. The following parts are added: 2 MCU Basic Modules; 5 Factory Test Mode; 6 Important SDK Modules; 7 Vendor Model Introduction; 9 8258 MESH Project Introduction; 13 Connect with a Platform; 14 Factory Reset; 15 SIG Remote provision Demo; 16 Fast bind Mode(PROVISION_FLOW_SIMPLE_EN Mode); 17 Private Fast provision Function Demo; 18 Private online status Function Demo; 19 OTA Test Brief; 20 Network Sharing; 21 Control Nodes via INI Demo
V1.3.0	This is the fourth release, compare with last version, the following parts have be updated: Delete draft feature
V1.4.0	Minor edits and corrections.

Contents

Revision History	3
1 SDK Overview	9
1.1 SDK File Architecture	9
1.1.1 main.c	10
1.1.2 app_config.h	11
1.1.3 BLE stack entry	11
1.2 Demo Project	12
1.3 LIGHT_TYPE_SEL Introduction	13
1.4 Version ID(VID) and Product ID(PID) Configuration	15
1.5 Mesh Application Packet Tx/Rx Processing	16
1.5.1 Packet Transmission Function	16
1.5.2 Packet Transmission Flow	16
1.5.3 Packet Reception Flow	18
1.5.4 Packet Reception Callback Function Introduction	18
1.5.5 SIG_mesh Channel	19
1.6 Telink Debug Method Introduction	19
1.6.1 Tdebug Tool Debugging	19
1.6.2 Log Print Debugging	21
2 MCU Basic Modules	25
2.1 Flash and RAM map	25
2.1.1 Flash map Introduction (512K flash)	25
2.1.2 RAM map (8258 64K)	25
2.1.3 Stack overflow check	27
2.2 Clock	27
2.2.1 System clock & System Timer	27
2.2.2 System Timer Usage	29
3 MESH Spec Introduction	31
3.1 Layered architecture	31
3.1.1 Model layer	32
3.1.2 Foundation Model layer	32
3.1.3 Access layer	32
3.1.4 Transport layer	32
3.1.5 Network layer	32
3.1.6 Bearer layer	33
3.2 Architectural concepts	33
3.2.1 States	33
3.2.2 Bound states	33
3.2.3 Messages	33
3.2.4 Node & Elements	33
3.2.5 Models	33
3.2.6 Publish & subscribe	34
3.2.7 Security	34
3.2.8 Sequence Number Storage	34
3.2.9 Friendship	35

3.2.10	Features	35
3.2.11	Mesh Topology	36
3.3	Mesh networking	36
3.3.1	Network layer	36
3.3.2	Access layer	38
3.3.3	Transport layer	39
3.3.4	Mesh beacon	39
3.3.5	iv update flow	39
3.3.6	Heartbeat	40
3.3.7	Health	40
4	Debugging Tool Instructions	43
4.1	Download Firmware	43
4.2	BLE Connection and Adding Light in Gateway USB Mode	48
4.3	BLE Connection and Adding Light in Gateway UART Mode	53
4.4	BLE Connection and Adding Light in GATT master dongle Mode	53
4.5	Control Corresponding Nodes	54
4.5.1	UI Display and on/off Control of Single/All Node(s)	54
4.5.2	Group Control (Subscription Demo)	56
4.5.3	Configure Node Parameter with UI	57
4.6	Time model operation	62
4.7	Scene model operation	64
4.8	Scheduler model operation	67
5	Factory Test Mode	71
5.1	Purpose	71
5.2	Factory Test Mode Parameters	71
5.3	Default Test-able Commands	71
6	Important SDK Modules	72
6.1	Configure Mesh SDK Default Feature	72
6.2	Share model introduction	72
6.3	Heartbeat demo	73
6.4	Mesh Receiving Transmitting Self-defined Packet	74
7	Vendor Model Introduction	76
7.1	Adding vendor model	76
7.2	Adding vendor command register reference	76
7.2.1	mesh_cmd_sig_func_t introduction	76
7.2.2	Add acknowledge command (request command with status response)	77
7.2.3	Add Unacknowledged command	78
7.2.4	Publish function registration	79
8	Global Configuration File Introduction	80
8.1	mesh_config.h	80
8.2	mesh_node.h	83
8.3	app_mesh.h	83
8.3.1	Macro introduction	83
8.3.2	Function introduction	84
8.4	app_provision.c	85
8.5	mesh_node.c	85
8.6	mesh_common.c file introduction	85

8.7	cmd_interface.h file introduction	90
8.8	vendor_model.c file introduction	91
8.9	mesh_test_cmd.c file introduction	92
9	8258 MESH Project Introduction	93
9.1	app_config_8258.h	93
9.2	app.c file introduction	94
9.2.1	Customization of Adv packet and Adv response packet	94
9.2.2	Configuration of fifo part	94
9.2.3	app_event_handler ()	94
9.2.4	main_loop ()	95
9.2.5	user_init()	95
9.2.6	void proc_ui()	96
9.3	app_att.c file introduction	96
9.4	light.c file introduction	96
10	Provisioner (Gateway) Project Introduction	101
10.1	Provisioner Function Introduction	101
10.1.1	adv-bearer and gatt-bearer	101
10.2	Provisioner Principle	101
10.2.1	Command Interaction of Provisioner	101
10.2.2	Timing Sequence Chart of adv Provisioner	102
10.2.3	Timing Sequence Chart of gatt Provisioner	105
10.3	app.c file introduction	106
10.4	Provisioner operation and ports	107
10.4.1	Format of SIG_MESH_TOOL ini file	107
10.4.2	SIG model format, g_all_on as an example	108
10.4.3	Vendor Model Format, CMD-vendor_on as example	109
10.4.4	Burn Nodes	109
10.4.5	Add Light via Provisioner	110
10.4.6	app_key binding	114
10.4.7	Light on/off Control	115
10.4.8	Provisioner Control Flow Chart	117
11	Mesh LPN Project Introduction	118
11.1	LPN Node and Implementation Method	118
11.1.1	LPN and friend	118
11.1.2	Friendship Parameters	118
11.1.3	Establish Friendship	119
11.1.4	Friendship Message Exchange	120
11.1.5	Security	121
11.1.6	Friendship Termination	121
11.2	LPN Demo	121
11.2.1	Hardware	121
11.2.2	Test method	122
11.3	app.c file introduction	125
11.4	mesh_lpn.c file introduction	125
12	SWITCH Project Introduction	127
12.1	Switch function introduction	127
12.2	Switch principle	127

12.3	app.c file introduction	127
12.4	Configuration of Switch Part	128
12.4.1	key table	128
12.4.2	Configure IOs for Drive Pins and Scan Pins	128
12.4.3	Turn on/off Light via Switch	129
12.5	Switch Operation Guide	130
12.6	Flow chart for Switch RC	132
12.7	Flow chart for sleep processing	133
13	Connect with a Platform	134
13.1	Normal Mode	134
13.1.1	No OOB provision mode	134
13.1.2	static OOB provision mode	134
13.1.2.1	Light Node Burn Static oob	134
13.1.2.2	Light node Device uuid	134
13.1.2.3	User Customized uuid Method	135
13.1.2.4	Provisioner static oob database	135
13.1.2.5	Test steps	136
13.2	Ali Tmall Genies Platform	137
13.2.1	Configuration	137
13.2.2	Apply tri-truple from Ali	138
13.2.3	Use SDK Default tri-truple	138
13.2.4	Provision via Tmall Genie	138
13.2.5	Provision via Firmware	139
13.2.6	Dual Modes of static oob and no oob	139
13.3	Xiaomi Xiao'ai Platform	139
13.3.1	Configuration	139
13.3.2	Certification Data Setting	140
13.3.3	Provision Test	140
13.4	Dual Vendor Mode (Tmall Genies and Xiaomi Xiaoai)	140
13.4.1	Function Introduction	140
13.4.2	Configuration	141
14	Factory Reset	142
14.1	8258_mesh/8269_mesh Node	142
14.1.1	Function Introduction	142
14.1.2	Default trigger action	142
14.1.3	Method to modify power-on sequence	143
14.1.4	After the reset action is triggered, the function of the previous mesh network can be restored	144
14.2	Gateway Node + firmware	144
14.3	GATT master dongle + Firmware	145
14.4	LPN Node	145
14.5	Switch Node	146
15	Fast bind Mode (PROVISION_FLOW_SIMPLE_EN Mode)	147
15.1	Function Introduction	147
15.2	Configuration	147
15.3	Function Demo	147
15.3.1	Firmware Configuration	147

15.3.2 APP Interface Configuration	148
16 Private Fast provision Function	149
16.1 Function Introduction	149
16.2 Configuration	149
16.3 Function Demo	149
17 Private online status function demo	152
17.1 Function Introduction	152
17.2 Configuration	152
17.3 Packet Format	152
17.4 GATT Master Dongle Firmware Demo	154
18 OTA Test Brief	155
18.1 GATT master dongle OTA for firmware update of BLE directly connected nodes	155
18.2 OTA OTA where the GATE WAY node updates its firmware	157
19 Network Sharing	159
19.1 APP Provision by Gateway or GATT master dongle, then share with APP	159
19.2 Provision by APP mode, then share with Gateway or GATT master dongle	163
20 Control Nodes via INI Demo	166
20.1 Provision Device	166
20.2 Configuration Operations	170
20.2.1 Key add /bind Operation	170
20.2.2 Subscription Configuration	172
20.2.3 publish configuration	172
20.2.4 Relay/Friend Function Configuration	172
20.2.5 Heartbeat setting	173
20.3 Control Operations	173
20.3.1 Control Generic model Demo	173
20.3.2 CTL model	174
20.3.3 HSL model	175
20.3.4 Vendor model	176
20.3.5 Gateway Transmit Long Packet to LPN	177

1 SDK Overview

Telink SIG Mesh SDK supplies demo code for SIG_mesh protocol application development, based on which user can develop his own application program.

Current projects of the SDK apply to Telink IC 825x/8278/8269, Telink provides PC tool (sig_mesh_tool.exe) connecting master dongle through USB to realize provisioner function, please note, master dongle supports only 8269 SoC series. For demo of basic functions, please refer to:

[BLE SIG Mesh Quick Start](#)

1.1 SDK File Architecture

File architecture for Telink SIG Mesh SDK includes APP (application) layer and BLE&SIG_mesh protocol layer. After the SDK project is imported in Telink IDE (please refer to AN_IDEUG-E1_Telink IDE User Guide.pdf for project importing, AN_16063000-E1_Guide for Adding New Project on Existing SDK.pdf for adding new project), the file structure is shown in figure below, containing the following top-layer folders.

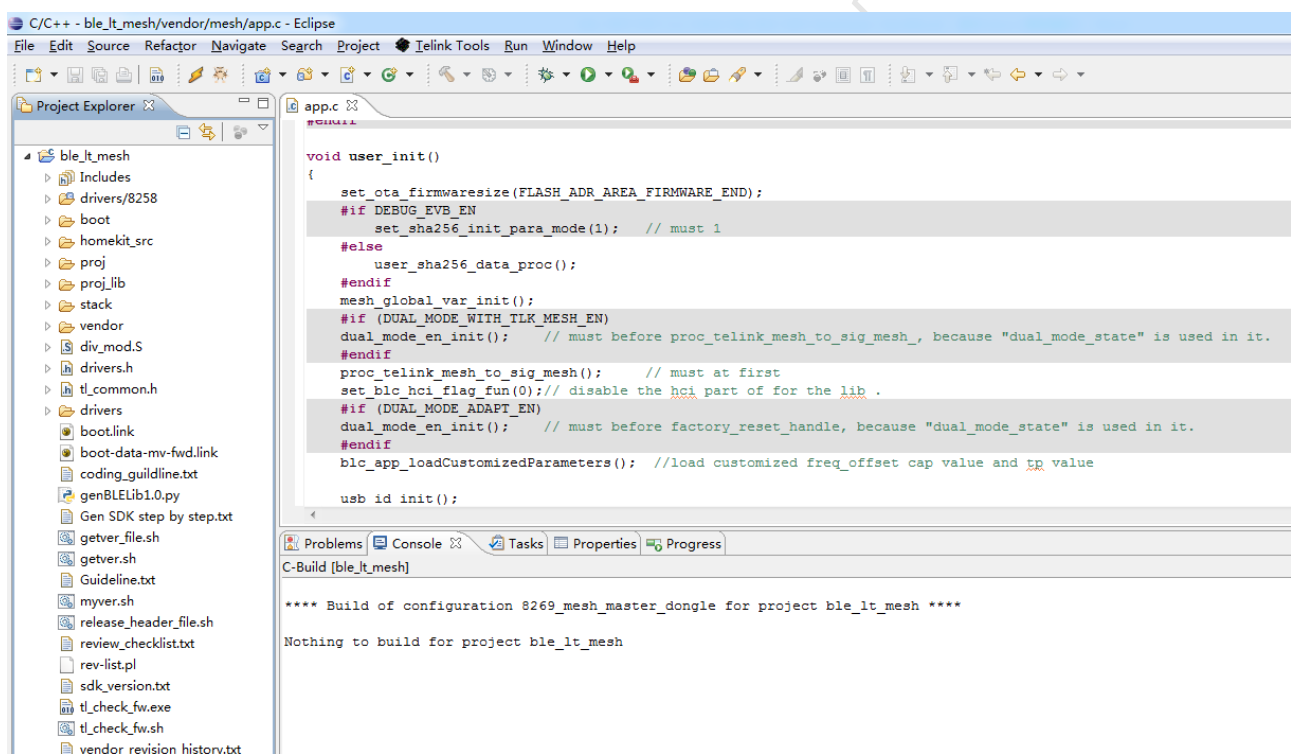


Figure 1.1: File Architecture

- **boot**: This folder contains bootloader of the SoC, i.e., the compiling process after MCU boot or awake from deep sleep, this is the environment base for later C programs.
- **drivers**: This folder contains configuration files and driver programs for hardware peripherals related with MCU, like clock, flash, i2c, usb, gpio, uart and etc.
- **proj**: This folder contains MCU related peripheral driver, such as flash, I2C, USB, GPIO, UART driver, and etc.

- **proj_lib**: This folder contains library files necessary for MCU running, e.g. BLE stack, RF driver, PM driver. Since this folder is supplied in the form of library files, the source files are not open to users, e.g. BT stack library file "libbt_8269_mesh.a", library file for SIG_mesh common node "libsig_mesh.a", library file for SIG_mesh low power node "libsig_mesh_LPN.a", library file for SIG_mesh provision node "libsig_mesh_prov.a".
- **stack**: This folder contains BLE protocol related header files. The source files are compiled into library files, and are not open to users.
- **vendor**: This folder contains user APP-layer code, including:
 - **8267_master_kma_dongle**: Firmware used for host test. In combination with host tool (sig_mesh_tool.exe) of GATT mode, it can act as a provisioner and it's used for demonstration and debugging.
 - **common**: It mainly contains common modules in mesh/ mesh_lpn/ mesh_provision/ mesh_switch, e.g. SIG mesh model processing, LED module, factory initialization module, test command module.
 - **mesh/ mesh_gw_node_homekit/ mesh_lpn/ mesh_provision/ mesh_switch/ spirit_lpn** have the same structure, which all include "app.c", "app.h", "app_att.c", "app_config.h" and "main.c". "app.c/app.h" contains initialization and bottom-layer callback function; "app_att.c" is description of BT ATT table and interface functions; "app_config.h" defines corresponding macros and declarations in projects; main.c is main function and interrupt function entry.

1.1.1.1 main.c

This includes the main function entry, system initialization related functions, and the infinite loop while(1), it is not recommended to verify this file, please follow the fixed codes.

```
int main (void) {
FLASH_ADDRESS_CONFIG;
#ifdef PINGPONG_OTA_DISABLE
    ota_fw_check_over_write(); // no-pingpong-OTA firmware
#endif
bld_pm_select_internal_32k_crystal(); //choose internal 32k rc as the clock source of 32k
↪ counter
cpu_wakeup_init();//MCU basic hardware initialization
int deepRetWakeUp = pm_is_MCU_deepRetentionWakeUp(); //wake up from deep retention or not?
rf_drv_init(RF_MODE_BLE_1M); //RF initialization
gpio_init(!deepRetWakeUp); //gpio initialization, user configure related parameters in
↪ app_config.h
clock_init(SYS_CLK_16M_Crystal);
if( deepRetWakeUp ){
    user_init_deepRetn ();//fast initialization when wake up from deep retention
}
else{
    user_init_normal ();//ble initialization, system initialization, user configure
}
irq_enable(); //enable interrupt
while (1) {
```

```

#if (MODULE_WATCHDOG_ENABLE)
    wd_clear(); //clear watch dog
#endif
    main_loop (); //include ble send/receive, low power management, mesh and user tasks
}
}

```

1.1.2 app_config.h

User configure file to configure all system related parameters, including BLE parameters, GPIO parameters, PM low power management configure parameters and etc.

The definition of each parameter in app_config.h in later parts of this document when each module is introduced.

1.1.3 BLE stack entry

There are 2 entry functions of BLE stack code in Telink BLE SDK

- a) BLE related interrupt handler entry of irq_handler in main.c file irq_blt_sdk_handler.

```

_attribute_ram_code_ void irq_handler(void)
{
    .....
    irq_blt_sdk_handler ();
    .....
}

```

- b) BLE logic and data processing function entry in application file main_loop blt_sdk_main_loop.

```

void main_loop (void)
{
    mesh_loop_proc_prior();//process with high priority, leap over the 10ms interval mailoop
    ↪ 8269
    //////////////////// BLE entry //////////////////////
    blt_sdk_main_loop();

    //////////////////// UI entry //////////////////////
    factory_reset_cnt_check();//5 times of boot factory reset
    mesh_loop_process();//mesh related loop function
    .....

    //////////////////// PM configuration //////////////////////
    .....
}

```

1.2 Demo Project

Telink SIG MESH SDK provided multiple BLE demo.

Users can check the result of each demo, or verify the demo code to develop your own applications.

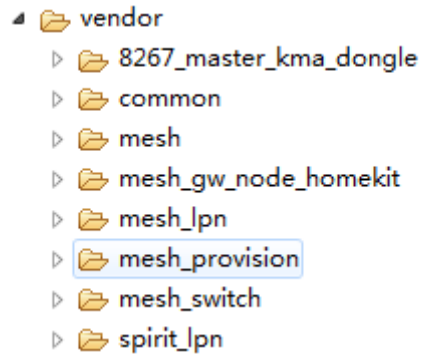


Figure 1.2: MESH SDK demo code

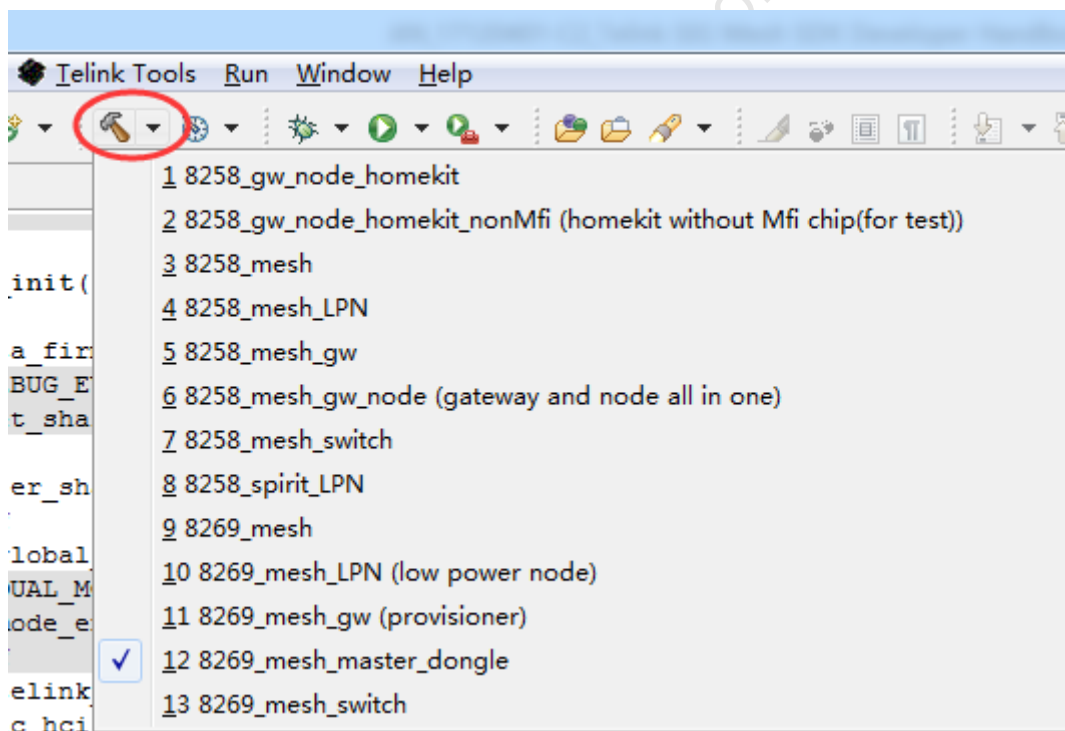


Figure 1.3: MESH SDK compiling options

The differences of each mesh demo are listed in the following table.

Demo	Vendor folder	Application	Mesh Feature
8258_mesh/ 8269_mesh	.\ mesh	CT/HSL light and etc	Relay, friend, proxy

Demo	Vendor folder	Application	Mesh Feature
8258_mesh_LPN\ 8269_mesh_LPN	.\ mesh_lpn	LPN	LPN, proxy
8258_mesh_gw\ 8269_mesh_gw	.\ mesh_provision	Gateway provisioner	adv provisioner, Relay, friend
8258_mesh _gw_node	.\ mesh_provision	Gateway+light node	adv provisioner, Relay, friend, proxy
8258_mesh_switch\ 8269_mesh_switch	.\ mesh_switch	RC	proxy
8258_spirit_LPN	.\ spirit_lpn	TMall Genies LPN mode	proxy
8258_gw_node _homekit	.\ mesh_gw_node \ _homekit	Gateway+light node+homekit	adv provisioner, Relay, friend, proxy
8269_mesh_ master_dongle	.\ 8267_master _kma_dongle	Tool	GATT provisioner

- 8269_mesh_master_dongle compiling option: supports GATT provisioner function, no rely nor friend function. Firmware uses this for host test, together with GATT mode, the firmware can act as a provisioner to demonstrate and debug.
- 8258_mesh, 8269_mesh compiling option: compiling project of normal SIG Mesh nodes, can be configured by provisioner, supports relay, friend, proxy function, no provision function.
- 8258_mesh_LPN, 8269_mesh_LPN compiling option: compiling project of LPN MESH nodes, receive message via friendship, does not support relay, friend, proxy, provision functions.
- 8258_mesh_gw, 8269_mesh_gw compiling option: compiling project of gateway provisioner nodes, supports adv provisioner, can configure other nodes, supports relay, friend functions.
- 8258_mesh_switch, 8269_mesh_switch compiling option: compiling project of (switch) MESH nodes. To lower power consumption, after the switch is provisioned, it send message with other receiving. Does not support relay, friend function.
- 8258_gw_node_homekit, 8258_gw_node_homekit_nonMfi compiling option: supports gateway adv provisioner + mesh node + homekit function.
- 8258_spirit_LPN compiling option: self-defined LPN mode of TMALL Genies.

1.3 LIGHT_TYPE_SEL Introduction

This macro is used to choose the pre-configured light types.

```
#define LIGHT_TYPE_NONE      0
#define LIGHT_TYPE_CT        1
#define LIGHT_TYPE_HSL       2
#define LIGHT_TYPE_XYL       3
```

```
#define LIGHT_TYPE_POWER          4
#define LIGHT_TYPE_CT_HSL        5
#define LIGHT_TYPE_DIM            6 // only single PWM
#define LIGHT_TYPE_PANEL          7 // only ON/OFF model
#define LIGHT_TYPE_LPN_ON/OFF _LEVEL 8 // only ON/OFF , LEVEL model
```

LIGHT_TYPE_CT:

CT is the abbreviation of color temperature light, and the corresponding product is color temperature light, contains color temperature related model, e.g., Light CTL Server, Light CTL Setup Server, Light CTL Temperature Server, and corresponding extend model, e.g. Generic On/off Server, Generic Level Server, Light Lightness Server and etc.

LIGHT_TYPE_HSL:

The corresponding product is RGB light, contains Light HSL Server, Light HSL Hue Server, Light HSL Saturation Server, Light HSL Setup Server and corresponding extend model, e.g. Generic On/off Server, Generic Level Server, Light Lightness Server and etc.

LIGHT_TYPE_XYL:

The corresponding product is XYL light, contains Light xyL Server, Light xyL Setup Server and corresponding extend model, e.g. Generic On/off Server, Generic Level Server, Light Lightness Server and etc.

LIGHT_TYPE_POWER:

The corresponding product is power adapter, contains generic Power Level Server, Generic Power Level Setup Server and corresponding extend model, e.g. Generic On/off Server, Generic Level Server and etc.

LIGHT_TYPE_CT_HSL:

The corresponding is CT light + HSL Light, contains CT and HSL related model, and Generic On/off Server, Generic Level Server, Light Lightness Server. CT light use the same lightness and on/off parameter with HSL light. Only one light is lighted at one time.

LIGHT_TYPE_DIM:

The corresponding light is dimming light, contains Light Lightness Server, Light Lightness Setup Server and the corresponding extend model, e.g. Generic On/off Server, Generic Level Server.

LIGHT_TYPE_PANEL:

The corresponding product is switch panel, which is the server, controlled by instruments like app and executing on/off switch. The default switch number is 3(defined by LIGHT_CNT).

LIGHT_TYPE_LPN_ONOFF_LEVEL:

The corresponding product is LPN equipment, contains Generic On/off Server model by default, and mesh OTA model is disabled. This is mainly for demo LPN function.

Note:

- When use LPN equipment, 825x retention RAM size should not exceed 32K.
- All models in node will appear in composition data (global variable model_sig_cfg_s_cps).

14 Version ID(VID) and Product ID(PID) Configuration

Configure file: vendor -> common -> version.h, this file will be also used in compiling codes.

For example:

```
#define MESH_PID_SEL          (LIGHT_TYPE_SEL)
#define MESH_VID             (VERSION_GET(0x33, 0x30))
#define FW_VERSION_TELINK_RELEASE (VERSION_GET(0x33, 0x30))
```

- 1) PID and VID in Composition data are defined by MESH_PID_SEL, MESH_VID
- 2) The 3rd to 6th bits in firmware file is the PID and VID here.

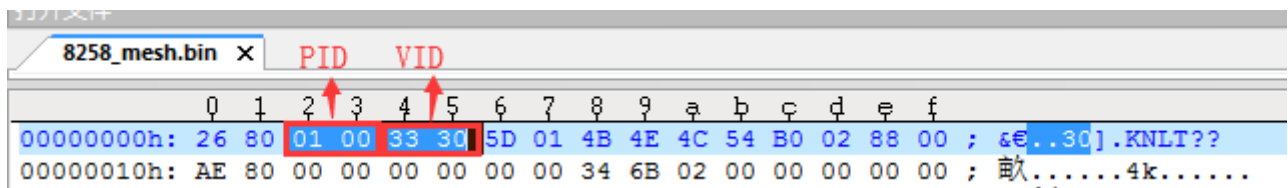


Figure 1.4: PID and VID

- 3) It showed in ATT UI of general APP in the following way:

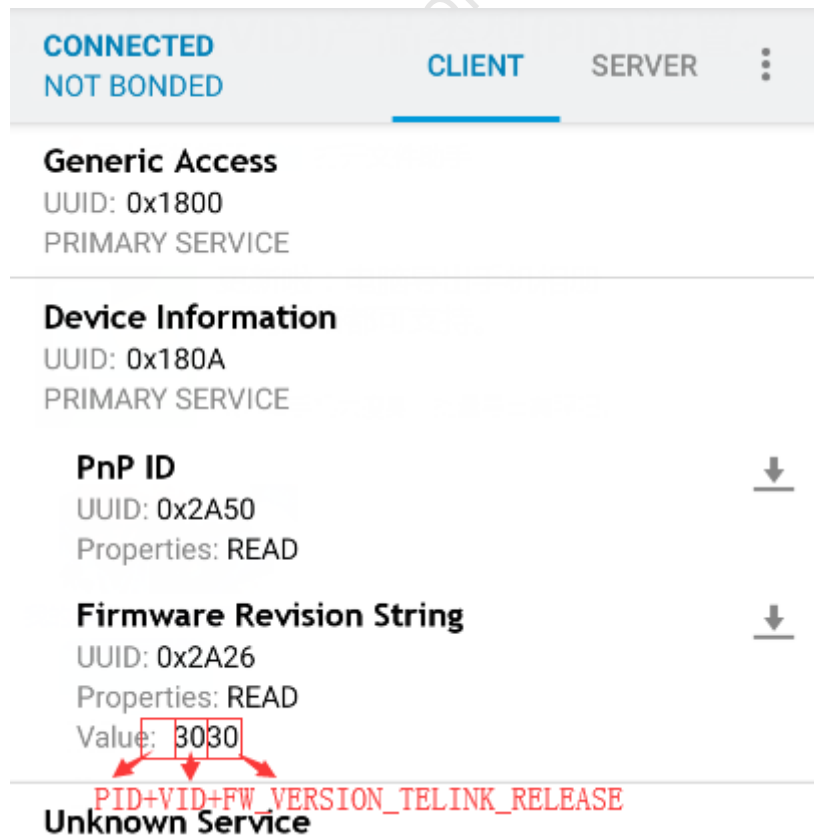


Figure 1.5: ATT user interface

MESH_PID_SEL(PID): general APP will show information in ASCII, so "0x00 0x01" is not visible. Users need to verify it to their own PID.

MESH_VID(VID): "0x33, 0x30" is shown as "30" in ASCII. Users need to verify it to their own PID.

FW_VERSION_TELINK_RELEASE: "0x33, 0x30" is shown as "30" in ASCII. This is the version ID when Telink release the SDK, users should not verify this.

- 4) Users can verify MESH_PID_SEL and MESH_VID according to their own requirement.

1.5 Mesh Application Packet Tx/Rx Processing

1.5.1 Packet Transmission Function

Packet Transmission between Nodes

Data transmitted by invoking the function "mesh_tx_cmd2normal_primary()" follows SIG mesh protocol. Commands such as "access_cmd_on/off ()" are derived by assembling the "mesh_tx_cmd2normal_primary()".

Developers can enable the function "sim_tx_cmd_node2node()" (it's masked by default) to demonstrate command transmission. The effect is: After power on, "ON" command and "OFF" command are automatically and alternately sent with the interval of three seconds. This function will be introduced in detail in subsequent section.

Master Packet Transmission from Directly-Connected Node to Master

Packet transmission from directly-connected node to Master will invoke the "bls_att_pushNotifyData()". Please refer to section 3.4.3.10 in Telink document "AN_17092700_Telink 826x BLE SDK Developer Handbook" for its usage guide. This method serves to send data of customized format.

Note:

- New UUID should be introduced when employ this method, otherwise it may conflict with current UUID protocol. Users can define new UUID in my_Attributes_provision[]/my_Attributes_proxy[]/my_Attributes[] to define BLE service.

1.5.2 Packet Transmission Flow

Note: red font shows library functions.

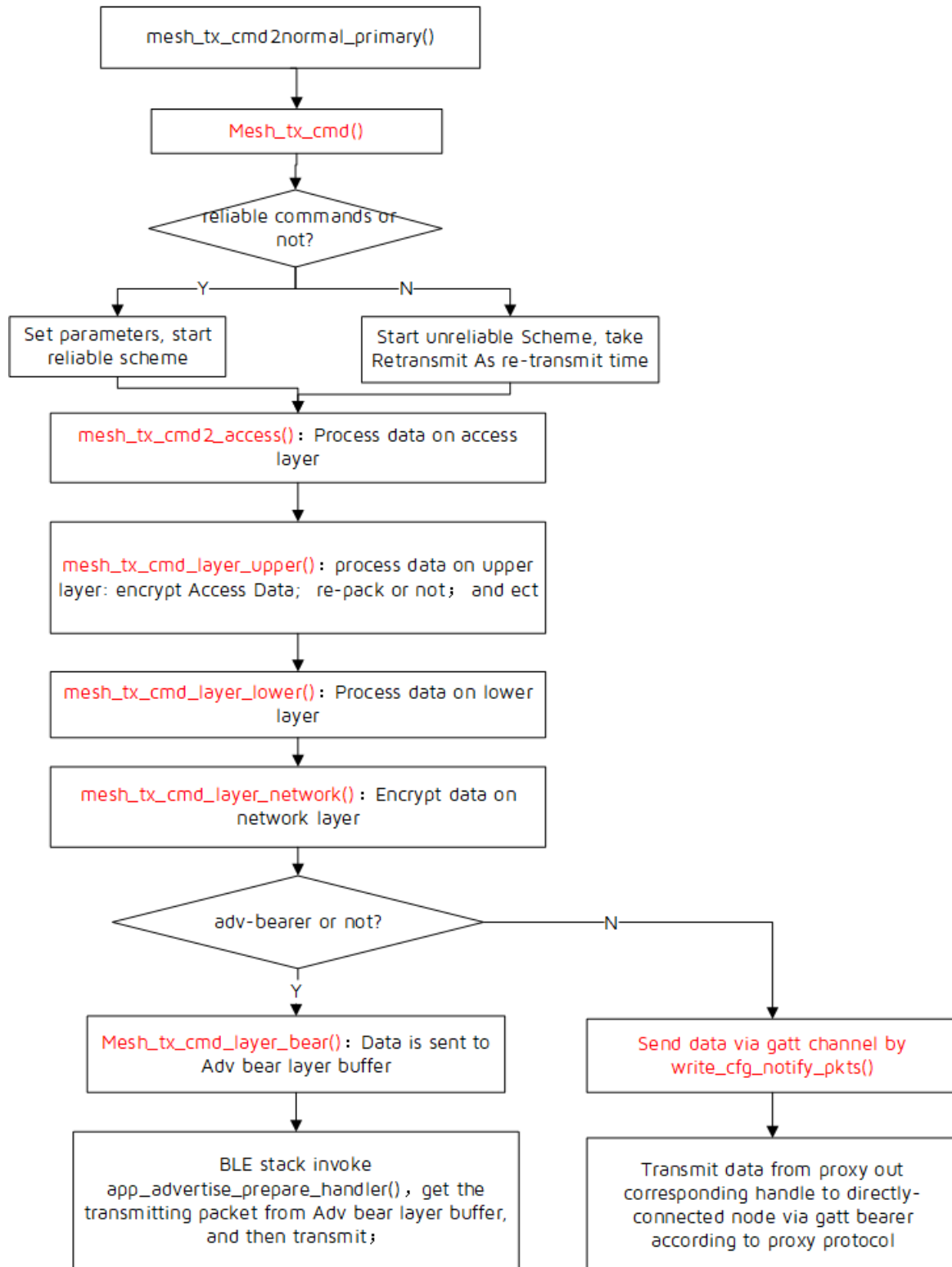


Figure 1.6: Packet Transmission Flow

1.5.3 Packet Reception Flow

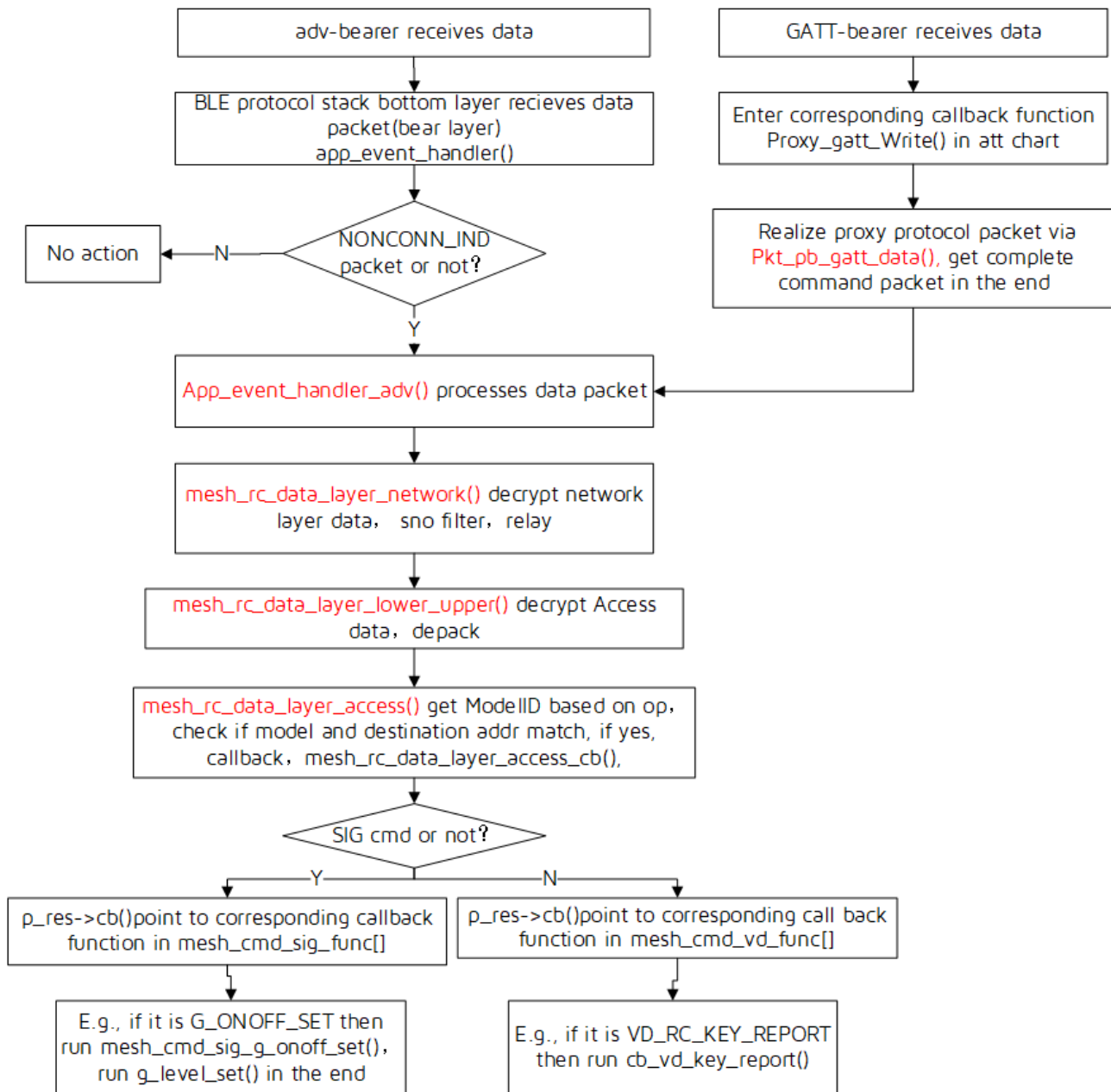


Figure 1.7: Packet Reception Flow

1.5.4 Packet Reception Callback Function Introduction

generic model:

The interface of generic model is in the file "vendor/common/generic_model.c". For callback function, please see the structure "mesh_cmd_sig_func[]". E.g. generic on message command. After this command is received, as specified in the packet reception flow, procedure will finally flow to the "mesh_cmd_sig_g_on/off _set()", in which user implements the effect of turning on/off light or setting gradient parameter. The gradient effect is processed in the "light_transition_proc" and the processing

interval is LIGHT_ADJUST_INTERVAL(20ms).

vendor model:

The interface of vendor model is in the file "vendor/common/vendor_model.c". Refer to "mesh_cmd_vd_func[]". User can add vendor command as needed. If such command is received, as specified in the SIG mesh spec, procedure will flow to corresponding callback function, e.g. "cb_vd_key_report()".

1.5.5 SIG_mesh Channel

SIG_mesh supports two types of communication channels:

adv-bearer: Implement mutual communication based on advertising mechanism, no need to establish BLE connection, the communication channel is the standard 37/38/39.

gatt-bearer: Implement communication based BLE connection, the communication channel is 1-36.

1.6 Telink Debug Method Introduction

1.6.1 Tdebug Tool Debugging

This is a stable/reliable debugging method with no effect of MCU performance, it can check/verify global variables on real time; it can also check running status of functions. To check if a function is executed or not, user can define a global variable, then count, and by checking the global variable via Tdebug, user can know if the function is executed or not. E.g.:

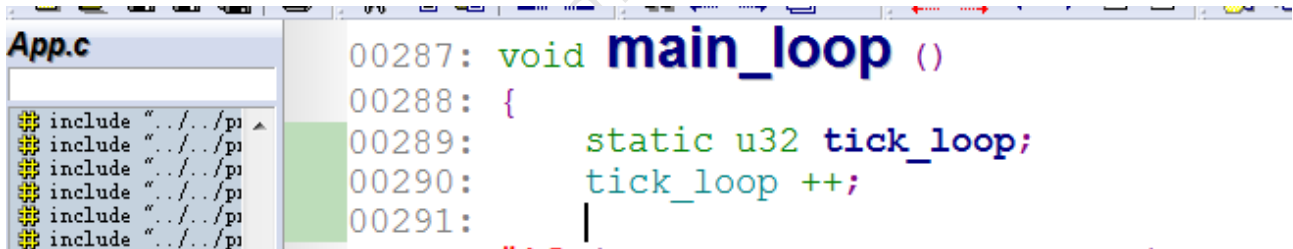


Figure 1.8: Check global variable via Tdebug

Tdebug is described as following, please check "Help" -> "User guide" in BDT tool for detail.

Note:

- right click in the 8th step to get this manual, as shown in figure below.

Sort the 9th step by name, then find tick_loop (tick_loop is static variable, the name maybe duplicate in the codes, so the compiler add a suffix of .12397 to distinguish), right click, then click Refresh, it will read all global variables and refresh, the value of tick_loop will refresh, too.

If you want to change global variables, change in value chart, then press enter. To read back, right click, then press Refresh.

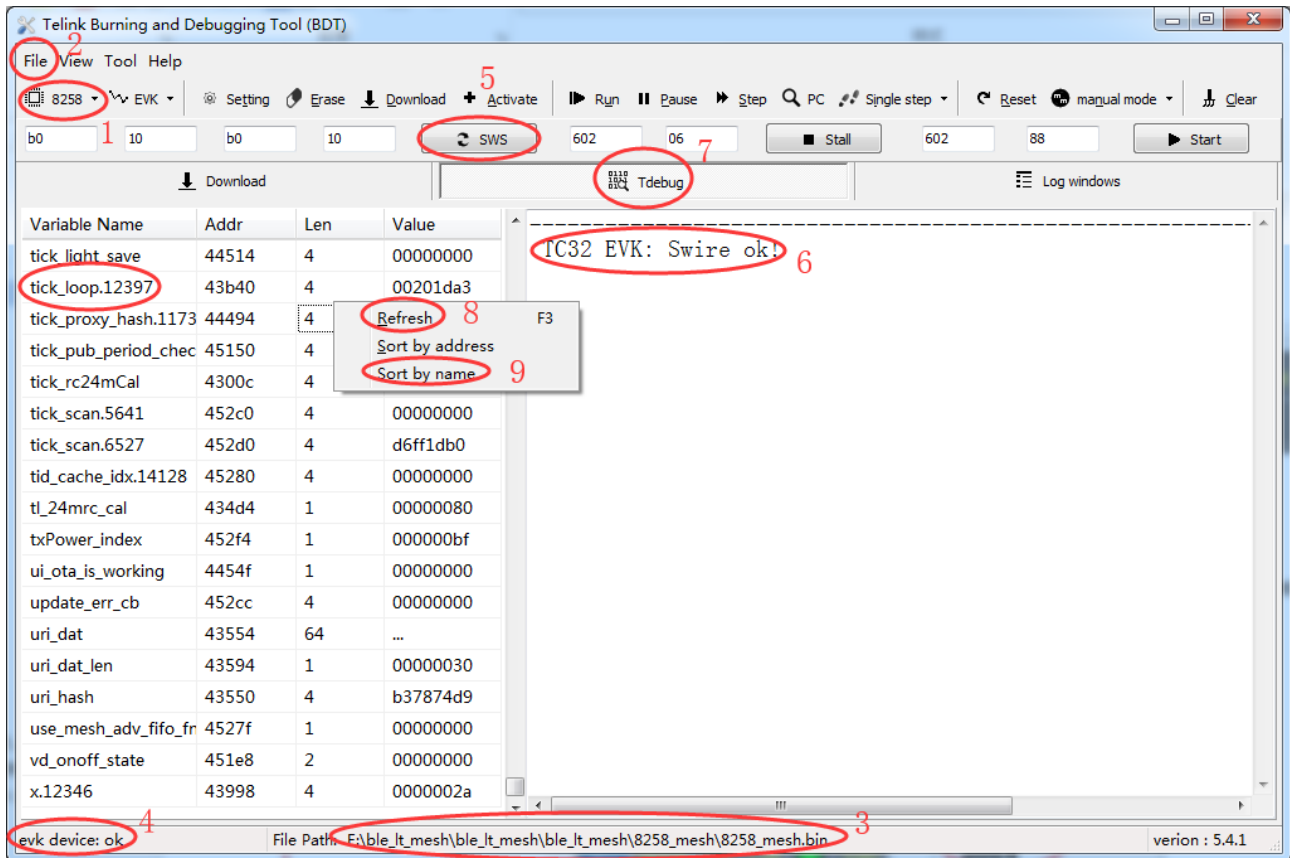


Figure 1.9: Tdebug overview

To read structure variables/arrays longer than 4 bytes but shorter than 1K bytes, click "... " in 1, and the read value will show in 2.

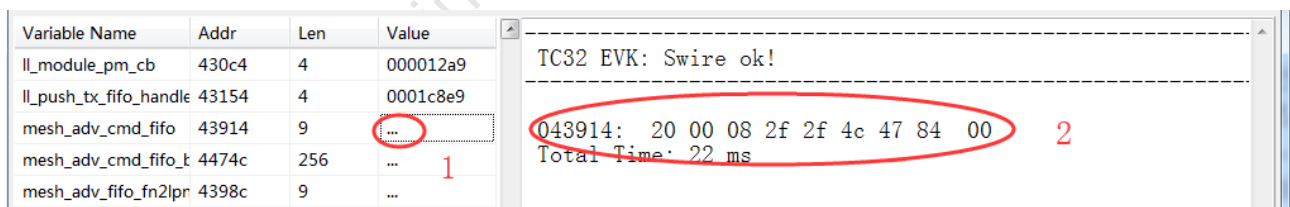


Figure 1.10: Read structure variables or arrays

If it is longer than 1k byte, a file will be generated and saved to Telink Burning and Debugging Tool -> config -> user -> Read.bin in BDT tool.

Variable Name	Addr	Len	Value	
blt_notify_fifo_b	4484c	2304	...	1
blt_ota_finished_flag	4449b	1	00000000	
blt_ota_finished_time	444c0	4	00000000	
blt_ota_start_tick	45490	4	00000000	
blt_ota_terminate_flag	444a0	1	00000000	
blt_ota_timeout_us	43630	4	01c9c380	

1024 bytes have finished!
2048 bytes have finished!
2304 bytes have finished!
All 2304 bytes have been saved!
Total Time: 152 ms

Figure 1.11: Read.bin file

1.6.2 Log Print Debugging

SDK versions after 2.9 support log print via uart output port simulated by GPIO, the default speed is 1Mbps. Please be noted, to guarantee the correctness of log output, when run log output, `irq_disable()` is executed by default, so too much log data will slow down MCU performance and RF packet processing, as well as make the mesh unstable, thus harm the debugging procedure. So please use log output as little as possible, and shut as many log as possible after the debug is finished.

Log can configure print level(`TL_LOG_LEVEL`) and print module(`TL_LOG_SEL_VAL`).

To rule out unnecessary logs in firmware by default, `TL_LOG_LEVEL` is set to `TL_LOG_LEVEL_ERROR`, only print level less than or equal to `TL_LOG_LEVEL_ERROR` will be printed. `TL_LOG_LEVEL_LIB` is for printing library codes, or important non-library-code log. `TL_LOG_LEVEL_USER` is for user, and is not in library. It is recommended to use `LOG_USER_MSG_INFO()` for printing. To use `LOG_MSG_INFO()`, please verify `TL_LOG_LEVEL`.

```

#define TL_LOG_LEVEL_DISABLE 0
#define TL_LOG_LEVEL_USER 1U // never use in
#define TL_LOG_LEVEL_LIB 2U // it will not be
#define TL_LOG_LEVEL_ERROR 3U
#define TL_LOG_LEVEL_WARNING 4U
#define TL_LOG_LEVEL_INFO 5U
#define TL_LOG_LEVEL_DEBUG 6U
#define TL_LOG_LEVEL_MAX TL_LOG_LEVEL_DEBUG

.....

#define TL_LOG_LEVEL TL_LOG_LEVEL_ERROR

```

Figure 1.12: Print level

Print module (i.e., `TL_LOG_SEL_VAL`): to print this module, the corresponding module like `TL_LOG_USER` should be included.

```
typedef enum{
    TL_LOG_MESH           = 0,
    TL_LOG_PROVISION      = 1,
    TL_LOG_LOWPOWER       = 2,
    TL_LOG_FRIEND         = 3,
    TL_LOG_PROXY          = 4,
    TL_LOG_GATT_PROVISION = 5,
    TL_LOG_WIN32          = 6,
    TL_LOG_GATEWAY        = 7,
    TL_LOG_KEY_BIND       = 8,
    TL_LOG_NODE_SDK       = 9,
    TL_LOG_NODE_BASIC     = 10,
    TL_LOG_REMOTE_PROV    = 11,
    TL_LOG_CMD_RSP        ,
    TL_LOG_COMMON         ,
    TL_LOG_CMD_NAME       ,
    TL_LOG_NODE_SDK_NW_UT ,
    TL_LOG_IV_UPDATE      ,
    TL_LOG_USER           , // never use in library.
    TL_LOG_MAX,
} ? end printf_module_enum ? printf_module_enum;
```

Figure 1.13: Print module

The log will be print only when the corresponding print level and Print module are right.

Log Printing Setup

Step 1 Define HCI_LOG_FW_EN as 1

PRINT_DEBUG_INFO means to use GPIO to simulate UART, and it can only support TX UART, but not RX UART.

Step 2 Set print pin

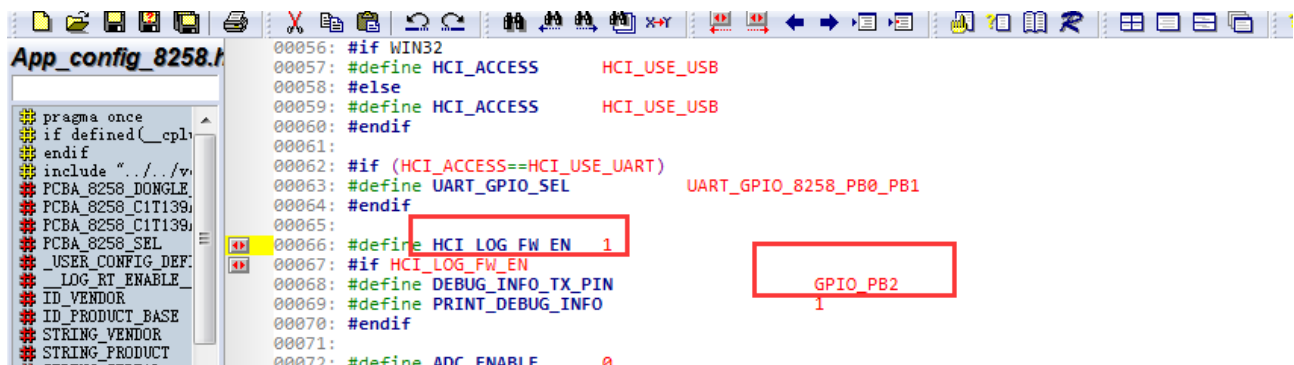


Figure 1.14: Set print pin

Step 3 Set the baud rate. Default we use 1 Mbps.

When simulate UART output by IO, the interrupt is disabled (SIMU_UART_IRQ_EN=1), so the speed of log is the fast the better under this mode, do not reduce band rate.

Note:

- some USB adapter board may have serious mis-alarmed when the speed is under 1M rate, to avoid this, user can reduce UART speed(which will reduce log printing speed), or change USB adapter, e.g. change to CH340G.

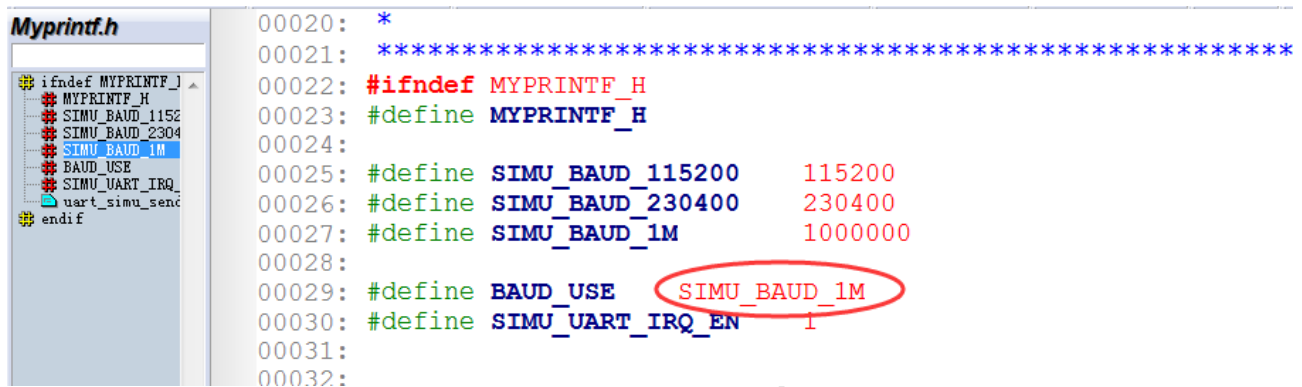


Figure 1.15: Set baud rate

Step 4 Choose log module

In the grading log definition, besides the above print level, there is log module. To print log, print level and log module should both be set to the right value.

To simplify log in Demo SDK, for TL_LOG_SEL_VAL, only TL_LOG_USER is enabled, other log are disabled.

TL_LOG_USER is not be called anywhere by default, to add print, user may run the following commands:

LOG_USER_MSG_INFO(pbuf, len, format,...)

Where:

- pbuf: for transferring a buffer into character and printing. If there is no, set this to 0.
- len: length of pbuf.

If other API is needed, e.g. LOG_MSG_INFO, check TL_LOG_LEVEL first.

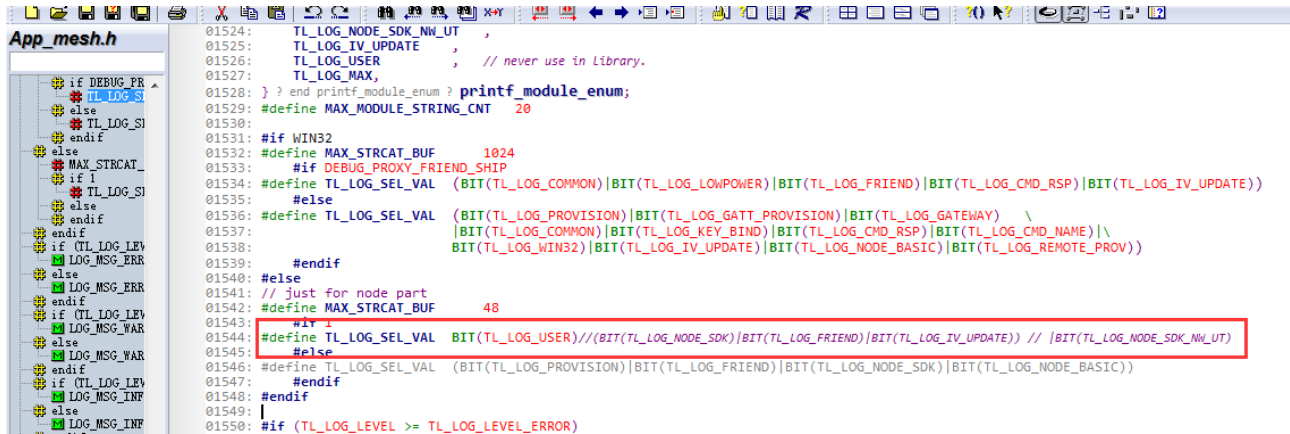


Figure 1.16: Choose log module

Step 5 To open pre-set debug log, set TL_LOG_SEL_VAL, e.g.:

```

#define TL_LOG_SEL_VAL (BIT(TL_LOG_USER)|BIT(TL_LOG_PROVISION)|BIT(TL_LOG_FRIEND)|
BIT(TL_LOG_NODE_SDK)|BIT(TL_LOG_NODE_BASIC))

```

2 MCU Basic Modules

This part introduces mesh related information, for detail, please refer to MCU Basic Modules in AN_19011501-C2_Telink Kite BLE SDK Developer Handbook.

2.1 Flash and RAM map

2.1.1 Flash map Introduction (512K flash)

0x80000	user data area
0x78000	
0x77080	32kRC
0x77041	tp high (1/28269)
0x77040	tp low (1/28269)
0x77000	frequency offset
0x76000	mac address
	para area
0x74000	
0x73000	OTA type flag
	para area
0x70000	OTA new bin storage area
0x40000	para area
0x30000	old firmware bin
0x00000	

8269/8258-SIG mesh

Figure 2.1: Flash Map

2.1.2 RAM map (8258 64K)

Check `./boot.link` for detailed configuration file.

0x840000	64K RAM
	vector
	ram_code
	cache(2.25K)
	data (retention)
less than 0x848000	bss (retention)
	data (no retention)
	(irq stack) bss (no retention)
	normal stack
0x850000	

Figure 2.2: RAM Map

- data(retention): contains variables with attribute_data_retention prefix, and all global variables whose initial value is not 0, default attribute is retention data.
- bss(retention): contains variables with attribute_bss_retention prefix, and all global variables whose initial value is 0, default attribute is retention bss.
- data(no retention): contains global variables with attribute_no_retention_data prefix.
- bss(no retention): contains global variables with attribute_no_retention_bss. Also contains irq_stack, whose size is defined by IRQ_STK_SIZE, default is 0x300, the size is smaller than 0x200 in demo SDK.
- normal stack: after bss, before 64K RAM. The initial value is 0xffffffff, to speed up RAM initialization, only 3K RAM will be initialized.

Note:

- attribute_bss_retention and attribute_no_retention_bss are not applicable to variables whose initial value is not 0, otherwise the initialization is invalid, default value is 0.

2.1.3 Stack overflow check

First, bss(no retention) end address must be less than RAM ending address, i.e., reserve space for normal stack.

End of bss(no retention) can sort variables by address via tdebug Tool, the address right after the last variable is the end address.

It can also be calculated by the *.lst file generated by compiling.

Then, test all functions, check if normal stack and irq_stack overflow.

Normal stack: the initial value is 0xffffffff. Demo SDK need a stack of around 2K, because all used stack is less than 3K, to speed up RAM initialization, only 3K RAM will be initialized. Check by reading RAM, if the 61K address, i.e., 0x84F400–0x84F403 is not 0xffffffff, that means the stack is over 3K, contact us for further confirmation and analysis.

irq_stack: the initial value is 0x00000000, check irq_stk[] via tdebug, if irq_stk[0–3] is not 0, then it overflows.

2.2 Clock

MCU clock is defined by CLOCK_SYS_CLOCK_HZ. The default value is 16MHz for 825x, and 32MHz for 8269.

2.2.1 System clock & System Timer

System clock is the clock of MCU programs.

System timer is a read-only timer, providing timing for BLE time sequence, as well as for users.

For Telink 826x IC, the time source of system timer is system clock; for Telink 8x5x IC, system timer is separated with system clock. As shown in figure below, system timer is 16M, generated by 2/3 divider from the external 24M Crystal Oscillator.

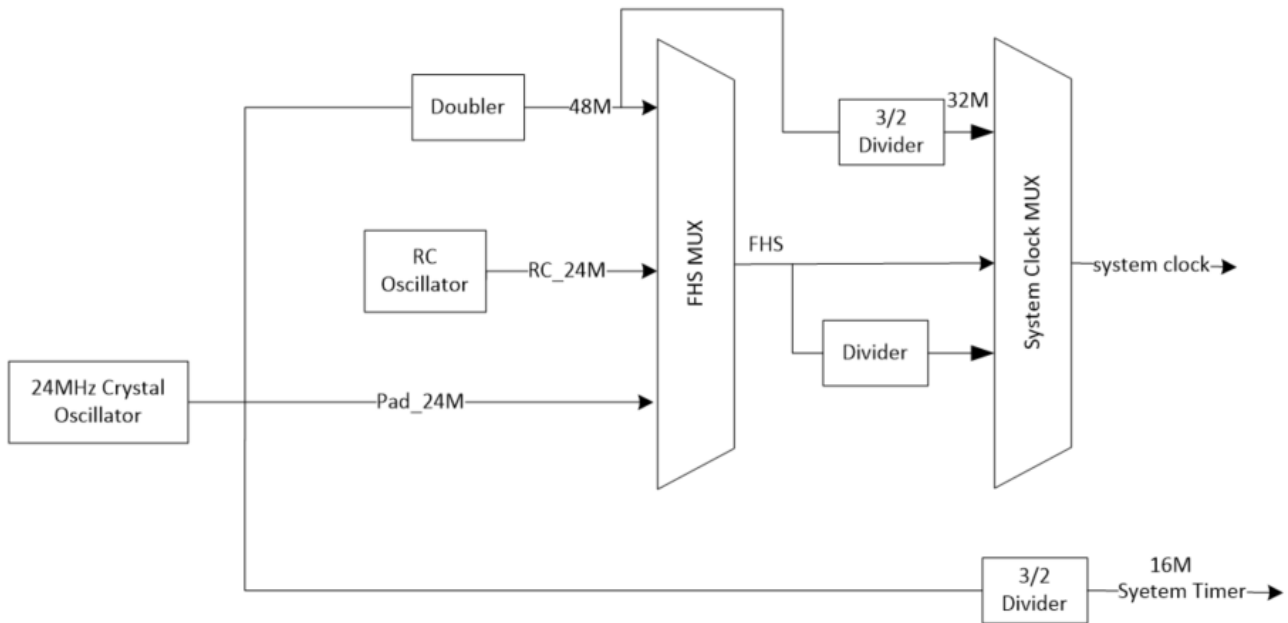


Figure 2.3: System Clock & System Timer

As can be seen in above figure, the external 24M Crystal Oscillator will be double to 48M, then divided by the dividers and generate 16M/24M/32M/48M as system clock, such clocks are called crystal clock (e.g., 16M crystal system clock, 24M crystal system clock); internal 24M RC Oscillator can also generate 24M RC clock, 32M RC clock and 48M RC clock, which we call RC clock(BLE SDK does not support RC clock).

For BLE SDK, we recommend crystal clock.

Call the following API configuration system clock when initialization, choose corresponding clock in the definition of enum variable SYS_CLK_TYPEDEF.

```
void clock_init(SYS_CLK_TYPEDEF SYS_CLK)
```

8x5x System Timer is different from system clock, user should know the clock source of each hardware module in MCU, whether it is system clock or system timer. The following case is an example, where the system clock is 24M crystal, system clock is 24M, and system timer is 16M.

In app_config.h, the definitions of system clock and the corresponding S, mS, uS are shown as below:

```
#define CLOCK_SYS_CLOCK_HZ      24000000
enum{
    CLOCK_SYS_CLOCK_1S = CLOCK_SYS_CLOCK_HZ,
    CLOCK_SYS_CLOCK_1MS = (CLOCK_SYS_CLOCK_1S / 1000),
    CLOCK_SYS_CLOCK_1US = (CLOCK_SYS_CLOCK_1S / 1000000),
};
```

All hardware modules whose clock source is system clock must use only CLOCK_SYS_CLOCK_HZ, CLOCK_SYS_CLOCK_1S when set module clock; in other words, if the module's clock set is the clock defined above, then the clock source of the module is system clock.

E.g., in PWM driver, PWM cycle and interval are set as following, means the clock source of PWM is system clock.

```
pwm_set_cycle_and_duty(PWM0_ID, (u16) (1000 * CLOCK_SYS_CLOCK_1US), (u16) (500 *  
↪ CLOCK_SYS_CLOCK_1US) );
```

System Timer is the fixed 16M, so for this timer, the S, mS, uS are defined as following in SDK code:

```
//system timer clock source is constant 16M, never change  
enum{  
    CLOCK_16M_SYS_TIMER_CLK_1S = 16000000,  
    CLOCK_16M_SYS_TIMER_CLK_1MS = 16000,  
    CLOCK_16M_SYS_TIMER_CLK_1US = 16,  
};
```

The following system timer related API should use similar CLOCK_16M_SYS_TIMER_CLK_xxx to define time, as shown below.

```
void sleep_us (unsigned long microsec);  
unsigned int clock_time(void);  
int clock_time_exceed(unsigned int ref, unsigned int span_us);  
#define ClockTime      clock_time  
#define WaitUs         sleep_us  
#define WaitMs(t)      sleep_us((t)*1000)
```

System Timer is BLE timing standard, so, all BLE timing related parameters and variables should use CLOCK_16M_SYS_TIMER_CLK_xxx to define time.

2.2.2 System Timer Usage

System Timer will start working after cpu_wakeup_init in Main function finishes initialization, user can read the value of System Timer tick.

System Timer tick is 32bit long, it will increase by 1 for each time cycle, i.e, 1/16 us, the value is range from 0x00000000 to 0xffffffff. The value of tick is 0 when system boot, it takes about (1/16) us * (2³²) = 268s to reach the maximum value, i.e., System Timer tick repeats the cycle every 268s.

System tick will not stop when MCU is running.

User can read System Timer tick via clock_time() function.

```
u32 current_tick = clock_time();
```

The whole BLE timing sequence is designed based on System Timer tick, and System Timer tick is widely used in the program for timing and over timing record, it is highly recommended to use System Timer tick for timing and over timing determination.

For example, a simple software timing, it is based on inquiry, with moderate real-time character and accuracy, it is for application with lower requirement for time error. Method:

- 1) start timing: set a u32 variable, read and record current System Timer tick.

```
u32 start_tick = clock_time(); // clock_time() return System Timer tick value
```

- 2) Check the difference between current System Timer tick and start_tick, see if it is surpass the timing value, if it is, then the timer is triggered, the program will take corresponding action, and the timer will be cleared or start a new timing cycle depends on requirement.

Suppose the timing time is 100 ms, to inquire if the time is up in the following way:

```
if( (u32) ( clock_time() - start_tick) > 100 * CLOCK_16M_SYS_TIMER_CLK_1MS)
```

The different value is switched to u32, so the range of system clock tick can be large than 0~0xffffffff. To deal with the u32 issue caused by different system clock, SDK provide a unified function, user can use the following function to inquire for any system clock source.

```
if( clock_time_exceed(start_tick, 100 * 1000)) //the unit for the second parameter is us
```

Please be noted, one cycle of 16M clock is 268s, so this function is only applicable to timing no more than 268s. Timing longer than 268s need an extra software counter.

For example, after 2s A is triggered (for only once), the program will take B action.

```
u32 a_trig_tick;
int a_trig_flg = 0;
while(1)
{
    if(A){
        a_trig_tick = clock_time();
        a_trig_flg = 1;
    }
    if(a_trig_flg &&clock_time_exceed(a_trig_tick,2 *1000 * 1000)){
        a_trig_flg = 0;
        B();
    }
}
```

3 MESH Spec Introduction

This section follows the chapter order in MshPRFv1.0.1.pdf, this is only a brief, for detail, please refer to respective chapters in SIG MESH spec.

3.1 Layered architecture

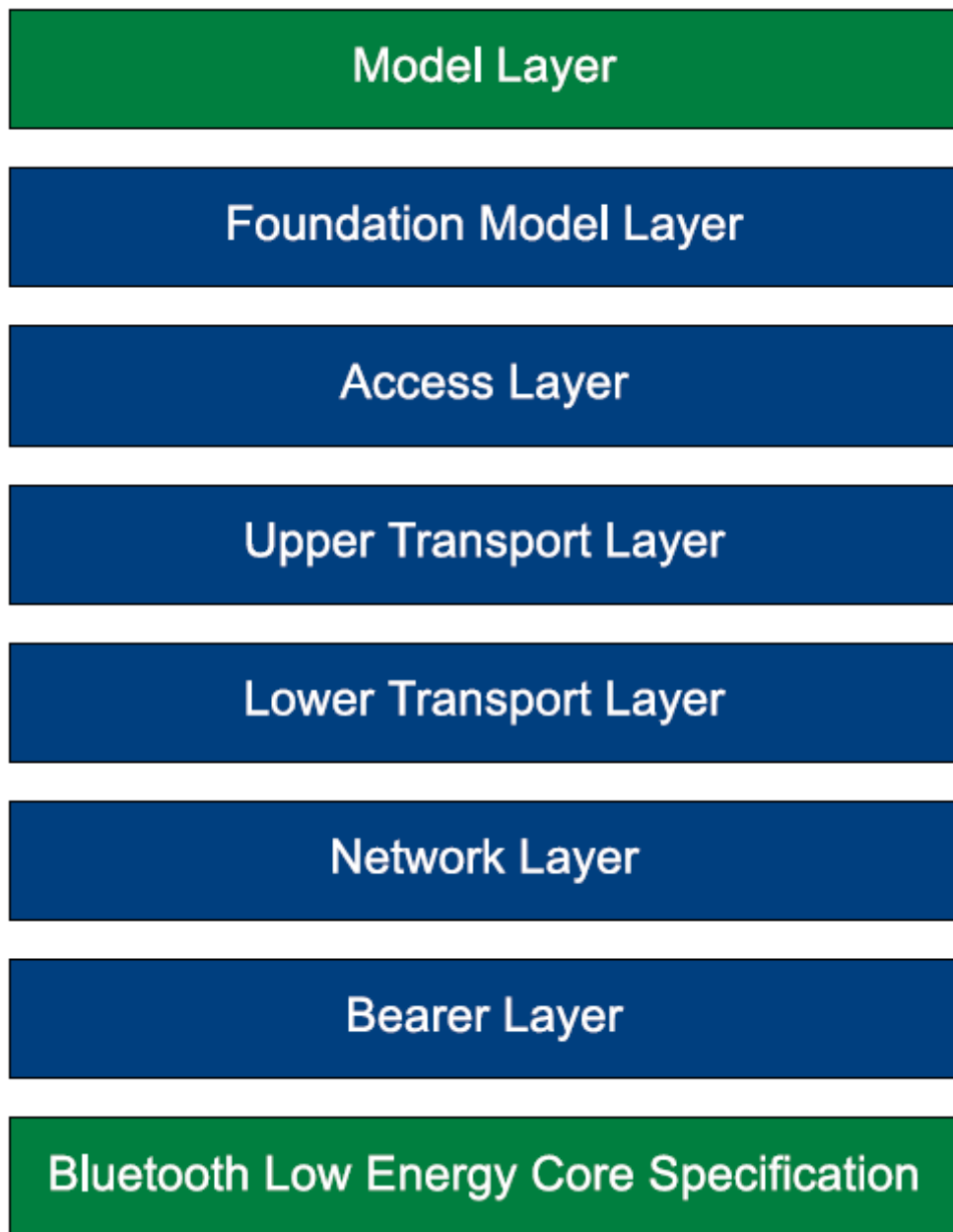


Figure 3.1: Layered Architecture

3.1.1 Model layer

Model defines a node supporting function, each model defines its own op code and status. E.g., generic on/off model defines Generic ON/OFF/GET/STATUS.

When provision, provisioner will get all model id that the node supports via get composition data, and get to know what functions the node supports. Only when the node supports a certain model, the corresponding op code defined by the same model will be sent to the node.

There are 2 kinds of models, server model and client model.

Server model: it is a model which can be controlled. It has its own status, can be changed/obtained by other nodes, e.g., on/off server model can receive on/off set/get command, can response to on/off status command, but it cannot send out on/off set/get command, nor handle on/off status command.

Client model: it can control server node, it has no status of its own. E.g., on/off client model, it can send out on/off set/ get command, it can also handle received on/off status command, but it cannot send out on/off status command, nor handle on/off set/get command.

3.1.2 Foundation Model layer

Foundation Model is similar to model, it is basic model, contains Configuration Server model, Configuration Client model, Health Server model, Health Client model.

All configured nodes must contain Configuration Server model, all provisioner must contain Configuration Client model. The 2 models contain subscription add/delete op code, and both of the models' access layers are encrypt with device key, so only provisioner node can send out set/get command of configuration model.

3.1.3 Access layer

Combine op code with parameter in prescribed format.

3.1.4 Transport layer

Decrypt/encrypt with app key or device key (for configuration model). Determine if it need segmentation or reassembly.

To compliant with protocols that does not support long packet like BLE4.2, the maximum payload is set to 31byte.

3.1.5 Network layer

For transmit: contains sequence number for packet, encrypt data with network key and iv index. Sequence number will increase by 1 after transmission.

For reception: decrypt data with network key and iv index, then determine if sequence number is valid (if it is bigger than received value), waive if invalid.

3.1.6 Bearer layer

Send out encrypted packet to mesh network via LL_TYPE_ADV_NONCONN_IND(0x02).

3.2 Architectural concepts

3.2.1 States

Node states, e.g., on/off States, lightness States.

3.2.2 Bound states

2 bound states, like on/off and lightness. When on/off switch from 1 to 0, lightness will switch to 0, too; when on/off switch from 0 to 1, lightness will switch from 0 to the value before turn off. Similarly, when lightness switch from 0 to non-0 values, on/off value will switch from 0 to 1.

3.2.3 Messages

The encrypted packet sent to mesh network. Also called as mesh packet/mesh command.

3.2.4 Node & Elements

Node is a complete node or Bluetooth module, while element is an addressable entity within a node.

Node has only 1 address, while element can have 1 or multiple continuous addresses. The first element address is called primary address, which is the same with Node address.

Multiple element addresses are needed when a node has multiple same type states. E.g., a switch with 3 sockets need to control on/off state by Generic ON/OFF command, the task cannot be completed if there is only 1 address, in this case, and multiple element addresses are needed.

Although CT light has only one on/off states, it needs 2 generic level models, one is for lightness, the other is for temp; thus it needs 2 elements.

Similarly, HSL light needs 3 elements because it needs 3 generic level models for lightness, Hue and Sat, respectively.

When build network, node will report element number in provision flow interaction flow. E.g., if the number is 2, provisioner will assign an address to node, for example, 0x0002, node will then assign 0x0002 to element 1 and 0x0003 to element 2. Provisioner will assign from 0x0004 for the next node when provisioning.

3.2.5 Models

Please refer to 3.3.1 Model layer.

3.2.6 Publish & subscribe

- Publish: element send out status spontaneously, configure publish address and publish cycle parameter with command Config Model Publication Set. When publish address is configured, each time when status changes, node will execute publish status action spontaneously. Cycle publish parameter will determine whether it need to send it by a certain cycle.
- Subscribe: when a node receives published status message (e.g., generic on/off status) or control message (e.g. generic on/off), it will determine whether to handle this message based on the model's Subscribe list[].Subscribe list[] contains group address or virtual address, unicast address and 0xffff is invalid. Add new elements with Config Model Subscription Add, Config Model Subscription Virtual Address Add.

Determination rule

- 1) When received destination address is not unicast address, check if it can find a matching address in corresponding model's Subscribe list.
- 2) When destination address is unicast address, check if it matches its own element address.
- 3) When destination address is 0xffff, then means the message should be received/handled.

3.2.7 Security

Encrypt/decrypt need Network key, IV index, App key or device key.

A message need to be encrypted twice, encrypt the whole access layer(including op code, parameters) with app key or device key, and encrypt network layer with network key + iv index, network layer is the packet sent to mesh network, contains source address, destination address and sequence number.

When encrypt access layer, if the corresponding model of op code is config model, use device key, otherwise use app key.

For segment message, because the access layer was encrypted by the whole payload, so the decryption will be done only when all the segment packets are received.

SDK supports 2 network keys (NET_KEY_MAX) and 2 app keys (APP_KEY_MAX) by default.

Multiple network key manage multiple network.

Multiple app keys manages products of different security levels. For example, there are lights and lock in the same mesh network, and lock has higher security level, in this case, user can assign an independent app key to the lock, and the app key is only open to certain mobile app(provisioner), and will not share when sharing network, thus guarantee a higher security level.

3.2.8 Sequence Number Storage

As described in 3.1.5 Network layer, Sequence Number(SNO) of mesh message increases by 1 each time it sends out command, when reception determines SNO, it should be bigger than received value, otherwise the value is invalid. This requires store SNO to flash every time it sends out commands, which is too often. To avoid this, we defined the following: store SNO only when it increase by MESH_CMD_SNO_SAVE_DELTA(default value is 0x80). To guarantee SNO is bigger than

used value, the reading SNO will be added by MESH_CMD_SNO_SAVE_DELTA when boot up. Check MESH_CMD_SNO_SAVE_DELTA mesh_flash_save_check() and mesh_misc_retrieve for reference.

3.2.9 Friendship

Friendship is the relationship built between Friend Node (FN) and Low Power Node(LPN) according to pre-scribed establish friend ship flow. After friendship is established, when LPN sleep, FN will store any message sent to LPN by other nodes. When LPN wakes up, it will send POLL inquiry command to FN, and FN will answer with stored message. This method can lower power consumption, but it need friend node in the network, and will cause delay in command receiving and answering.

3.2.10 Features

Mesh features:

- **Relay feature:** node will reduce message's TLL value by 1 after it receive the message, and then send out relay, when the received TLL value is less than or equal to 1, then no relay. With relay, the mesh network can obtain longer transmission distance. TLL is to control the delay time of the last node that receives the message. The TLL default value is TTL_DEFAULT(0x0A) in SDK, this macro is verify-able, provisioner can also configure this with Config Default TTL Set, maximum value is 127.
- **Proxy feature:** proxy is the protocol for mobile app connect to mesh network. In mesh network, app is an independent node, with its own Node address. Most app can not define transmitting packet discretionarily, neither monitoring mesh network all the time (switch to wifi for some time), so mobile app need to connect to a node with BLE GATT, the node will send out the data it receives from the app, and when it receives the answer message from the mesh network, it will sends back to mobile app with GATT according to proxy protocol.
- App sends message to node with ATT_OP_WRITE_CMD(0x52), node reply app with notify, i.e. ATT_OP_HANDLE_VALUE_NOTI(0x1B).
- **Low Power feature:** please refer to Friendship introduction in 3.2.9.
- **Friend feature:** please refer to Friendship introduction in 3.2.9.

3.2.11 Mesh Topology

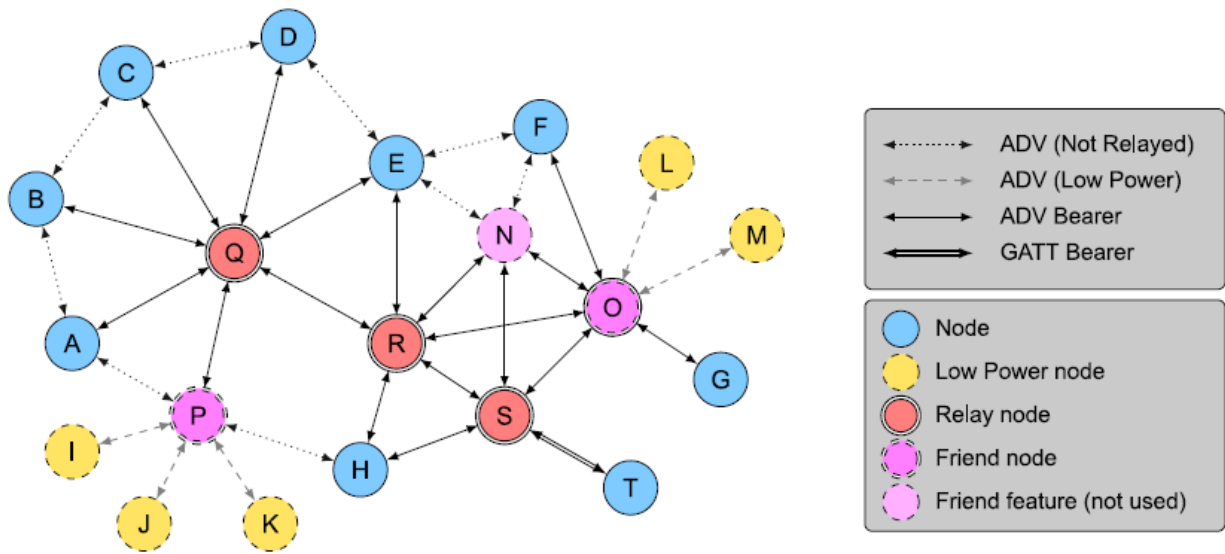


Figure 3.2: Mesh Topology

All nodes without low power support Relay, Friend. All nodes support ADV provisioning or GATT provisioning by default in this SDK.

3.3 Mesh networking

3.3.1 Network layer

Address:

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxxxx (excluding 0b0000000000000000)	Unicast Address
0b10xxxxxxxxxxxxxxxx	Virtual Address
0b11xxxxxxxxxxxxxxxx	Group Address

Figure 3.3: 16 bit Address Allocation

- Unassigned address: 0 for Unassigned address
- Unicast address for element address
- Group address: for group control and publish—subscribe scheme.
- Virtual address: use together with 16BYTE label UUID, Virtual address is the value that calculate UUID with hash algorithm. When group address (total 16384) is not enough, this can be used to expand.

Network PDU:

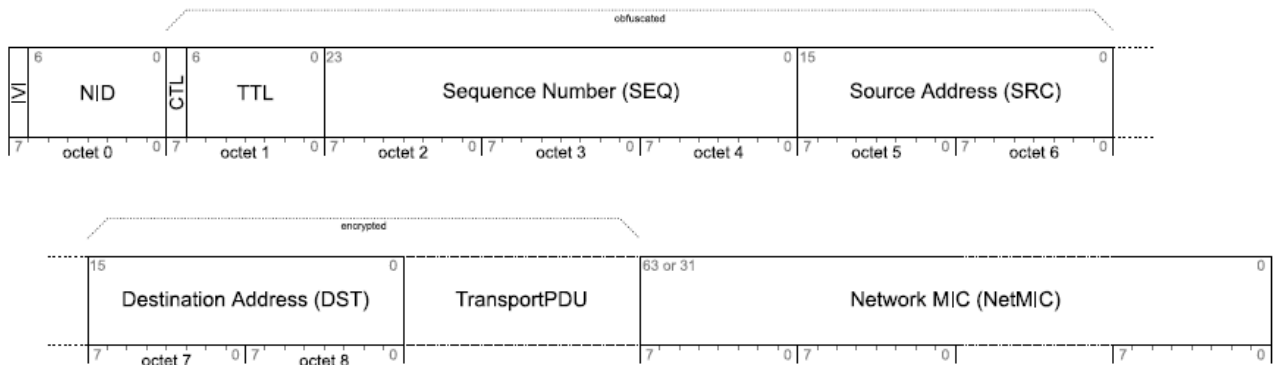


Figure 3.4: Network PDU Format

Field Name	Bits	Notes
IVI	1	Least significant bit of IV Index
NID	7	Value derived from the NetKey used to identify the Encryption Key and Privacy Key used to secure this PDU
CTL	1	Network Control
TTL	7	Time To Live
SEQ	24	Sequence Number
SRC	16	Source Address
DST	16	Destination Address
TransportPDU	8 to 128	Transport Protocol Data Unit
NetMIC	32 or 64	Message Integrity Check for Network

Figure 3.5: Network PDU Field Definitions

- IVI: iv index (i.e., the lowest bit of iv_idx_st.tx[3], stored as big endian)
- NID: network key related
- CTL: flag for control message

Network transmit count/interval

Network transmit count is how many times it need to repeat to send out a command. The rf packet is exactly the same, including SNO. The purpose of send command repeatedly is to increase reception success rate. For example, for a 2-node mesh network, if the success rate of each transmit is 80%, that means the packet losing rate is 20%, so, theoretically, the packet losing rate is 6th power of 20%, which is 0.0064%, i.e., the success rate is 99.993%. This of course also depends on RF environment.

Our SDK's default re-transmit count is 5, i.e., TRANSMIT_CNT_DEF(5), the total transmit time = n+1 = 6.

Network transmit interval is the interval of 2 adjacent re-transmitted packets, the default value in SDK is 30-40ms, defined by TRANSMIT_INVL_STEPS_DEF, calculate in the following way: $((\text{TRANSMIT_INVL_STEPS_DEF} + 1) * 10 + (0 - -10))\text{ms}$.

Network transmit count and transmit interval can also be configured by SIG config command CFG_NW_TRANSMIT_SET.

In conclusion, to send a network packet, e.g., generic ON/OFF no ack (this command need no de-packing), SDK need about $40 * 6 = 240\text{ms}$.

Reliable retry

Reliable retry is the retry on app layer, used for commands with status answer, e.g., generic ON/OFF. When send out a network packet (including network transmit), the program will check if the status is received or not, if not, then it will retry, and the sequence number in the network packet will thus change. The program will retry twice at the maximum by default.

3.3.2 Access layer

Field Name	Size (octets)	Notes
Opcode	1, 2, or 3	Operation Code
Parameters	0 to 379	Application Parameters

Figure 3.6: Access Payload Field

Opcode Format	Notes
0xxxxxxx (excluding 01111111)	1-octet Opcodes
01111111	Reserved for Future Use
10xxxxxx xxxxxxxx	2-octet Opcodes
11xxxxxx zzzzzzzz	3-octet Opcodes

Figure 3.7: Opcode Format

There are 3 kinds of op code, 1byte, 2byte and 3byte. 1byte and 2byte are defined by SIG, 3byte is defined by vendor, in which 2 bytes are vendor ID (CID), a vendor id supports at most 64 vendor opcode in the whole mesh network.

Access layer contains op code and parameter, supports 380 byte at the maximum.

3.3.3 Transport layer

To compliant with protocols that does not support long packet like BLE4.2, the adv's maximum payload is set to 31byte. The effective payload of a single packet is 11bytes, others are occupied by communication protocols, so when Access layer is longer than 11byte, it need to be segmented, thus, for vendor op code, if the parameters is longer than 8 byte(8=11-3), the mesh protocol stack will segment the message spontaneously.

3.3.4 Mesh beacon

The following figure shows unprovisioned device beacon PDU.

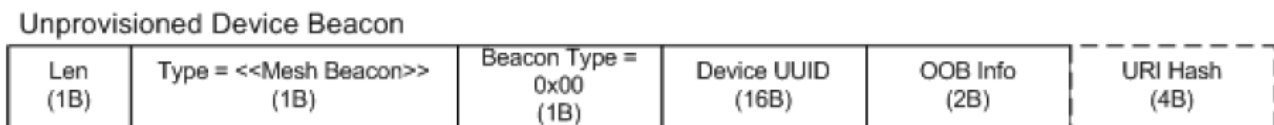


Figure 3.8: Unprovisioned device beacon PDU

Node can be identified by Device UUID. For some mobile phone, for example, IOS cannot get mac, nor can get mac in future remote provision, so in SIG mesh, node is identified by Device UUID instead of by mac.

Unprovisioned beacon is transmit via non-connectable ADV packet, used for PB-ADV provision mode.

Please refer to spec 3.9.2 Unprovisioned Device beacon for Oob info and URI Hash.

Before provision, unprovisioned beacon will be transmit via unprov_beacon_send(), the transmit interval is defined by beacon_send.inter = MAX_BEACON_SEND_INTERVAL, the default value is 2s.

After provision, security beacon will be sent out via mesh_tx_sec_nw_beacon(). User can also enable-disable this transmit command via CFG_BEACON_SET. Please refer to SecNwBc operation in 4.4, the transmit interval is defined by SEC_NW_BC_INV_DEF_100MS, the default value is 10s.

3.3.5 iv update flow

This is iv index update flow. Both network layer and access layer decryption/encryption need iv index. As described before, mesh network requires network PDU's sequence number accumulate all the time, and the length of sequence number is 3 bytes. So when sequence number approach its maximum, iv index need to be updated, otherwise the sequence number will be reset to 0, and will be invalid in reception end. So iv index is the expand of sequence number.

Nodes will start and execute iv index update flow spontaneously. When a node is noted that its sequence number is bigger than IV_UPDATE_START_SNO (0xC00000), it will start iv update flow. The iv index increases by 1 for each Iv update.

Check SPEC V1.0 3.10.5 IV Update procedure for detail.

3.3.6 Heartbeat

Mesh heartbeat, sent out periodically, can be used for on/off line check (periodical publish can do the same thing), and hops calculation, i.e., to calculate how many times heartbeat message hopped before it get received.

Count received heartbeat, calculate the hops of each heartbeat, get the min hops and max hops, thus know the structure of the whole network and the message transmission reliability of each node. One heartbeat subscription configuration can only monitor/calculate 1 node.

hops is calculated in the following way:

$$\text{hops} = \text{InitTTL} - \text{RxTTL} + 1。$$

- InitTTL: TTL in heartbeat publish set
- RxTTL: TTL in received message network PDU

Node send out no heartbeat by default, check 5.6 for detailed configuration.

3.3.7 Health

Health model related message is used for node's warning/error status, e.g., battery warning/error messages. Check spec 4.2.15.1 Current Fault for detail, as shown in table below.

Value	Description
0x00	No Fault
0x01	Battery Low Warning
0x02	Battery Low Error
0x03	Supply Voltage Too Low Warning
0x04	Supply Voltage Too Low Error
0x05	Supply Voltage Too High Warning
0x06	Supply Voltage Too High Error
0x07	Power Supply Interrupted Warning
0x08	Power Supply Interrupted Error
0x09	No Load Warning
0x0A	No Load Error
0x0B	Overload Warning
0x0C	Overload Error
0x0D	Overheat Warning
0x0E	Overheat Error

Value	Description
0x0F	Condensation Warning
0x10	Condensation Error
0x11	Vibration Warning
0x12	Vibration Error
0x13	Configuration Warning
0x14	Configuration Error
0x15	Element Not Calibrated Warning
0x16	Element Not Calibrated Error
0x17	Memory Warning
0x18	Memory Error
0x19	Self-Test Warning
0x1A	Self-Test Error
0x1B	Input Too Low Warning
0x1C	Input Too Low Error
0x1D	Input Too High Warning
0x1E	Input Too High Error
0x1F	Input No Change Warning
0x20	Input No Change Error
0x21	Actuator Blocked Warning
0x22	Actuator Blocked Error
0x23	Housing Opened Warning
0x24	Housing Opened Error
0x25	Tamper Warning
0x26	Tamper Error
0x27	Device Moved Warning
0x28	Device Moved Error
0x29	Device Dropped Warning
0x2A	Device Dropped Error
0x2B	Overflow Warning



Value	Description
0x2C	Overflow Error
0x2D	Empty Warning
0x2E	Empty Error
0x2F	Internal Bus Warning
0x30	Internal Bus Error
0x31	Mechanism Jammed Warning
0x32	Mechanism Jammed Error
0x33-0x7F	Reserved for Future Use
0x80-0xFF	Vendor Specific Warning / Error

Telink Semiconductor

4 Debugging Tool Instructions

4.1 Download Firmware

Please refer to "Help" -> "User guide" for BDT tool detailed instruction. The following is instruction for frequent used operations.

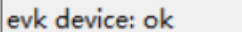
Before start, download 8258_mesh.bin into each node (8258 Dongle), then burn provisioner nodes, there are 2 provisioner modes, master dongle mode connected via GATT and gateway mode(ADV mode).

GATT mode: download 8269_mesh_master_dongle.bin in to 8269 Master Dongle (8269 Dongle).

Gateway mode: download 8258_mesh_gw.bin to 8258 gateway Dongle.

For example, user can download firmware to 8258 light node wit the following steps:

1) Hardware connection: connect miniUSB port on EVK board with PC USB port with USB cable, the light on EVK board will flash if the connection succeed. 8258 Dongle connect with EVK board USB port via USB port.

evk device: ok

will show in the left lower corner of the tool.

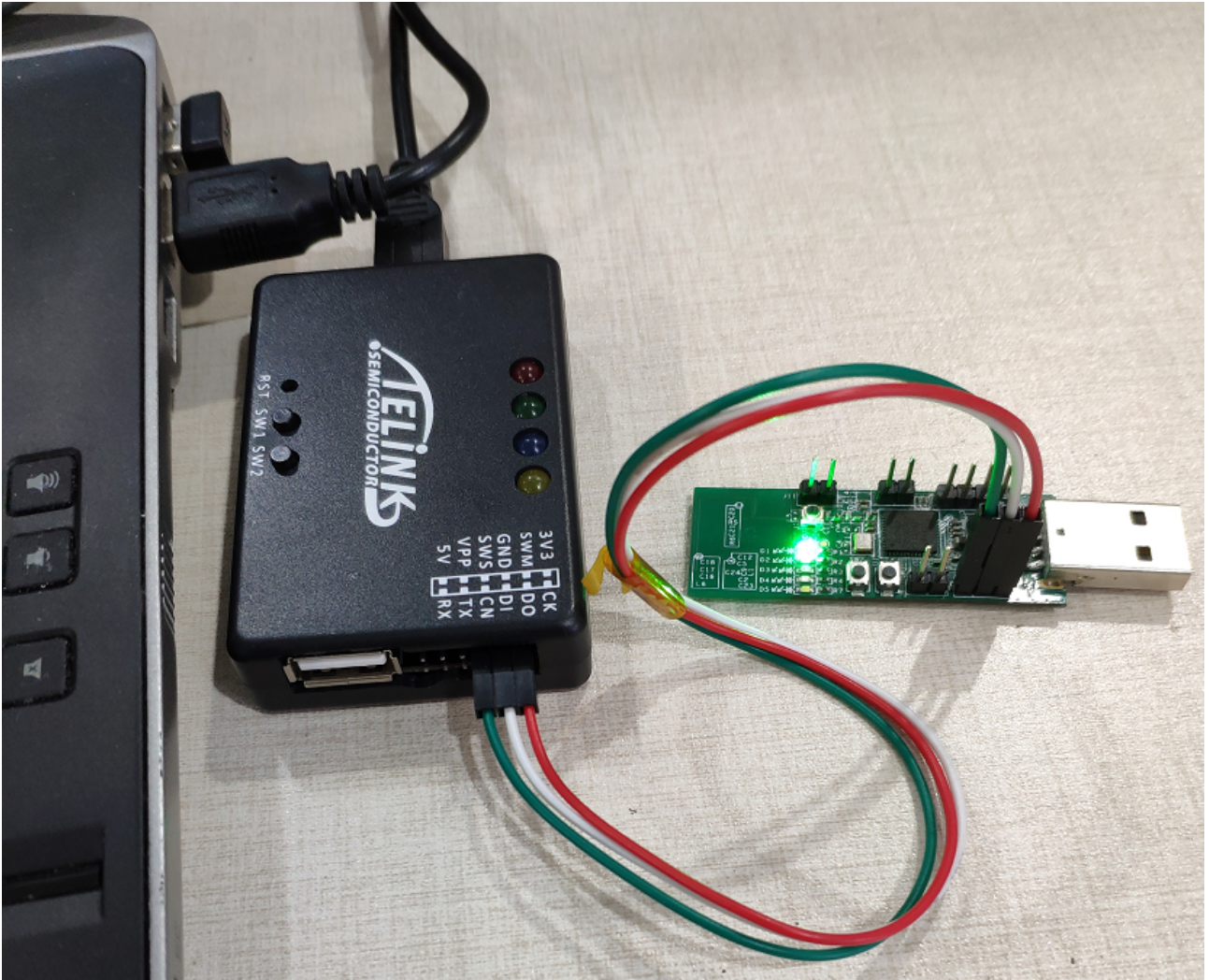


Figure 4.1: Hardware connection

2) Download 8258_mesh.bin to 8258 Dongle flash with Telink BDT tool.

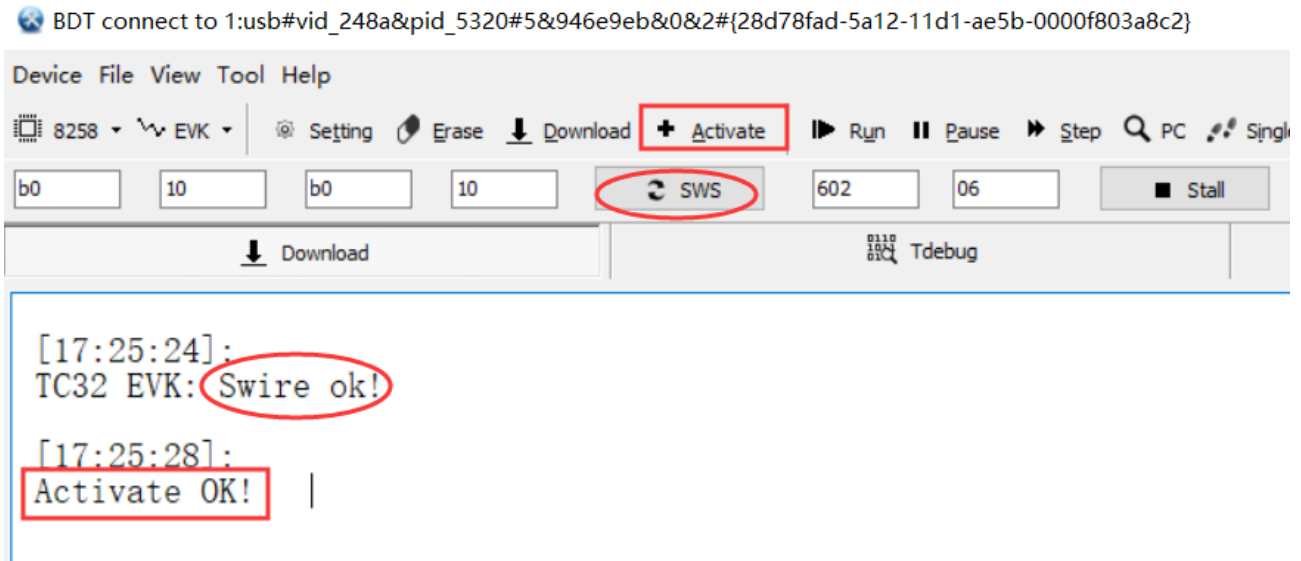

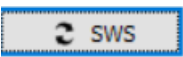

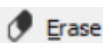
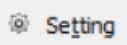


Figure 4.2: BDT interface

Step 1 Open BDT, click  to choose the right part no, then click  to check if EVK and 8258 dongle can communicate normally, if yes, Swire ok will show. If not, it may because the SoC is in sleep mode, click  to awake the SoC, this is especially important for low power equipment, Activate OK will show when succeed. Active contains MCU restart.

Step 2 Click  to erase 8258 Dongle flash.

Note: click  to set start address and size for the erase action.

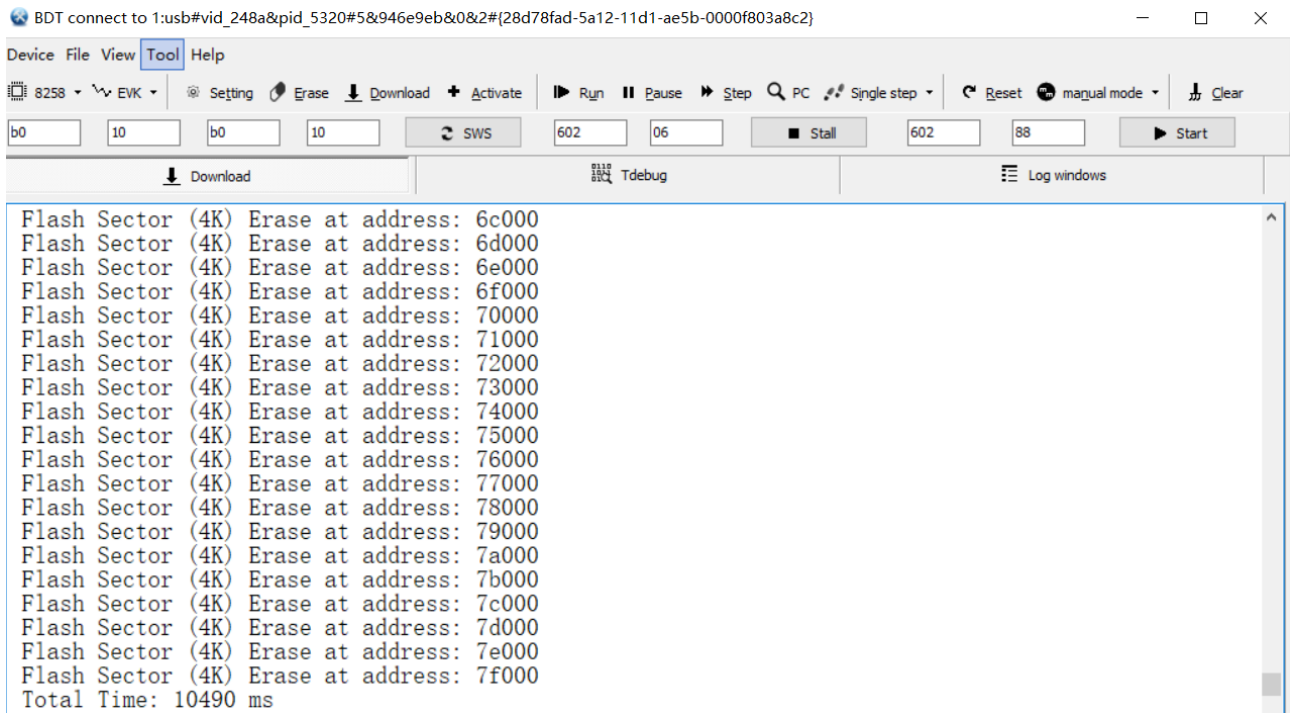


Figure 4.3: Erase Flash

Step 3 Click "File", then click "open", choose corresponding "8258_mesh.bin" file, click to open, then the BDT corresponding file:

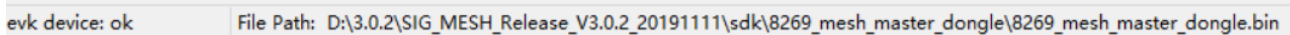
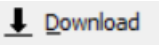
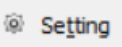


Figure 4.4: Bin file

Step 4 Click  , burn the chosen 8258_mesh.bin in flash address start from 0.

Note: click  to set start address and size for the download action, default value is 0.

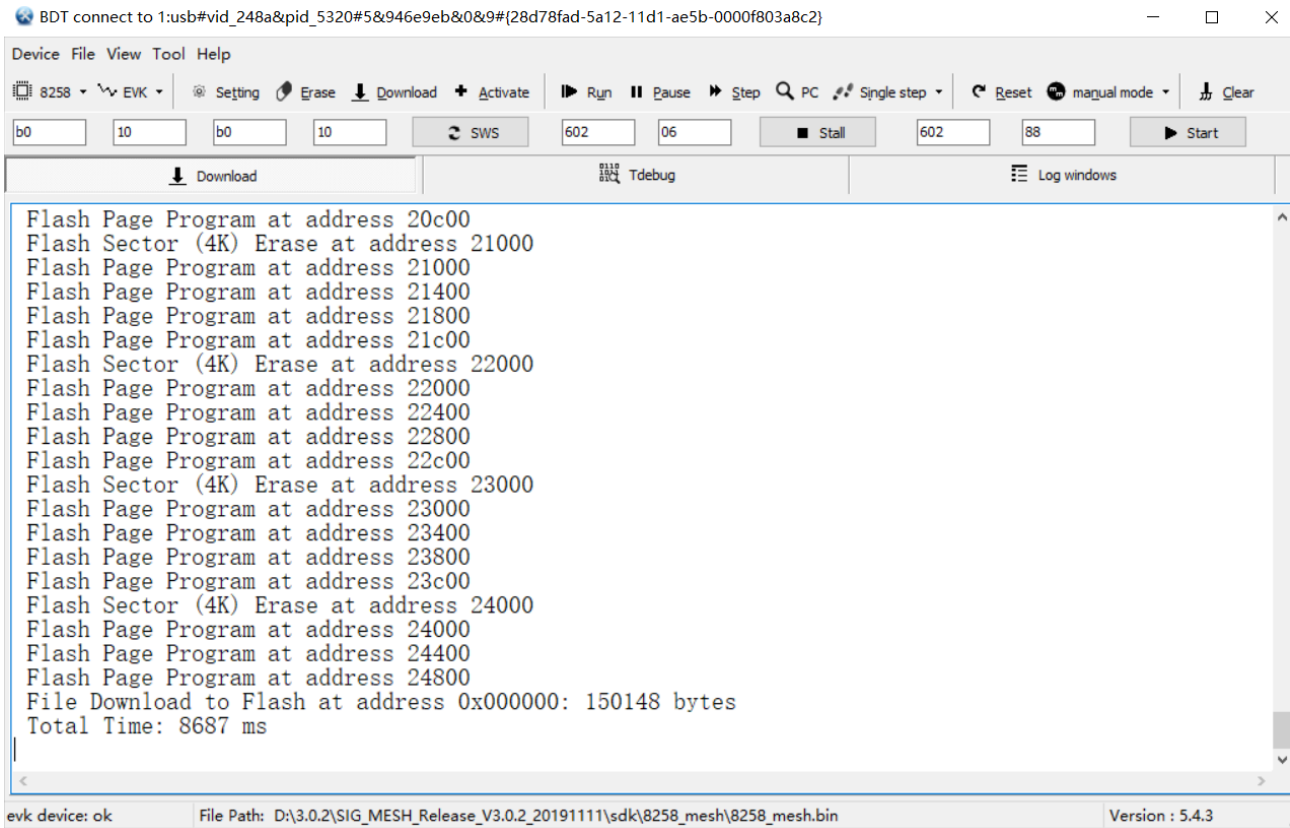


Figure 4.5: bin file burned into flash

Firmware Burning Steps

Step 1 Click “Tool”—“Memory Access”, dialogue box pops up.

Note:

- this action only applicable when burning light nodes and gateway nodes, GATT master dongle does not need this.

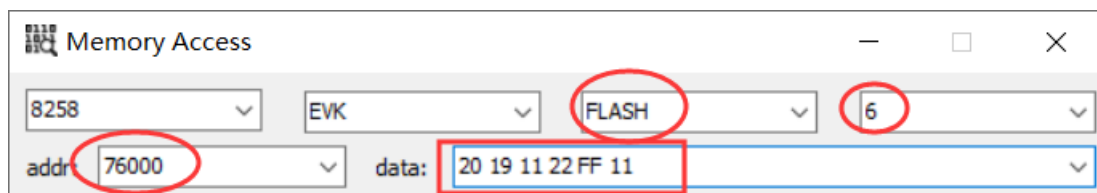


Figure 4.6: Input information

Input the 6-byte MAC as shown above, click enter in data field to write. Click Tab in Addr field to read, this is the read-back confirmation.

If the mac field keeps empty, when 8258 dongle reboot, it will detect 0x76000 has no mac, then it will assign a random mac and save it in 0x76000 in flash.

Step 2 After reboot, 8258 Dongle can work as a light node.

Above is the BDT frequent used operations. Users can also click "Help" -> "User guide" for details of other functions and operations.

4.2 BLE Connection and Adding Light in Gateway USB Mode

- 1) Open sig_mesh_tool.exe, plug the gateway dongle with burned 8258_mesh_gw.bin in PC USB port.
- 2) As shown below, "Found" means 8258 gateway Dongle connected correctly with PC tool, and the communication works. The tool will choose tl_node_gateway.ini automatically based on connected hardware.

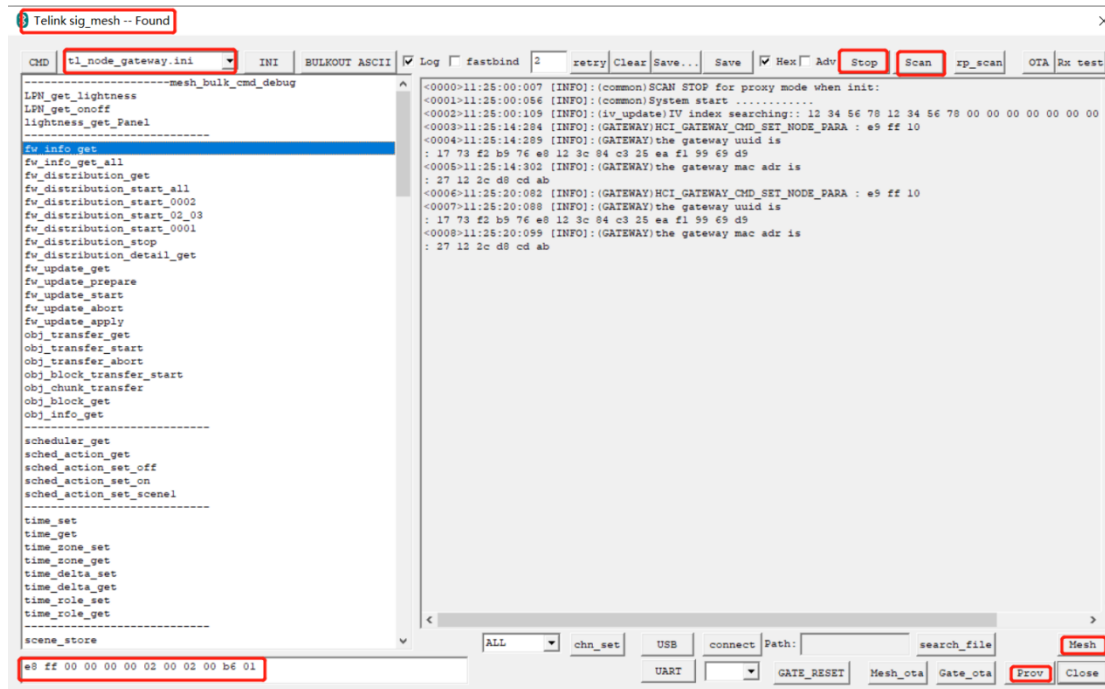


Figure 4.7: SIG_MESH_TOOL interface

- 3) Boot 8258mesh node.
- 4) Click "Scan" to open "ScanDev" window, showing corresponding mac address, including rssi and frequency offset.

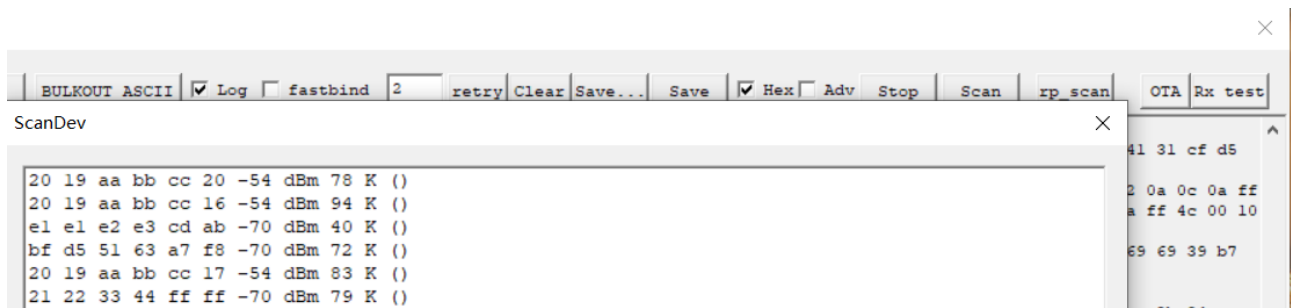


Figure 4.8: ScanDev window

- 5) Click corresponding item in "ScanDev" to choose node.

Gateway mode: double click to choose the node, no connection or command transmission.

8258 gateway Dongle mode: double click will build BLE connection, if the red light on 8258 gateway Dongle turns on, means BLE connection is built successfully, "Stop" is to stop current BLE connection, when the white light on 8258 gateway Dongle turns on, means BLE connection is stopped. Currently supports only single node BLE GATT connection.

6) Click "Prov" to open "provision" window.

Note:

- "Provision" and "bind_all" is forbidden after initialization, users can not use the 2 button at the same time.

"network_key" is generated randomly when first open "provision", it can be modified before click "Set-Pro_internal".




Figure 4.9: provision window

7) Click "SetPro_internal" to set network initial parameters, print "Set internal provision success" in log window to show that the parameters are set successfully.

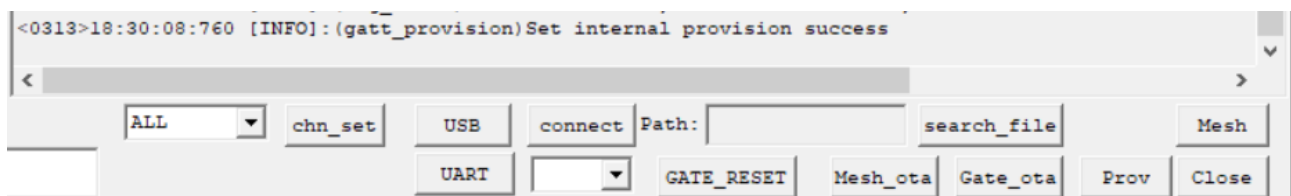


Figure 4.10: Set internal provision success window

Once click "SetPro_internal", the corresponding parameters like netkey cannot be modified, so "Set-Pro_internal" will turn to grey. Parameters will be saved in mesh_database.json and will be read

automatically next time open the tool. If network_key need to be modified, then the whole network should be dismissed, and reset to Factory settings.

Now "Provision" is enabled. unicast_addr is the primary address to be assigned to provision, user can change it manually, but it highly recommended not to.



Figure 4.11: Provision enabled

8) Click "Provision" to execute SIG provision flow to add corresponding node into network. The red LED will flash 4 times to show the connection success. Log information is shown below:

Gateway mode log:

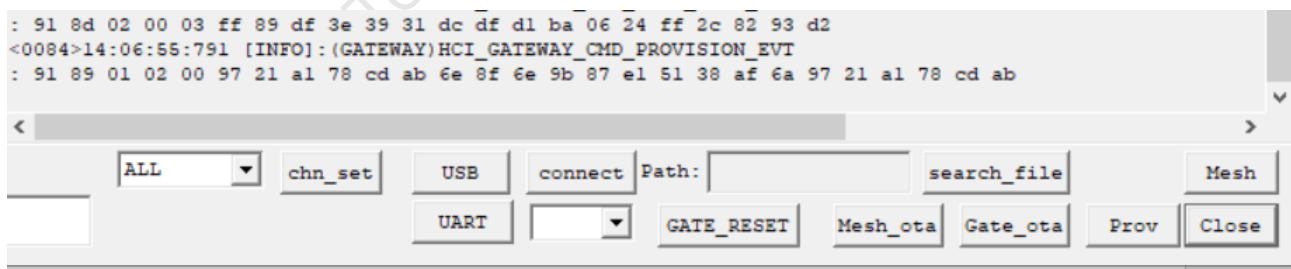


Figure 4.12: Gateway mode log

GATT Master dongle log:

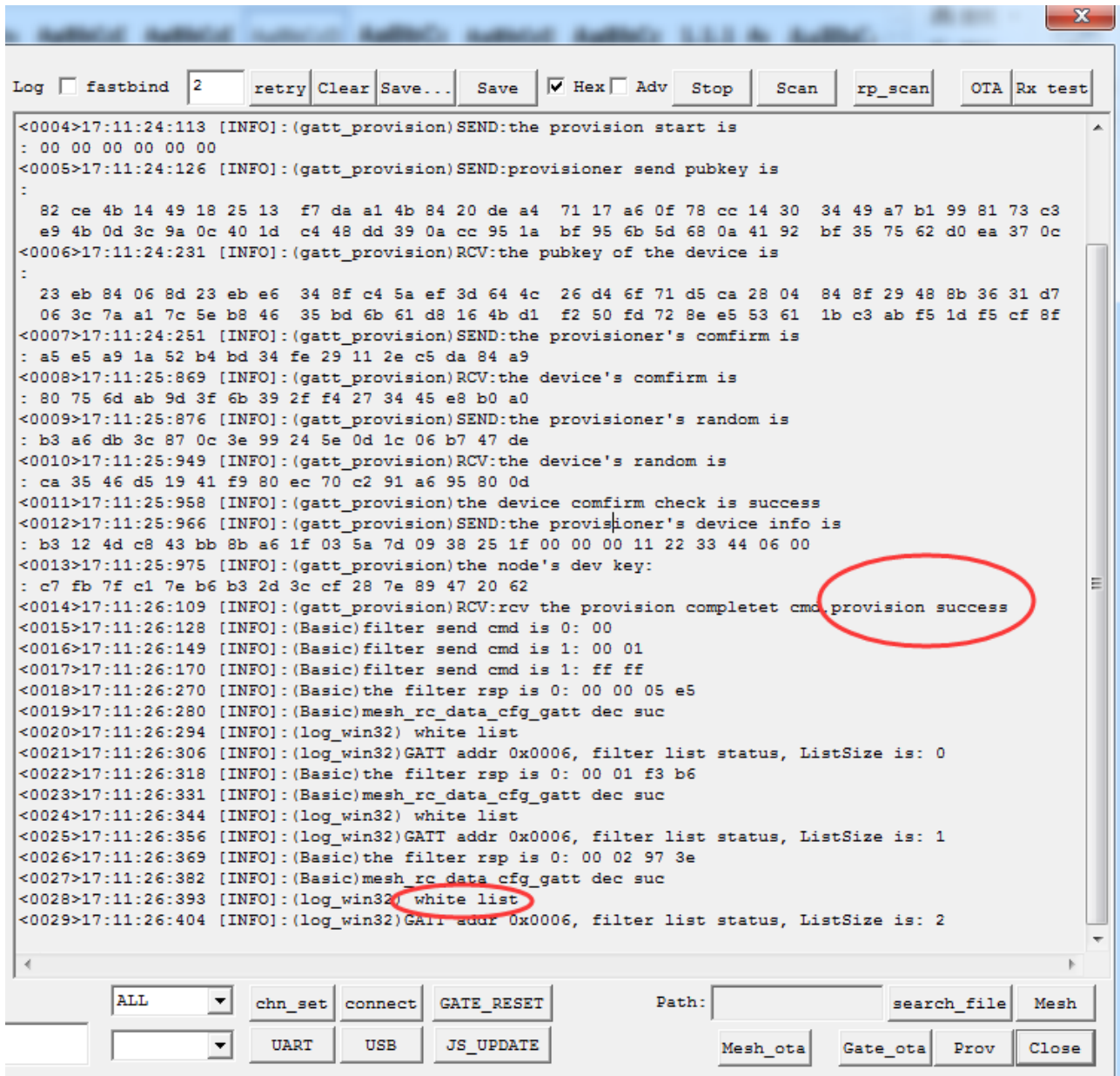


Figure 4.13: GATT Master dongle log

9) Click "bind_all" after configure app_key, first get composition data will be sent to get all model ids, then bind app_key to all models.

After Bind_all, unicast_adr will automatically accumulate based on the elements number the current node contains, and calculate primary address for next node provision.(e.g., CT light has 2 elements, then unicast_adr will increase by 2 each time a CT light is added).

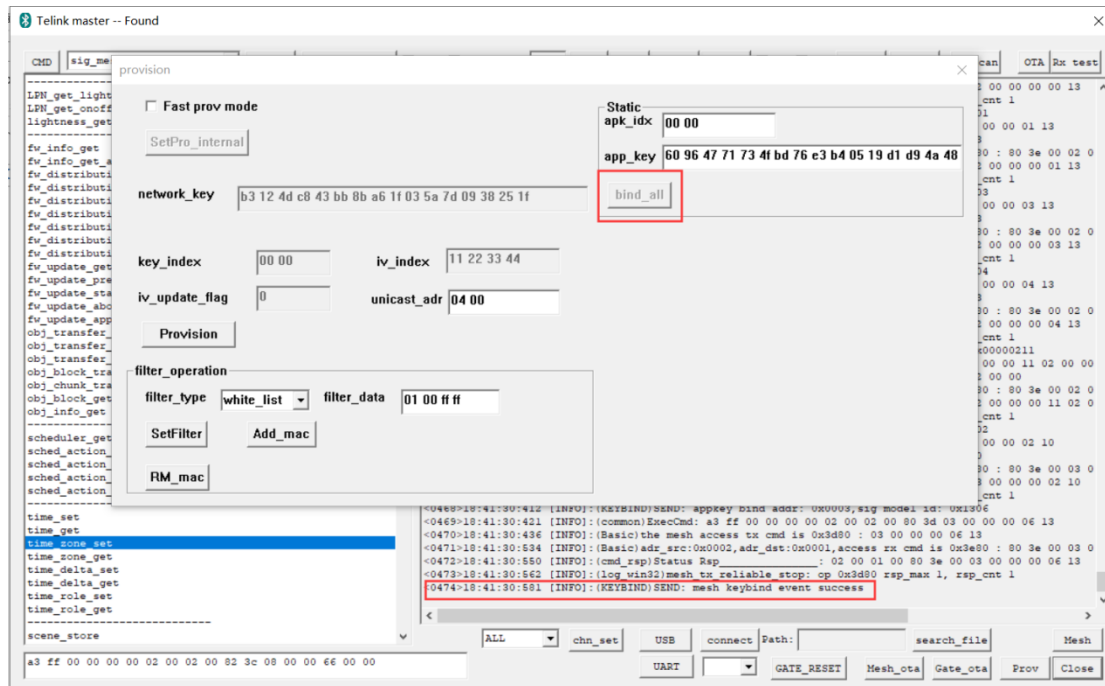


Figure 4.14: Click bind_all

10) After binding App_key, click mesh to enter mesh UI, user can turn on/off light here.

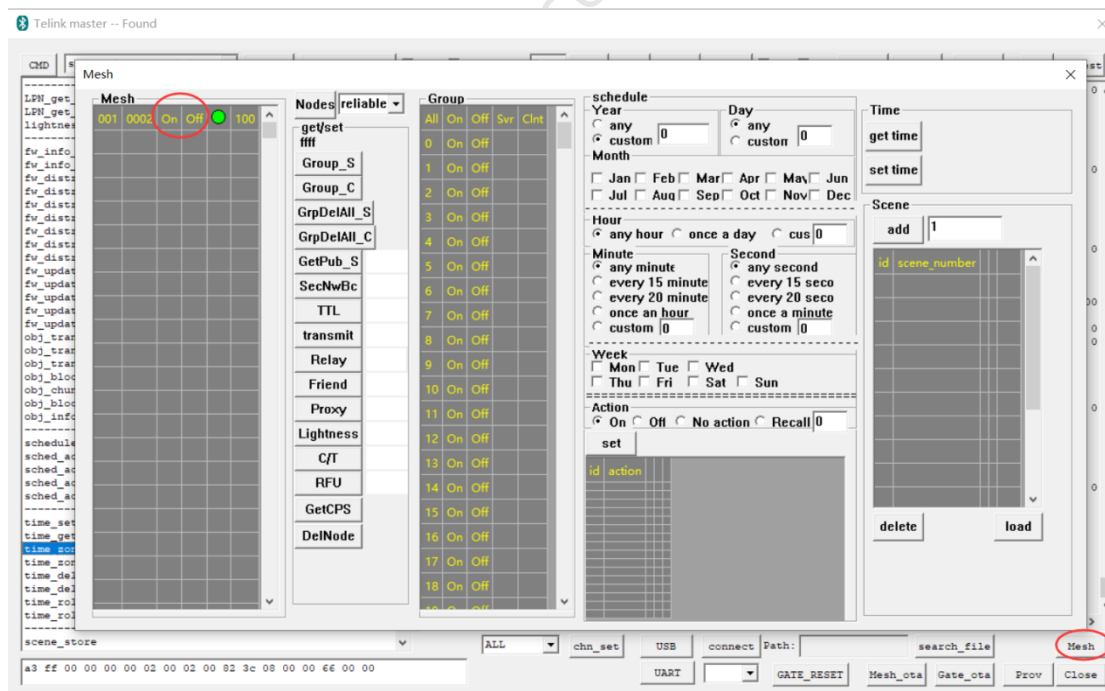


Figure 4.15: mesh UI

11) Dismiss network

Both GATT master dongle and Gateway mode can follow the following way:

Choose a node, then click "DelNode" to delete this node. Refer to 4.5.3 "DelNode" instruction for detail.

Note:

- in GATT master dongle mode, please delete no-GATT-direct-connected nodes first, then delete GATT direct connected nodes. Delete GATT directly connected node will disconnect current GATT.

4.3 BLE Connection and Adding Light in Gateway UART Mode

Configure UART ports:

- 1) Choose HCI_USE_UART in HCI_ACCESS in gateway firmware, re-compile.
- 2) Insert port tool to PC, connect tx/rx with gateway rx/tx.
- 3) Open "sig_mesh_tool.exe".
- 4) Click UART then choose the pop-up COM port.
- 5) Click "Connect", if the connection succeed, the button will change to Disconnect. Now UART can execute gateway functions.
- 6) All other operations are similar with that of USB mode, please refer to 4.2 for detail.

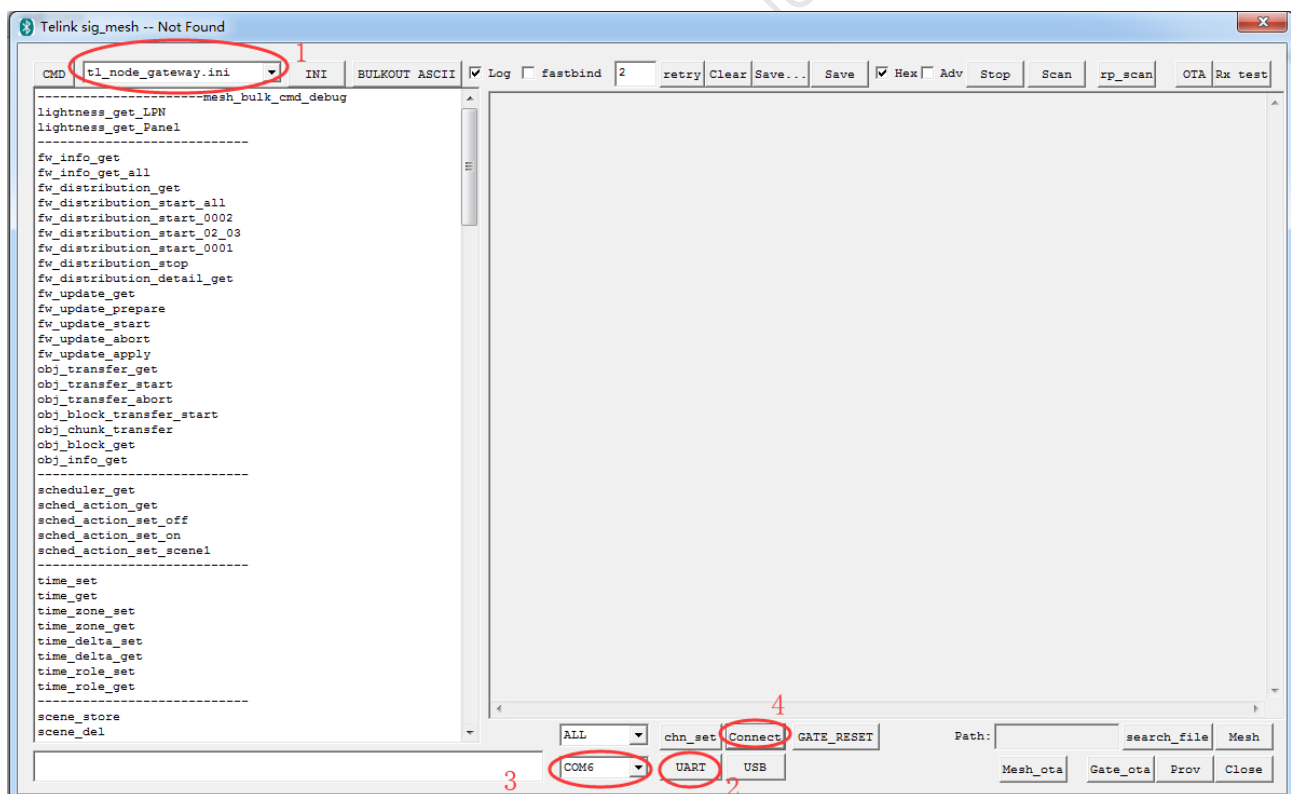


Figure 4.16: Configure UART port

4.4 BLE Connection and Adding Light in GATT master dongle Mode

- 1) Open sig_mesh_tool.exe", plug 8269 Master Dongle with burned program in PC USB port.

2) As shown below, "Found" means 8269 Master Dongle connected correctly with PC tool, and the communication works. The tool will choose sig_mesh_master.ini automatically based on connected hardware.

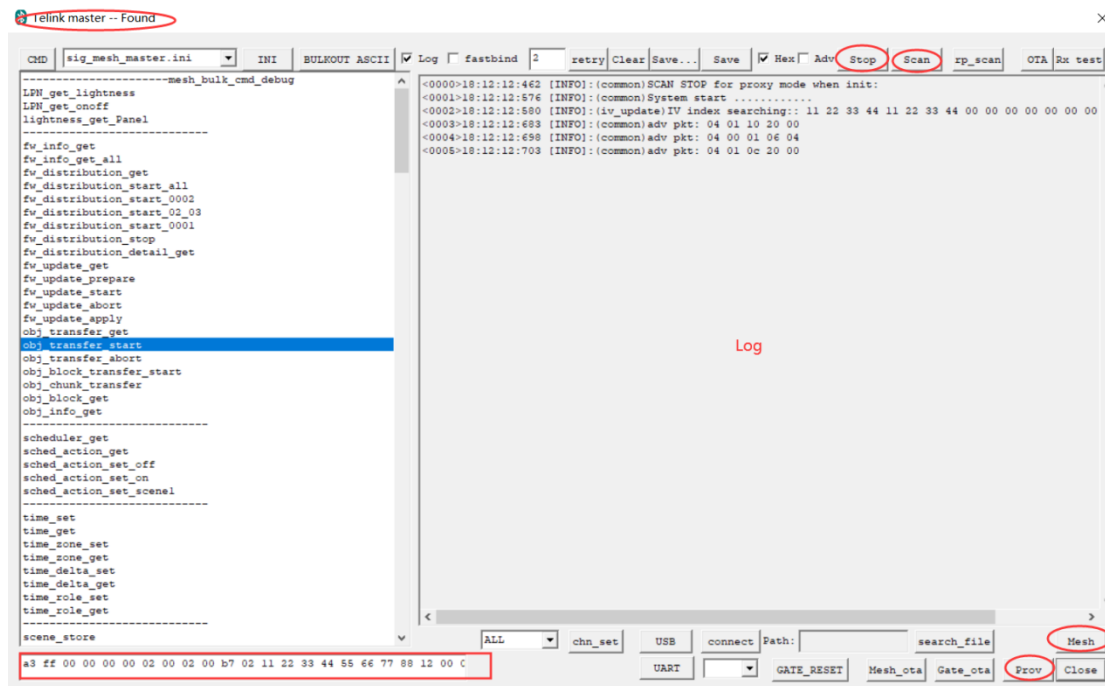


Figure 4.17: SIG_MESH_TOOL interface

3) Please refer to 4.2 for further steps.

4.5 Control Corresponding Nodes

GATT master dongle and gateway have the same operation and UI.

4.5.1 UI Display and on/off Control of Single/All Node(s)

1) Click "Mesh". A "Mesh" window will pop up.

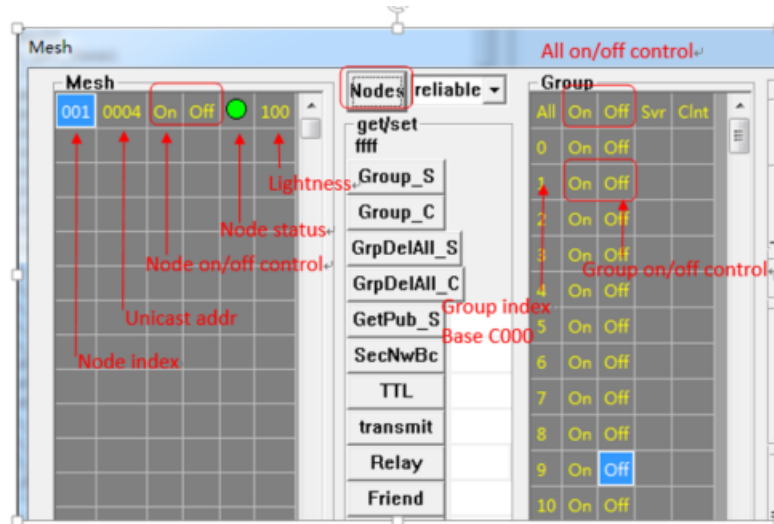


Figure 4.18: mesh window

2) By clicking "Nodes" in "Mesh" window, user can refresh all light status.

If click "Nodes" when choose reliable in the right drop-down box, it will send out lightness get command. The UI will be refreshed according to lightness status.

If click "Nodes" when choose unreliable in the right drop-down box, it will send out lightness get command. But on/off command is no ack. Node will not reply status in this case, so the UI will not be refreshed, use publish to refresh UI.

If click "Nodes" when choose online status in the right drop-down box, it will not send out command, only initialize UI to null, then refresh UI according to returned online status data.

Note: online status is a private mode, the node's firmware should enable ONLINE_STATUS_EN.

The lightness display in 0-100 scale, which is switched from SIG defined 0-65536 scale.

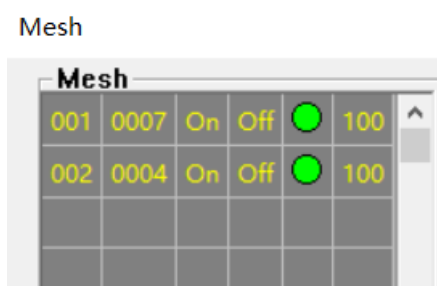


Figure 4.19: Node status

3) Single node operation: click "On"/"Off", the corresponding light will control the switch status. The node status will be reported to the tool to refresh the corresponding status.

4) : "on", : "off", : off-line



Figure 4.20: Single node control

5) All on all off control: Click "On"/"Off" besides "All", all nodes in the mesh network will switch to on/off.



Figure 4.21: All node control

4.5.2 Group Control (Subscription Demo)

Click index number of the light node, e.g., 002, to get the node address and show in position in below figure. The default value is 0xffff, means no node is chosen.



Figure 4.22: Obtain node address

User can click/right click "Svr" box in Group Control, to add/delete this light node in/from corresponding group. ✓ in "Svr" means the node has been added to the group, blank in "Svr" means the node is not in the group.

- "Svr" column is for generic on/off server model (0x1000)
- "Clnt" column is for generic on/off client model (0x1001)

Normally, node supports only server model, so only "Svr" column is operated.

Group index and group address's relation is described as: group address = group index + 0xC000.

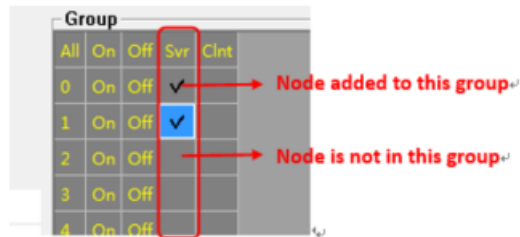


Figure 4.23: Allocate one light to multiple groups

User can click the corresponding "On"/"Off" to control the group in Group controls.

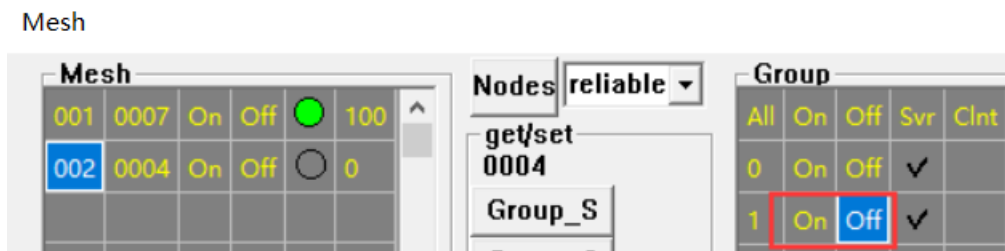


Figure 4.24: Group control

4.5.3 Configure Node Parameter with UI

As shown in figure below, click position 1 to choose the node, then the tool will send get group (subscription list) command out automatically to get the node's group and show it in corresponding UI. Then determine if current node supports scene, time, scheduler function, if yes, send get commands out automatically to get scene, time and scheduler list.

Send SCENE_REG_GET to get all valid scene index, and display in UI list.

Send TIME_GET to get time of current node, and display in UI. If the node is just booted, the time will be 0, which means the node is waiting for configuration, in this case, the time will keep 0, and do not do the timing action. User can configure time in 2 ways, 1, send time set command via app/gateway, 2, when the node boots, configure its time model's publish character to get time status other nodes published.

Send SCHD_GET to get all valid scheduler index, then based on the returned index value, send SCHD_ACTION_GET respectively to get detail parameters, and display in UI list. As shown in below figure, the provision is just completed, no scheduler is added, so no need to send SCHD_ACTION_GET.

For the same reason, group, scene, time and scheduler are all blank.

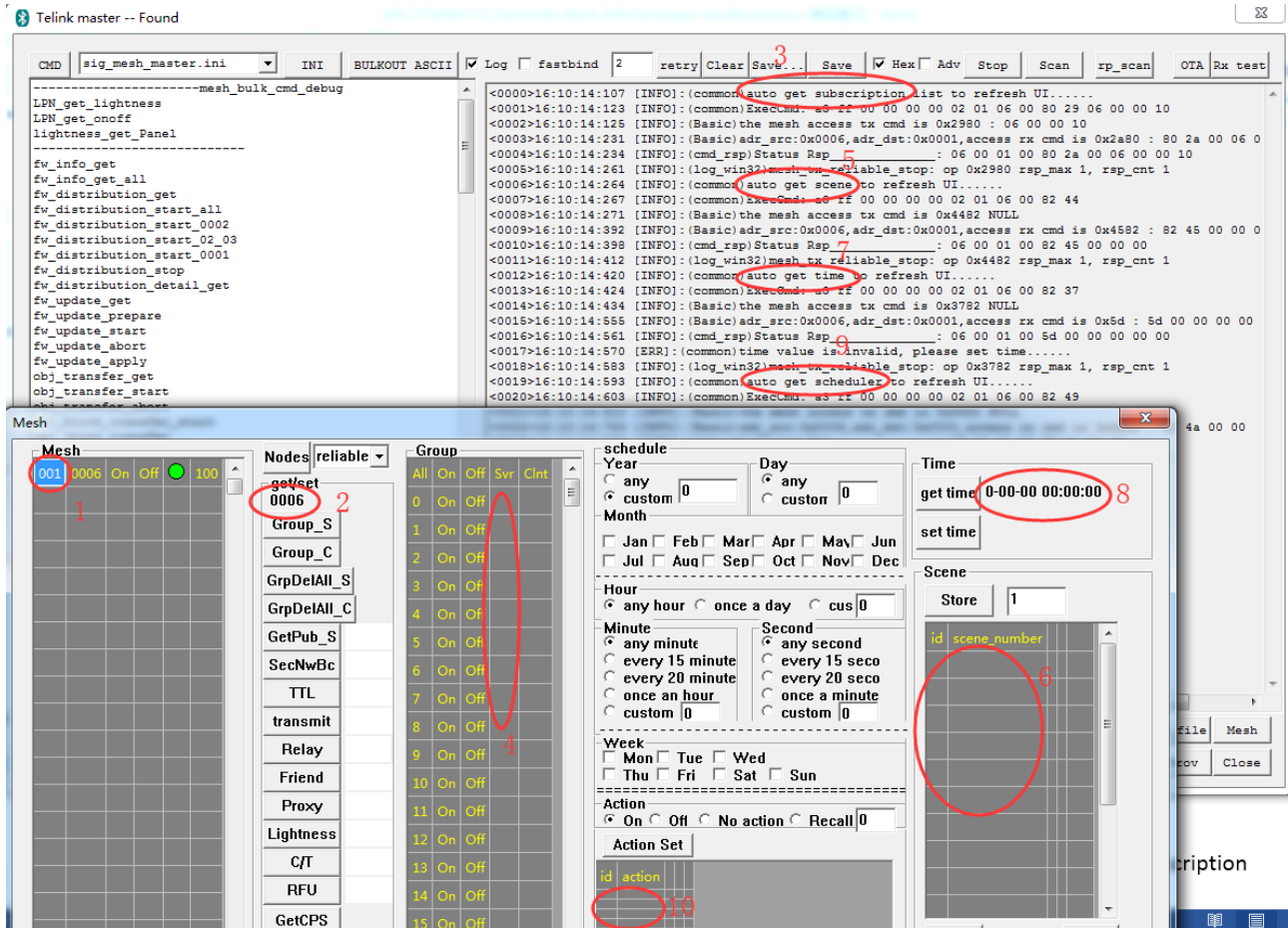


Figure 4.25: Configure node parameter with UI

The following button is based on current chosen node.

"Group_S"

By clicking this button, CFG_SIG_MODEL_SUB_GET will be sent to get subscription address list of on/off server model of current node, and display it in "Svr" column.

"Group_C"

By clicking this button, CFG_SIG_MODEL_SUB_GET will be sent to get subscription address list of on/off client model of current node, and display it in "Cln" column.

Most nodes do not support on/off client model, so this is not a frequent used button.

"GrpDelAll_S"

By clicking this button, CFG_MODEL_SUB_DEL_ALL will be sent to delete subscription address list of on/off server model of current node, and clear "Svr" column.

"GrpDelAll_C"

By clicking this button, CFG_MODEL_SUB_DEL_ALL will be sent to delete subscription address list of on/off client model of current node, and clear "Cln" column.

"GetPub_S"

By clicking this button, CFG_MODEL_PUB_GET will be sent to get publish address of on/off server model of current node, and display the return value in later display box.

Modify publish address in input box, then press "Enter", CFG_MODEL_PUB_SET will be sent, and the corresponding publish address is the value in input box.

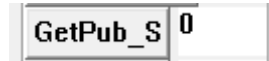


Figure 4.26: "GetPub_S"

Other parameters are default values. Check command sending log:

```
ExecCmd: a3 ff 00 00 00 00 02 01 06 00 03 06 00 00 00 00 00 ff 00 15 00 10
```

Figure 4.27: Command sending log

Other publish parameters can be modified by cfg_pub_set_sig of INI command, modify the parameter to wanted value, then send.

"SecNwBc"

By clicking this button, CFG_BEACON_GET will be sent, the return value will display in right display box. This command determine whether to send security network beacon or not.

Modify value in input box, then press "Enter", CFG_BEACON_SET will be sent, and the corresponding parameter value is the value in the input box.

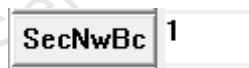


Figure 4.28: "SecNwBc"

"TTL"

By clicking this button, CFG_DEFAULT_TTL_GET will be sent, the return value will display in right display box. This command gets the default TTL value of the node. SDK default value is defined by TTL_DEFAULT.

Modify TLL value in input box, then press "Enter", CFG_BEACON_SET will be sent, and the corresponding parameter value is the value in the input box.



Figure 4.29: "TTL"

"transmit"

By clicking this button, CFG_NW_TRANSMIT_GET will be sent, the return value will display in right display box. This command gets the network transmit value of the node. The lower 3bit is network transmit count, the higher 5bit is network transmit interval.

SDK default values are defines as following:

```
: #define TRANSMIT_CNT_DEF (5)
: #define TRANSMIT_INVL_STEPS_DEF (2)
```

Figure 4.30: SDK default value

Note:

- transmit count(5) + network transmit interval(2) is 0x15.

Modify network transmit value in input box, then press "Enter", CFG_NW_TRANSMIT_SET will be sent, and the corresponding parameter value is the value in the input box.

transmit	15
----------	----

Figure 4.31: "transmit"

"Relay"

By clicking this button, CFG_RELAY_GET will be sent, the return value will display in right display box. This command gets the relay enable value of the node.

Modify Relay value in input box, then press "Enter", CFG_RELAY_SET will be sent, and the corresponding parameter value is the value in the input box.

Relay	1
-------	---

Figure 4.32: "Relay"

"Friend"

By clicking this button, CFG_FRIEND_GET will be sent, the return value will display in right display box. This command gets the friend feature enable value of the node.

Modify Friend value in input box, then press "Enter", CFG_FRIEND_SET will be sent, and the corresponding parameter value is the value in the input box.

Friend	1
--------	---

Figure 4.33: "Friend"

"Proxy"

By clicking this button, CFG_GATT_PROXY_GET will be sent, the return value will display in right display box. This command gets the proxy feature enable value of the node.

Modify Proxy value in input box, then press "Enter", CFG_GATT_PROXY_SET will be sent, and the corresponding parameter value is the value in the input box.

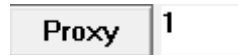


Figure 4.34: "Proxy"

"Lightness"

By clicking this button, LIGHTNESS_GET will be sent, the return value will first switch from 0-65535 scale to 0-0x64 and then display in right display box.

Modify Lightness value in input box, then press "Enter", LIGHTNESS_SET will be sent, and the corresponding parameter value is the value in the input box.

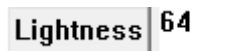


Figure 4.35: "Lightness"

"C/T"

By clicking this button, LIGHT_CTL_TEMP_GET will be sent, the return value will first switch from 800-20000 scale to 0-0x64 and then display in right display box.

Modify C/T value in input box, then press "Enter", LIGHT_CTL_TEMP_SET will be sent, and the corresponding parameter value is the value in the input box.



Figure 4.36: "C/T"

"RFU":

Reserve for future.

"GetCPS"

By clicking this button, COMPOSITION_DATA_GET will be sent, the return value will display in right display box.

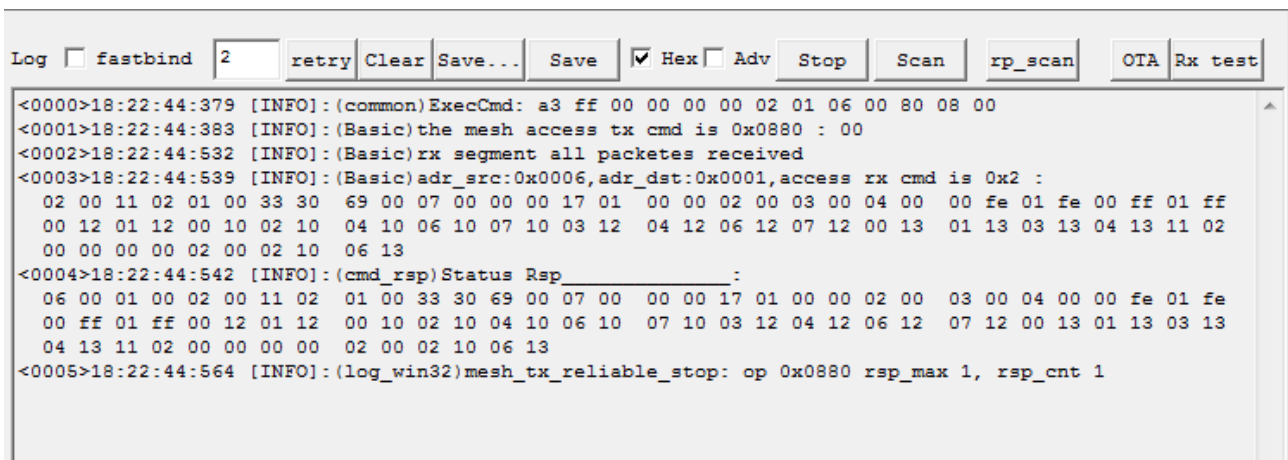


Figure 4.37: Return value

"DelNode":

By clicking this button, NODE_RESET will be sent to delete current node from the network. If the deletion succeed, the red LED light of the node will flash for 8 times, and the node will run reboot operation.

4.6 Time model operation

1) The node's time model is disabled by default in Firmware, MD_TIME_EN need to be set to 1. Gateway 8269 is disabled by default, need to be enabled, gateway 8258 is enabled by default.

2) Double click to choose the node.

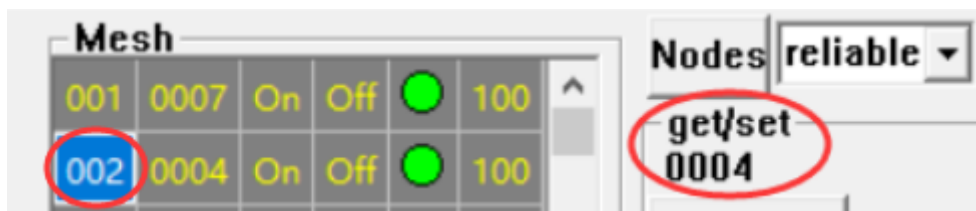


Figure 4.38: Double click to choose the node

3) Click "set time", the tool will send current time of the PC to the node via "TIME_SET". Time will display as following, and will refresh automatically.

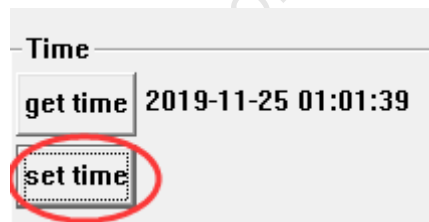


Figure 4.39: "set time"

Note: the parameters of time set when sending is shown in detail below:

```
typedef struct{
    u32 TAI_sec;           // 32bit is enough for 2000 ~ 2099 year
    u8 TAI_sec_rsv;
    u8 sub_sec;
    u8 uncertainty;
    u16 time_auth          :1;
    u16 TAI_UTC_delta      :15;
    u8 zone_offset;
}time_status_t;
```

Figure 4.40: time set parameters

- TAI_sec: the value compares with time zone 0.
- zone_offset: set current time zone, unit is 15 minutes.

e.g.: Beijing time 2019/1/1 09:00:00(UTC+8) configuration:

```
void tx_cmd_time_set_local_sample()
{
    // beijing: 2019/1/1 09:00:00 (time zone: east 8)
    s8 zone_hour = 8;    // Positive numbers are eastwards
    mesh_utc_t UTC = {0};
    UTC.year = 2019;
    UTC.month = 12;
    UTC.day = 4;
    UTC.hour = 10 - zone_hour; // translate to 0 time zone.
    UTC.minute = 0;
    UTC.second = 0;
    u32 TAI_sec = get_TAI_sec(&UTC);

    time_status_t time_set = {0};
    time_set.TAI_sec = TAI_sec;
    time_set.zone_offset = get_time_zone_offset(zone_hour*60);

    access_cmd_time_set(0xffff, 1, &time_set);
}
```

Figure 4.41: Switch between TAI and local time

The above function is to show how to switch local time to TAI, but normally it not in this way. Time set is set from Mobile APP or PC, and both have API to get current TAI_sec and zone_offset, e.g., PC firmware operate in the following way:

(OFFSET_1970_2000 is because PC's base time is 1970 while SIG MESH's base time is 2000)

```
void CTLMeshDlg::OnBnClickedSetTime()
{
    if(0 != Sel_Ele_Check()) {
        return ;
    }
    time_status_t setttime={0};
    //mesh.UTC_t set_utc;
    CTime time = CTime::GetCurrentTime();
    u32 ntSeconds2 = (u32)time.GetTime();
    setttime.TAI_sec=ntSeconds2-OFFSET_1970_2000;
    TIME_ZONE_INFORMATION tz;
    u32 dwRet = GetTimeZoneInformation(&tz);
    setttime.zone_offset = get_time_zone_offset(-tz.Bias);
    if(is_support_model_dst(mesh_sel,SIG_MD_TIME_S,1)) {
        access_cmd_time_set(mesh_sel, 0, &settime);
    }else{
        LOG_MSG_INFO (TL_LOG_COMMON, 0, 0, "Node not support time model",0);
    }
}
```

Figure 4.42: PC firmware operate

4.7 Scene model operation

- 1) The node's scene model is disabled by default in Firmware, MD_SCENE_EN need to be set to 1. Gateway 8269 is disabled by default, need to be enabled, gateway 8258 is enabled by default.
- 2) Double click to choose the node.
- 3) Set the node's status to the scene wanted one via UI or INI. E.g., generic on/off set and lightness set.
- 4) Input scene number, then click "Store", and scene adding command (SCENE_STORE) will be sent, to set node's current status to corresponding scene ID, and list all the configured scene ID in the list, as shown in figure below.

Note:

- (SCENE_STORE) have only scene number, no light status information. Node will automatically save currently status information like on/off, lightness as scene status when receive scene adding command.

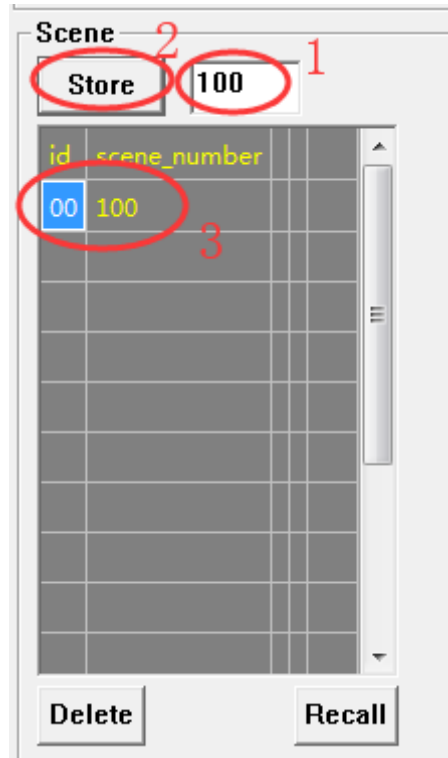


Figure 4.43: Input scene number

5) Recall scene, i.e., set light status to status defined by scene. Click buttons in the following figure, then click "Recall".

Note:

- Recall scene will change light status, but it is not reported because publish status is not configured, to refresh UI, configure publish parameter in corresponding model.

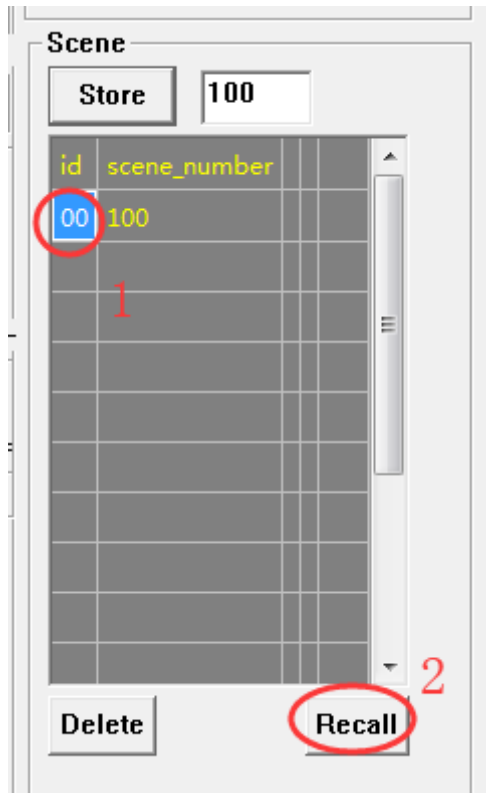


Figure 4.44: Recall scene

- 6) Modify scene. No modify command, modify with scene store.
- 7) Delete scene. Click buttons in the following picture, then press "Delete".

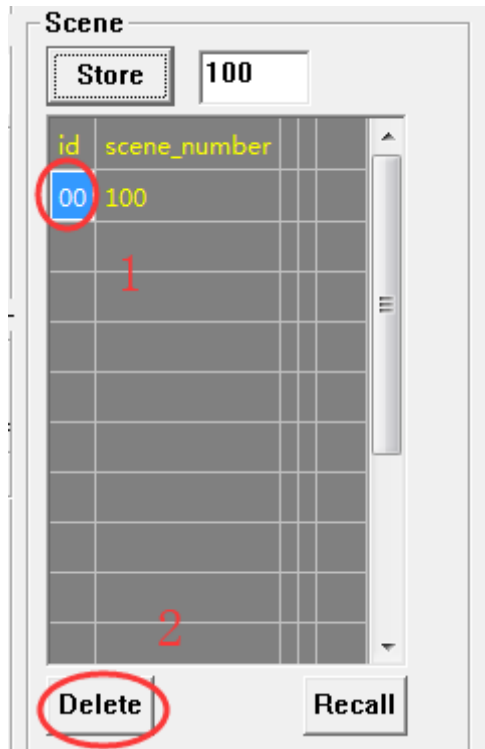


Figure 4.45: Delete scene

4.8 Scheduler model operation

1) Parameters (refer spec for detail)

- Year: any: each year, custom: a specific year, base: year 2000, ie, 0 means year 2000, 19 means year 2019
- Month: can choose 1/multiple/all. Note: both blank and all means choose all
- Day: any: each day, custom: a specific day
- Week: can choose 1/multiple/all. Note: both blank and all means choose all
- Hour: any: each hour, once a day: randomly respond once a day, and the random number is generated daily; custom: a specific year,
- Min: any: each minute, every 15 means responds on 0/15/30/45, every 20 means responds on 0/20/40, once an hour: randomly respond once an hour, and the random number is generated hourly, custom: a specific minute
- Second: similar with Min.

2) Double click to choose the node. If the action column is blank, that means the ID's schedule of the node is not configured yet.

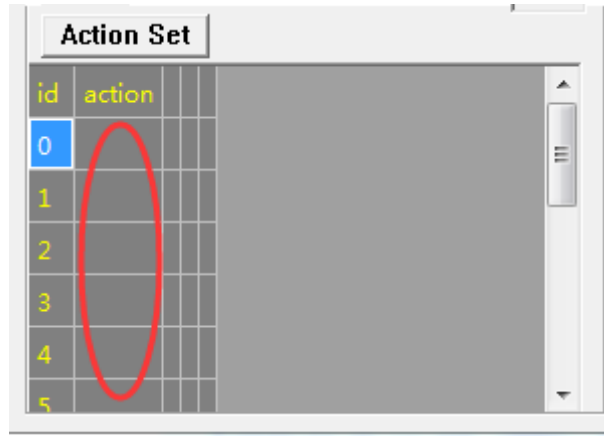


Figure 4.46: Action Set

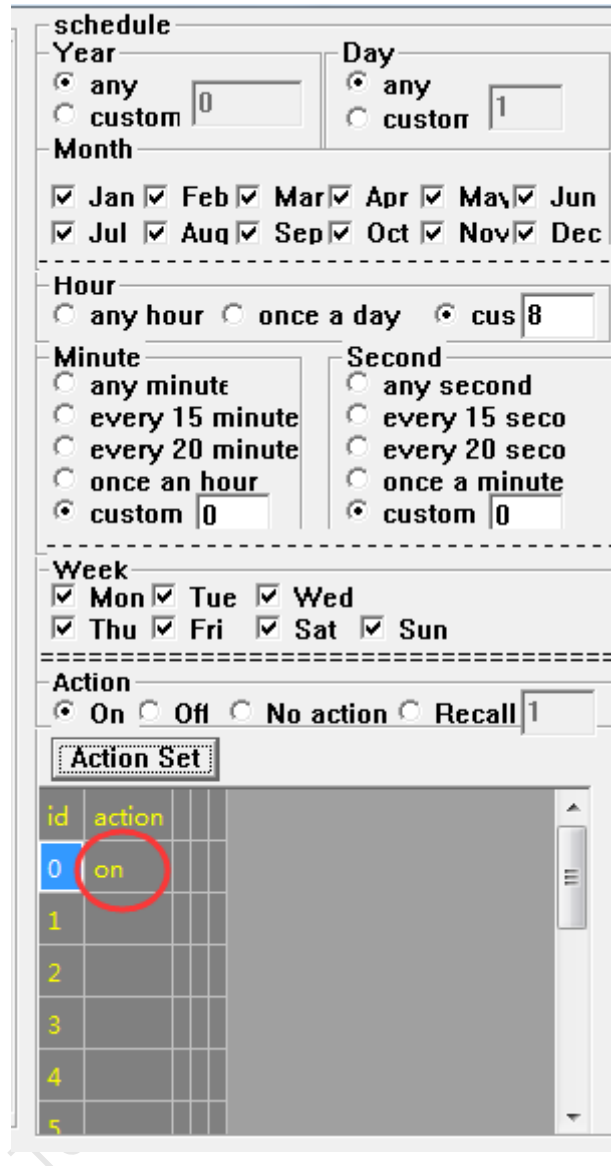
3) Click ID column, choose the to-be configured scheduler's ID (maximum 16 by definition in SIG, range from 0-15), the chosen ID will show in blue background, and the schedule parameter of the ID will be refreshed to the UI above.

The screenshot shows a configuration window for a schedule. The 'schedule' section includes fields for Year, Day, Month, Hour, Minute, and Second, each with radio button options for 'any', 'custom', or specific intervals. The 'Week' section has checkboxes for days of the week. The 'Action' section has radio buttons for 'On', 'Off', 'No action', and 'Recall'. Below these is an 'Action Set' table with columns 'id' and 'action'. The 'id' column is highlighted with a red circle, and the value '0' is visible in the first row.

id	action
0	
1	
2	
3	
4	
5	

Figure 4.47: Click "id"

- 4) User can modify schedule parameter, then click "Action Set" to send SCHD_ACTION_SET to configure. Because of the UI display limit, only action parameters are shown in the list, i.e., "on", "off", "no action", "recall", if the field is blank, that means the schedule is not configured yet.
- Click a specific value in ID column to check detail information of a schedule id.



schedule

Year
☒ any ☐ custom

Day
☒ any ☐ custom

Month
☒ Jan ☒ Feb ☒ Mar ☒ Apr ☒ May ☒ Jun
☒ Jul ☒ Aug ☒ Sep ☒ Oct ☒ Nov ☒ Dec

Hour
☐ any hour ☐ once a day ☒ custom

Minute
☐ any minute
☐ every 15 minute
☐ every 20 minute
☐ once an hour
☒ custom

Second
☐ any second
☐ every 15 seco
☐ every 20 seco
☐ once a minute
☒ custom

Week
☒ Mon ☒ Tue ☒ Wed
☒ Thu ☒ Fri ☒ Sat ☒ Sun

Action
☒ On ☐ Off ☐ No action ☐ Recall

Action Set

id	action
0	on
1	
2	
3	
4	
5	

Figure 4.48: schedule parameter

5) Delete Schedule

No specific delete command in SIG. Set action of the schedule to "No action" to delete the schedule.

5 Factory Test Mode

5.1 Purpose

Factory test mode is used to manufacture, to execute some common control tests without provision, e.g., on/off, lightness control, CT control and etc. Gateway and GATT master dongle support this mode while APP does not support for now.

5.2 Factory Test Mode Parameters

- unicast address: default is the lower 15bit of MAC, if the lower 15bit is 0, then take 1 as unicast address.
- network key, app key, device key, iv index use the compiled default value.

5.3 Default Test-able Commands

The control-able models are defined by `factory_test_model_array[]`, while the useable commands of configure model are defined by `factory_test_cfg_op_array[]`.

```
const u16 factory_test_model_array[] = {  
    SIG_MD_G_ONOFF_S, SIG_MD_LIGHTNESS_S, SIG_MD_FW_UPDATE_S,  
    SIG_MD_LIGHT_CTL_S, SIG_MD_LIGHT_CTL_TEMP_S, SIG_MD_LIGHT_HSL_S,  
    SIG_MD_LIGHT_XYL_S  
};  
const u16 factory_test_cfg_op_array[] = {COMPOSITION_DATA_GET, NODE_RESET};
```

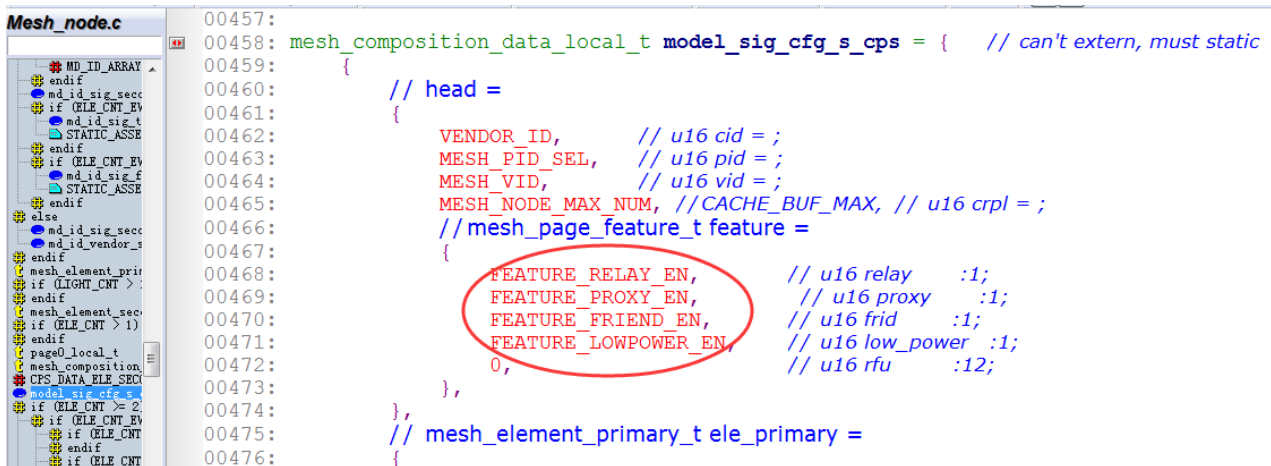
Figure 5.1: Default testable commands

Controls under factory mode do not need provision, please refer to 4.5. Please be noted, the mode needs all nodes unprovisioned, including gateway and master dongle.

6 Important SDK Modules

6.1 Configure Mesh SDK Default Feature

- 1) Mesh nodes describe their supporting features in composition data (model_sig_cfg_s_cps.page0.head.feature), SDK initialization is shown as below:



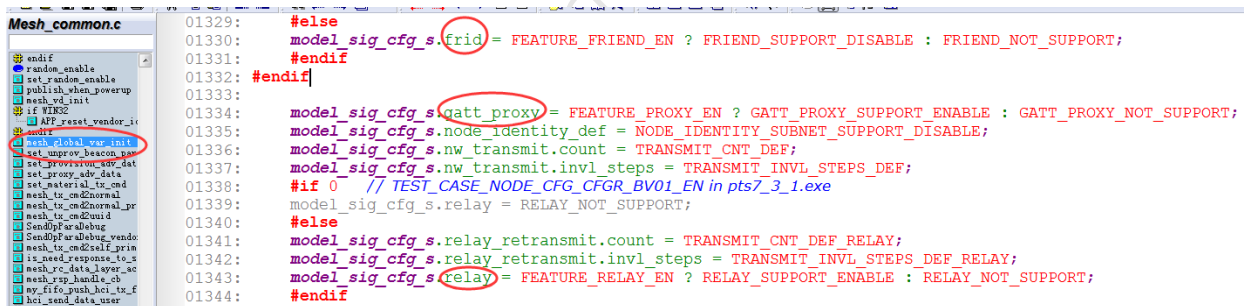
```

00457:
00458: mesh_composition_data_local_t model_sig_cfg_s_cps = { // can't extern, must static
00459: {
00460: // head =
00461: {
00462:     VENDOR_ID, // u16 cid = ;
00463:     MESH_PID_SEL, // u16 pid = ;
00464:     MESH_VID, // u16 vid = ;
00465:     MESH_NODE_MAX_NUM, // CACHE_BUF_MAX, // u16 crpl = ;
00466:     // mesh_page_feature_t feature =
00467:     {
00468:         FEATURE_RELAY_EN, // u16 relay :1;
00469:         FEATURE_PROXY_EN, // u16 proxy :1;
00470:         FEATURE_FRIEND_EN, // u16 frid :1;
00471:         FEATURE_LOWPOWER_EN, // u16 low_power :1;
00472:         0, // u16 rfu :12;
00473:     },
00474: },
00475: // mesh_element_primary_t ele_primary =
00476: {

```

Figure 6.1: SDK initialization

- 2) Composition data defines supports or not, enable/disable can also be defined under “support” status. Please refer to the configuration action model_sig_cfg_s.frid in mesh_global_var_init().



```

01329: #else
01330: model_sig_cfg_s.frid = FEATURE_FRIEND_EN ? FRIEND_SUPPORT_DISABLE : FRIEND_NOT_SUPPORT;
01331: #endif
01332: #endif
01333:
01334: model_sig_cfg_s.gatt_proxy = FEATURE_PROXY_EN ? GATT_PROXY_SUPPORT_ENABLE : GATT_PROXY_NOT_SUPPORT;
01335: model_sig_cfg_s.node_identity_def = NODE_IDENTITY_SUBNET_SUPPORT_DISABLE;
01336: model_sig_cfg_s.nw_transmit.count = TRANSMIT_CNT_DEF;
01337: model_sig_cfg_s.nw_transmit.invl_steps = TRANSMIT_INVL_STEPS_DEF;
01338: #if 0 // TEST_CASE_NODE_CFG_CFGFR_BV01_EN in pts7_3_1.exe
01339: model_sig_cfg_s.relay = RELAY_NOT_SUPPORT;
01340: #else
01341: model_sig_cfg_s.relay_retransmit.count = TRANSMIT_CNT_DEF_RELAY;
01342: model_sig_cfg_s.relay_retransmit.invl_steps = TRANSMIT_INVL_STEPS_DEF_RELAY;
01343: model_sig_cfg_s.relay = FEATURE_RELAY_EN ? RELAY_SUPPORT_ENABLE : RELAY_NOT_SUPPORT;
01344: #endif

```

Figure 6.2: Enable/disable the configuration

During working procedure, user can enable/disable these features with the following commands: CFG_FRIEND_SET, CFG_RELAY_SET and CFG_GATT_PROXY_SET.

During working procedure, user can enable/disable these features with the following commands: CFG_FRIEND_SET, CFG_RELAY_SET and CFG_GATT_PROXY_SET.

- 3) For default features of each compiling project, please refer to Demo Project in SDK Instruction.

6.2 Share model introduction

In SIG Mesh spec, each model has an independent configured group index, so, each model has an independent index number data, the maximum storage no. is 8(SUB_LIST_MAX), as shown below.

```
typedef struct{
    u16 ele_adr; // use as primary address for model_sig_cfg_s_t
    u8 no_pub :1; // means not support publish function
    u8 no_sub :1; // means not support subscription function; must before pub and sub par
    u8 pub_trans_flag :1; // transition process was ongoing flag.
    u8 pub_2nd_state :1; // eg: lightness and lightness linear.
    u8 rsv2;
    bind_key_t bind_key[BIND_KEY_MAX];
    u8 pub_uuid[16];
    cb_pub_st_t cb_pub_st; // no need to save, fix later
    u32 cb_tick_ms; // no need to save, fix later
    u16 pub_adr; // pub_adr and pub_par must existed if sub_list existed // offset:32
    mesh_model_pub_par_t pub_par;
    u8 rfu3[1];
    u16 sub_list[SUB_LIST_MAX]; // pub_adr, pub_par, sub_list must follow com if existed
    u8 sub_uuid[SUB_LIST_MAX][16];
}model_common_t;
```

Figure 6.3: Configure group index

In some app, when configuring group index for a certain model, user may hope some other related models can also be configured automatically at the same time to say group index number configure time. Share model is designed for this scene, the corresponding macro switch is SUBSCRIPTION_SHARE_EN, user can put all sig models need to be bound in sub_share_model[], while all vendor models need to be bound in ub_share_model_vendor[].

Please refer to sub_share_model[] introduction in Chapter 8 for detail.

6.3 Heartbeat demo

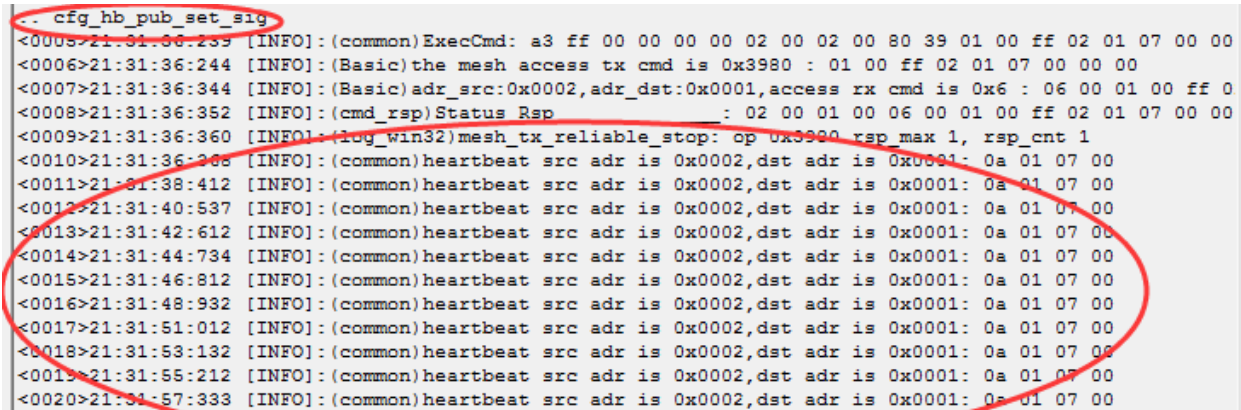
No heartbeat message is sent by default, user can configure this by sending command HEART-BEAT_PUB_SET. After the command is sent, the node will send out heartbeat message. Below is an example: send heartbeat message every 2 seconds, and the corresponding INI command:

```
CMD-cfg_hb_pub_set_sig
=a3 ff 00 00 00 00 00 00 02 00 80 39 01 00 ff 02 05 07 00 00 00
```

The parameters are described as following:

- 80 39: op code
- 01 00: destination address of heartbeat is 0x0001
- ff: CountLog, 0xff means infinity
- 02: PeriodLog, period is 2 powers (02-1) i.e. 2 seconds
- 05: InitTTL, set TTL value when send heartbeat message
- 07 00: features, once any of relay, friend, proxy feature changes status(switches between enable and disable) the heartbeat message will be immediately reported.
- 00 00: NetKeyIndex.

Heartbeat packet can be seen in firmware tool.



```

.. cfg_hb_pub_set_sig
<0005>21:31:36:239 [INFO]:(common)ExecCmd: a3 ff 00 00 00 02 00 02 00 80 39 01 00 ff 02 01 07 00 00
<0006>21:31:36:244 [INFO]:(Basic)the mesh access tx cmd is 0x3980 : 01 00 ff 02 01 07 00 00 00
<0007>21:31:36:344 [INFO]:(Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x6 : 06 00 01 00 ff 0
<0008>21:31:36:352 [INFO]:(cmd_rsp)Status Rsp : 02 00 01 00 06 00 01 00 ff 02 01 07 00 00
<0009>21:31:36:360 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x3980 rsp_max 1, rsp_cnt 1
<0010>21:31:36:368 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0011>21:31:38:412 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0012>21:31:40:537 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0013>21:31:42:612 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0014>21:31:44:734 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0015>21:31:46:812 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0016>21:31:48:932 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0017>21:31:51:012 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0018>21:31:53:132 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0019>21:31:55:212 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00
<0020>21:31:57:333 [INFO]:(common)heartbeat src adr is 0x0002,dst adr is 0x0001: 0a 01 07 00

```

Figure 6.4: Heartbeat packet

- 0a: heartbeat opcode, please note that heartbeat is control message.
- 01: InitTTL, same value as that of heartbeat set message parameter.
- 07: Features, same value as that of heartbeat set message parameter.

64 Mesh Receiving Transmitting Self-defined Packet

Self-defined Packet Transmitting

SDK will call `bls_set_advertise_prepare (app_advertise_prepare_handler)` to register packet and send call back function when initialization, the packet can be visited and modified before sending out, SDK call the packet sending function once every 10ms by default. If user want to send self-defined packet, send it in a similar way of sending mesh-connectable packet in `gatt_adv_prepare_handler`. Control packet sending interval by `clock_time_exceed` software timing, `rf_packet_adv_t * p` to packet to be sent, user can modify the contents of the packet pointed to by `p` according to packet format (please refer to `set_adv_provision()`), then set the return value `ret` to 1, means will send packet.

Receiving/Filtering Connectable Packet

SDK call `adv_filter_proc()` during `rf rx` interrupt to filter received packets, return 0 to abandon received packet, return 1 to receive and compress into `blt_rxfifo` without filtering. All connectable packet will be filtered by default. If user want to receive connectable packet, then open `USER_ADV_FILTER_EN`, in `user_adv_filter_proc()`, set the packet you want to return 1. It is not recommended to set all connectable packet to return 1, because this will do no filter to the packets, all packets will be pushed into `blt_rxfifo`, including those packets sent by other no-mesh BLE products, this may be beyond the storage capability of our receiving buffer, thus result in losing mesh message as well as the mesh packet receiving.

`blt_sdk_main_loop ()` will check `blt_rxfifo`, if there is data need to be processed, it will call `app_event_handle()`, use may process the received connectable packet in the `if(LL_TYPE_ADV_NONCONN_IND != (pa->event_type & 0x0F))` branch of this callback function, as shown below:

```
int app_event_handler (u32 h, u8 *p, int n)
{
    static u32 event_cb_num;
    event_cb_num++;
    int send_to_hci = 1;

    if (h == (HCI_FLAG_EVENT_BT_STD | HCI_EVT_LE_META)) //LE event
    {
        u8 subcode = p[0];
        #if MI_API_ENABLE
        telink_ble_mi_app_event(subcode,p,n);
        #endif

        //----- ADV packet -----
        if (subcode == HCI_SUB_EVT_LE_ADVERTISING_REPORT) // ADV packet
        {
            event_adv_report_t *pa = (event_adv_report_t *)p;
            if (LL_TYPE_ADV_NONCONN_IND != (pa->event_type & 0x0F)) {
                return 0;
            }

            #if DEBUG_MESH_DONGLE_IN_VC_EN
            send_to_hci = mesh_dongle_adv_report2vc(pa->data, MESH_ADV_PAYLOAD);
            #else
            send_to_hci = app_event_handler_adv(pa->data, ADV_FROM_MESH, 1);
            #endif
        }
    }
}
```

Figure 6.5: Receiving and filtering connectable packet

7 Vendor Model Introduction

7.1 Adding vendor model

Normally, it is not necessary for users to add model, because currently SIG model is completed, vendor model can use the already added vendor model: `VENDOR_MD_LIGHT_C`, `VENDOR_MD_LIGHT_S`, only need to add op code.

If user want to publish multiple status for current vendor model, new model need to be added. It not recommended to do so. Please contact us, or refer to `MD_SCENE_EN` to add new model.

7.2 Adding vendor command register reference

Command and opcode are refer to the same item in this article, op code (1BYTE) + vendor id(2BYTE).

Vendor model have 64 op codes in total. Note, it's not each product type has 64, it's all the product with the same vendor id in the whole mesh network has 64 in total. When `MESH_USER_DEFINE_MODE` chooses `MESH_NORMAL_MODE`, 32 must be reserved for Telink, i.e., `0xCO—0xDF`, and `0xE0—0xFF` is for users. It it recommended to use them by sub-commands way. Some Telink self-defined function will be disabled if user use more than 32 op codes. Please contact us in this case.

The maximum length of vendor command parameters is 377byte, but SIG mesh bottom layer will automatically de-pack packets longer than 8 byte. It is recommended that keep frequently used control commands not longer than 8 byte.

Please refer to `mesh_cmd_sig_func_t mesh_cmd_vd_func[] = {... ...}` in `vendor_model.c` for vendor model's register, as shown below:

```
#if DEBUG_VENDOR_CMD_EN
{VD_LIGHT_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set, VD_LIGHT_ONOFF_STATUS},
{VD_LIGHT_ONOFF_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_get, VD_LIGHT_ONOFF_STATUS},
{VD_LIGHT_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set, STATUS_NONE},
{VD_LIGHT_ONOFF_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C, cb_vd_light_onoff_status, STATUS_NONE},
#endif
```

Figure 7.1: Code model

Note:

- `mesh_cmd_vd_func[]` is a const array, so this is a read-only array, thus can save RAM space.

If the added command code need TID field, it need to be registered in `is_cmd_with_tid_vendor()`, see 3.2.2 for detail.

7.2.1 mesh_cmd_sig_func_t introduction

```
typedef struct{
    u16 op;
    u16 status_cmd;
    u32 model_id_tx;
```



```
u32 model_id_rx;
cb_cmd_sig2_t cb;
u32 op_rsp;
}mesh_cmd_sig_func_t;
```

- **op**: new-added command's opcode, no matter it is SIG command or vendor command, is expressed as u16, vendor command do not need to fill the vendor id bytes, library bottom layer will add automatically.
- **status_cmd**: If the opcode is "status command" corresponding to certain "acknowledge request command", e.g. VD_LIGHT_ON/OFF_STATUS, the "status_cmd" should be set as 1; otherwise it should be set as 0. When model_id_rx is client model, "status_cmd" should be set as 1. This status_cmd flag is used in Library.
- **model_id_tx**: Corresponding model ID sending this command. E.g., when publish status, first, check mesh_cmd_vd_func[] according to op to get model_id_tx, then get the corresponding global variables, such as model_sig_g_on/off_level.on/off_srv, then get the model_sig_g_on/off_level.on/off_srv->com. pub_adr and its publish parameter, finally publish status.
- **model_id_rx**: Corresponding model ID receiving this command. If the node does not have corresponding model id in composition data, this opcode won't be processed.

When supports a specific model, the model parameters should be checked to determine if the model has bond corresponding app key, when the destination address is a group address, if the model has follow the corresponding group.

- When this command is received, callback processing function mesh_rc_data_layer_access_cb()->p_res->cb() is invoked, users can process their own app in this callback function.
- **op_rsp**: If this opcode is "acknowledge request command", the "op_rsp" should be set as corresponding ack command; otherwise it should be set as "STATUS_NONE". The sending end will use this to determine if it has received the corresponding status response after it send the command.

7.2.2 Add acknowledge command (request command with status response)

Take "VD_LIGHT_ON/OFF_SET" as an example:

- 1) Add the content below in the "mesh_cmd_vd_func[]":

```
{VD_LIGHT_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, &cb_vd_light_onoff_set,
  VD_LIGHT_ONOFF_STATUS}
```

- 2) This command needs the "TID (Transmit ID)" field. Therefore, it's needed to add corresponding branch in the "is_cmd_with_tid_vendor()", and mark the location of the TID field in the access payload.

In the library, there will be a global variable mesh_tid that manages TID uniformly. When a command is sent, it will be automatically increased by one and copied to the TID field of the command parameter area. Please refer to [Vendor model command format] for detail.

TID purpose: If repeated TID is received within a specific duration (currently it's set as 6s by default), the corresponding action won't be implemented, but it will respond with response. This recognition action is implemented in the library, while the upper APP can directly judge the flag "cb_par->re-transaction".

TID normally is for light status control commands. TID is to prevent acting repeatedly when receive retry in a specific duration, thus cause wrong delay time, for example, when the node receive OFF command, the corresponding delay time is 1s, and when the delay time passes for 0.8 s, receives retry command, if this command is executed, the delay time will re-count for 1 s, so the phenomena is that the node will be off after 1.8s. This will also cause light flash, e.g., 2 app control the same node at the same time, one act generic on, the other generic off, and both have retry action (message have different sequence numbers but same TID), for this case, what we expected is, light only execute 1 on and 1 off, the final status is determined by the last received command. This can be done if there is TID identification, without TID, the light may execute multiple on/off actions, thus causes flash.

In SIG standard commands, only light control command, such as on/off set, level set, lightness set, CT set, use TID.

Commands like lightness get, and config set/get in config model do not use TID.

Do not add vendor command if not necessary, do not waste any byte of the limited effective bytes.

- 3) Compile the function "cb_vd_light_on/off_set()", invoke the "light_on/off_idx()" to execute light on/off action.
- 4) Since this command needs ack response, it's needed to compile corresponding ack function "vd_light_on/off_st_rsp()", and invoke the "mesh_tx_cmd(VD_LIGHT_ON/OFF_STATUS,.....)" in the "vd_light_on/off_st_rsp()" to implement ack response.

Note:

- Call mesh_tx_cmd_rsp() instead of mesh_tx_cmd2normal_primary() to reply status after receiving a command, because, in a network with multiple network keys and app keys, when reply status, you have to use the same encrypt key with the decrypt key when receiving the packet.
- mesh_tx_cmd2normal_primary() use the first key by default to send, normally is used for send command, when reply status, do not use this function.

- 5) Assemble the interface "vd_cmd_on/off()" sending the "VD_LIGHT_ON/OFF_SET" command.

"rsp_max" indicates the number of nodes that need response.

- When the "adr_dst" is unicast, the "rsp_max" can be set as 1 (recommended) or 0.
- When the "adr_dst" is group, the "rsp_max" should be set as the number of elements owned by group according to the record in APP database.

7.2.3 Add Unacknowledged command

Take "VD_LIGHT_ON/OFF_SET_NOACK" as an example.

- 1) Add the content below in the "mesh_cmd_vd_func[]":

```
{VD_LIGHT_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, &cb_vd_light_onoff_set,
  STATUS_NONE}
```

- 2) This command needs the "TID (Transmit ID)" field. Please refer to the method of adding acknowledge command.
- 3) Compile the function "cb_vd_light_on/off_set()" (shared with "VD_LIGHT_ON/OFF_SET"). Please refer to the method of adding acknowledge command.
- 4) This command does not need ack response.
- 5) Assemble the interface "vd_cmd_on/off()" sending the "VD_LIGHT_ON/OFF_SET" command. Please refer to the method of adding acknowledge command.

7.24 Publish function registration

When the publish parameter of the light node model is set by the CFG_MODEL_PUB_SET command, the model has the publish function. When the model status changes, it will automatically publish a status message to the publish address configured by the publish parameter. In addition, the publish parameter can also be configured to send periodically, please check publish command parameter definition spec in [4.3.2.16 Config Model Publication Set] for detail.

In order to implement the above automatic publish function, you need to register the publish function for the model to send status messages, as shown below:

```
void mesh_model_cb_pub_st_register()  
{  
    .....  
    MODEL_PUB_ST_CB_INIT(model_vd_light.srv, &vd_light_onoff_st_publish);  
    .....  
}
```

Figure 7.2: publish function

When the status changes, or after the publish cycle time expires, the mesh stack will call back this function and send a status message.

8 Global Configuration File Introduction

8.1 mesh_config.h

PROXY_HCI_SEL:

For debugging, developer can choose PROXY_HCI_GATT by default.

DEBUG_VENDOR_CMD_EN:

Enable/disable vendor model debug command. Enabled by default.

FAST_PROVISION_ENABLE:

This is a private mode, can provision with multiple nodes at the same time, supports group action and relay network. Disabled by default.

MESH_USER_DEFINE_MODE:

Define authentication mode during provision, MESH_NORMAL_MODE: no OOB mode; others are static OOB mode, please refer to Chapter 13.

SUBSCRIPTION_SHARE_EN:

Private mode. When receiving command to setting group index for on/off model, firmware will automatically add the same group index to models listed in sub_share_model_sig[] and sub_share_model_vendor[].

PROVISION_FLOW_SIMPLE_EN:

Same as the standardized provision, provision nodes one by one, i.e., only one node is configuring network at the same time. When node receives app key add, automatically binds key to every model. Provisioner does not need to send key bind command. Simplify provision process and reducing provision time.

AIS_ENABLE / MI_API_ENABLE:

Please refer to Chapter 13.

LIGHT_TYPE_SEL:

Select light type. Currently supported light types are mutually exclusive. Please refer to 1.3.

The following are model on/off control macro, e.g., MD_LIGHTNESS_EN, when it is enables, whether it enables client or server model, or both, is determined by MD_SERVER_EN, MD_CLIENT_EN and MD_CLIENT_VENDOR_EN. Check introductions of these 3 macros below.

LIGHT_TYPE_CT_EN:

Enable / Disable CT light related model, includes Light CTL Server, Light CTL Setup Server, Light CTL Temperature Server, Light CTL Client.

LIGHT_TYPE_HSL_EN:

Enable / Disable HSL light related model, includes Light HSL Server, Light HSL Hue Server, Light HSL Saturation Server, Light HSL Setup Server, Light HSL Client.

MD_LIGHT_CONTROL_EN:

Enable / Disable (Default) Lighting Control related model on/off control, includes Light LC Server, Light LC Setup Server, Light LC Client.

MD_LIGHTNESS_EN:

Enable / Disable Lightness related model, includes Light Lightness Server, Light Lightness Setup Server, Light Lightness Client.

MD_LEVEL_EN:

Enable (Default) / Disable Generic Level Model. Each status can have a corresponding level model.

MD_MESH_OTA_EN:

Enable / Disable (Default) Mesh_OTA_Model interface.

MD_ONOFF_EN:

Enable (Default) / Disable Generic On/off Model.

MD_DEF_TRANSIT_TIME_EN:

Enable (Default) / Disable Generic Default Transition Time Model.

MD_POWER_ONOFF_EN:

Enable (Default) / Disable Generic Power On/off Model. Enable / Disable at the same time with MD_DEF_TRANSIT_TIME_EN, because the parameters of these 2 models are save in the same flash sector.

MD_TIME_EN:

Disable (Default) / Enable Time Model.

MD_SCENE_EN:

Disable (Default) / Enable Scene Model.

MD_SCHEDULE_EN:

Disable (Default) / Enable Schedule Model. Disable / Enable at the same time with MD_TIME_EN, because schedule depends on time.

MD_PROPERTY_EN:

Enable / Disable Property model, includes Generic User Property Server, Generic Admin Property Server, Generic Manufacturer Property Server, Generic Client Property Server, Generic Property Client.

MD_LOCATION_EN:

Enable/Disable Location model, includes Generic Location Server, Generic Location Setup Server, Generic Location Client.

MD_SENSOR_EN:

Enable/Disable Sensor model, includes Sensor Server, Sensor Setup Server, Sensor Client.

MD_BATTERY_EN:

Enable/Disable Battery model, includes Generic Battery Server, Generic Battery Client

MD_SERVER_EN:

Enable the SIG and vendor models of the enabled server models when the value is set to 1, e.g., lightness server and VENDOR_MD_LIGHT_S.

MD_REMOTE_PROV:

Enable/Disable Remote provision model. Default disable.

MD_CLIENT_EN:

Enable the client SIG model when the value is set to 1, e.g. lightness client.

MD_CLIENT_VENDOR_EN: Enable client vendor model: VENDOR_MD_LIGHT_C when the value is set to 1.

MD_VENDOR_2ND_EN:

Enable the second vendor server model VENDOR_MD_LIGHT_S2 when the value is set to 1. Normally vendor model needs only 1.

Note:

- Generally nodes do not need Client Model, so Client Model is disabled for light side by default so as to save RAM. Client model is controlled by MD_CLIENT_EN and MD_CLIENT_VENDOR_EN, some light node need to enable vendor client model but not SIG client model.

FACTORY_TEST_MODE_ENABLE:

Enable (Default) / Disable factory test mode.

For the convenience of factory test, in the case of no provision, default key can be used to implement simple operations such as turning on/off node, adjusting luminance.

MANUAL_FACTORY_RESET_TX_STATUS_EN:

Set whether to send NODE_RESET_STATUS to notify gateway or app after 5 times of booting/reset.

KEEP_ONOFF_STATE_AFTER_OTA:

Set whether to keep the on/off status of the light before reset.

ELE_CNT_EVERY_LIGHT:

Element no. of each light. E.g., a CT light need 2 elements, most model will put it in the first element, only Light CTL Temperature Server and corresponding level model are in the second element.

Since lightness and CTL Temperature can both be controlled by level model commands, if there is only 1 element address, when receiving level set command, it is impossible to determine whether it is to control lightness or Temperature.

Note the difference between LIGHT_CNT and ELE_CNT.

LIGHT_CNT is how many same lights in the BLE module, e.g., 2 CT lights.

$ELE_CNT = ELE_CNT_EVERY_LIGHT * LIGHT_CNT$ is how many elements in this node. It is also the element address no. when provision.

FEATURE_FRIEND_EN:

Set whether to support Friend Feature

FEATURE_LOWPOWER_EN:

Set whether to support Low Power Feature.

FEATURE_PROV_EN:

Provision switch, need to enable.

FEATURE_RELAY_EN:

Set whether to support Relay Feature.

FEATURE_PROXY_EN:

Set whether to support Proxy Feature.

MAX_LPN_NUM:

Set the number of low power nodes supported by one friend node. Currently it's set as 2, it is recommended to limit this value less than 10(the maximum verified number) if the user need to modify this number. Too big value will cause higher possibility of packets conflict when friend reply respond to multiple LPN, and thus cause time delay and higher power consumption of LPN node, the RAM consuming will also increase.

USER_DEFINE_SET_CCC_ENABLE:

Must enable. Set whether App controlled node report notify/indication.

SEND_STATUS_WHEN_POWER_ON:

Set whether to send luminance state packet when power on, default sending address is 0xffff.

8.2 mesh_node.h

SUB_LIST_MAX:

Maximum supporting subscription address number (group index number). Cannot be modified, because this macro is used in library.

BIND_KEY_MAX:

Maximum supporting bind key number, cannot be modified, because this macro is used in library.

SCENE_CNT_MAX:

Maximum configurable scene number, can be modified.

8.3 app_mesh.h

8.3.1 Macro introduction

TRANSMIT_CNT_DEF:

Set message default transmit cnt, i.e., retry time of each command

Retry time = TRANSMIT_CNT_DEF + 1.

TRANSMIT_INVL_STEPS_DEF:

Set message default transmit interval, i.e., retry interval.

Retry interval = (TRANSMIT_INVL_STEPS_DEF + 1) * 10ms + (0-10) ms.

TRANSMIT_CNT_LPN_ACCESS_CMD:

For LPN node, control commands' transmit cnt is defined by TRANSMIT_CNT_DEF, e.g., friend request, friend poll, other message is defined by TRANSMIT_CNT_LPN_ACCESS_CMD, e.g., on/off status.

TRANSMIT_CNT_DEF_RELAY, TRANSMIT_INVL_STEPS_DEF_RELAY:

Relay's transmit count and transmit interval.

MESH_ADV_CMD_BUF_CNT:

Set message transmitting buffer size, excludes relay message.

MESH_ADV_BUF_RELAY_CNT:

Set relay message of relay message.

SEC_NW_BC_INV_DEF_100MS:

Set security beacon's transmitting interval when provision, unit is 100ms.

8.3.2 Function introduction

mesh_tx_cmd(material_tx_cmd_t *p)

This is a common function to send command.

1) Parameters:

```
type typedef struct{
union{ //point to parameter address
u8 *par;
u8 *p_ac;
};
union{ //parameter length
u32 par_len;
u32 len_ac;
};
u16 adr_src; //source address
u16 adr_dst; //destination address
u8* uuid; //point to virtual address
model_common_t *pub_md; // point to model parameter
u32 rsp_max; //number of nodes that need response
u16 op; // command code
u16 nk_array_idx; // network_key index
u16 ak_array_idx; // app_key index
u8 retry_cnt; // number of retry times
}material_tx_cmd_t;
```

Parameter values are determined by the parameters of the invoking function "mesh_tx_cmd2normal_primary(u16 op, u8 *par, u32 par_len, u16 adr_dst, int rsp_max)".

2) Return value

If the return value is 0, it indicates successful command execution.

If the return value is not zero, it indicates transmission failure, e.g. currently there's a command being sent, new command cannot be accepted (busy state), certain parameter is illegal, and etc.

```
int mesh_tx_cmd_primary(u16 op, u8 *par, u32 par_len, u16 adr_dst, int rsp_max)
```

This function serves to fix "adr_src" as "ele_adr_primary", and then assemble "mesh_tx_cmd()".

84 app_provision.c

u8 is_provision_success():

Get the status if the node is provisioned successfully.

u8 is_provision_working():

Get the status if the node is in provision process.

8.5 mesh_node.c

is_own_ele():

Determine if node's adr is the element address of its own.

8.6 mesh_common.c file introduction

HCI fifo:

hci_tx_fifo and hci_rx_fifo are fifos to define and transmit data by peripherals, e.g., gateway nodes and gateway firmware USB communication.

mesh_get_proxy_hci_type():

Define proxy type, PROXY_HCI_GATT by default. PROXY_HCI_USB is debug mode, not open to user.

mesh_tid_save():

Function to save TID. E.g. Commands such as generic on/off need to use tid. If deep sleep mode is not executed, it's not needed to save the tid (just initialize it as 0 after power on). If deep mode is executed, e.g. switch, each key press will initialize all variables including tid, in this case, the tid should be saved.

adv_filter_proc():

IRQ RX will filter the received verified correct packets with this function, e.g., abandon connectable packet. See code for detail.

const u16 sub_share_model[]:

```
const u16 sub_share_mode= {
    SIG_MD_G_ONOFF_S, SIG_MD_G_LEVEL_S, SIG_MD_LIGHTNESS_S, SIG_MD_LIGHTNESS_SETUP_S,
    SIG_MD_LIGHT_CTL_S, SIG_MD_LIGHT_CTL_SETUP_S, SIG_MD_LIGHT_CTL_TEMP_S,
    SIG_MD_LIGHT_HSL_S, SIG_MD_LIGHT_HSL_SETUP_S, SIG_MD_LIGHT_HSL_HUE_S,
    SIG_MD_LIGHT_HSL_SAT_S,
    SIG_MD_SCENE_S, };
```

Refer to SUBSCRIPTION_SHARE_EN introduction in mesh_config.h.

These Models are bonded by default, i.e. when setting subscribing address (assign group), these models will take effect at the same time.

How to call:

Receive CFG_MODEL_SUB_ADD -> mesh_rc_data_layer_access_cb() -> mesh_cmd_sig_cfg_model_sub_set()
-> share_model_sub_by_rx_cmd() -> share_model_sub();

entry_ota_model():

The SDK will callback this function after OTA start command is received.

ota_condition_enable():

Condition to allow GATT OTA. When GATT connection is successful, and "set proxy filter" is received, "pair_login_ok" will be set as 1. (Note: set proxy filter need to be encrypted/decrypted with network key when receiving/transmitting.)

proc_telink_mesh_to_sig_mesh():

It serves to detect whether firmware type before OTA is SIG mesh or other SDK, e.g. Telink mesh. If it is not SIG mesh, product switch and parameter initialization will be executed.

mesh_ota_reboot_proc():

After mesh OTA is finished, delay for 1.5s and then reboot.

How to call: main_loop() -> mesh_loop_process() -> mesh_ota_reboot_proc()

mesh_ble_connect_cb:

Callback this function when GATT connect successfully.

mesh_ble_disconnect_cb:

Callback this function after GATT disconnect.

update_para_change_MTU():

It serves to request for BLE connection parameter update as needed, and prevent starting parameter update during discovery and provision.

gatt_adv_prepare_handler():

1) relay_adv_prepare_handler()

Relay buffer is independent, use different fifo with TX command, relay buffer can transmit packet during TX command transmit interval.

Priority: TX command -> relay -> connectable packet.

2) Others are GATT packets.

app_advertise_prepare_handler ():

When BLE stack bottom layer allows to send adv packet, it will callback this function. If there's adv packet (including connectable adv packet, beacon packet) to be sent in current task, it's only needed to set the parameter "p" as the pointer of the structure.

Priority: message Friend Node send to LPN after it receive LPN poll > TX command > relay > connectable adv packet.

1) get_adv_cmd():

The return value is pointer of mesh message packet to be sent. If it's non-zero value, it indicates there's packet to be sent, including MESH_ADV_TYPE_MESSAGE, MESH_ADV_TYPE_BEACON or SECURE_BEACON type.

2) mesh_adv_cmd_set()

Copy packet to be sent to BLE stack.

3) p_bear -> trans_par_val:

It includes transmit count and transmit interval.

4) mesh_rsp_random_delay_step

When the node receive a group address as destination address, it need to add Random delay for response. Check mesh_rc_data_layer_access_cb() for detail.

5) adv_retry_flag

Serves for cancelling network transmit interval, continuous transmission and etc., e.g., poll sent by LPN after build friendship.

app_l2cap_packet_receive ():

When BLE stack receives packet with payload, it will callback this function, and then invoke the function "blc_l2cap_packet_receive()" to analyze the data. During debugging, developer can print out the data for the convenience of analysis.

chn_conn_update_dispatch():

Negligible currently.

sim_tx_cmd_node2node():

It serves to send unreliable light ON/OFF command with the interval of three seconds for demo demonstration.

usb_id_init():

It's used to configure USB ID. When multiple dongles are connected with one PC simultaneously, they must be configured with different IDs so as to be recognized as different devices by PC.

ble_mac_init():

When parameter location of MAC is illegal value, it will randomly generate a MAC and save it.

mesh_scan_rsp_init():

It serves to fill in the fixed field of "scan rsp" during initialization, e.g. mac address. Mac needs to be used when building database, and iOS system cannot directly obtain it from the AdvA field of adv packet data. Therefore, it should be marked in the content of scan response.

mesh_scan_rsp_update_adr_primary():

It serves to fill in the fixed field of "scan rsp" during initialization, e.g. mac address. Mac needs to be used when building database, and iOS system cannot directly obtain it from the AdvA field of adv packet data. Therefore, it should be marked in the content of scan response.

publish_when_powerup():

Boot, send corresponding status after a random interval (publish_powerup_random_ms), notify app or gateway node to get online.

mesh_vd_init():

Common processing part related to mesh of multiple projects. It's invoked in "mesh_init_all()".

mesh_global_var_init():

Initialization function of global structure variable, executed before reading related parameters stored in flash. It serves to set default value of compiling, and if there are related parameters in flash, the values in flash will be used.

model_sig_cfg_s_cps:

I.e. composition data. For related definitions, please refer to the structure definition of model_sig_cfg_s_cps and spec.

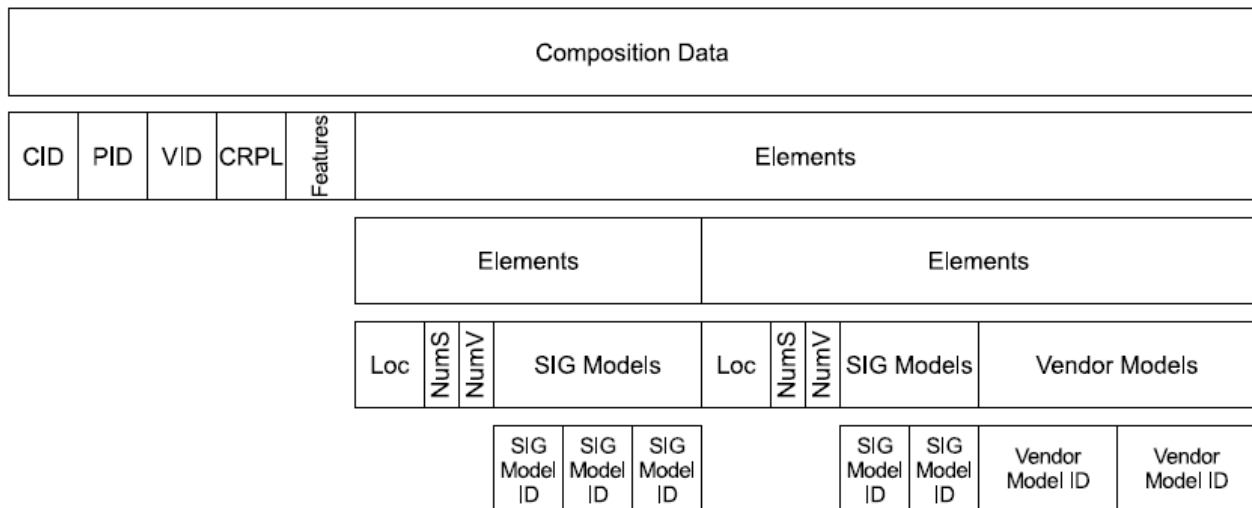


Figure 8.1: Composition data

set_unprov_beacon_para():

- p_uuid: pointer to set uuid. The length is 16 bytes.
- p_info: pointer to set oob_info. The length is 2 bytes.
- p_hash: pointer to set hash value of URI. The length is 4 bytes.

- uri_para: pointer of uri connection. The maximum length is 40 bytes.
- uri_len: Length of actually used data in uri part. The maximum length does not exceed 40.

The parameters above serve to configure beacon packet of unprovisioned node.

set_provision_adv_data():

- p_uuid: pointer to set uuid. The length is 16 bytes.
- oob_info: pointer to set oob_info. The length is 2 bytes.

The parameters above serve to configure parameters of adv packet part when provision is not finished.

set_proxy_adv_data():

- p_hash: pointer to set hash. The length is 8 bytes.
- p_random: pointer to set random. The length is 8 bytes.
- node_identity: It indicates adv packet type.

If "node_identity" is 0, it indicates adv type is "advertising with Network ID", in this case "p_hash" and "p_random" won't take effect.

If "node_identity" is 1, it indicates adv type is "advertising with Node Identity", in this case "p_hash" and "p_random" will take effect.

The parameters above serve to configure adv packet to send proxy connection after provision is finished.

uart_drv_init()/usb_bulk_drv_init():

Serial port and USB initialization, select serial port or USB via the macro "HCI_ACCESS". Use "bhc_register_hci_handler" to register callback function. User can invoke "my_fifo_push_hci_tx_fifo" to push data to be reported into "hci_tx_fifo".

set_material_tx_cmd():

Set transmission parameter:

- 1) op: vendor op code, input 1 byte, no need to input vendor id, it will be fulfilled automatically.
- 2) rsp_max: only effect to status replied command, serves to detect whether receives enough status. When destination address is unicast, the value is 0 or 1(0 and 1 are the same, the detection is done when receiving the status), when destination address is group, the value is the node number of the group.
- 3) retry_cnt: only effect to status replied command, when the command is sent for a while, and no specified rsp_max status is received, then trigger retry flow, the retry time is determined by retry_cnt.
- 4) uuid: when the sending destination address is virtual address, need to input uuid, otherwise it is 0.
- 5) nk_array_idx: mesh supports multiple netkey, this is to set the array index no. in netkey array, note, it is not the provision global netkey index.
- 6) ak_array_idx: mesh supports multiple appkey, this is to set the array index no. in appkey array, note, it is not the provision global appkey index.

- 7) pub_md: when execute publish status, it need to be set as pointer point to model, because when sending message, mesh_model_pub_par_t parameter in pub_md-> pub_par need to be used, e.g., ttl, network transmit, network count, appkey index, but not use default values. Set pub_md to 0 when not execute publish status.

mesh_tx_cmd2normal_primary():

Node actively send command API, see set_material_tx_cmd() for parameter detail.

SendOpParaDebug_vendor():

In WIN32 mode, analysis of gateway, app, par_tmp[2:3], see ini format analysis.

is_need_response_to_self():

Set if need reply status when receive command sending by the function itself and need to answer with status.

mesh_rc_data_layer_access_cb():

When node receives a command sent to itself (condition: model supports, destination address matches) will call this function.

- 1) Vendor Op code range in VD_OP_RESERVE_FOR_TELINK_START and VD_OP_RESERVE_FOR_TELINK_END is opcode reserved for Telink, not open to users.
- 2) mesh_need_random_delay : when node receives group address as destination address, need to add a Random delay to avoid multiple nodes respond at the same time when response.
- 3) p_res->cb: this is the corresponding callback function for each op code, mesh_cmd_sig_func[]->cb and mesh_cmd_vd_func[]->cb.

mesh_rsp_handle_cb():

Reports status message gateway received to firmware, via USB or UART.

hci_send_data_user():

Buffer data of hci tx fifo, the first 2 byte is len, the third is data type to tell data type. Here is HCI_RSP_USER, other please refer to hci_type_t.

mesh_tx_reliable_stop_report():

Callback function when gateway send reliable command, and the stop condition is fulfilled.

app_hci_cmd_from_usb():

In blt_sdk_main_loop() function, callback app_hci_cmd_from_usb() to handle commands sent by firmware, analyze and execute the commands via app_hci_cmd_from_usb_handle().

app_hci_cmd_from_usb_handle ():

The corresponding data is in ini format, check ini chapter for detail.

8.7 cmd_interface.h file introduction

access_cmd_get_level():

This function serves to obtain level value of element by setting "opcode" as "G_LEVEL_GET" and then assembling "mesh_tx_cmd()".

access_cmd_set_level():

This function serves to set level value of element by assembling "mesh_tx_cmd()". When ack is 1, set opcode as "G_LEVEL_GET". When ack is 0, set opcode as "G_LEVEL_SET_NOACK". The SDK will manage and implement tid parameters used in this command, so these parameters are negligible for upper development.

access_set_lum():

This function serves to set level value by inputting "lum" (range is 0-100) and assembling "access_cmd_set_level ()".

access_cmd_onoff():

This function serves to set on/off value of element by assembling "mesh_tx_cmd()". When ack is 1, set opcode as "G_ON/OFF_GET". When ack is 0, set opcode as "G_ON/OFF_SET_NOACK". The SDK will manage and implement tid parameters used in this command, so these parameters are negligible for upper development.

8.8 vendor_model.c file introduction

This file mainly introduces transmission of opcode corresponding to vendor model, as well as corresponding callback function to be executed after this opcode is received.

Note: non-provisioner nodes use only 1 vendor id, and will show this id in composition data, Vendor model has 64 op code in total. Please be noted, this is not 64 for a product, it's 64 for all products. So please use it wisely, use as many sub-commands as possible.

Telink also uses some vendor op code for self-define features, so currently 0xC0—0xDF is reserved for Telink, and 0xE0—0xFF is for other users.

Register of vendor opcode

See Chapter 3.2.

mesh_search_model_id_by_op_vendor():

This function serves to search for related resources in the array "mesh_cmd_vd_func[]" via opcode. User does not need to modify this function.

vd_cmd_key_report():

This function serves to report key press event and it's used in the project "8258_mesh_switch".

It can be considered that "int SendOpParaDebug(u16 adr_dst, u8 rsp_max, u16 op, u8 *par, int len);" is equivalent to "mesh_tx_cmd_primary()".

is_cmd_with_tid_vendor():

This function serves to check whether this opcode needs to carry tid and return value accordingly. If this opcode needs to carry tid, it will return 1, and return the location of tid in parameter area via "tid_pos_out"; otherwise, it will return 0.

8.9 mesh_test_cmd.c file introduction

This file serves to save implementation of test commands.

Telink Semiconductor

9 8258 MESH Project Introduction

9.1 app_config_8258.h

PCBA_8258_SEL:

Select PCBA, default is 48pin dongle board, the other 2 are reference board.

FLASH_1M_ENABLE:

Enable this macro when internal flash is 1M because the flash map is different, and for the initial configuration, MAC is 0Xff000 while 512K MAC is 0x76000.

HCI_ACCESS:

Set HCI interface.

- HCI_USE_USB: use USB
- HCI_USE_UART: use UART

HCI is not needed for data transmission by default.

UART_GPIO_SEL:

Set UART IO.

HCI_LOG_FW_EN:

Firmware disables this function by default, user can enable this if needed, refer to log output chapter for detail.

ADC_ENABLE:

Set whether to enable ADC or not.

ONLINE_STATUS_EN:

Private mesh SDK online status function. Send real time status data, optimize real-time function of publish.

DUAL_MODE_ADAPT_EN:

SIG mesh + ZigBee dual modes.

DUAL_MODE_WITH_TLK_MESH_EN:

Enable SIG mesh + private mesh SDK dual modes.

TRANSITION_TIME_DEFAULT_VAL:

Default transition time is used when power on, and receiving a command supporting transition variable, e.g., generic on/off, but there is no transition variable in this command, the light will act according to TRANSITION_TIME_DEFAULT_VAL. The default transition time is 1s, to disable transition, set TRANSITION_TIME_DEFAULT_VAL to 0. Refer to trans_time_t for detail.

SW1_GPIO/SW2_GPIO:

Two buttons on dongle board, used for debugging, disabled by default.

To enable, first verify IO, then modify corresponding PULL_WAKEUP_SRC_XXX and XXX_INPUT_ENABLE.

PWM_R/ PWM_G/ PWM_B/ PWM_W:

Set IO corresponding to PWM.

GPIO_LED:

Set led light IO, e.g., when the provision is completed, reset to factory configuration, the definition of light flashing.

9.2 app.c file introduction

9.2.1 Customization of Adv packet and Adv response packet

Advertising packet

Connectable adv packet: Currently SIG MESH spec has already defined all fields for connectable adv packet format. Please refer to the structure "PB_GATT_ADV_DAT" or spec for details.

Advertising response packet

User can customize adv response packet by modifying the array "u8 tbl_scanRsp [] = {}" via "mesh_scan_rsp_init()". The maximum length of adv response packet can reach 31 bytes, only a part of which is used currently. User can configure the "rsv" field as needed in "mesh_scan_rsp_init()".

```
typedef struct{
    u8 len;
    u8 type;
    u8 mac_adr[6];
    u16 adr_primary;
    u8 rsv_telink [10]; // not for user
    u8 rsv_user[11];
}mesh_scan_rsp_t;
```

9.2.2 Configuration of fifo part

```
MYFIFO_INIT(blt_rxfifo, 64, 16);
MYFIFO_INIT(blt_txfifo, 40, 32);
```

The two functions serve to configure packet Rx buffer and Tx buffer in BLE stack bottom layer. Generally it's not recommended to modify them unless RAM size is not large enough.

9.2.3 app_event_handler ()

Callback processing function: When BLE stack receives adv (include connectable adv packet, beacon packet, etc), connect request packet, BLE connection parameter update packet, BLE connection termination, and etc, this callback function will be invoked after the event to process correspondingly.

Currently all beacons used for SIG MESH communication are processed in the branch "(subcode == HCI_SUB_EVT_LE_ADVERTISING_REPORT)".

For processing of other events, please refer to corresponding code. Currently only simple LED indicating light processing is contained, and related functions can be added if needed.

HCI_SUB_EVT_LE_ADVERTISING_REPORT:

Processing branch after receiving adv packet. User can add corresponding event and processing function under this branch.

HCI_SUB_EVT_LE_CONNECTION_COMPLETE:

Event callback generated after BT connection is established. BLE stack bottom layer will callback to this branch after BT connection is established.

HCI_CMD_DISCONNECTION_COMPLETE:

After BT connection is terminated, BLE stack bottom layer will callback to this branch.

9.24 main_loop ()

- mesh_loop_proc_prior(): function with high priority for real time features
- blt_SDK_main_loop (): main_loop function of BLE stack.
- proc_led(): LED indicating light event processing function.
- factory_reset_cnt_check(): factory reset processing function. Support reset method of five power on operations. Please refer to factory reset section.
- mesh_loop_process(): SIG mesh related loop function, including retry mechanism of reliable command, segment ack timeout response, TID timeout detect mechanism, and etc.
- sim_tx_cmd_node2node(): Demo demonstration interface of ON/OFF command timed transmission.

9.2.5 user_init()

- proc_telink_mesh_to_sig_mesh(): To implement OTA between sig mesh and telink mesh with incompatible parameter format, it's needed to initialize parameters. In current SDK, when mesh type change is detected, parameter area to be used by new mesh will be cleared.
- bls_ll_setAdvParam(): Define parameters including adv packet interval, currently not recommended to modify.
- blc_ll_setAdvCustomedChannel(): Customize adv channel. Sig mesh requires to use standard channel 37/38/39. However, during test process, for the convenience of debugging, the channel can be changed.
- bls_ll_setAdvEnable(1): Enable transmission of adv packet.
- rf_set_power_level_index (MY_RF_POWER_INDEX): Set Tx power as 8dbm.
- mesh_init_all(): sig mesh related initialization.

9.2.6 void proc_ui()

This function mainly implements UI related processing, e.g. button detect function, as well as corresponding test code.

9.3 app_att.c file introduction

pb_gatt_provision_out_ccc_cb():

Enable transmission of "provision out" part. Only when "provision_Out_ccc" is set as "01 00", can mesh node normally return command.

pb_gatt_Write ():

Callback function corresponding to uuid of "my_pb_gatt_in_UUID" and used to process provision command.

proxy_gatt_Write():

Callback function to process proxy command. The command head of proxy command include three types: MSG_PROXY_CONFIG, MSG_MESH_BEACON, MSG_NETWORK_PDU.

- MSG_PROXY_CONFIG: It's used to configure white list and black list for proxy communication.
- MSG_MESH_BEACON: It's used to control reception of beacon command (notify).
- MSG_NETWORK_PDU: It's used to control ON/OFF command.

attribute_t my_Attributes[]:

Service list in SIG_mesh containing basic att, as well as att contents related to SIG_mesh part.

9.4 light.c file introduction

Modify IO pins

It's only needed to modify IO pins corresponding to PWM_R / PWM_G / PWM_B / PWM_W.

```
#define PWM_R    GPIO_PC2        //red
#define PWM_G    GPIO_PC3        //green
#define PWM_B    GPIO_PB6        //blue
#define PWM_W    GPIO_PB4        //white
typedef struct{
    u32 gpio;
    u8 id;        // pwm id
    u8 invert;    // pwm invert feature
    u8 func;      // PWM first function or second function
    u8 rsv[1];
}light_res_hw_t;
light_res_hw_t light_res_hw[LIGHT_CNT][4];
```

light_res_hw defines PWM IO features.

func: GPIO_PC1, GPIO_PC4, GPIO_PD5 have 2 PWM output functions, here defines whether to use function 1 or function 2.

In PWM macro definition, all 4 leds, i.e., RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B/ RES_HW_PWM_W, on dongle board/reference board are listed, but for a specific type of light, not all of them are needed, light_res_hw is to define this, e.g.:

LIGHT_TYPE_CT: select RES_HW_PWM_R and RES_HW_PWM_G, red is for warm light bead, green is for cold light bead.

LIGHT_TYPE_HSL: RES_HW_PWM_R, RES_HW_PWM_G, RES_HW_PWM_B are corresponding to RGB beads respectively, when modify light, change HSL to RGB in dim_refresh(), to drive LED.

LIGHT_TYPE_LPN_ONOFF_LEVEL: select RES_HW_PWM_R, currently can only control onoff because the low power consumption of retention.

LIGHT_TYPE_PANEL: default value is 3, occupying 3 element addresses, with 3 onoff server models, and the corresponding beads of these 3 onoff models are RES_HW_PWM_R/ RES_HW_PWM_G/ RES_HW_PWM_B.

Set PWM Frequency

Just modify PWM_FREQ.

Note: PWM tick overflow will cause STATIC_ASSERT(PWM_MAX_TICK < 0x10000) error when compiling. PWM tick is 16 bits, and the default PWM clock is PLL clock, when PWM_FREQ is too small, PWM tick will overflow, in this case, user can set PWM frequency division, i.e., PWM_CLK_DIV_LIGHT. Generally it is not needed.

ct_flag:

Used only in LIGHT_TYPE_CT_HSL mode. There are CT bead and HSL bead in this mode, but only 1 will light up at the same time. ct_flag is 1, indicates this is CT bead and 0 indicates HSL bead.

light_res_sw_save:

This variable includes all light status related parameters need to be saved, e.g., lightness, CT and etc. Note, all data are transfer to generic level format (range from -32768 ~ 32767) before saved.

The reason why save data in level format: (1) all status value can transfer to level, (2) level is the most accurate (3) save only 1 parameter for the same status, e.g., for CT value, you can't save both CT value and the transferred level value, because these 2 values may not synchronize.

In general, all status are saved in level format, otherwise may lose accuracy.

Nonlinear correspondence of luminance and PWM value

Developer can modify the array "rgb_lumen_map[]" according to actual light characteristic.

// 0-100% (pwm's value index: this is pwm compare value, and the pwm cycle is 255*256)

```
const u16 rgb_lumen_map[101] = {}
```

mesh_global_var_init_light_sw():

The initial value when first booting is the default compiling value, when corresponding parameter is saved to flash, then use the saved value.

light_res_sw_load():

Status parameter for load light from flash.

light_pwm_init():

Call this function after read light status. Set to OFF after initializing PWM register, then check if need to enable and if it need transition parameter.

light_par_save_proc():

The light status will save to flash 3s after it changed to avoid writing flash too often.

light_dim_set_hw():

Set PWM output. It will be executed after node receives "G_LEVEL_SET"/"G_LEVEL_SET_NOACK" command.

Idx and idx2: light_res_hw[idx][idx2].

idx: light count index, e.g., when a BLE module has 2 CT lights

Idx2: light bead index.

light_dim_refresh():

When light status changes, call this function to refresh PWM output value. In this function, users can get lightness, CT value and etc., users can calculate PWM value based on this value according to their own dimming algorithm.

The default algorithm is, change the CT value of standard lightness to 0—100 scale, then check rgb_lumen_map[101] to find the corresponding PWM output value.

get_light_pub_list():

Check all status need to be published when light status changes.

temp_to_temp100():

Change 800—20000 CT value to 0—100 scale.

temp100_to_temp():

Change 0—100 scale to 800—20000 CT value.

light_g_level_set_idx_with_trans():

```
typedef struct{
    s32 step_1p32768;    // (1 / 32768 level unit)
    u32 remain_t_ms;     // unit ms: max 26bit: 38400*1000ms
    u16 delay_ms;        // unit ms
    s16 present;         // all value transfer into level, include CT.
    s16 present_1p32768; // (1 / 32768 level unit)
    s16 target;
}st_transition_t;
```

Set transition parameter when receive command like level set/lightness set and transition is needed. "1p32768" in st_transition_t means dividing one level scale in 32768 units, i.e., the unit of this value is 1/32768, thus can avoid floating calculation and enhance calculation efficiency.

step_1p32768: the changing value for each LIGHT_ADJUST_INTERVAL during transition.

remain_t_ms: remain value in the command changes to ms.

delay_ms: delay_ms value in the command changes to ms.

present: real time value of level during transition.

present_1p32768: remaining part of present, unit is 1/32768 level scale.

target: target level.

light_transition_proc():

Transition polling processing function. When the transition finishes, i.e., level reach target level, if this transition is triggered by scene load, call scene_target_complete_check(i); to label that the scene is valid.

When the transition finishes, check and transmit publish status.

led_onoff_gpio():

In deep retention sleep or deep sleep mode, the output during sleep is done by setting pull-up/pull-down. In this case, PWM stops working, as well as gpio function, gpio output enable, gpio output registers, only analog registers setting pull-up/pull-down works.

proc_led():

LED indicating light polling processing function.

rf_link_light_event_callback ():

It's LED indicating light register event. By using this method, light blinking is executed in main_loop and it won't influence processing of other events.

In LPN mode, proc_led() can not be polled all the time because the SDK enters deep mode, thus the light flashes slowly, not as we expected. Current solution is to set faster flashing parameter when led indicating light is needed, then keep polling proc_led(), process other function after the flash ends. Normal LPN products need no LED, only for development.

Flashing scene introduction:

LGT_CMD_SET_MESH_INFO(LGT_CMD_PROV_SUC_EVE): light flashing when provision succeeds.

LGT_CMD_FRIEND_SHIP_OK: scene generated by LPN when LPN builds friendship with friend successfully.

LGT_CMD_SET_SUBSCRIPTION: receiving message to modify subscription address

LGT_CMD_BLE_ADV: BLE disconnecting scene, disable by default, only for debug mode.

LGT_CMD_BLE_CONN: BLE connecting scene, disable by default, only for debug mode.

LGT_CMD_SWITCH_POWERON: flash once when power switches to on.

LGT_CMD_SWITCH_PROVISION: switches to PROVISION mode.

LGT_CMD_SWITCH_CMD: switch sends press button command.

PROV_START_LED_CMD: gateway starts provision flow to a node.

PROV_END_LED_CMD: provision flow finishes.

LGT_CMD_DUAL_MODE_MESH: switch modes in dual mode status.

show_ota_result():

It's processing function of light blinking indication after OTA is finished.

show_factory_reset():

It's processing function of light blinking indication after implementing factory reset operation.

How to Introduce Customized Dimming Algorithm:

Default dimming algorithm: refer to light_dim_refresh().

Users can modify dimming algorithm by changing light_dim_refresh(), i.e, get standard value with this function, then call their own dimming algorithm:

Lightness:

```
st_transition_t *p_trans = P_ST_TRANS(idx, ST_TRANS_LIGHTNESS);  
u16 lightness = get_lightness_from_level(p_trans->present);
```

CT value:

```
u16 temp = light_ctl_temp_presents_get(idx);  
HSL(CT) value:
```

Refer to light_dim_refresh(), get HSL.h/HSL.s/HSL.l or RGB.r/ RGB.g/ RGB.b based on dimming algorithm needs.

On/off: defined by lightness.

10 Provisioner (Gateway) Project Introduction

10.1 Provisioner Function Introduction

10.1.1 adv-bearer and gatt-bearer

Provisioning is to add an unallocated device into mesh network via Provisioner, so that the device can become a node in the mesh network.

The provision process mainly allocates network key and IV index (key parameters to determine whether it's the same network), as well as unicast adr (address allocated in the network).

The Provisioner can use network parameters and unicast adr to access and control corresponding node, e.g. turn on/off light, adjust luminance.

Provision supports two types of link channels:

- Implement communication in adv-bearer channel via adv packet. This section mainly introduces the adv-bearer part.
- Implement communication via BLE connection and gatt-bearer. For the implementation of gatt-bearer, please refer to section 7.2, and Android/iOS APP corresponding to SIG_mesh can implement corresponding functions.

10.2 Provisioner Principle

10.2.1 Command Interaction of Provisioner

The provisioner uses "adv-bearer" to add unprovisioned device (unpaired node) into network, and adopts BT channel 37, 38 and 39 to communicate with unprovisioned device. By transferring parameters (e.g. random, key) between the provisioner and unprovisioned device, network parameters and address allocation are exchanged to finally add the unprovisioned device into the network. Please refer to section 5.3 in sig_mesh document "Mesh_v1.0" for details.

10.2.2 Timing Sequence Chart of adv Provisioner

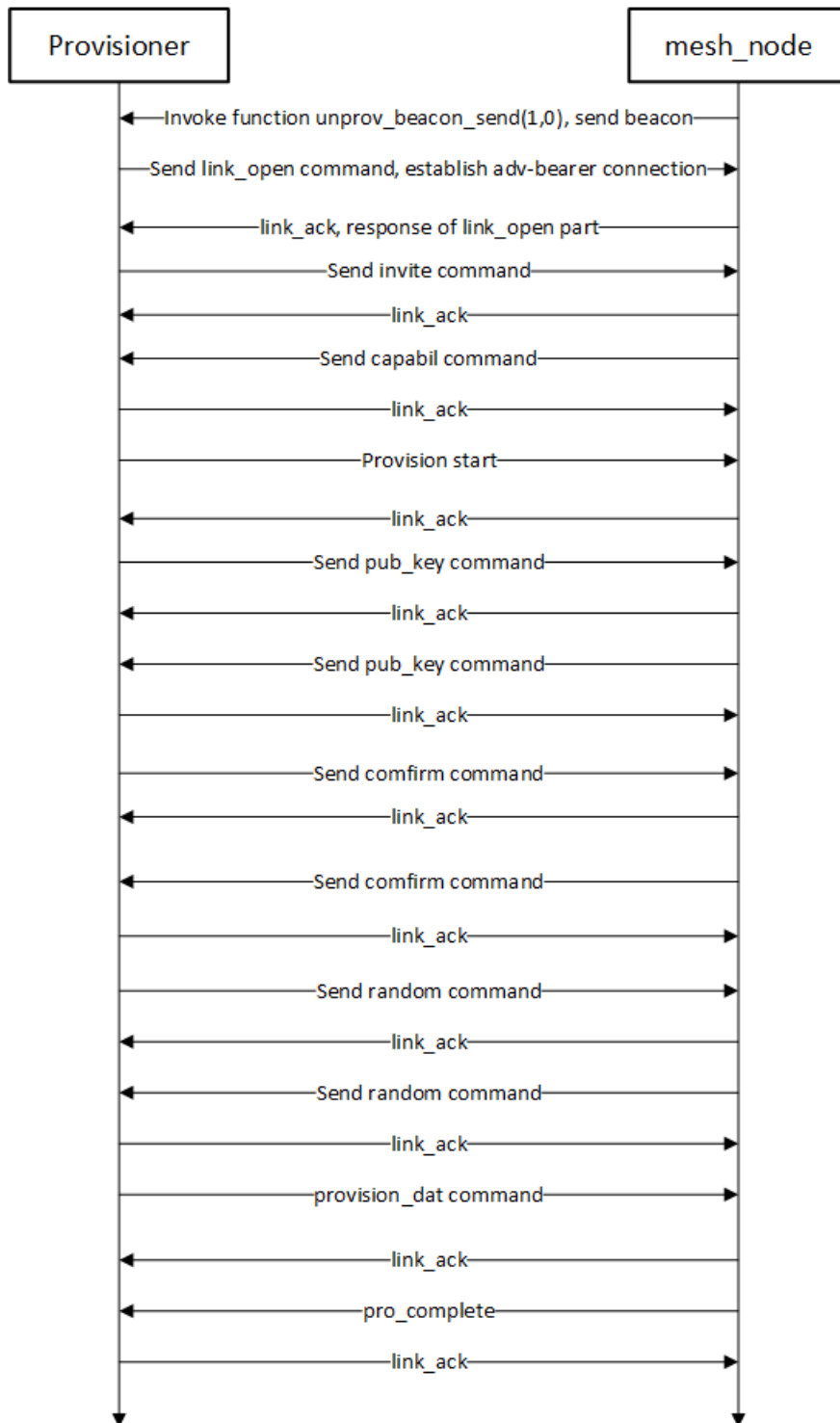


Figure 10.1: adv Provisioner Timing Sequence Chart

The “provision_dat” command contains three network parameters including network key, IV index and unicast adr, and implements the function of network formation.

Function invoking relationship chart for the packet Tx part of adv-provision:

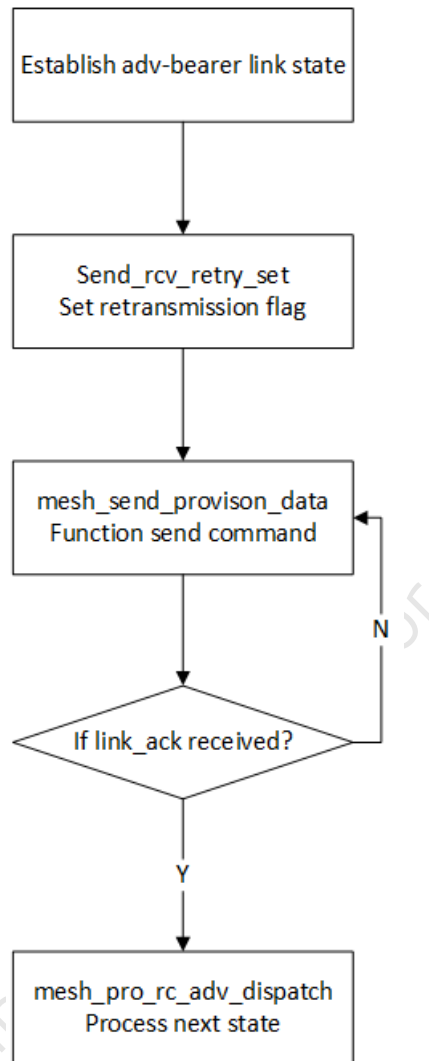


Figure 10.2: Function Invoking Relationship Chart for the packet Tx Part of Adv-provision

Function invoking relationship chart for the packet Rx part of adv-provision:

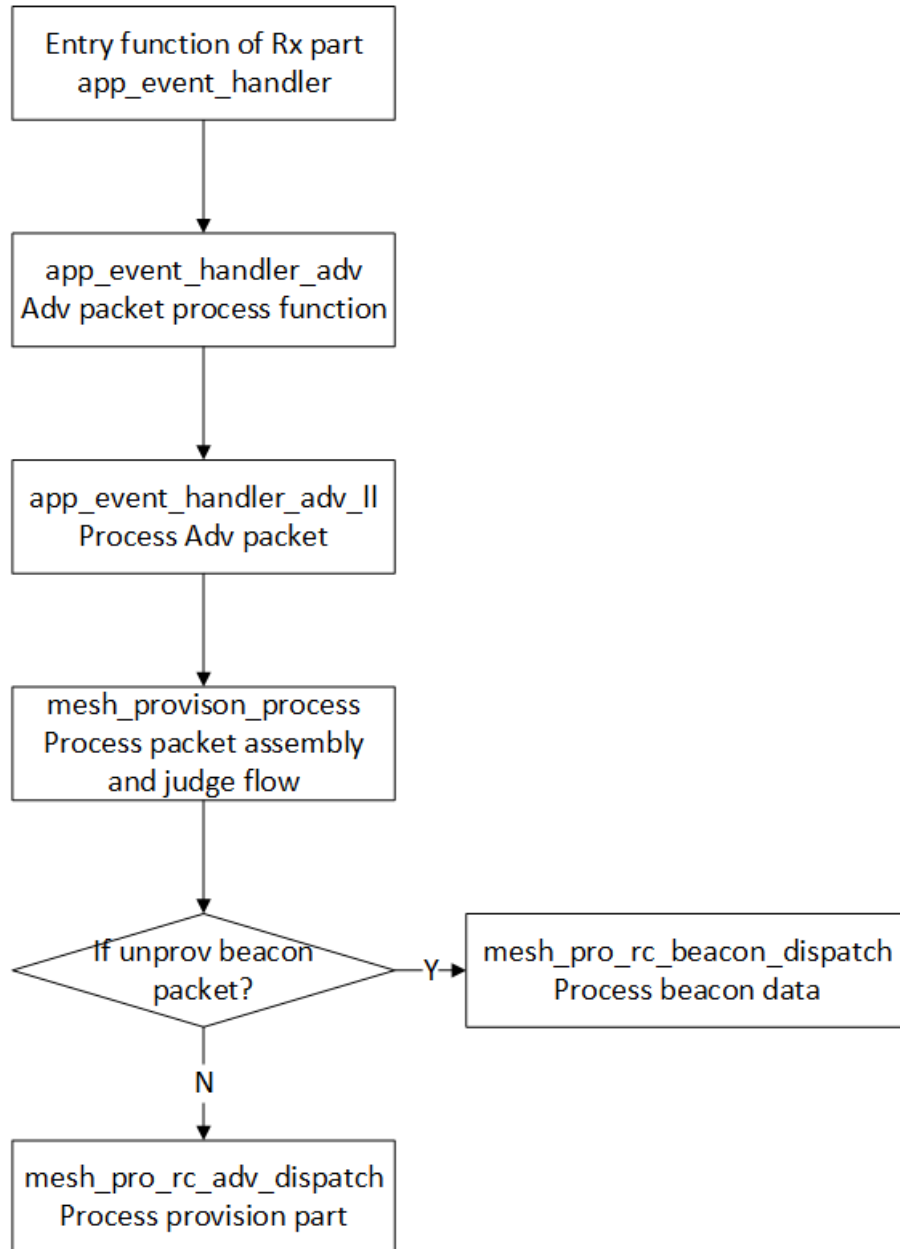


Figure 10.3: Function Invoking Relationship Chart for the Packet Rx Part of Adv-provision

10.2.3 Timing Sequence Chart of gatt Provisioner

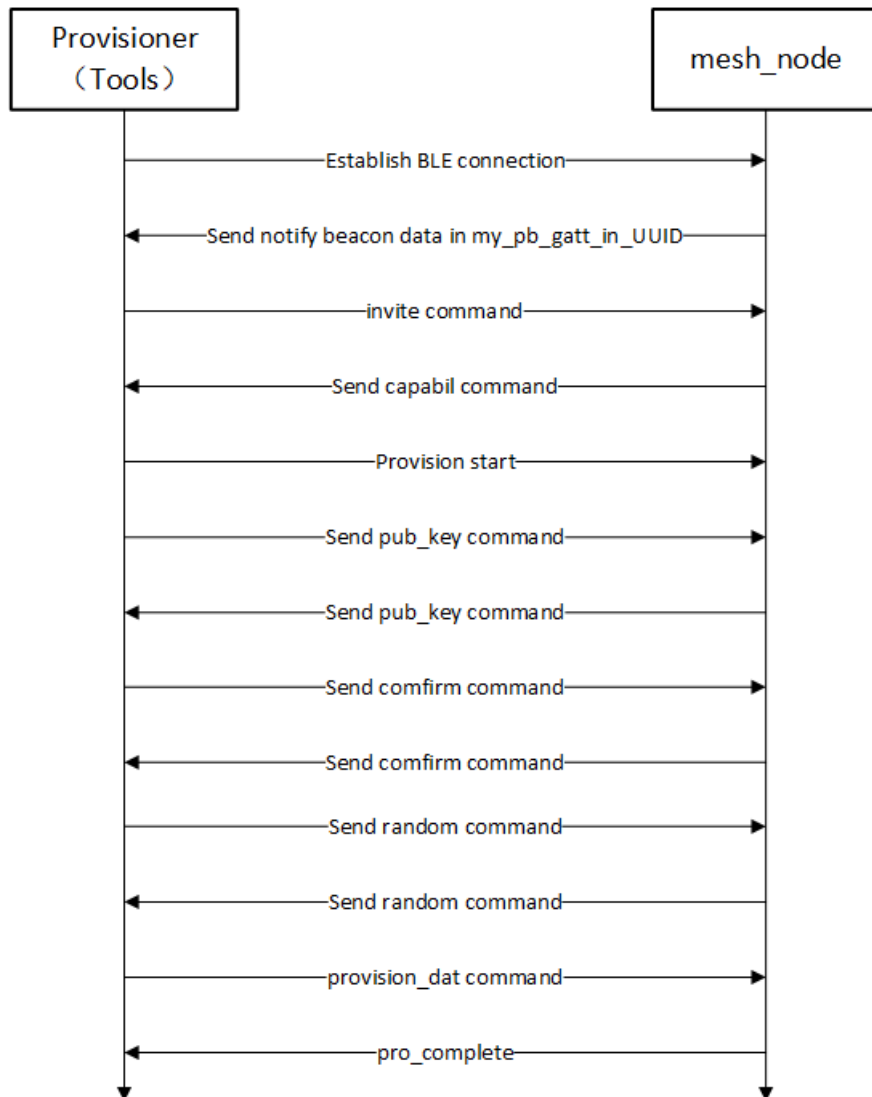


Figure 10.4: gatt provisioner Timing Sequence

By using the method of gatt-provision, the function of provision can be implemented more quickly. The "int pb_gatt_Write (void *p)" is the entry function of "gatt_provision" part.

Packet Tx function entry of gatt_provision:

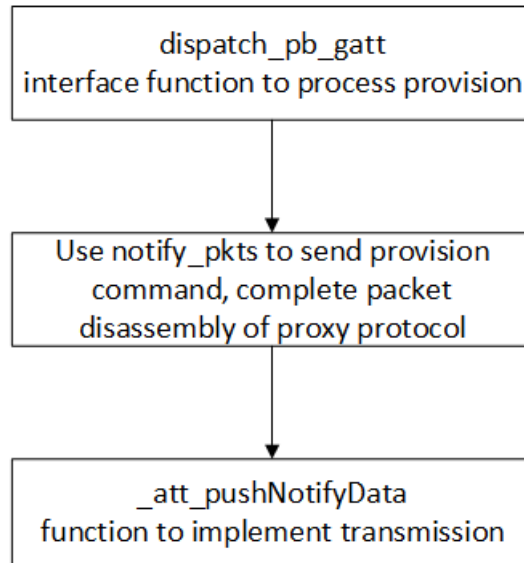


Figure 10.5: Packet Tx Function Entry of gatt_provision

Packet Rx function entry of gatt_provision:

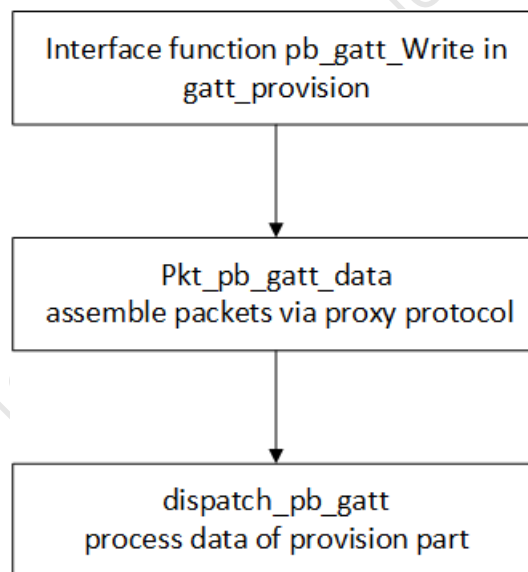


Figure 10.6: Packet Rx Function Entry of gatt_provision

10.3 app.c file introduction

In the provisioner project of current SDK, only the "app.c" file needs customized modifications.

Customization of Adv packet and Adv response packet

Please refer to Section 9.2.1.

Configuration of fifo part

Please refer to Section 9.2.2.

HCI (USB/UART) Report Data

my_fifo_push_hci_tx_fifo (u8 p, u16 n, u8 head, u8 head_len)

p: point to address of the data to be sent

n: data length

head: head of the data

head_len: length of the head (0 if not specified)

Call my_fifo_push_hci_tx_fifo (u8 p, u16 n, u8 head, u8 head_len) to send data to hci_tx_fifo, the hci_tx_fifo data will be sent in the callback function. Call back function is registered in user_init.

- UART: blc_register_hci_handler (blc_rx_from_uart, blc_hci_tx_to_uart);
- USB: blc_register_hci_handler (app_hci_cmd_from_usb, blc_hci_tx_to_usb);

app_event_handler ():

Refer to section 9.2.3.

main_loop ():

Refer to section 9.2.4.

user_init():

Refer to section 9.2.5.

proc_ui():

The "proc_ui" function configures IO pin scanning with the interval of 40ms to detect IO change. The interface function "access_cmd_onoff" is finally used to send ON/OFF command. By pressing SW1/SW0, an ON/OFF command will be sent with the interval of 100ms.

104 Provisioner operation and ports

Protocol stack runs in provisioner port. Node information are saved in provisioner flash with the address of FLASH_ADR_VC_NODE_INFO(0x3f000), 1 sect or is 4K, so the maximum saved node number is 200. If the node number is bigger than 200, call HCI_GATEWAY_CMD_CLEAR_NODE_INFO, i.e., "e9 ff 06" to clear node information saved in gateway.

104.1 Format of SIG_MESH_TOOL ini file

Gateway operation will use "SIG_MESH_TOOL". User can click control buttons on the interface, or send out command via the left "command list" window (double click cmd line) or the bottom "edit control" window (compile cmd and press Enter). The provisioner will handle this in the corresponding branch of app_hci_cmd_from_usb_handle after it receives the command.

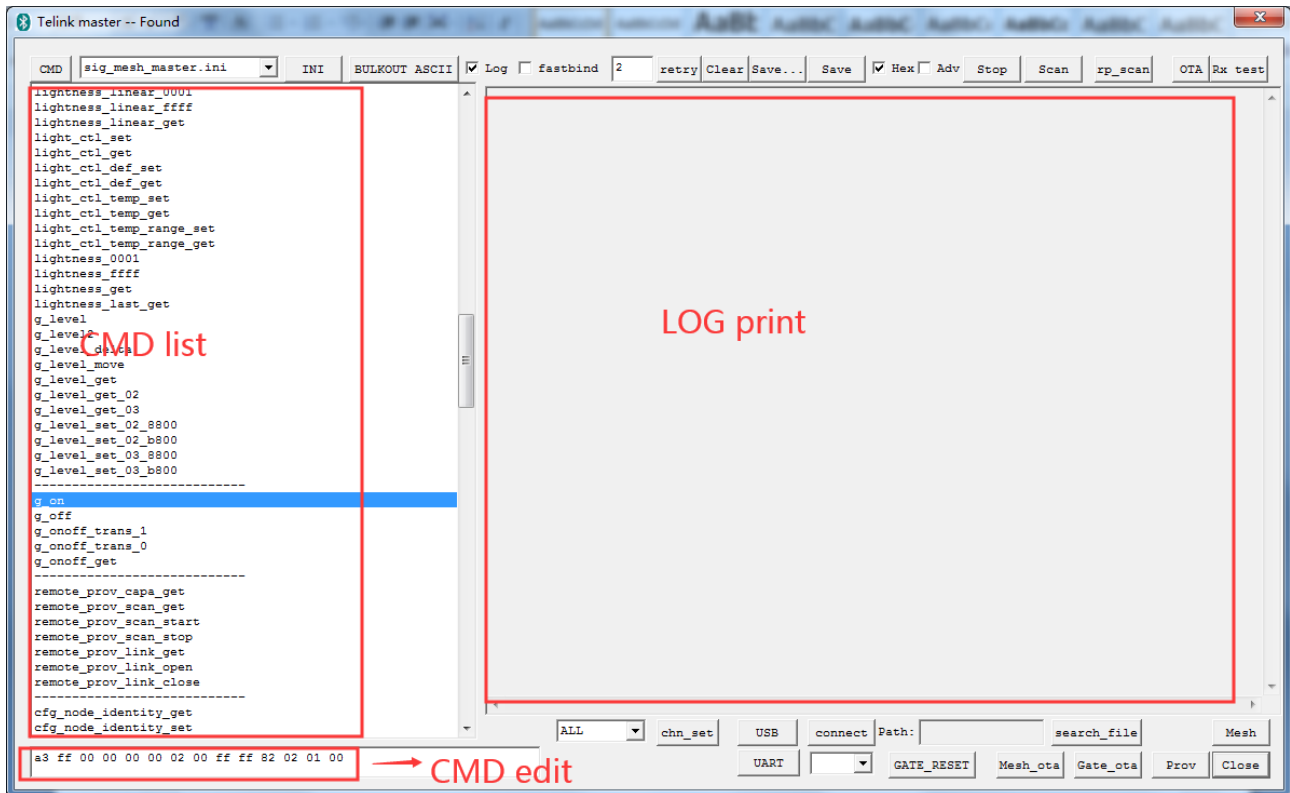


Figure 10.7: SIG_MESH_TOOL

The command has 2 formats in ini file, SIG model and vendor model.

104.2 SIG model format, g_all_on as an example

1		2	3	4	5	6	7	8	9	10	11
Flag		nk_idx	ak_idx	reliable retry cnt	reliable rsp_max	dst	op	Onoff	TID	Transition time	Delay
e8	ff	0000	0000	00	00	ff ff	82 02	1	0	nc	nc

Figure 10.8: g_all_on

The first 2 bytes are identifier, defined by Telink, used to identify communication packet head, for gateway it is 0xE8FF, for app(including mobile app and kma dongle firmware) is 0xA3FF.

Parameter structure as below:

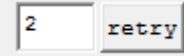
```
typedef struct{
    u16 nk_idx;      //netkey index
    u16 ak_idx;      //app_key index
    u8  retry_cnt;   // time of app layer, when rsp_max is not received, app layer will retry
```



```

u8 rsp_max;      // expected answer number
u16 adr_dst;     // destination address
u8 op;           // first byte of op_code
u8 par[MESH_CMD_ACCESS_LEN_MAX]; //rest byte of op_code and parameter
}mesh_bulk_cmd_par_t;

```

Among them, retry_cnt is in VC tool, if set to 0, it means to use the value of "retry" control, the default is 2, . VC tool will automatically change the value of retry_cnt in the INI data to 2. If it is set to 0xFF, it means that no retry is required, that is, the VC tool will automatically change the value of retry_cnt in the INI data to 0.

Refer to set_material_tx_cmd() for detail of nk_idx, ak_idx, rsp_max.

Note:

- if TID is 0, then it will be maintained and managed by protocol stack, if TID is not 0, then use this as TID, so that users can maintain TID on their own.

104.3 Vendor Model Format, CMD-vendor_on as example

1		2	3	4	5	6	7	8	9	10	11
Flag		nk_idx	ak_idx	reliable retry cnt	eliable rsp_max	dst	op	op_rsp	tid_pos	para[0]	para[1]
e3	ff	0000	0000	02	00	ff ff	C2 1102	c4	2	1	0

Figure 10.9: CMD vender on

Different with SIG model, there are 2 more parameters, op_rsp and tid_pos, these 2 parameters are pseudo parameters, and will not be sent to light node.

op_rsp: set corresponding response opcode(vendor id is not compulsory)

tid_pos: set tid position (0 means no tid bytes, 1 means tid is in para[0], 2 means para[1]...), op_rsp and tid_pos 's configuration should be unified with that of firmware.

The purpose of these 2 parameter, is that provisioner need to support more vendor id's op code, and vendor op may be added at any time, we cannot compile all these information in the program, so we add these information via ini.

104.4 Burn Nodes

Burn 2 8258 dongles: 1 8258 provisioner node(8258_mesh_gw.bin), 1 dongle node(8258_mesh.bin).

Check 4.1 for burning steps.

104.5 Add Light via Provisioner

The gateway cooperates with the lighting process of the SIG_mesh_tool tool and the corresponding command format (hexadecimal representation).

- 1) Plug the provisioner dongle to the USB port. Start the "SIG_MESH_TOOL", and select "tl_node_gateway.ini".

The top side of the tool will show "Found", which indicates the gateway device (provisioner) is found.

The tool will automatically obtain the uuid and mac address of the gateway. The command format is:

HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_GET_UUID_MAC

i.e., e9 ff + 10

The gateway will report uuid and mac after receiving it, the format is:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_UUID+uuid(16 bytes) +mac(6 bytes), i.e., 91+99+uuid(16 bytes)+mac(6 bytes).

- 2) Power on the 8258 dongle, and then click the "Scan" button on the tool to start scanning for devices.

The corresponding command of the scan control is HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_START: e9 ff + 00

The command corresponding to the stop control is HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_STOP: e9 ff + 01

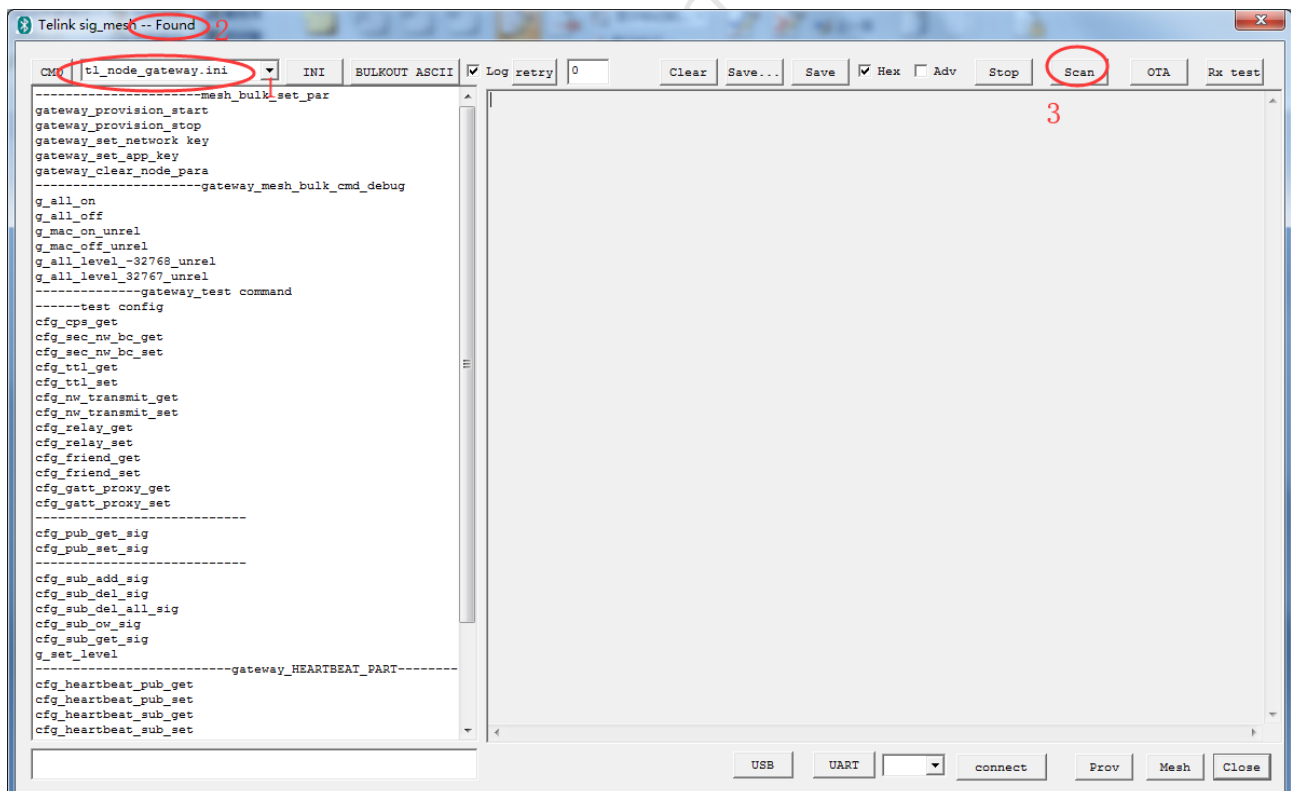


Figure 10.10: Add light via provisioner

- 3) After clicking the start control, the gateway will report the received unprovision beacon in the following format:

TSCRIPT_GATEWAY_DIR_RSP+ HCI_GATEWAY_CMD_UPDATE_MAC+unprovision beacon. i.e.:91+88+mac(6 bytes)+ unprovision beacon.

The scanned devices will be shown in the device list. Double click the target device which needs provision, the corresponding command is:

HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_ADV_FILTER+6 byte mac address

i.e.:e9 ff + 08 + mac(6 bytes).

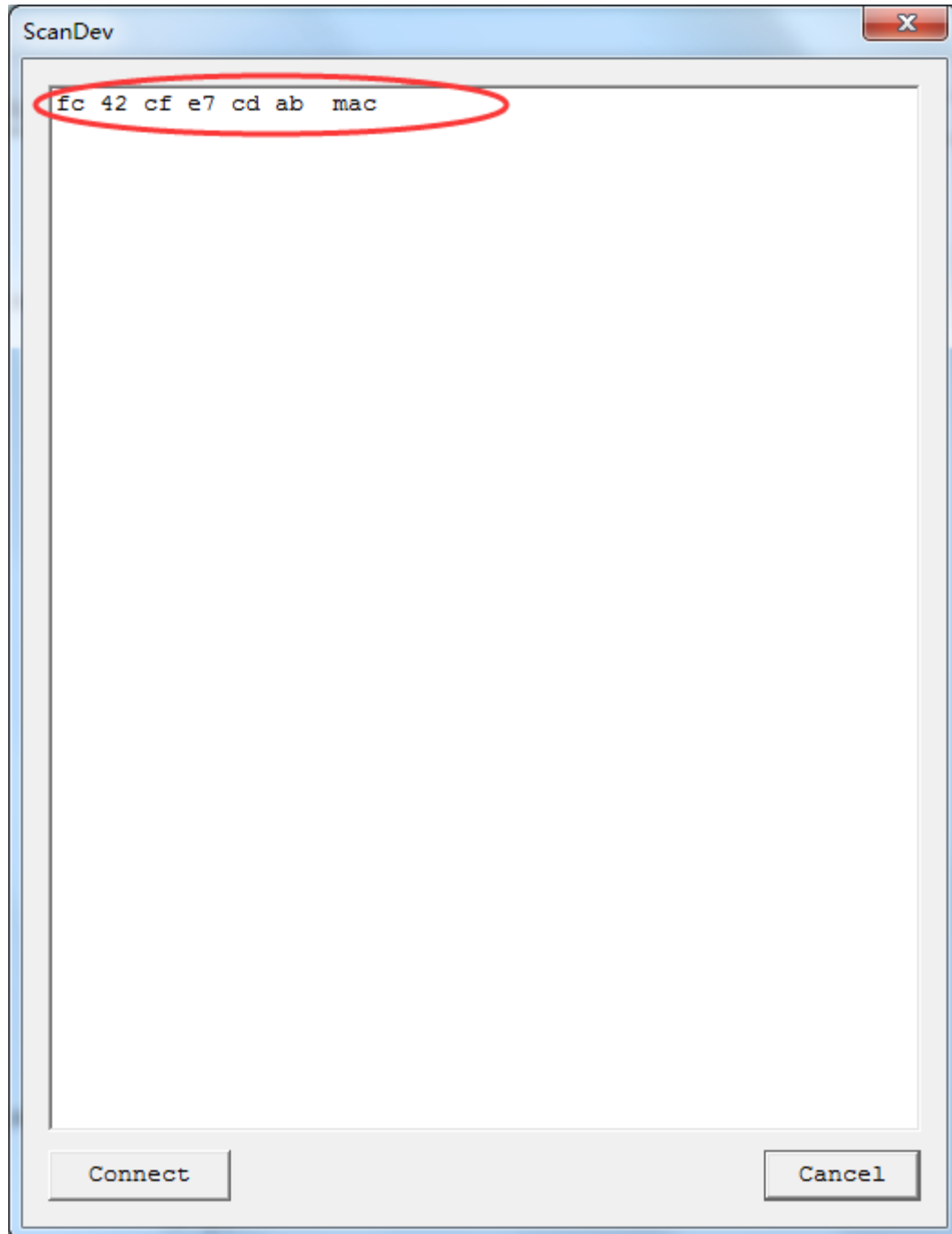


Figure 10.11: unprovision beacon

- 4) Click "Prov" to enter provision interface.

- a. The corresponding command of the Provision control is: HCI_GATEWAY_CMD_GET_PRO_SELF_STS. i.e. e9 ff 0c.
- b. After receiving the command, the gateway will return whether there is configuration information and the number of elements of the gateway. The corresponding command format is: TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_PRO_STS_RSP + provision_flag + pro_net_info.

That is: 91 8b + provision_flag + pro_net_info. pro_net_info is 25 bytes of provision data. The format is as follows:

```
typedef struct{
    u8  net_work_key[16];    //network key
    u16  key_index;          //network key index
    union{
        mesh_ctl_fri_update_flag_t prov_flags;
        u8  flags; // iv update flag
    };
    u8  iv_index[4]; // iv index
    u16  unicast_address;
}provision_net_info_str;
```

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_SEND_ELE_CNT+total element: i.e., 91+8c+ total element.

- c. If the provision flag is 0, it means that the gateway has no configuration information. The SetPro Internal control is enabled. Fill in the provision interface with relevant parameters of pro_net_info and click SetPro_interval to set the gateway configuration information.

The corresponding command is: HCI_CMD_GATEWAY_CTL + HCI_GATEWAY_CMD_SET_PRO_PARA + pro_net_info. i.e.,:e9 ff + 09 + pro_net_info.

HCI_CMD_GATEWAY_CTL+HCI_GATEWAY_CMD_SET_PRO_PARA+unicast address +device key: i.e.,:e9 ff + 0d +gateway address+device key

If the "SetPro Internal" button is disabled, it indicates the gateway has configured network parameters. Just skip parameter setting step.

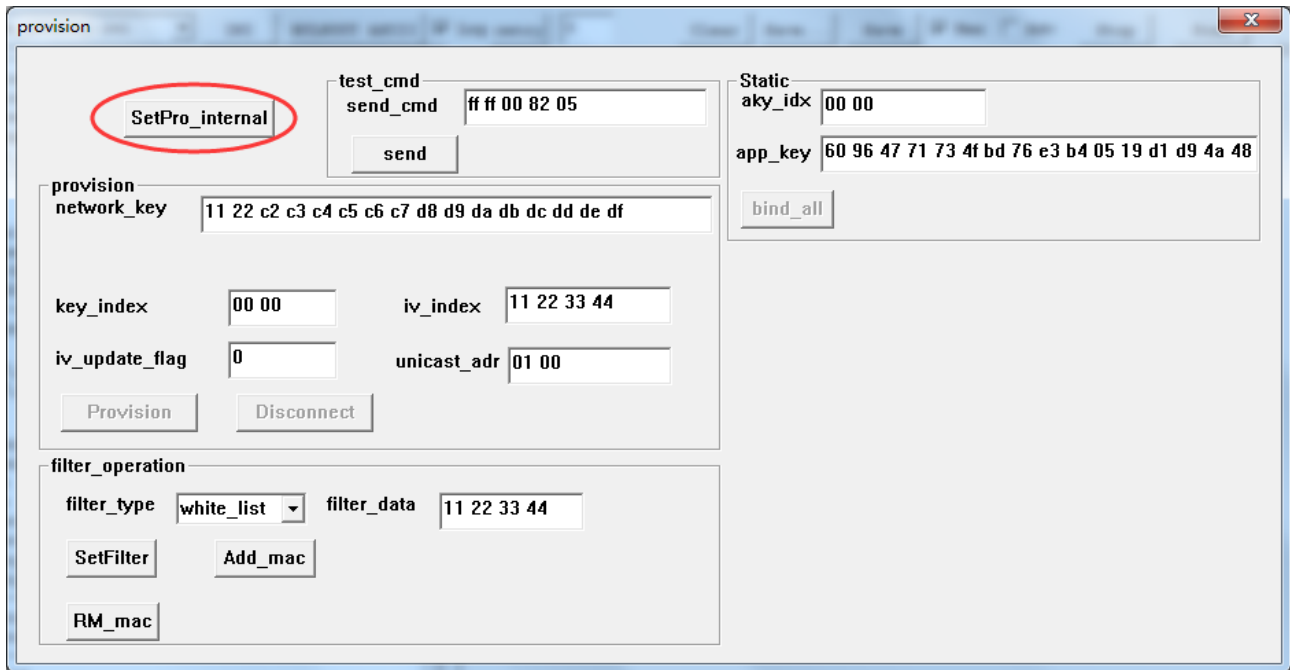


Figure 10.12: SetPro Internal

5) Click the "Provision" button to implement provision. During provision process, related log will be printed out and shown on the main interface. After successful provision, the "bind_all" button becomes enabled.

a. The corresponding command of the Provision control is: HCL_CMD_GATEWAY_CTL+HCL_GATEWAY_CMD_SET_NODE_PARA+ pro_net_info

i.e. e9 ff+0a+ ro_net_info.

b. During the Provision process, the allocated addresses are reported in the following format:

TSCRIPT_GATEWAY_DIR_RSP +HCL_GATEWAY_RSP_UNICAST+unicast addr,

i.e.91+80+unicast address.

c. The node information will be reported after the provision is completed in the following format:

TSCRIPT_GATEWAY_DIR_RSP+ HCL_GATEWAY_CMD_SEND_NODE_INFO+ VC_node_info_t

i.e.91+8d+ VC_node_info_t.

VC_node_info_t is defined as following:

```
typedef struct{
    u16 node_addr;    // primary address
    u8 element_cnt;
    u8 rsv;
    u8 dev_key[16];
}VC_node_info_t;
```

d. The status of the provision will be reported after the provision is completed in the following format:

TSCRIPT_GATEWAY_DIR_RSP+HCI_GATEWAY_CMD_PROVISION_EVT+ gateway_prov_event_t

i.e.:91 + 89 + gateway_prov_event_t.

gateway_prov_event_t is defined as following:

```
typedef struct{
    u8 eve;//1 means success
    u16 adr;
    u8 mac[6];
    u8 uuid[16];
}gateway_prov_event_t;
```

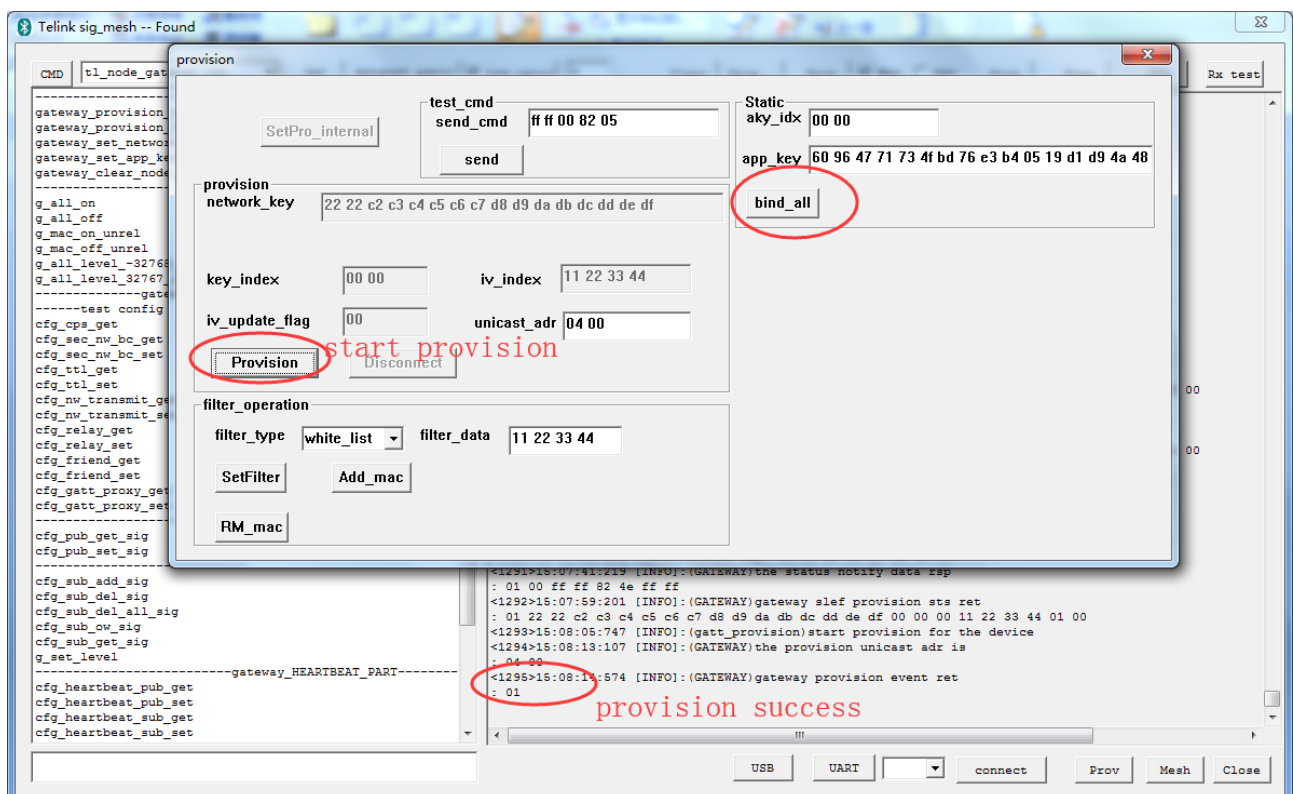


Figure 10.13: Provision

104.6 app_key binding

After provision is finished, it's also needed to bind the app_key for model by clicking the "bind_all" button.

- The command corresponding to bind_all is: HCI_CMD_GATEWAY_CTL+ HCI_GATEWAY_CMD_START_KEYBIND + fast_bind ++app_key index(2 byte)+app_key(16 bytes).

i.e. e9 ff + 0b + fast_bind + app_key index(2 byte)+app_key(16 bytes).

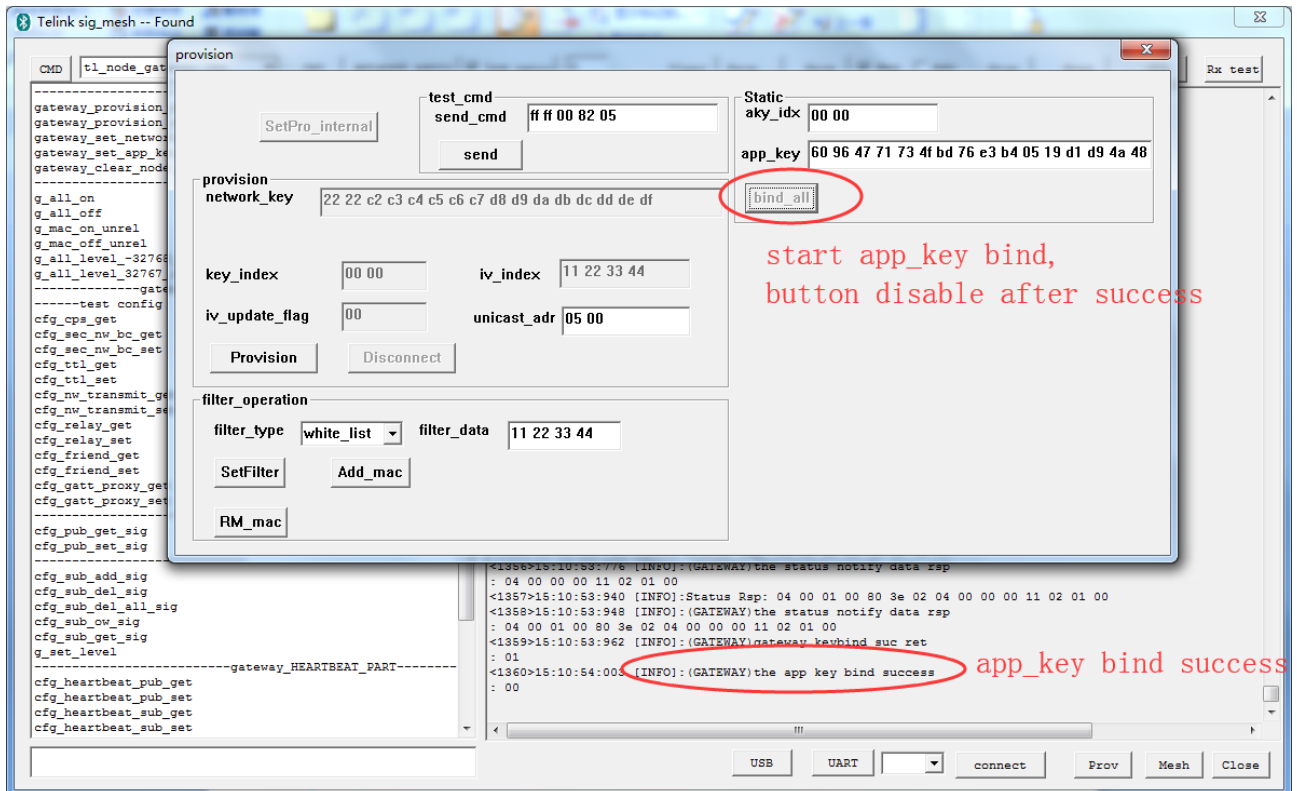


Figure 10.14: bind_all

When fast_bind is 1: the gateway will only send appkey add command. The provisioned device needs to enable the default binding function (PROVISION_FLOW_SIMPLE_EN is set to 1).

When fast_bind is 0: the gateway binds all model ids by default. To save time, users can choose the model ids to be bound. The gateway opens the macro MD_BIND_WHITE_LIST_EN. For the model ids to be bound, refer to the master_filter_list [] in the Mesh_common.c file. Users can modify it as needed.

- During the App_key bind process, the gateway will call u8 gateway_model_cmd_rsp (u8 * para, u8 len) to return the status information of the bound model in the format: TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_RSP_OP_CODE + parameter. i.e.: 91 + 81 + appkey bind status
- App_key bind will return HCI_GATEWAY_CMD_KEY_BIND_EVT after completion, indicating success or time_out. The format is:

TSCRIPT_GATEWAY_DIR_RSP + HCI_GATEWAY_CMD_KEY_BIND_EVT +result

i.e.: 91 + 8a + result. (1:success 2:time_out)

104.7 Light on/off Control

After app_key binding, click "Mesh" to enter mesh interface, or directly double click "g_all_on/g_all_off" in the left command window of the main interface to implement on/off control.

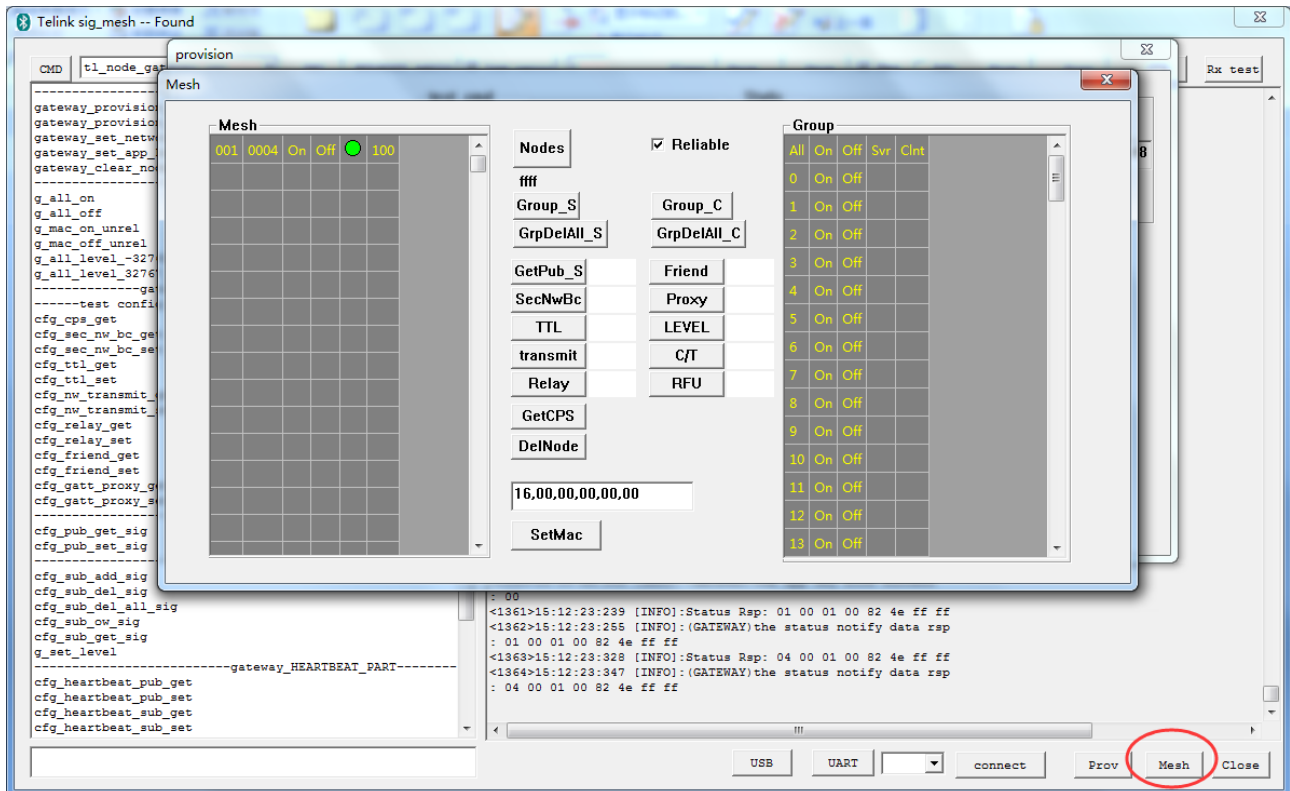


Figure 10.15: Light on/off control

104.8 Provisioner Control Flow Chart

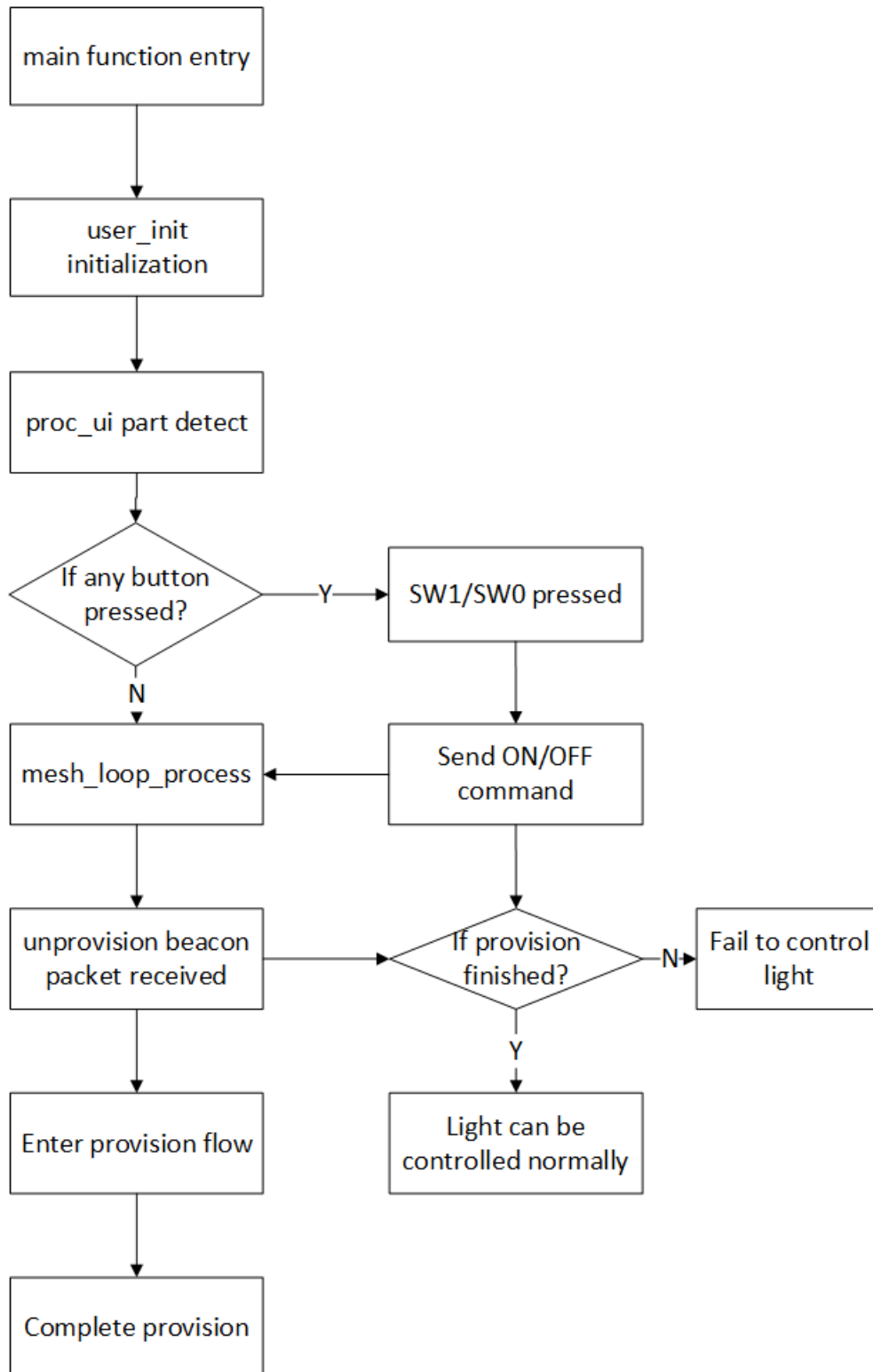


Figure 10.16: Provisioner Control Flow Chart

11 Mesh LPN Project Introduction

11.1 LPN Node and Implementation Method

Note: All figures in this section are derived from SIG MESH spec.

11.1.1 LPN and friend

Low-Power feature: Rx side can run with obviously low duty cycle in mesh network. By enabling radio receiver only when necessary, the duty cycle is minimized to decrease node's power consumption. This is implemented by establishing friendship between LPN (Low Power Node) and FN (Friend Node).

A LPN can only establish friendship with a single FN, while a FN can establish friendship with multiple LPNs.

After friendship is established, the LPN will initiate polling with a relatively long period to check whether there's new message from its FN. If yes, it will obtain this message.

Friend feature: To help LPN running, the FN will store the information to be sent to the LPN, and only initiate transmission when there's obvious request from the LPN.

11.1.2 Friendship Parameters

LPN needs to find FN and initiate a "Friendship Establish" process to establish friendship with it. Following shows some key parameters which are configured during the "Friendship Establish" process and serve to manage LPN behavior.

- 1) ReceiveDelay: After LPN sends request to its FN, it needs to wait a duration of "ReceiveDelay" to start listening for response from the FN, so that the FN can have enough time to make preparations and send back the response.
- 2) ReceiveWindow: It indicates the duration for LPN to listen for response. The figure below shows the timing sequence of "ReceiveDelay" and "ReceiveWindow".

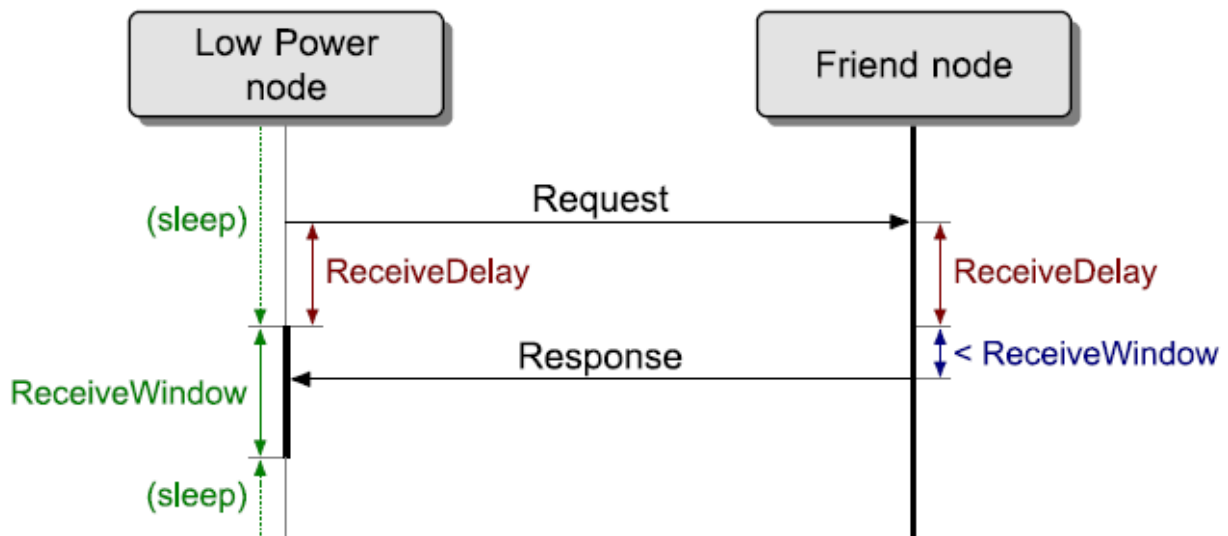


Figure 11.1: Timing Sequence of ReceiveDelay and ReceiveWindow

- 3) PollTimeout: It specifies the maximum duration between two adjacent requests from LPN to its FN. If the FN fails to receive request from the LPN before the "PollTimeout" timer expires, their friendship will be terminated.

11.1.3 Establish Friendship

The process to establish friendship in BT mesh network is shown as below:

Step 1 LPN issues a "Friend Request" message which does not support relaying. Only the FN within the direct radio range will process this message, and other nodes without "friend" features will discard this message. The "Friend Request" message contains parameters of LPN, including "ReceiveDelay", "ReceiveWindow" and "PollTimeout".

Step 2 If a FN nearby supports specific requirement in the "Friend Request" message, it will prepare a "Friend Offer" message and send it back to the LPN. This message contains various parameters, including supported ReceiveWindow size, available message queue size, available subscription list size, and RSSI value measured by the FN.

Step 3 When the LPN receives the "Friend Offer" message, it will adopt a specific algorithm to select suitable FN. This accurate algorithm may take various cases into consideration: Some device may give priority to the ReceiveWindow size, so as to minimize power consumption; some device may pay more attention to the RSSI value, so as to ensure high-quality link with FN. It depends on product developer.

Step 4 The LPN will send a "Friend Poll" message to the selected FN.

Step 5 After the "Friend Poll" message from the LPN is received, the FN will respond with a "Friend Update" message to finish "Friendship Establish" process and supply security parameters.

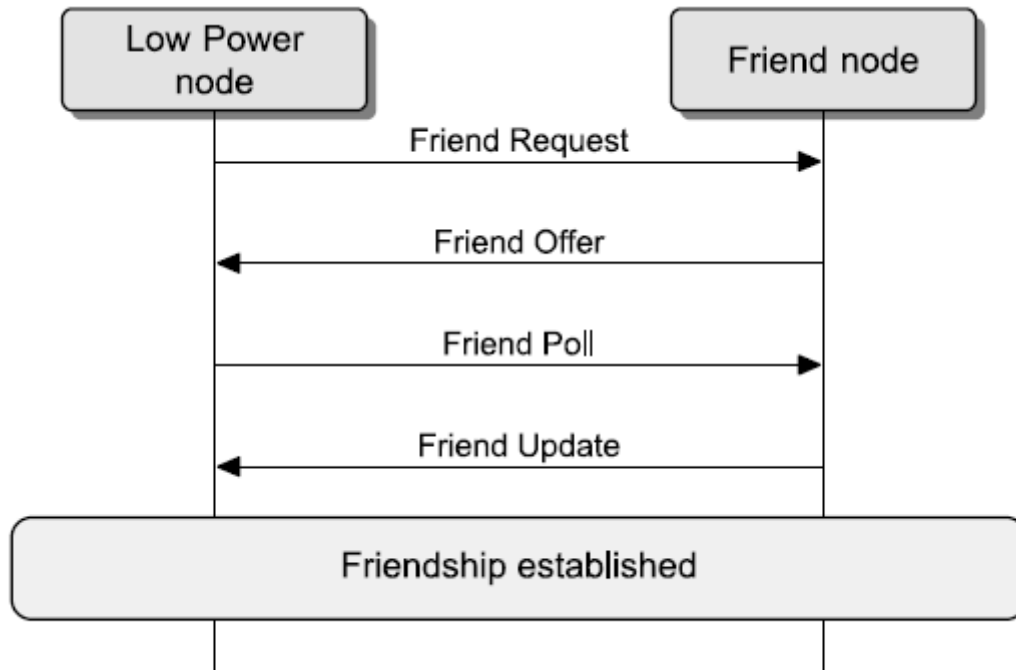


Figure 11.2: Establish friendship

11.14 Friendship Message Exchange

After friendship is established, the FN will store all messages of the LPN in the "Friend Queue". These messages are so-called "stored message". The figure below shows message exchange between the FN and the associated LPN.

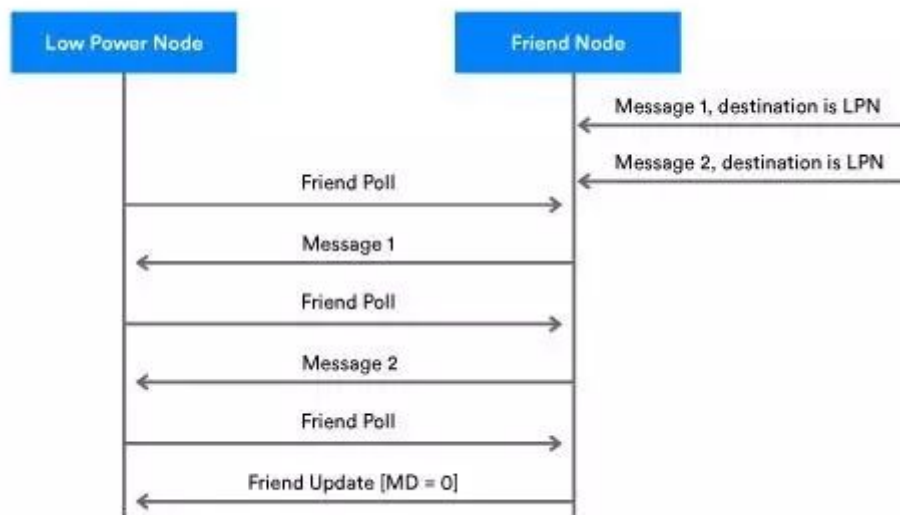


Figure 11.3: Friendship Message Exchange

When the FN receives a message from the LPN addressing to this node, the FN will buffer this message by storing it in the "Friend Queue" area. As shown in the figure above, the FN stores "Message 1" and "Message

2" for the LPN.

The LPN will periodically enable its transceiver, and send "Friend Poll" message to the FN so as to check whether there's any stored message buffered for the LPN.

The FN will first send a stored message to the LPN as the response to the "Friend Poll".

After each reception of message from the FN, the LPN will continue to send "Friend Poll" message until it receives a "Friend Update" message with the "MD (More Data)" field set as "0". "MD=0" means there's no more message buffered in the FN for the LPN. Then the LPN stops the polling to the FN.

11.1.5 Security

Master Security Material: It's derived from network key (NetKey), and it can be used by other nodes within the same network. Message encrypted by using "Master Security Material" can be decoded by any node within the same network.

Friend Security Material: It's derived from network key (NetKey), as well as extra counter number generated by the LPN and FN. Message encrypted by using "Friend Security Material" can only be decoded by the LPN and the FN processing this message.

Friend messages encrypted by using "Friend Security Material" include: "Friend Poll", "Friend Update", and "Friend Subscription List".

Friend messages encrypted by using "Master Security Material" include: "Friend Clear" and "Friend Clear Confirm".

Any other non-control message from the LPN to the FN will set the "credential_flag" in corresponding model publish parameter as needed, so as to determine whether the encryption method is "Master Security Material" or "Friend Security Material". The default value of the "credential_flag" is 0, corresponding to "Master Security Material" encryption.

11.1.6 Friendship Termination

If the FN fails to receive a "Friend Poll", "Friend Subscription List Add" or "Friend Subscription List Delete" message before the "PollTimeout" expires, the friendship between the FN and the LPN is terminated.

The LPN can initiate friendship termination program by sending a "Friend Clear" message to the FN, so that the FN will terminate their friendship.

11.2 LPN Demo

11.2.1 Hardware

This demo is based on the GATT master dongle mode. The operation steps of the APP and gateway modes are similar to the GATT master dongle mode. Note that in the gateway mode, the gateway node itself also supports the friend function.

One 8269 GATT master dongle and two 8258 mesh dongle (one burns 8258_mesh.bin, supports Friend function by default. The other burns 8258_mesh_LPN.bin, which is the LPN node).

LPN supports generic ONOFF, generic Level by default, and does not enable lightness and light CT.

11.2.2 Test method

The time-related macros mentioned below can be modified by customers according to their actual needs.

Step 1 Mesh friend node (FN) is powered on and provisioned with SIG_MESH_TOOL.

Step 2 Powered on unprovisioned LPN node, at this time, the LPN is in awake state.

After the LPN node is powered on, the red LED will be in the ON mode. In the unprovision state, do not enter the sleep mode, the purpose is to support GATT provision and ADV provision. In this state, if the provisioning process has not started after 1 minute (LPN_SCAN_PROVISION_START_TIMEOUT_MS), then the system will enter the deep sleep mode, and ADV will not be sent, LED will be turned off, the purpose is to save power consumption and avoid working in high power consumption mode for too long. If LPN have entered deep sleep, you need to press SW1 or SW2 defined in mesh_lpn_key_map [] to wake up. After wakeup, LPN will start sending ADV again and waiting for the provision flow.

Step 3 Provision and bind key process for the unconfigured LPN in the awake mode.

When Bind key is successful. After 3 seconds (LPN_START_REQUEST_AFTER_BIND_MS), LPN will automatically reboot, and then set lpn_provision_ok to 1, and enter LPN mode, starting to send a friend request command every 2 seconds (FRI_REQ_TIMEOUT_MS).

When the provision is successful, in order to reduce the processing of invalid network messages and reduce power consumption, LPN only receives messages sent from FN through friend ship. If you want to receive ordinary network messages, initialize mesh_lpn_rx_master_key to 1.

Step 4 When there is FN, it will automatically establish a friend ship. Only when the establishment is successful (red light flashes 3 times), LPN can receive message.

After receiving the friend request, FN will automatically reply to the friend offer, and then establish the friendship. If the establishment is successful, the friend_ship_establish_ok_cb_lpn () will be called back and the red light will flash 3 times (LGT_CMD_FRIEND_SHIP_OK). Then it starts sending friend POLL in a 2 second period (FRI_POLL_INTERVAL_MS). After the FN receives the POLL, if there is a cache message that needs to be sent to the LPN, it will send the message to the LPN. The default maximum number of Cache message (network PDU) is 4 ($2^{\wedge} \text{FN_CACHE_SIZE_LOG}$).

If there is no FN responding to Friend Request, LPN will keep sending friend request in 2 second cycle.

Step 5 The "mesh" window displays the LPN node and ONOFF operation.

First open the "mesh" window, click the "LPN_get_level" INI command, a3 ff 00 00 00 00 00 00 04 00 82 05 appears in the lower left corner of the figure below, where 04 00 is the unicast address of LPN (if it is incorrect, it needs to be modified), press the Enter key to send the command. After receiving the LPN level status reply, the LPN node will be displayed in the UI, and then you can initiate the ONOFF operation on the LPN, as shown below.



Figure 11.4: LPN_get_level

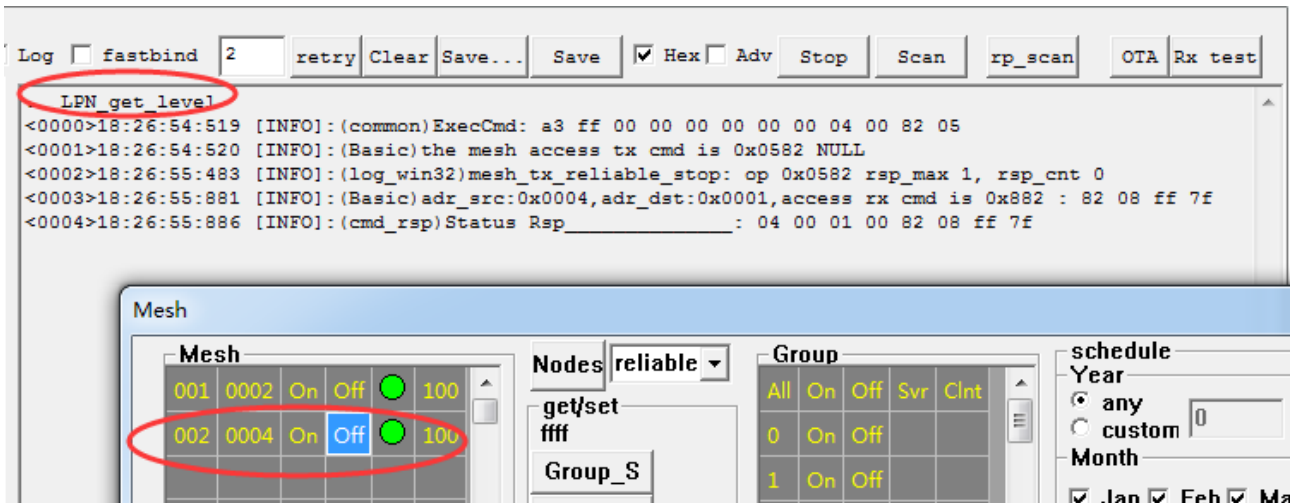


Figure 11.5: ONOFF operation on LPN

Because LPN does not receive the message whose destination address is 0xffff, it needs to send the command in unicast mode. If you have configured a group number for LPN, you can also send commands in group mode.

Also note that after clicking the "Nodes" button or reopening the "Mesh" window, the VC tool will put all the nodes offline, and then send the lightness get all (destination address is 0xffff) command to regain the node status, but There is no separate send command to the LPN node by unicast destination address, so you need to manually click the "LPN_get_level" command or click the ON / OFF command in the mesh window to display the online status, otherwise it is offline.

Step 6 group operation is the same as normal node operation, please refer to "4.5.2 Group Control (ie Subscription Function Demo)"

Step 7 The LPN detects that the FN is powered off and automatically searches for a new FN.

When the FN is powered off, LPN retry 8 times (FRI_POLL_RETRY_MAX) POLL command, where the POLL interval is 170ms (FRI_REC_DELAY_MS + FRI_REC_WIN_MS), if the LPN still does not receive a reply from the FN, it is considered that the FN has been powered off, it will disconnect the friend ship and callback friend_ship_disconnect_cb_lpn (), if you need to perform led flashing operation, please add it in the callback function, then resend the friend request to find a new friend node.

Step 8 For now, one friend node of demo SDK establishes a friend ship with two LPNs at the same time by default. If you need to modify it, just set MAX_LPN_NUM. The maximum value is 16.

When the LPN is powered off, the FN will detect for 10 seconds (LPN_POLL_TIMEOUT_100MS). If the POLL command has not been received, the node is considered to be powered off. At this time, the FN will clear the LPN information.

Step 9 Press the key to send the ALL ON / OFF command.

When the LPN is in the retention sleep mode, press SW2 (MESH_LPN_CMD_KEY) to wake up the LPN, and then detect the key through suspend_handle_next_poll_interval ()-> mesh_lpn_wakeup_key_io_get (), and then execute the test_cmd_wakeup_lpn () function to alternately send ALL ON / OFF commands. LPN spontaneously sends the access layer command to use master security credentials to encryption by default.

Step 10 Reset to Factory Setting

Long press the button SW1 (MESH_LPN_FACTORY_RESET_KEY) for 3 seconds (LONG_PRESS_TRIGGER_MS) to trigger the factory reset.

11.3 app.c file introduction

Customization of Adv Packet and Adv Response Packet

Please refer to Section 9.

Configuration of fifo part

Please refer to Section 9.

app_event_handler ():

Please refer to Section 9.

main_loop ():

Please refer to Section 9.

user_init():

Please refer to Section 9.

proc_ui():

This function mainly implements UI related processing, e.g. button detect function, as well as corresponding test code. This function is reserved currently.

test_cmd_wakeup_lpn():

When pressing the corresponding command button (SW2 in current demo dongle) to wake up the program, the function "test_cmd_wakeup_lpn()" will be executed. This function will send ON/OFF command. After sending the command, it will enter sleep.

This function is only used for demo demonstration.

11.4 mesh_lpn.c file introduction

mesh_lpn_proc_suspend ():

This function handles the sleep processing function of LPN. lpn_sleep.op indicates what command or event needs to go to sleep, and handles the subsequent actions of the event after waking up.

For example, when lpn_sleep.op is equal to CMD_CTL_POLL, it means that the POLL message has just been sent, and then you need to enter the retention sleep time of receive delay, and then wake up to enter the receive window, as shown below:

```

void mesh_lpn_proc_suspend ()
{
    .....

    if(lpn_sleep.op && is_lpn_support_and_en){
        .....
        u8 r = irq_disable();
        if((CMD_ST_SLEEP == lpn_sleep.op) || (CMD_ST_NORMAL_UNSEG == lpn_sleep.op)){
            .....
        }else if(CMD_ST_NORMAL_SEG == lpn_sleep.op){ // no need to enter retention deep sleep
            .....
        }else{
            // wake up for rx after send ctrl op, such as POLL,...
            // can't use HANDLE_RETENTION_DEEP_PRE now, because it is from lpn_quick_tx_and_suspe
            suspend_handle_wakeup_rx(HANDLE_SUSPEND_NORMAL);
        }
        irq_restore(r);
    }
}

```

Figure 11.6: mesh_lpn_proc_suspend

Other customized events are:

CMD_ST_SLEEP: After the interaction cycle of a friend ship is completed, it enters the retention sleep mode of 2 seconds (friend request interval or poll interval), and then wakes up to enter the interaction of the next cycle.

CMD_ST_NORMAL_UNSEG: After spontaneously sending the access layer's unsegment message, enter the same processing mode as CMD_ST_SLEEP.

CMD_ST_NORMAL_SEG: After spontaneously sending the segment message of the access layer, it goes to sleep, waits for a period of time, wakes up, and then receives the block ack, as well as the packet loss mechanism for processing the segment message and so on.

CMD_ST_POLL_MD: After sending the POLL, the FN reply MD (more data) is 1, then sleep for 100ms (FRI_POLL_DELAY_FOR_MD_MS), wake up, and continue to send POLL to receive the remaining message.

suspend_handle_next_poll_interval():

Execute the POLL processing function of the next cycle, which contains the HANDLE_RETENTION_DEEP_PRE and HANDLE_RETENTION_DEEP_AFTER branches, because after the 825x performs retention sleep, the MCU does not continue to execute backward, but will execute from main () -> user_init_deepRetn ()-> suspend_handle_next_pollinter_pollinter Subsequent processing.

mesh_feature_set_lpn():

Initialization of some configurable parameters of LPN. Mainly configure LPN_POLL_TIMEOUT_100MS, the default value is 10 seconds.

12 SWITCH Project Introduction

12.1 Switch function introduction

Switch mainly serves to add the function of remote control. The provisioner needs to add the switch node into the network, so that the buttons on the switch can be used to control nodes in the mesh network.

12.2 Switch principle

As a low power remote control node to control mesh, the switch must trigger pairing mode to implement provision. After the provisioner adds the switch to the mesh network, the switch can control nodes in the network.

12.3 app.c file introduction

Customization of Adv packet and Adv response packet

Please refer to Section 9.

Configuration of fifo part

Please refer to Section 9.

app_event_handler ():

Please refer to Section 9.

main_loop ():

Please refer to Section 9.

user_init():

Please refer to Section 9.

proc_ui ():

The "proc_ui" function configures key scan with the interval of 4ms. "mesh_proc_keyboard" is the interface function for key processing.

- When "keycode" is "RC_KEY_A_ON", the switch will send the all_on command to turn on all lights in the network.
- When "keycode" is "RC_KEY_A_OFF", the switch will send the all_off command to turn off all lights in the network.

proc_led():

First the configuration function "cfg_led_event" is used to configure LED blinking frequency and time. E.g. "cfg_led_event(LED_EVENT_FLASH_1HZ_4S)": configure LED to blink for 4s with the frequency of 1Hz. Then the function "proc_led" serves to control the processing of LED blinking part.

mesh_switch_init():

The "mesh_switch_init" contains setting of two parts:

- Code setting of wakeup IO of switch part, as well as enabling of the wakeup enable flag bit.
- IO setting of LED part. By default, LED pin is configured as GPIO mode with 100kohm pull-down resistor, and LED will blink four times after power on.

proc_rc_ui_suspend():

The processing function for sleep function part is "proc_rc_ui_suspend()".

The processing of sleep part in current SDK is set as below: In advertising state, if MCU directly enters deep state without sending packets, after wakeup by key press, MCU will continue to enter deep state when packet transmission is finished. After pairing mode is triggered, MCU will enter deep state 30s later, and it won't enter deep state temporarily in link state.

For the processing flow of sleep part, please refer to section 12.7.

kb_scan_key ():

"kb_scan_key" is the interface of matrix keyboard scan part. In current SDK, by default "numlock_status" is set as 0 to indicate the numlock in full keyboard, while "read_key" is the read key value.

124 Configuration of Switch Part

124.1 key table

```
#define KB_MAP_NORMAL  {\n    {RC_KEY_1_OFF,    RC_KEY_2_OFF,    RC_KEY_1_ON}, |\n    {RC_KEY_3_ON,     RC_KEY_3_OFF,    RC_KEY_2_ON}, |\n    {RC_KEY_4_ON,     RC_KEY_4_OFF,    RC_KEY_R},  |\n    {RC_KEY_A_OFF,    RC_KEY_A_ON,     RC_KEY_UP},  |\n    {RC_KEY_L,        RC_KEY_DN,       RC_KEY_M},   }
```

User can configure the contents of actual "key_table" according to the number of drive pins and scan pins which correspond to the number of columns and rows respectively.

124.2 Configure IOs for Drive Pins and Scan Pins

```
#define KB_DRIVE_PINS  {GPIO_PB4,  GPIO_PB5,  GPIO_PB6}\n#define KB_SCAN_PINS {GPIO_PE3,  GPIO_PE2,  GPIO_PE1,  GPIO_PE0,  GPIO_PD3}
```

Modify macros corresponding to "KB_DRIVE_PINS" and "KB_SCAN_PINS" according to actually used pins.

Then customize IO attributes for drive pins and scan pins, as shown below:

IO attribute setting corresponding to drive pins:

```
#define PB4_FUNC          AS_GPIO
#define PB5_FUNC          AS_GPIO
#define PB6_FUNC          AS_GPIO
#define PULL_WAKEUP_SRC_PB4  MATRIX_ROW_PULL
#define PULL_WAKEUP_SRC_PB5  MATRIX_ROW_PULL
#define PULL_WAKEUP_SRC_PB6  MATRIX_ROW_PULL
#define PB4_INPUT_ENABLE    1
#define PB5_INPUT_ENABLE    1
#define PB6_INPUT_ENABLE    1
```

IO attribute setting corresponding to scan pins:

```
#define PE3_FUNC          AS_GPIO
#define PE2_FUNC          AS_GPIO
#define PE1_FUNC          AS_GPIO
#define PE0_FUNC          AS_GPIO
#define PD3_FUNC          AS_GPIO
#define PULL_WAKEUP_SRC_PD3  MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE0  MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE1  MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE2  MATRIX_COL_PULL
#define PULL_WAKEUP_SRC_PE3  MATRIX_COL_PULL

#define PE3_INPUT_ENABLE    1
#define PE2_INPUT_ENABLE    1
#define PE1_INPUT_ENABLE    1
#define PE0_INPUT_ENABLE    1
#define PD3_INPUT_ENABLE    1
```

Suppose it's needed to modify "GPIO_PB6" as "GPIO_PB7" in drive pin part, the following parts should be modified accordingly.

```
1). #define PB6_FUNC      AS_GPIO----->>>#define PB7_FUNC      AS_GPIO
2). #define PULL_WAKEUP_SRC_PB6      MATRIX_ROW_PULL----->>
   #define PULL_WAKEUP_SRC_PB7      MATRIX_ROW_PULL
3). #define PB6_INPUT_ENABLE    1 ----->>
   #define PB7_INPUT_ENABLE    1
```

124.3 Turn on/off Light via Switch

According to different key values, different commands will be sent so as to process correspondingly.

Please refer to key processing program "mesh_proc_keyboard ()". Switch cannot control light nodes before it's added into the network. User can simultaneously press the "RC_KEY_A_ON" and "RC_KEY_1_OFF" button on the switch for more than 2 seconds to trigger pairing mode, and add the switch into the mesh network

via the provisioner, so that all light nodes in this network can be turned on/off via the "RC_KEY_A_ON" and "RC_KEY_A_OFF" button. Please refer to section 12.5 for details.

12.5 Switch Operation Guide

First follow the provision operations in section 10.4. Connect the switch with PC USB via Telink burning EVK (as shown in the figure below), and then burn the switch with corresponding firmware.



Figure 12.1: Switch Burning Connection

The switch buttons are shown as below:

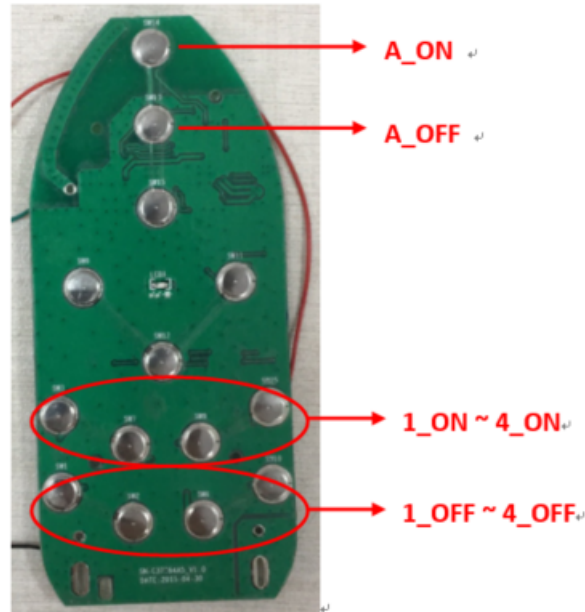


Figure 12.2: Switch button

Power on the switch device. After power on, as a low power node, the switch must trigger pairing mode by simultaneously pressing the "RC_KEY_A_ON" and "RC_KEY_1_OFF" for more than 2s, so that it can be added into mesh network via the provisioner.

After the switch triggers pairing mode, its LED light will continuously blink four times to indicate it enters pairing mode. Power on the provisioner (if it's powered down), and wait for 15s or so. The LED on the switch will continuously blink four times to indicate the switch has already been added into the network.

Then the "RC_KEY_A_ON" and "RC_KEY_A_OFF" on the switch can be used to turn on/off all light nodes in the network.

12.6 Flow chart for Switch RC

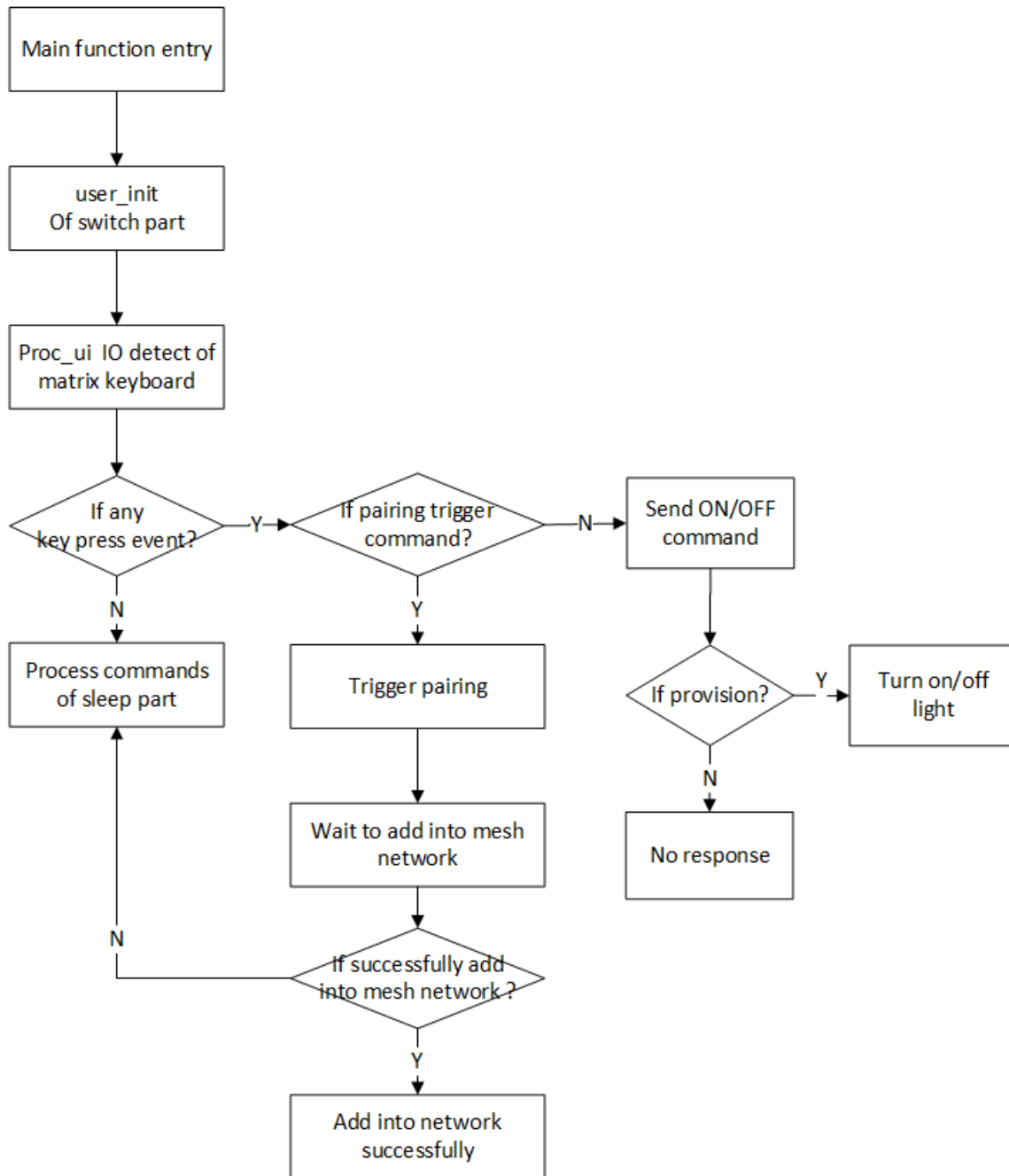


Figure 12.3: Flow chart for switch RC

12.7 Flow chart for sleep processing

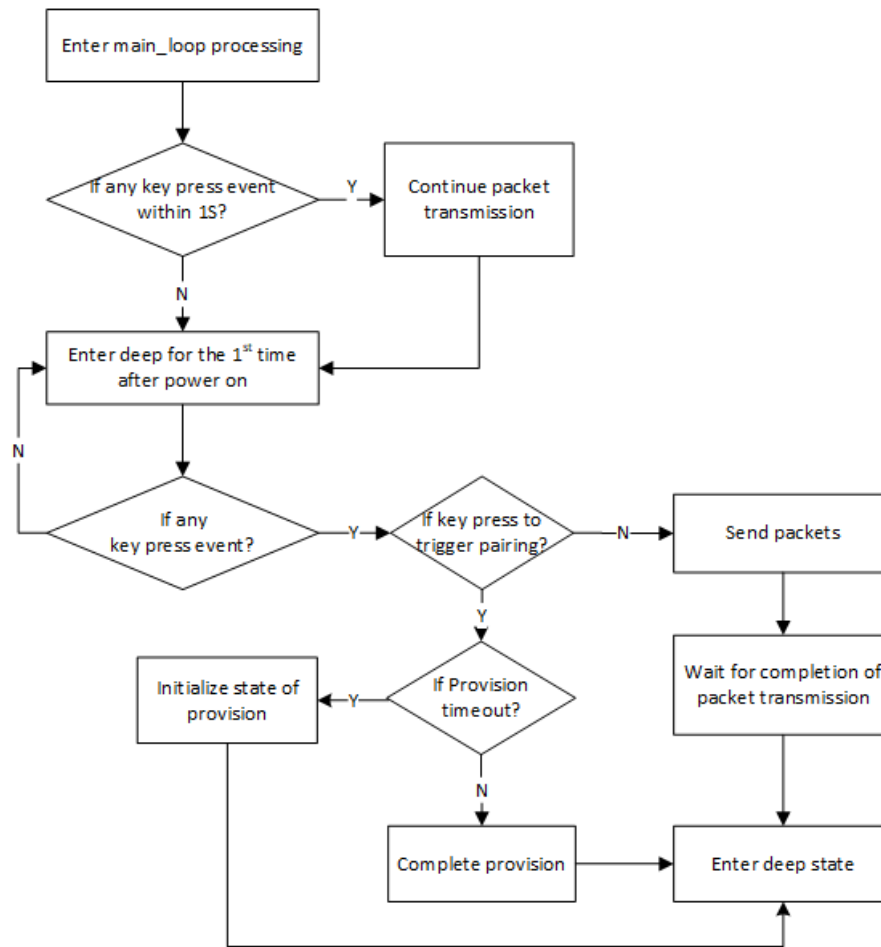


Figure 12.4: Flow chart for sleep processing

13 Connect with a Platform

When connect with a certain platform, you need to configure some options, especially the provision method. Select by configuring MESH_USER_DEFINE_MODE.

13.1 Normal Mode

The SDK defaults to Normal mode. No changes are required.

13.1.1 No OOB provision mode

Configuration method:

Provision uses MESH_NO_OOB mode.

VENDOR_ID is 0x0211

When testing, you can directly use our mobile app or host computer tools to provision.

13.1.2 static OOB provision mode

13.1.2.1 Light Node Burn Static oob

When burning the firmware, just write 16 bytes directly in the flash fixed location `FLASH_ADR_STATIC_OOB` (for example, `0x77800`). If there is no burning (all `0xff`), it means that no oob mode is used. If you need to modify the flash address, modify the macro `FLASH_ADR_STATIC_OOB`.

13.1.2.2 Light node Device uuid

The device uuid is generated by `user_prov_multi_device_uuid () -> uuid_create_by_mac (tbl_mac, prov_para.device_uuid)` by default and can be obtained by the following methods:

- 1) Read prov_para.device_uuid through BDT tool
- 2) Obtain unprovision broadcast package through the general APP

[illegible]

Figure 13.1: Device uuid

3) Obtain unprovision broadcast package through TI sniffer

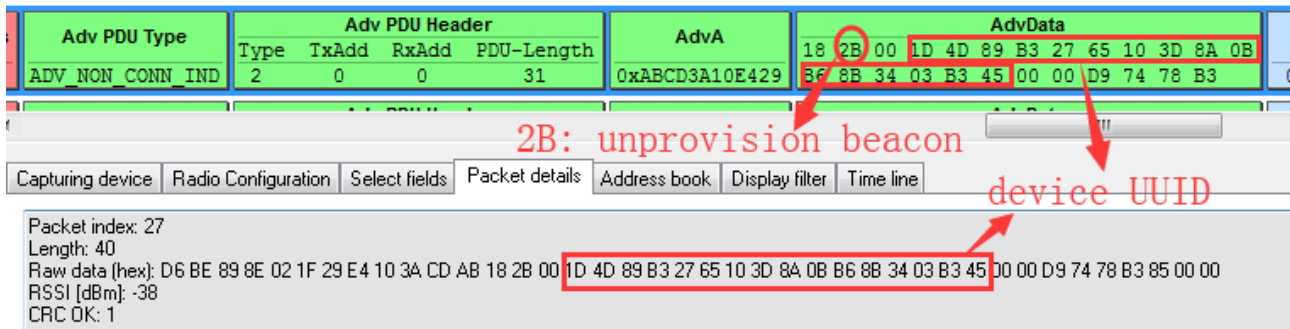


Figure 13.2: unprovision broadcast package

4) when the connection is successful, the device uuid will be printed out

```
<0027>23:03:16:292 [INFO]: (GattProv)CScanDlg::OnConnect:the device uuid is
: 1d 4d 89 b3 27 65 10 3d 8a 0b 29 e4 10 3a cd ab
```

Figure 13.3: Connection successful and print device uuid

5) In the case of gateway provision, when selecting a node obtained by scan, the device uuid will be printed.

```
<0005>00:02:49:081 [INFO]: (GattProv)CScanDlg::provision link:the device uuid is
: 1d 4d 89 b3 27 65 10 3d 8a 0b 29 e4 10 3a cd ab
```

Figure 13.4: Print device uuid at a scan node

13.1.2.3 User Customized uuid Method

If the user wants to customize the device uuid, set NORMAL_MODE_DEV_UUID_CUSTOMIZE_EN to 1.

13.1.2.4 Provisioner static oob database

The Provisioner needs to fill in the oob data of the node to the oob database file (oob_database.txt):

The data format of oob database is as follows:

device uuid(16byte) + oob(16byte)

For example: 1d4d89b32765103d8a0b29e4103acdab ff000000000000000000000000000000

Field analysis is as follows:

1d4d89b32765103d8a0b29e4103acdab: The device uuid of the provisioned node.

ff000000000000000000000000000000: The oob data of the provisioned node, that is, the value written by Light Firmware at the fixed flash location FLASH_ADR_STATIC_OOB (for example, 0x77800)

If there are multiple nodes, the line break can be increased, for example:

```
1d4d89b32765103d8a0b29e4103acdab ff000000000000000000000000000000
9f6f6e6e5e90943db9be6d79446a9e37 ffaa0000000000000000000000000000
```

13.1.2.5 Test steps

Please follow the general process for provision and testing.

The test results of the static oob provision success, the screenshot using GATT master dongle mode is as follows:

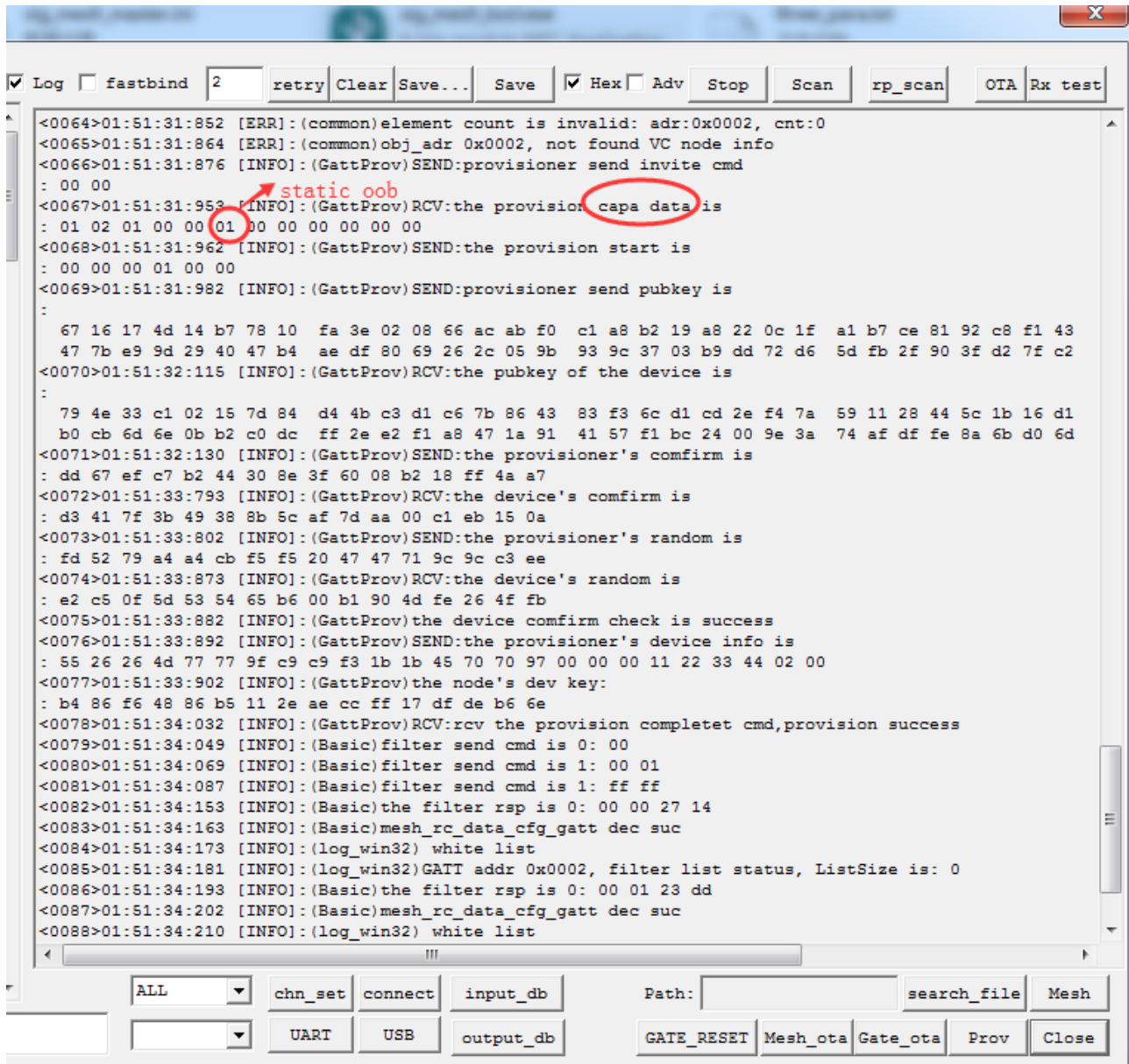


Figure 13.5: static oob provision success

Capability data, please refer to the following chapters of spec for more detailed analysis.

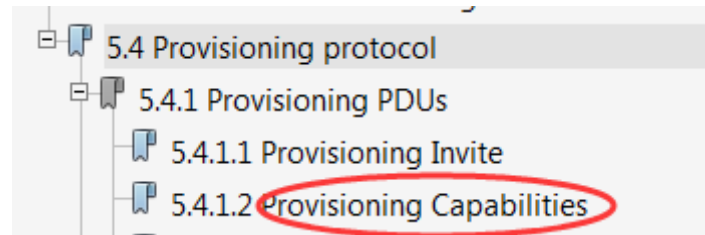


Figure 13.6: capability data

Data analysis is as follows:

01 02 01 00 00 01 00 00 00 00 00 00

01: Provisioning PDU Type, 01 indicates Provisioning Capabilities

02: Number of Elements

01 00: Algorithms

00: Public Key Type

01: Static OOB Type

00: Output OOB Size

00 00: Output OOB Action

00: input OOB Size

00 00: Input OOB Action

13.2 Ali Tmall Genies Platform

13.2.1 Configuration

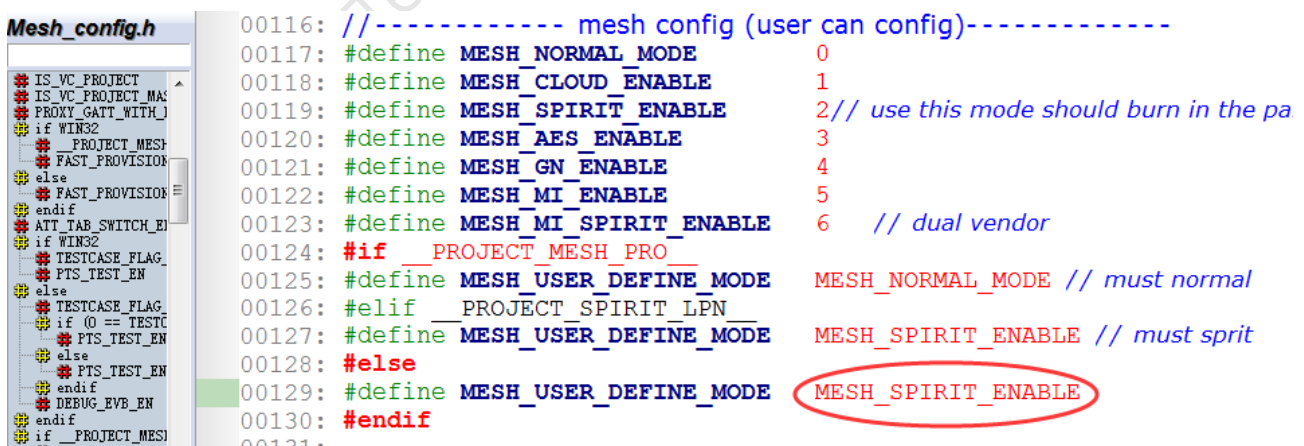


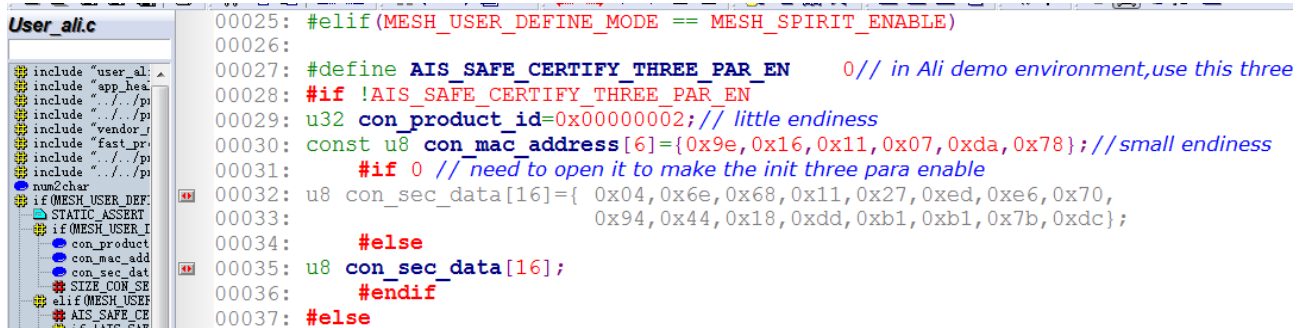
Figure 13.7: Configuration

Provision uses MESH_STATIC_OOB mode.

VENDOR_ID is 0x01A8.

13.2.2 Apply tri-truple from Ali

The default tri-truple information is null(`con_sec_data[16]` is null), code is in `user_ali.c`, shown as following:



```

00025: #elif(MESH_USER_DEFINE_MODE == MESH_SPIRIT_ENABLE)
00026:
00027: #define AIS_SAFE_CERTIFY_THREE_PAR_EN    0 // in Ali demo environment, use this three
00028: #if !AIS_SAFE_CERTIFY_THREE_PAR_EN
00029: u32 con_product_id=0x00000002; // little endiness
00030: const u8 con_mac_address[6]={0x9e,0x16,0x11,0x07,0xda,0x78}; // small endiness
00031:     #if 0 // need to open it to make the init three para enable
00032: u8 con_sec_data[16]={ 0x04,0x6e,0x68,0x11,0x27,0xed,0xe6,0x70,
00033:                     0x94,0x44,0x18,0xdd,0xb1,0xb1,0x7b,0xdc};
00034:     #else
00035: u8 con_sec_data[16];
00036:     #endif
00037: #else

```

Figure 13.8: Apply tri-truple

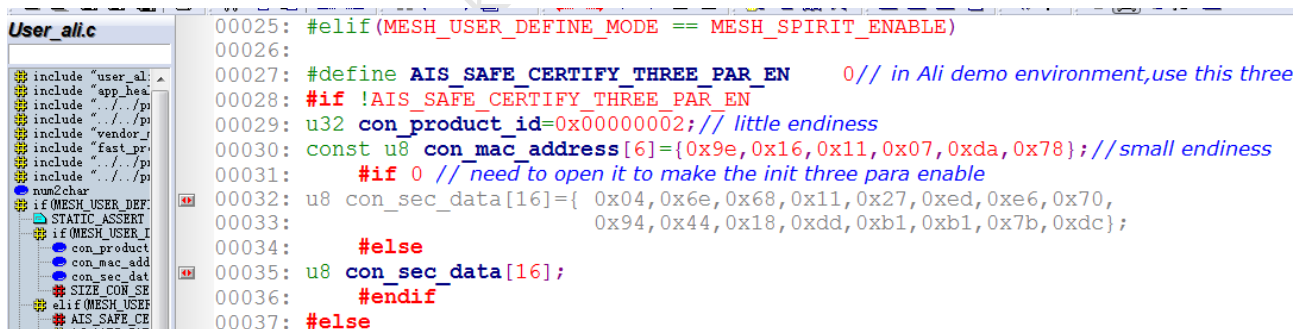
So user need to apply tri-truple from Ali:

(Total 24byte: PID(4byte, smaller end) + MAC(6byte, bigger end) + secret data(16 byte)),

Then burn to `FLASH_ADR_THREE_PARA_ADR(0x78000)`, and program will read parameter in `0x78000` automatically.

13.2.3 Use SDK Default tri-truple

User can use SDK default tri-truple for demo, Open it as follows: (Enable the preset `con_sec_data []` in the `user_ali.c` file)



```

00025: #elif(MESH_USER_DEFINE_MODE == MESH_SPIRIT_ENABLE)
00026:
00027: #define AIS_SAFE_CERTIFY_THREE_PAR_EN    0 // in Ali demo environment, use this three
00028: #if !AIS_SAFE_CERTIFY_THREE_PAR_EN
00029: u32 con_product_id=0x00000002; // little endiness
00030: const u8 con_mac_address[6]={0x9e,0x16,0x11,0x07,0xda,0x78}; // small endiness
00031:     #if 0 // need to open it to make the init three para enable
00032: u8 con_sec_data[16]={ 0x04,0x6e,0x68,0x11,0x27,0xed,0xe6,0x70,
00033:                     0x94,0x44,0x18,0xdd,0xb1,0xb1,0x7b,0xdc};
00034:     #else
00035: u8 con_sec_data[16];
00036:     #endif
00037: #else

```

Figure 13.9: Apply tri-truple

However, because there is only one default tri-truple, it can only be used for a single node demonstration when testing.

13.2.4 Provision via Tmall Genie

Provision can be done directly through Tmall Genie's voice commands.

13.2.5 Provision via Firmware

Because Tmall Genie mode only supports the static oob mode by default, oob and tri-truple have a binding relationship, so the firmware needs to know the information of the tri-truple to provision. So you need to add the tri-truple to this file of the firmware:

SIG_MESH_Release_Vxxx -> tools -> telink-ble-phone -> three_para.txt

SDK default tri-truple information has been added to this file. When adding new tri-truple information, just refer to this format.

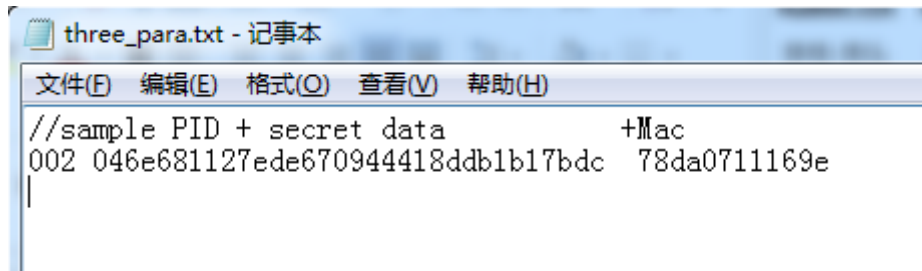


Figure 13.10: firmware file

Then refer to Provision part in chapter 4 for detail steps.

13.2.6 Dual Modes of static oob and no oob

Tmall Genie mode, the node end only responds to the static oob mode by default. If you need to support no oob mode at the same time, change ENABLE_NO_OOB_IN_STATIC_OOB from 0 to 1.

13.3 Xiaomi Xiao'ai Platform

13.3.1 Configuration

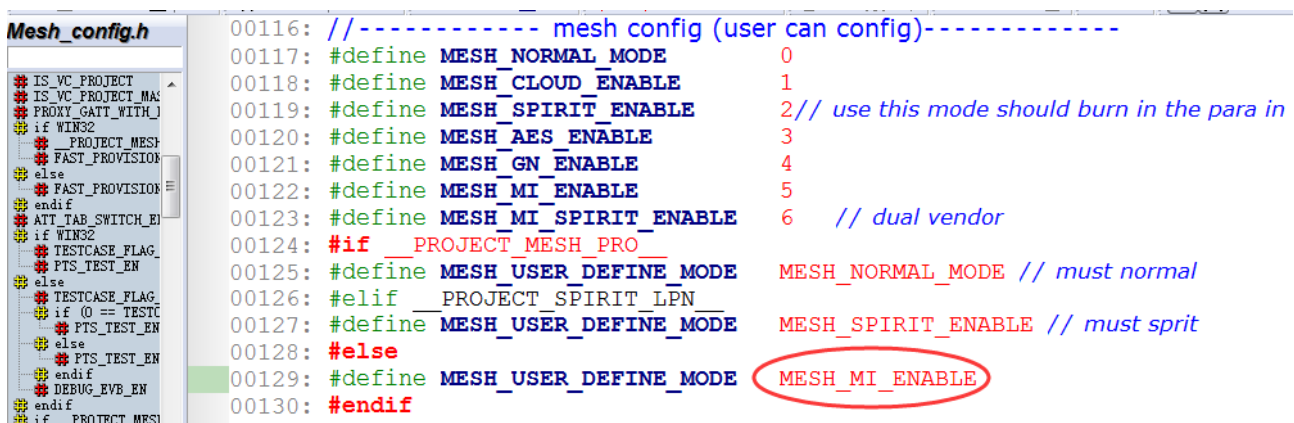


Figure 13.11: Xiaomi defined mesh provision mode

Provision uses Xiaomi defined mesh provision mode

VENDOR_ID is 0x038F.

13.3.2 Certification Data Setting

- R & D test mode: The default certification data of the SDK is used, dev_cert_pri [], so it can only be used in the single node test mode. There are 6 certificates by default. Because these certificates are public, if they have been used by others, conflict will happen. So it is recommended for users to use the certificate they applied for and then testing it in production mode.
- Production mode: When production or multi-node network testing is required, flash writing is required. The steps are as follows:

```
#define DEMO_CERT_TYPE0 0
#define DEMO_CERT_TYPE1 1
#define DEMO_CERT_TYPE2 2
#define DEMO_CERT_TYPE3 3
#define DEMO_CERT_TYPE4 4
#define DEMO_CERT_TYPE5 5

#define DEMO_CERT_TYPE DEMO_CERT_TYPE1
```

Figure 13.12: Apply certificate

Step 1 Define MI_CER_MODE as FLASH_CER_MODE, generate firmware

Step 2 Burn certification data to DEV_SK_FLASH_ADR(0x7f000)

13.3.3 Provision Test

After burning firmware, please make sure that the MAC address of the flash (512K flash is at 0x76000 and 1M flash is at 0xFF000) is empty (that is, all 0xff), otherwise it will prompt "Unable to connect" or "Provision failure".

When firmware is powered on for the first time, it will extract the MAC from the certificate, write it to the MAC address sector, and generate some necessary parameters.

After the node is powered on, it can be directly provisioned through Xiao'ai (Voice instruction example: "Xiao'ai, add device").

134 Dual Vendor Mode (Tmall Genies and Xiaomi Xiaoi)

134.1 Function Introduction

When the node leaves the factory, it will send adv. packets in Ali and Xiaomi modes at the same time, which can be provisioned by either Tmall Genies or Xiao'ai. Once provision is successful, subsequent functions are performed according to the selected mode, including parameters such as the vendor model and the transmit count. For parameter switching functions, see mesh_ais_global_var_set ().

The production test function in unprovisioned state of is performed according to the Xiaomi mode.

After provision is successfully, you can execute the kick light command or restore the factory settings to return to the dual vendor mode and select again.

Which mode you are currently in can be viewed through the provision_mag.dual_vendor_st variable.

134.2 Configuration

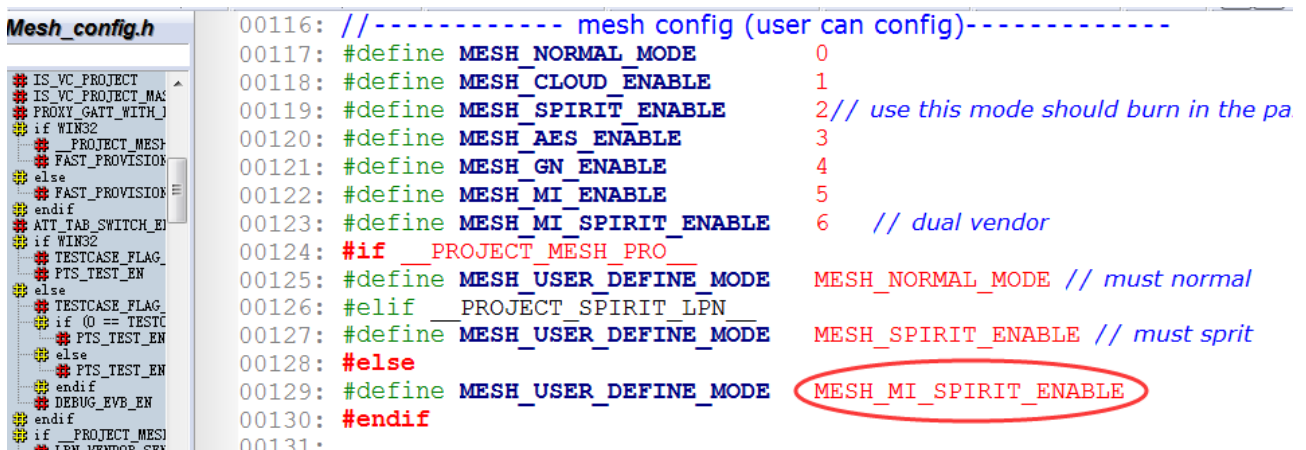


Figure 13.13: Provision method

Provision method and parameter configuration, please check 13.2 and 13.3.

14 Factory Reset

14.1 8258_mesh/8269_mesh Node

14.1.1 Function Introduction

Factory reset reset execution action, please refer to `factory_reset_handle ()` or `kick_out()`:

```
{
  irq_disable();
  factory_reset(); // Flash erasing
  #if DUAL_MODE_WITH_TLK_MESH_EN
  UI_resotre_TLK_4K_with_check();
  #endif
  show_ota_result(OTA_SUCCESS); // LED indication
  start_reboot(); // MCU reboot
}
```

If the customer has modified the flash map, or used the customer flash section (the default is 0x7a000—0x7f000, and erase is not performed on the area by default), you need to reconfirm the `factory_reset ()` function to confirm whether there are sector errors or missing erases.

Power-up sequence detection function `factory_reset_cnt_check ()`:

- (a) After power-on, the power-on sequence will not be detected until after `VALID_POWER_ON_TIME_US` (default 50ms). Because it is necessary to filter the pulse voltage generated when some power supplies are powered on.
- (b) `clear_st` is 4:

First `reset_cnt_get_idx ()` to obtain the sequence before power off, if it is an odd number, it means that the previous power-on sequence does not meet expectations, directly clear the power-on sequence and restart counting. If it is even, then it is as expected.

Then, check whether the stored power-on sequence value satisfies the condition that triggers a factory reset, and if it does, execute a factory reset. If not, immediately add 1 to the sequence value.

- (c) `clear_st` is 3, get the power-on sequence value by `get_reset_cnt ()`, get the timing time, and start the timing of the first phase.
- (d) `clear_st` is 2, the first phase timing meets the requirements, get the power-on sequence value through `get_reset_cnt ()`, get the timing time, and start the second phase timing.
- (e) `clear_st` is 1. If the second phase is over and power has not been turned off, it means that the power-on time is not as expected, and the power-on sequence is directly cleared.

14.1.2 Default trigger action

Low power node, e.g., LPN, cannot be triggered with booting sequence, because LPN cannot count time and determine timing sequence. User can trigger this with button-pressing, call functions described in 14.1.1.

User can follow the steps below to reset a SIG_mesh module (non-LPN) to factory default configuration:

Step 1 Power on the SIG_mesh module, and wait for more than 30s (equivalent to initial power on).

This operation will erase previous power on record, and ensure it conforms to the timing sequence requirement of initial power on rather than any power-on sequence defined by "factory_reset_serials[]".

Step 2 Power cycle the SIG_mesh module three times. Note: After each power on, the module must be powered down within the range of 0~3s. This operation conforms to the requirement of former three power-on sequences in the "factory_reset_serials[]".

Step 3 Power cycle the SIG_mesh module two times. Note: After each power on, the module must be powered down within the range of 3~30s. This operation conforms to the requirement of latter two power-on sequences in the "factory_reset_serials[]".

Step 4 Power cycle the SIG_mesh module. The "factory_reset_handle()" in the "user_init()" will detect and get the result that previous five power-on sequences match with the trigger requirement of "Factory Reset". The red LED light on the module will blink for 8s with the frequency 1Hz to indicate factory reset success.

Note:

- Since some module may need some time to finish power down, to ensure its MCU is stopped completely, it's needed to wait for a duration (e.g. 2s, depend on module) after power-down operation.

Power-on timing sequence is defined by the array below:

```
u8 factory_reset_serials[] = { 0, 3,
                               0, 3,
                               0, 3,
                               3, 30,
                               3, 30,};
```

14.1.3 Method to modify power-on sequence

To modify power-on sequence, it's only needed to modify/add/delete sequences in the array "factory_reset_serials[]" correspondingly, as long as the requirements below are met:

- The left value should be smaller than the right value.
- When it's needed to add/delete sequences, since one sequence corresponds to two values, multiple of two values must be added/deleted correspondingly.

E.g. To modify power-on sequence as six sequences, the following method can be followed.

```
u8 factory_reset_serials[] = { 0, 3,
                               0, 3,
                               0, 3,
                               3, 30,
                               3, 30,
                               3, 30,};
```

14.14 After the reset action is triggered, the function of the previous mesh network can be restored

The above 1 ~3 mode is the normal mode for restoring factory settings.

After trigger factory reset, if it does not re-configure the network within a certain period of time (such as 30 seconds), some users may hope to automatically restore the previous network information. SDK provides 2 APIs needed to implement this function:

- `mesh_reset_network (u8 provision_enable)`: Restore the network information in the ram to the default network state. Since the parameter `provision_enable` is 1, the device will send unprovision beacon and `pb_adv`, and the device can be reconfigured. At this time, only the RAM data is restored, but the flash information has not changed.
- `mesh_revert_network ()`: Reload network information from flash.

Implementation: When the user triggers the factory reset action and enters the `kick_out` function, it directly calls `mesh_reset_network (1)` to restore the network information in the ram to the default network, without calling `factory_reset ()` and `start_reboot ()`. If the user wants to restore the network, directly call `mesh_revert_network ()` to reload the mesh information from the flash.

14.2 Gateway Node + firmware

To reset the gateway, you need to perform two actions: one is to delete the `mesh_database.json`, and the other is to clear the parameter area of the flash of the gateway dongle (for example, by resetting the device 5 times). Because there are two actions that need to be performed, in order to facilitate the operation, a "GATE_RESET" button is added to the firmware. After executing this button, the above two things will be performed. After the gateway dongle clears its own flash parameter area, it will automatically call `start_reboot ()` for a soft restart.

Note: this button just resets the gateway itself and will not send commands to reset other nodes.

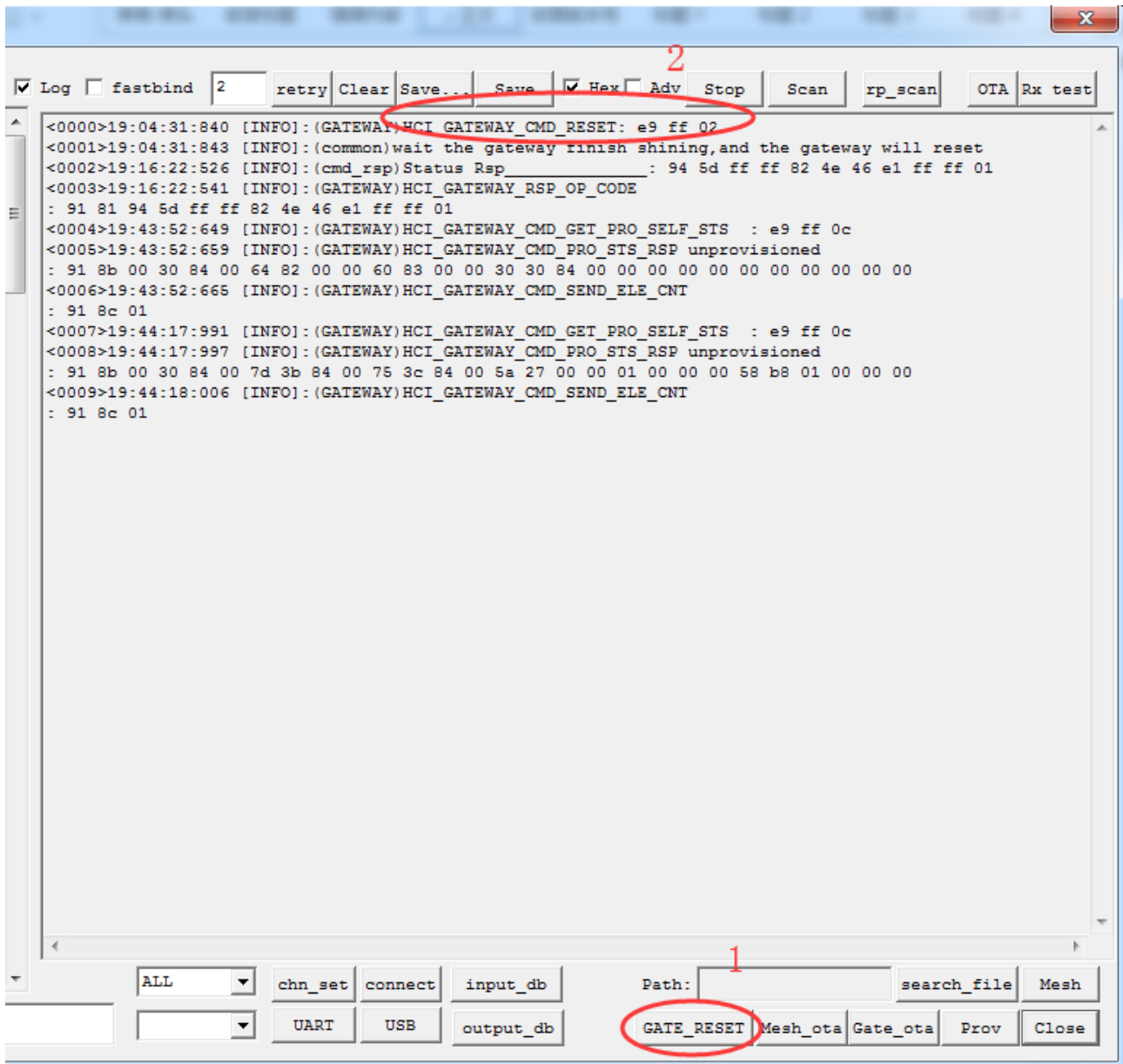


Figure 14.1: gateway reset

14.3 GATT master dongle + Firmware

There is no storage parameter in the flash of master dongle, so just delete the mesh_database.json file of the upper computer, and then reopen the upper computer tool.

14.4 LPN Node

Low-power nodes cannot be reset by powering on 5 times, because when it is in sleep state, it will take some time to consume power after power off. Therefore, LPN nodes generally need to press keys and other methods to trigger and call the functions in the first section of this chapter.

DEMO LPN: long pressed SW1 key (MESH_LPN_FACTORY_RESET_KEY) for more than three seconds (LONG_PRESS_TRIGGER_MS) .it will flash 4 times, indicating that the reset is successful..

14.5 Switch Node

Low-power nodes cannot be reset by 5 power-ups. Need to press keys, press and hold the key combination: RC_KEY_A_ON + RC_KEY_4_OFF for more than three seconds, it will flash 4 times, indicating that the reset is successful.

Telink Semiconductor

15 Fast bind Mode (PROVISION_FLOW_SIMPLE_EN Mode)

15.1 Function Introduction

This mode is a non-standard mode and is used to speed up the provision. The improvements are as follows: After the implementation of the Spec definition provision flow, the unicast address, netkey, and other information are assigned, you need to send get composition data, app key add in order, and perform key bind on each model in the composition data obtained. This is not so complicated, in most cases a device will only have an APPkey. So we defined this Fast bind pattern.

The Fast bind mode is mainly to optimize the key bind part. When the provisioner sends the app key add, it no longer sends the key bind command. After the node receives the app key add, it performs the key bind action on all of its own models. For details, see the code enclosed in the PROVISION_FLOW_SIMPLE_EN macro.

In addition, in order to further simplify provision, the demo app does not need to execute the get composition data command, it directly queries the database to obtain all information of the corresponding composition data through the PID in the device UUID obtained by provision. The following figure shows the function of firmware to write PID to device uuid.

```
void set_dev_uuid_for_simple_flow( u8 *p_uuid)
{
    simple_flow_dev_uuid_t *p_dev_uuid = (simple_flow_dev_uuid_t *)p_uuid;
    memcpy(&p_dev_uuid->cps_head, &gp_page0->head, sizeof(p_dev_uuid->cps_head));
    memcpy(p_dev_uuid->mac, tbl_mac, sizeof(p_dev_uuid->mac));
    // set uuid
}
```

Figure 15.1: Write PID to device uuid

If the customer does not want to obtain the composition data by querying the database, he/she can also modify the flow of the app and add the get composition data command.

15.2 Configuration

Node Firmware: set PROVISION_FLOW_SIMPLE_EN to 1.

15.3 Function Demo

15.3.1 Firmware Configuration

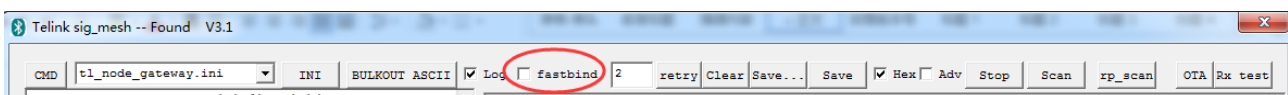


Figure 15.2: Firmware Configuration

15.3.2 APP Interface Configuration

APPs also need to select this mode through controls. For details, please refer to APP instructions.

Telink Semiconductor

16 Private Fast provision Function

16.1 Function Introduction

Remote provision is done node by node, when the network is big, the provision time is still too long, to solve this problem, we provide this private fast provision function, i.e., in the default key network, add vendor commands, e.g., VD_MESH_RESET_NETWORK, and send network key, app key, iv index to target address 0xffff(send only once, and the whole network can receive it), then assign unicast address one by one according to mac. In this way, user can provision nodes within multiple hops. Device keys are generated based on mac address according to a certain rule, no need to assign by mesh commands.

User can also add un-provisioned new devices into an existing mesh network by fast provision way.

16.2 Configuration

Set FAST_PROVISION_ENABLE to 1. Compile 8258_mesh project, download to 3(more than 1)8258 dongles.

GATT master dongle mode and APP support fast provision function by default while gateway does not for now.

16.3 Function Demo

The following demo shows how to add multiple (more than 2) 8258 nodes into mesh network at the same time.

- 1) Open the "sig_mesh_tool.exe" tool and insert the 8269 master Dongle that has been programmed into the USB port of the PC.
- 2) As shown in the figure below, "Found" in the upper left corner of the tool indicates that the 8269 Master Dongle and the PC tool are normally connected and can communicate normally. At this time, the tool will automatically select "sig_mesh_master.ini" according to the connected hardware.

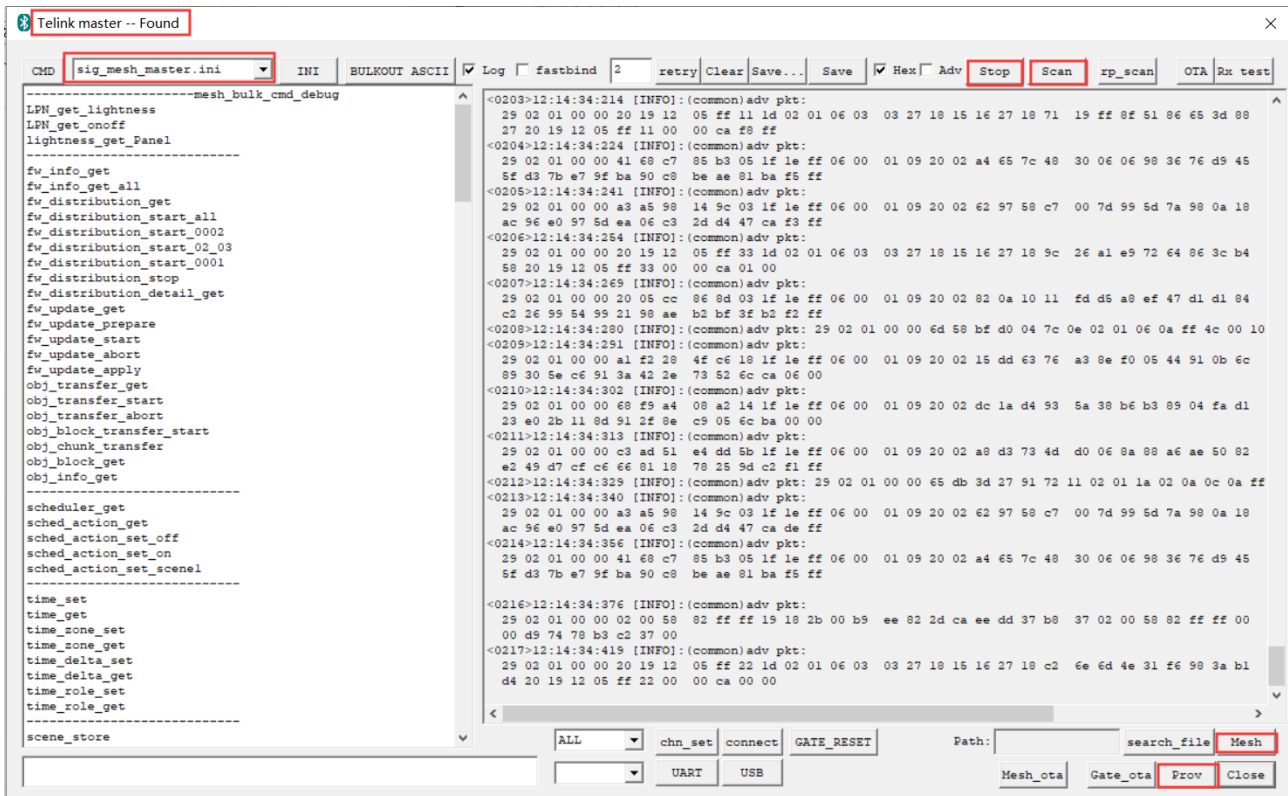


Figure 16.1: Normal connection

- 3) Boot 8258mesh node.
- 4) Click the "Scan" button in the upper right corner, the tool will open a "ScanDev" window, which will display the corresponding MAC address list, including rssi and frequency.

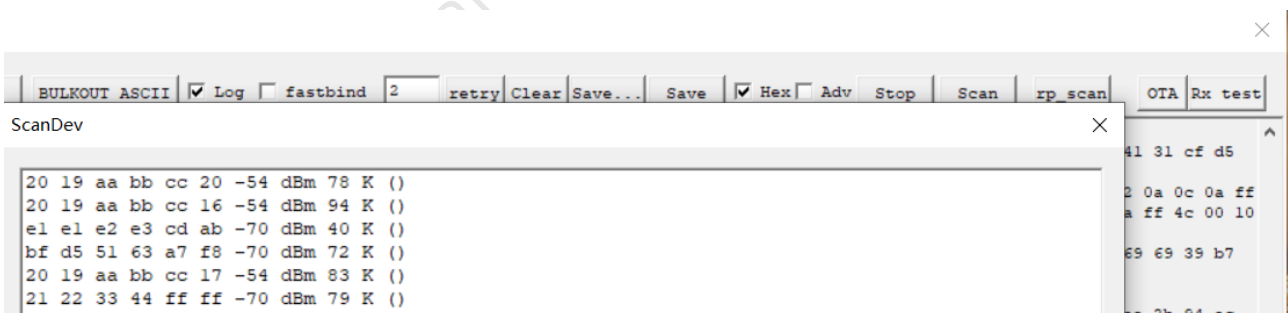


Figure 16.2: ScanDev window

- 5) Double-click the corresponding entry in the ScanDev window and select the node. 8269 Master dongle mode: A BLE connection will be established after double-clicking. If the red light on the 8269 Master dongle lights up, it means that the BLE connection is established normally. The "Stop" button is used to terminate the current BLE connection. The white light on the 8269 Master Dongle lights up, indicating that the BLE connection is disconnected. Currently only supports BLE connection for a single node.
- 6) Click the "Prov" button in the lower right corner to open the "provision" window. Click the 1/2/3 buttons in the order shown below. (Note that you must select Fast prov mode first, you do not need to click the bind_all button).



The screenshot shows the 'provision' window with the following elements:

- Fast prov mode**: A checkbox that is checked, circled in red, and labeled with a red '1'.
- SetPro_internal**: A button circled in red and labeled with a red '2'.
- network_key**: A text field containing the hex value 'b3 12 4d c8 43 bb 8b a6 1f 03 5a 7d 09 38 25 1f'.
- Static apk_idx**: A text field containing '00 00'.
- app_key**: A text field containing the hex value '60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48'.
- bind_all**: A button.
- key_index**: A text field containing '00 00'.
- iv_index**: A text field containing '11 22 33 44'.
- iv_update_flag**: A text field containing '0'.
- unicast_adr**: A text field containing '0d 00'.
- Provision**: A button circled in red and labeled with a red '3'.
- filter_operation**: A section containing:
 - filter_type**: A dropdown menu set to 'white_list'.
 - filter_data**: A text field containing '01 00 ff ff'.
 - SetFilter**: A button.
 - Add_mac**: A button.
 - RM_mac**: A button.

Figure 16.3: [Click in order](#)

After a few seconds, you will see that all nodes are flashing 3 times, indicating that they have been successfully provisioned. Click the “Mesh” button to enter the mesh interface to perform operations such as switching lights.

17 Private online status function demo

17.1 Function Introduction

Currently, the online and offline monitoring mechanism provided by the SIG mesh spec can be implemented through the heartbeat and publish mechanisms. The real-time monitoring of status such as on/off of nodes can be implemented through the publish mechanism.

When both online and offline monitoring and real-time monitoring of status are required, a publish mechanism is required. However, the publish mechanism has the following limitations:

- Publish messages generally need to set relay, so there will be more packets on air.
- The publish period cannot be set too short (it usually takes tens of seconds or longer), otherwise there will be too many packets on air, affecting normal control.
- Sometimes it takes several publish status messages to include all the statuses that need to be reported. At this time, there will be more packets on air.

Therefore, we have added an online status mechanism, the purpose of which is to achieve fast and effective online and offline detection, and to report important status of nodes, while also effectively reducing data packets in the network.

17.2 Configuration

Set the `ONLINE_STATUS_EN` of `app_config.h` on the light node side from 0 to 1.

GATT master dongle mode and APP support online status function by default while gateway does not for now.

17.3 Packet Format

The online status data packet is sent by adv of ADV_NON_CONN_IND, and the type of the payload is customized `MESH_ADV_TYPE_ONLINE_ST` (0x62), as shown in the figure below.

Time (us)	Channel	Access Address	Adv PDU Type	Adv PDU Header			AdvA	AdvData	CRC	RSSI (dBm)	FCS
+319991 =1599997	0x25	0x8E89BED6	ADV_NON_CONN_IND	Type	TxAdd	RxAdd	PDU-Length	1E 62 0F 07 31 56 F2 03 47 DA 76 D7 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 91 9F AB 4A	0x000080	-38	OK
				2	0	0	37				
								0xFFFFF82580002			
+360010 =1960007	0x25	0x8E89BED6	ADV_NON_CONN_IND	Type	TxAdd	RxAdd	PDU-Length	1E 62 0F 07 31 57 F2 03 47 DA 76 D5 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 7E 1D 3E 25	0x000076	-38	OK
				2	0	0	37				
								0xFFFFF82580002			
+319994 =2280001	0x25	0x8E89BED6	ADV_NON_CONN_IND	Type	TxAdd	RxAdd	PDU-Length	1E 62 0F 07 31 54 F2 03 47 DA 76 D4 23 28 CC 80 64 13 B8 41 57 93 BA 1C 6B 9E DD 0C C1 5E 66	0x000084	-38	OK
				2	0	0	37				
								0xFFFFF82580002			

Figure 17.1: Packet format

The effective payload length is 24 bytes. By default, each node requires 6 bytes. For details, please refer to

```
typedef struct{
    u16 dev_adr :15;      // don't change include type
    u16 rsv      :1;
    u8  sn;         // don't change include type
    u8  par[MESH_NODE_ST_PAR_LEN]; //lumen-rsv,
}mesh_node_st_val_t;
```

Figure 17.2: reference details

The effective length of each node is MESH_NODE_ST_PAR_LEN (3). The specific data filled can be customized according to each product. The default fill BYTE 0 is brightness, BYTE 1 is the color temperature value, and BYTE 2 is reserved. Modify device_status_update () according to customer needs.

```
void device_status_update()
{
    // packet
    u8 st_val_par[MESH_NODE_ST_PAR_LEN] = {0};
    memset(st_val_par, 0xFF, sizeof(st_val_par));
    // led_lum should not be 0, because app will take it to be light off
    st_val_par[0] = light_lum_get(0, 1);
    #if (LIGHT_TYPE_CT_EN)
    st_val_par[1] = light_ct_lum_get(0, 1);
    #else
    st_val_par[1] = 0xff;    // rsv
    #endif
    // end

    ll_device_status_update(st_val_par, sizeof(st_val_par));
}
```

Figure 17.3: device_status_update

Note: MESH_NODE_ST_PAR_LEN (3), can be modified, but the larger the length, the slower the transmission speed. And there is no length field to describe this length, so when defining the network, you must first determine the value of MESH_NODE_ST_PAR_LEN. If the MESH_NODE_ST_PAR_LEN values configured by nodes in the network are inconsistent, there will be compatibility issues, resulting in data format parsing errors.

174 GATT Master Dongle Firmware Demo

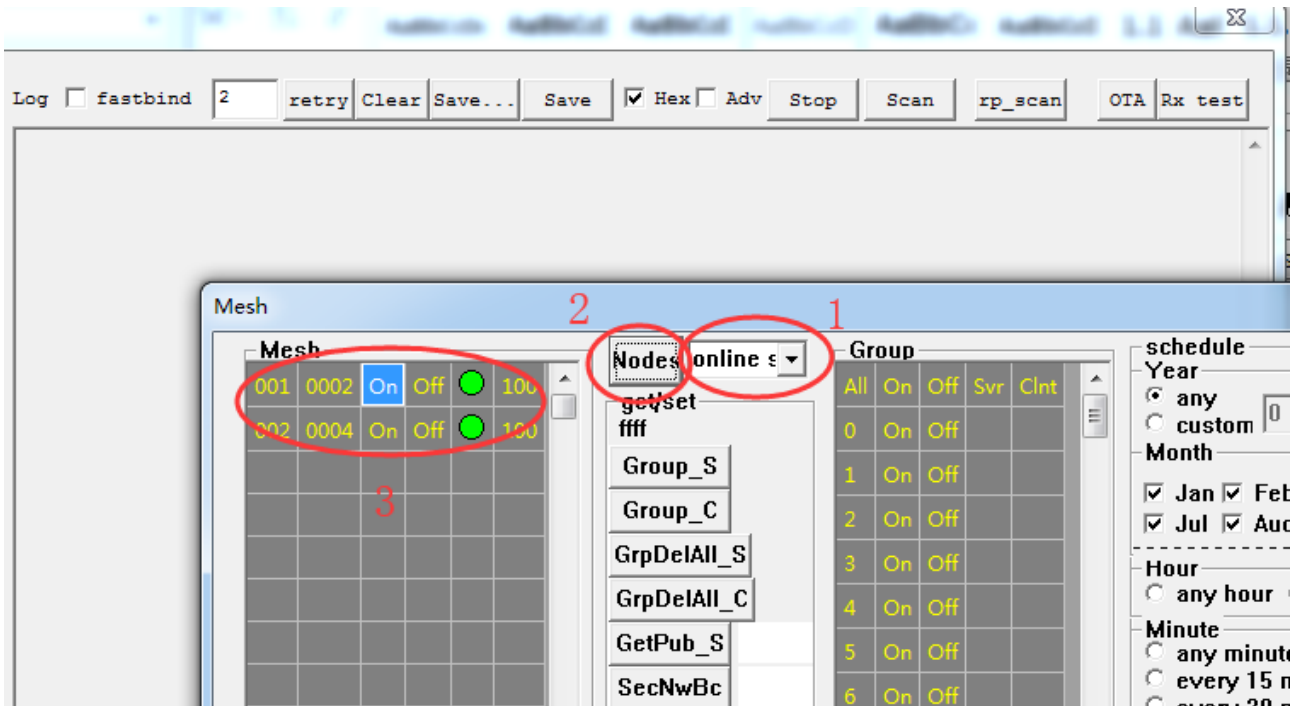


Figure 17.4: GATT master dongle firmware demo

Step 1 Choose online status as shown in figure above.

Step 2 Click Node button as shown in figure above. After clicking, you can see from the log window that you did not send similar commands such as lightness get. Instead, you can directly obtain the online status data in the directly connected node through the custom UUID.

Step 3 After clicking the Node button, the node display window on the left shows the node information of the current network.

18 OTA Test Brief

18.1 GATT master dongle OTA for firmware update of BLE directly connected nodes

This mode is a point-to-point BLE Direct Connect OTA.

- 1) Download the BIN file (New FW) that requires OTA to the flash address of 8269 master dongle starting from 0x20000 according to the instructions in 7.1, burn 8269_mesh_master_dongle.bin from address 0x0000.

Note: The difference of burning New FW:

BDT tool, please refer to the following steps:

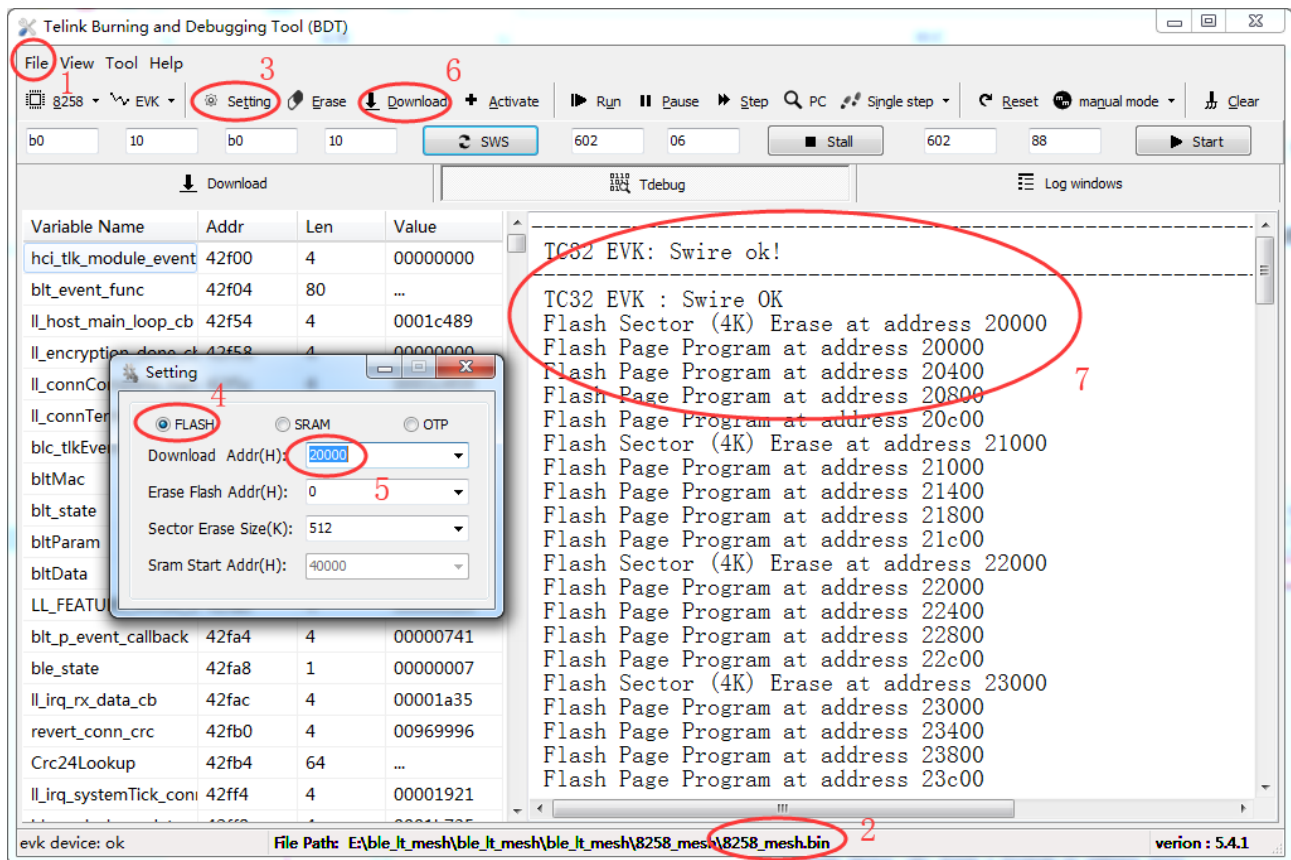


Figure 18.1: BDT tool

For wtcdB tool, click the “WF20000” button to start burning. Please refer to the following steps:

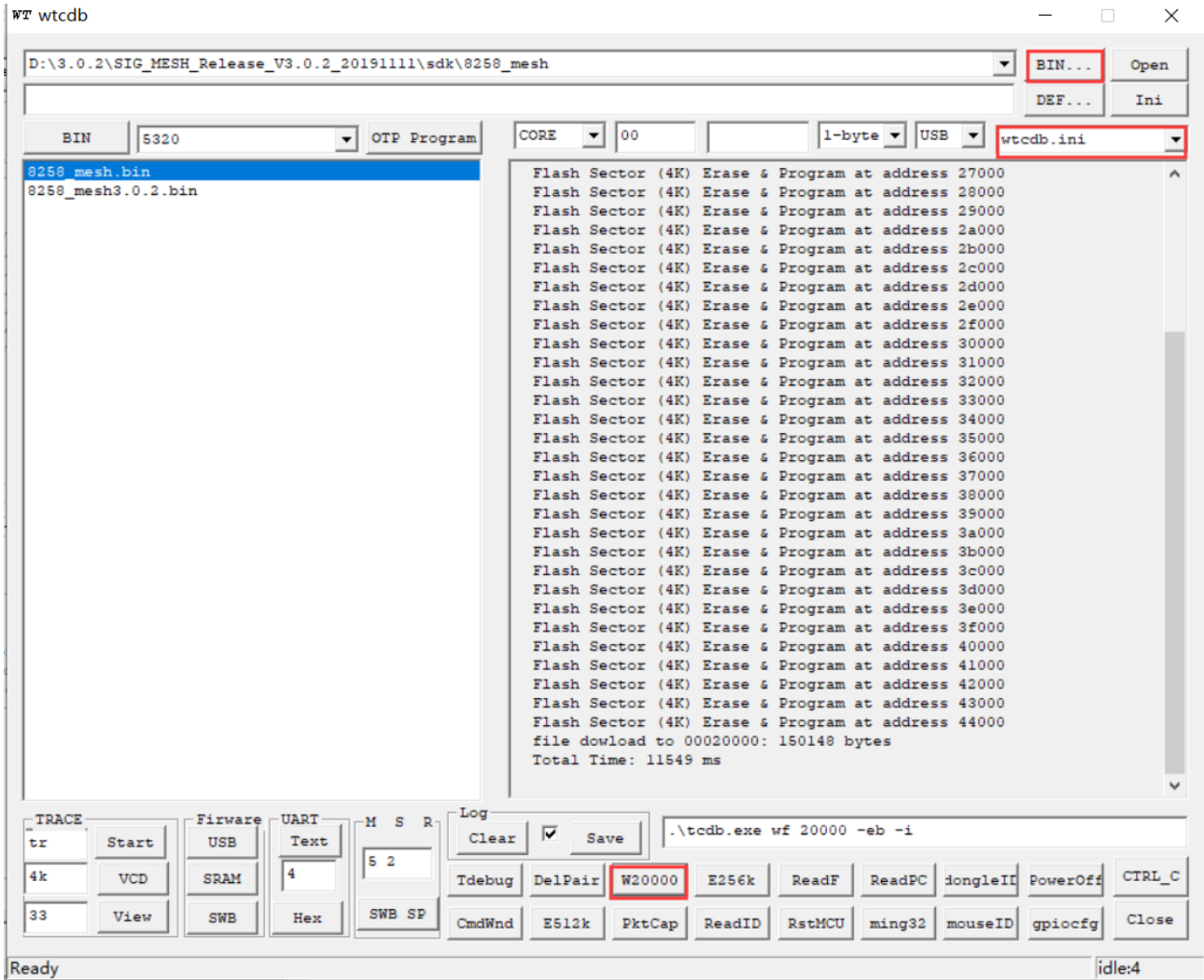


Figure 18.2: wtcdB tool

- 2) Refer to the steps (1) to (5) in Section 7.2 to establish a BLE connection between the target node and the tool.
- 3) Click the OTA button to start the OTA process. If the OTA is completed normally, the node will flash 8 times continuously.

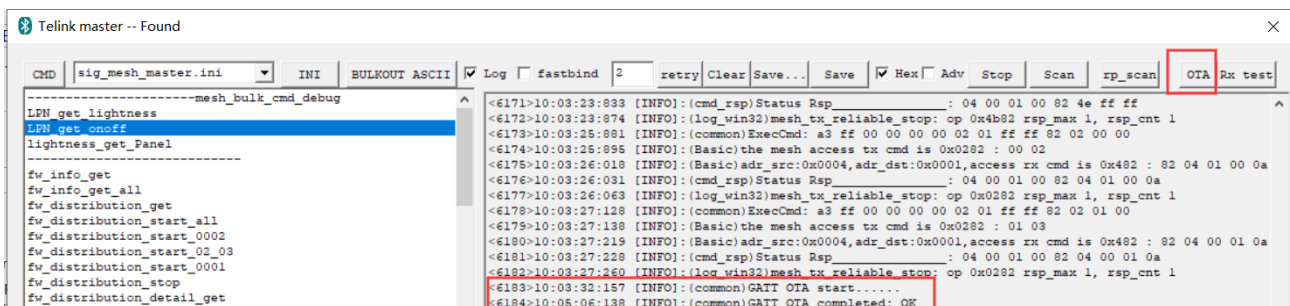


Figure 18.3: Start OTA

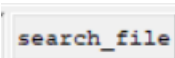
- 4) For the commands of the OTA part and the details of the protocol part, please refer to Chapter 6.4

of “AN_17092701_Telink 826x BLE SDK Developer Handbook”. There is a detailed description of the command and protocol format.

18.2 OTA OTA where the GATE WAY node updates its firmware

The purpose of Gateway Gate OTA is to upgrade the firmware of gateway itself.

- 1) Open the BDT tool and download 8258_mesh_gw.bin to 8258 dongle.
- 2) “Found” in the upper left corner of the tool means that the 8258 Dongle and the PC tool are normally connected and can communicate normally.

- 3) Click the  button in the lower right corner and select a different version of the 8258_mesh_gw.bin file;

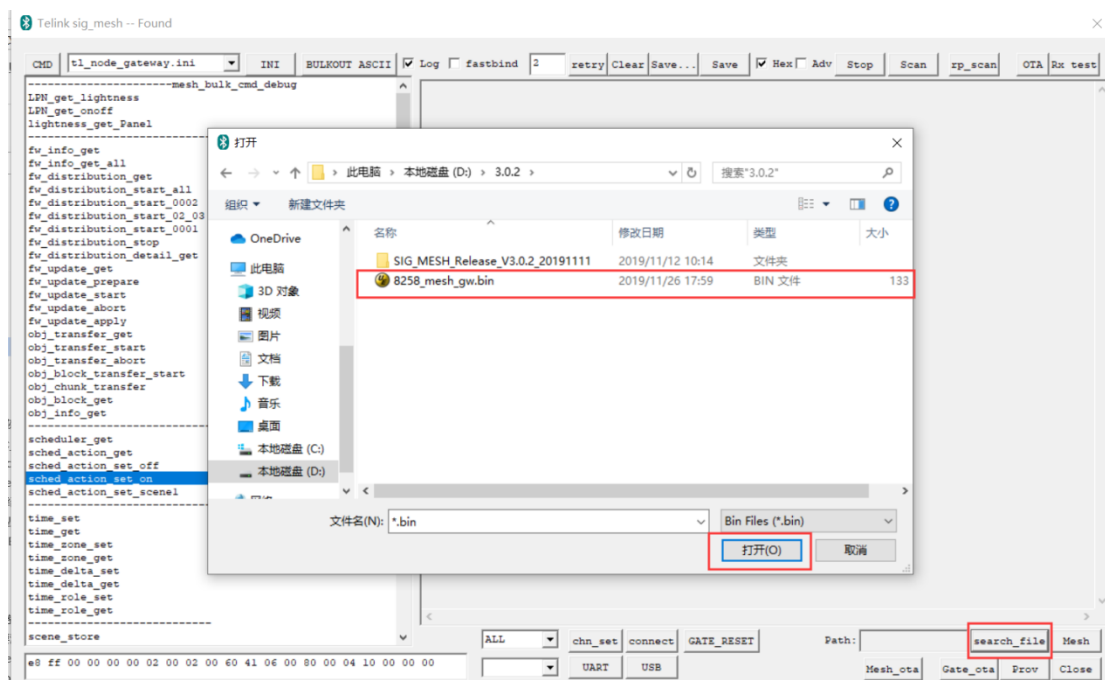


Figure 18.4: Select bin file

- 4) Click the Gate_ota button to start the upgrade. LOG prompts gateway firmware load suc to indicate that the upgrade was successful. After the upgrade is successful, the gateway will automatically restart and enable the new firmware.

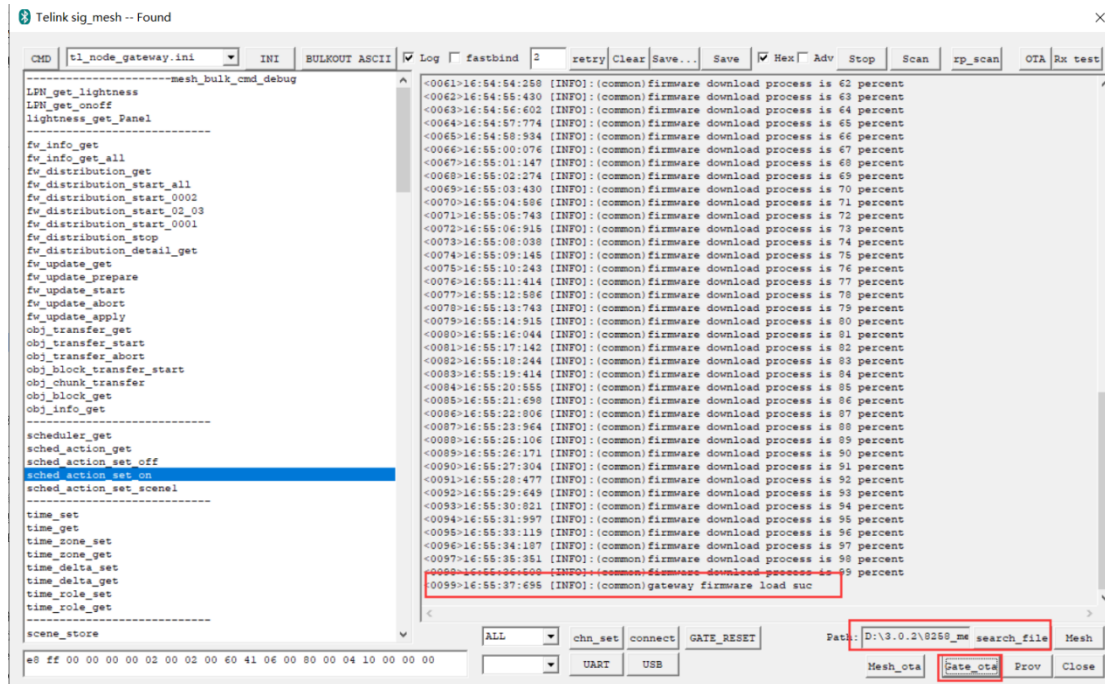


Figure 18.5: Click Gate_ota

19 Network Sharing

19.1 APP Provision by Gateway or GATT master dongle, then share with APP

Step 1 Provision with VC master or gateway, make sure 8258 dongle node can be provisioned and controlled normally

Step 2 Find the generated files in sig_mesh_tool.exe folder

Step 3 Import mesh.json into TelinkSigmesh APP;

IOS APP Steps:

Step 1 Connect your phone to a computer with iTunes installed.

Step 2 Click the phone icon in the upper left corner of iTunes to enter the iTunes device details interface.

Step 3 Select "File Sharing" on the left side of iTunes, then find and click the demo APP "TelinkSigMesh" in the app, and wait for iTunes to load the file.

Step 4 After the file is loaded, drag the json file on your computer into the "TelinkSigMesh" document on the right

Step 5 Click the IMPORT button in the APP to select the JSON file to load. Detailed steps are in the pictures below:

a) Open APP TelinkSig mesh, click Setting button.

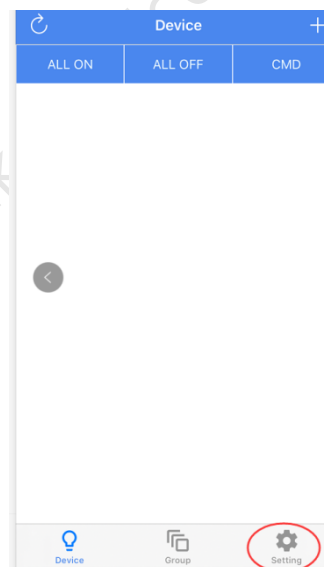


Figure 19.1: Click Setting



b) Click Share button

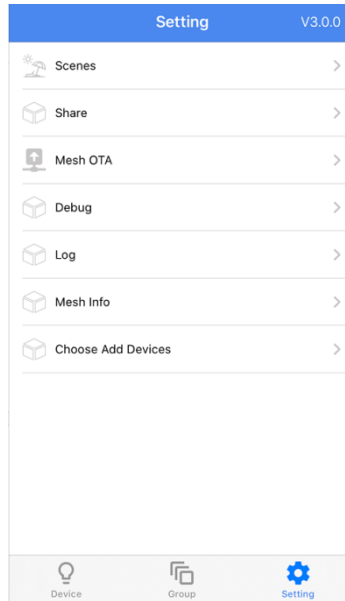


Figure 19.2: Click Share

c) Click IMPORT button



Figure 19.3: Click IMPORT

d) Click mesh.json, click IMPORT

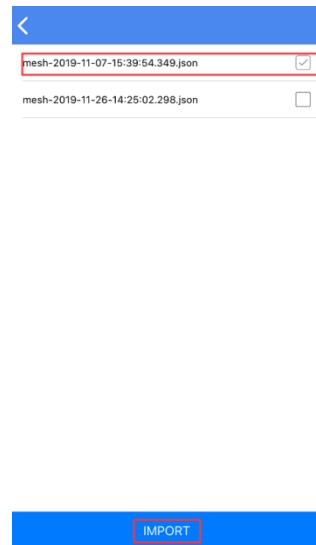


Figure 19.4: Click mesh.json


Import file with Android APP:

Step 1 Connect the phone to computer, import the mesh.json file into any folder on the phone, and remember the path to the folder.

- a) Open APP TelinkSig mesh, click Setting button.



Figure 19.5: Click Setting

- b) Click Share button  Share

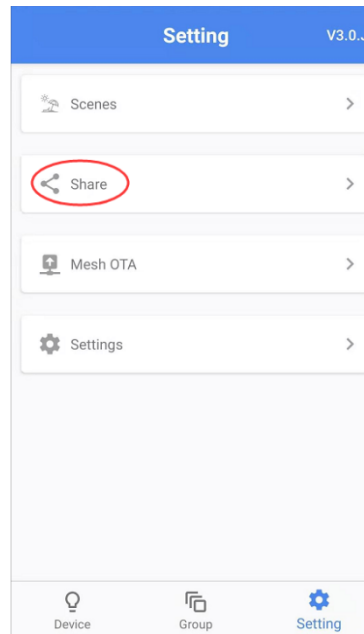


Figure 19.6: Click Share

c) Click IMPORT button

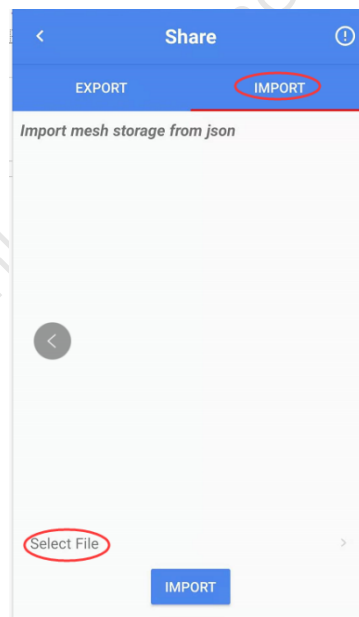


Figure 19.7: Click IMPORT

d) Click Select File to select mesh.json imported from the computer

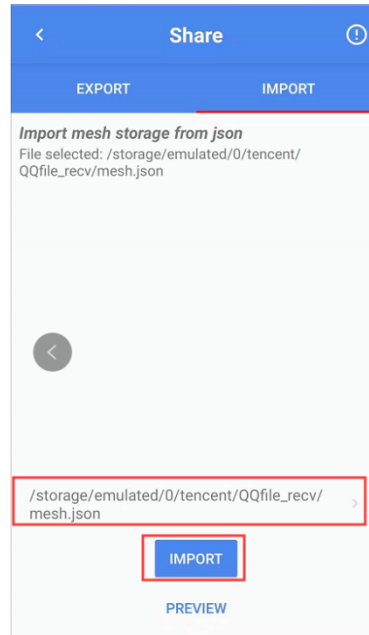


Figure 19.8: Click mesh.json

e) Click Import button



Step 2 After the import is successful, return to the main page: APP Device interface. At this time, the shared nodes will be displayed. These nodes are the nodes of the VC tool network. APP can be controlled, and both VC and APP can control the nodes.

19.2 Provision by APP mode, then share with Gateway or GATT master dongle

- 1) Provision with ios or android TelinkSIGmesh APP, and works normally.
- 2) Click Setting button, then click Share button to enter share interface, click EXPORT button, generate JSON file. The path of JSON file folder will be displayed on the interface.

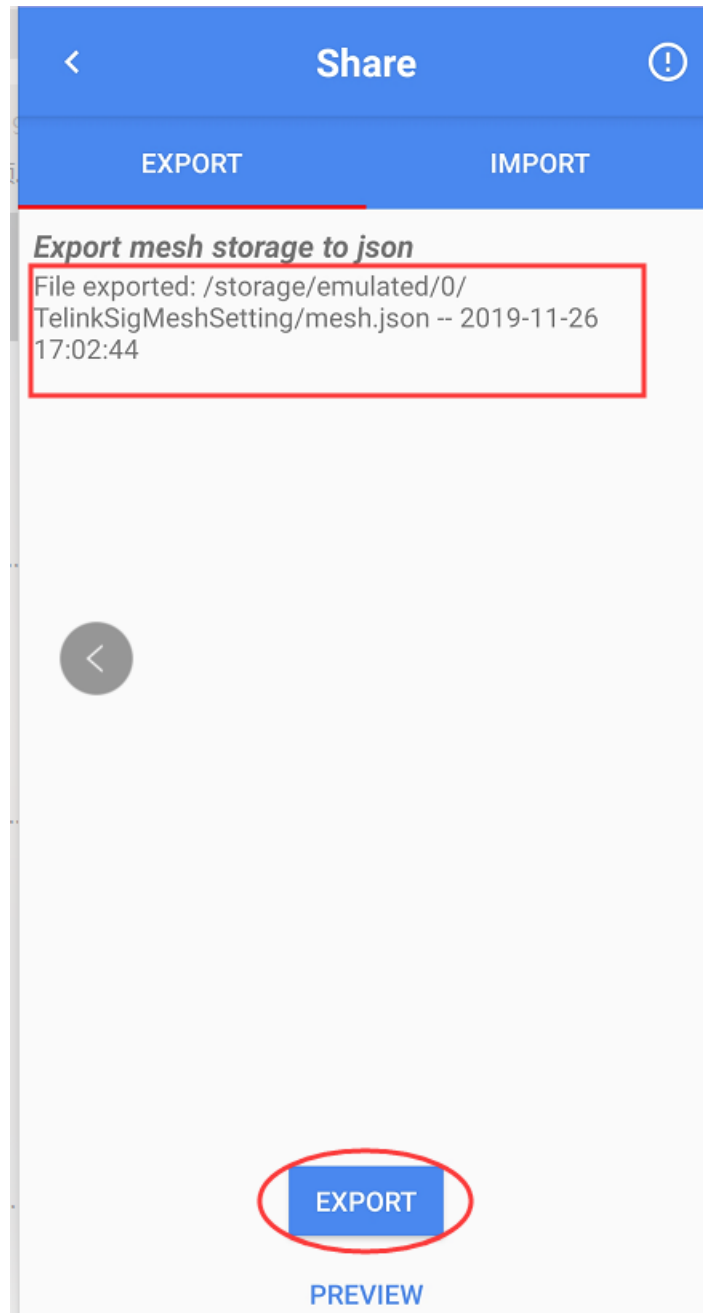


Figure 19.9: JSON file path

- 3) Copy the generated mesh.json in APP to SIG_MESH_TOOL folder in computer, if the folder is already existed, replace it.

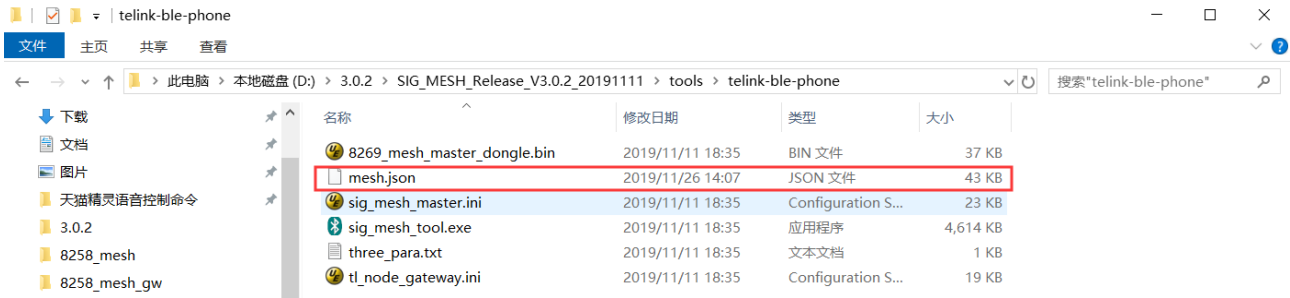


Figure 19.10: mesh.json file

- 4) Make sure master 8269 Dongle or gateway dongle is already connected with PC.
- 5) Open SIG_MESH_TOOL, now it shows that master 8269 Dongle or gateway 8258 Dongle is connected with PC tool.

Normally, Gateway dongle is not provisioned, if it is, all the information will be deleted, and replaced by imported data.

- 6) Click mesh button, the imported nodes will show in the mesh window, and can be controlled.
- 7) Not the sharing is completed, gateway/master dongle and APP can all control the nodes.

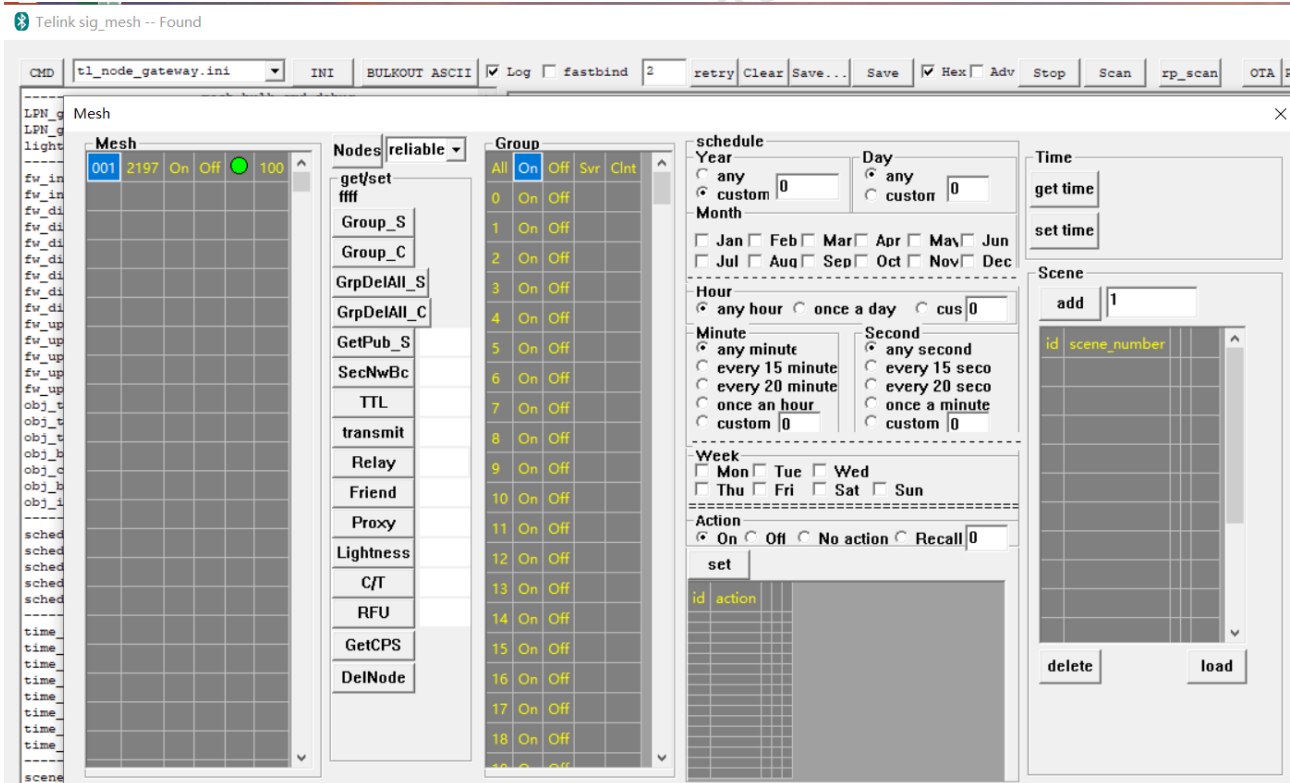


Figure 19.11: Complete sharing

20 Control Nodes via INI Demo

20.1 Provision Device

Device provision is slightly different for PB-GATT and PB-ADV, but it is the same for interface and operation procedure.

Step 1 Scan UNprovision_beacon adv devices;

Step 2 Connect according to MAC of the scanned UNprovision adv devices;

Step 3 Provision device;

Step 4 Bind model.

The following demo is to test with master dongle. Click stop, then click scan to enter scan mode, connect the scanned device, with the mac of: 112233445566.

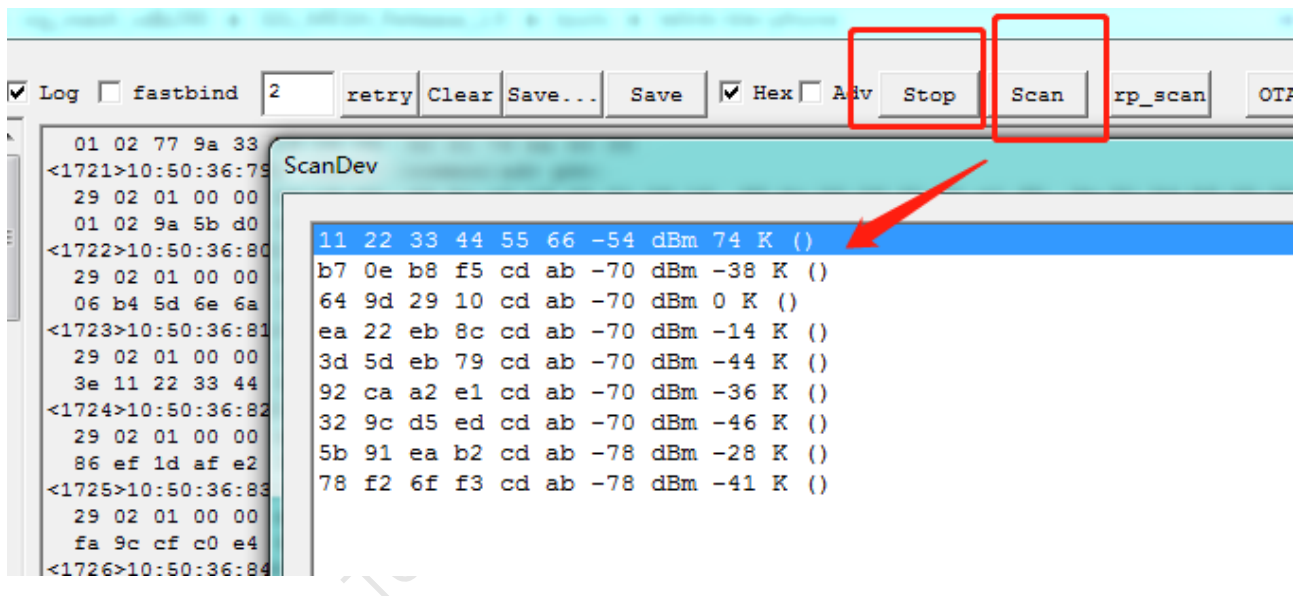


Figure 20.1: Connect device

Log information after the device is connected:

```

29 02 01 00 00 e4 4a d8 a7 6c 7f 1f 1e ff 06 00 01 09 20 02 23 73 90 d6 fc c3 94 59 c5 fa 87 08
86 ef 1d af e2 85 c0 41 6b 83 d8 ca 14 00
<11788>10:53:00:253 [INFO]:(common)adv pkt: 29 02 01 00 00 26 29 c0 89 d2 7b 0e 02 01 1a 0a ff 4c 00 1
<11789>10:53:00:266 <11790>10:53:00:396 [INFO]:(gatt_provision)CScanDlg::OnConnect:the device uuid is
: 91 2f 68 1d 5c 48 fb 3d b6 3e 11 22 33 44 55 66
[INFO]:(common)adv pkt:
29 02 01 00 00 11 22 33 44 55 66 1d 02 01 06 03 03 27 18 15 16 27 18 91 2f 68 1d 5c 48 fb 3d b6
3e 11 22 33 44 55 66 00 00 ca 4a 00
<11791>10:53:01:118 [INFO]:(common)Mesh Provisioning Service:
<11792>10:53:01:132 [INFO]:(common)uuid:dc 2a
<11793>10:53:01:144 [INFO]:(common)the handle:13
<11794>10:53:01:160 [INFO]:(common)Mesh Proxy Service:
<11795>10:53:01:177 [INFO]:(common)uuid:de 2a
<11796>10:53:01:191 [INFO]:(common)the handle:1c
<11797>10:53:01:201 [INFO]:(Basic)filter send cmd is 0: 00
<11798>10:53:01:222 [INFO]:(Basic)filter send cmd is 1: 4b 7e
<11799>10:53:01:242 [INFO]:(Basic)filter send cmd is 1: ff ff
<11800>10:53:01:264 [INFO]:(iv_update)app tx beacon with GATT,IV index step0: : 12 34 56 78 12 34 56 7
<11801>10:53:01:279 [INFO]:(iv_update)secure NW beacon:: 17 2b 01 00 44 cf 7a d5 44 8c f1 6e 12 34 56
<11802>10:53:01:359 [INFO]:(Basic)the filter rsp is 0: 00 00 46 6f
<11803>10:53:01:373 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11804>10:53:01:388 [INFO]:(log_win32) white list
<11805>10:53:01:405 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 0
<11806>10:53:01:422 [INFO]:(Basic)the filter rsp is 0: 00 01 74 0e
<11807>10:53:01:432 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11808>10:53:01:444 [INFO]:(log_win32) white list
<11809>10:53:01:456 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 1
<11810>10:53:01:467 [INFO]:(Basic)the filter rsp is 0: 00 02 85 01
<11811>10:53:01:479 [INFO]:(Basic)mesh_rc_data_cfg_gatt dec suc
<11812>10:53:01:494 [INFO]:(log_win32) white list
<11813>10:53:01:507 [INFO]:(log_win32)GATT addr 0x2211, filter list status, ListSize is: 2

```

Figure 20.2: log info

Provision Parameter Setting and Device Provision

- 1) Click prov button to enter provision interface, first click SetPro_internal to assign net key to dongle.
- 2) Click provision button to provision the connected device with MAC of 112233445566, during provision, assign netkeyindex, IVindex, unicast_addr generated by firmware to device.
- 3) Click bind_all button to bind APPkey(based on the model reported by devicecomposition data).

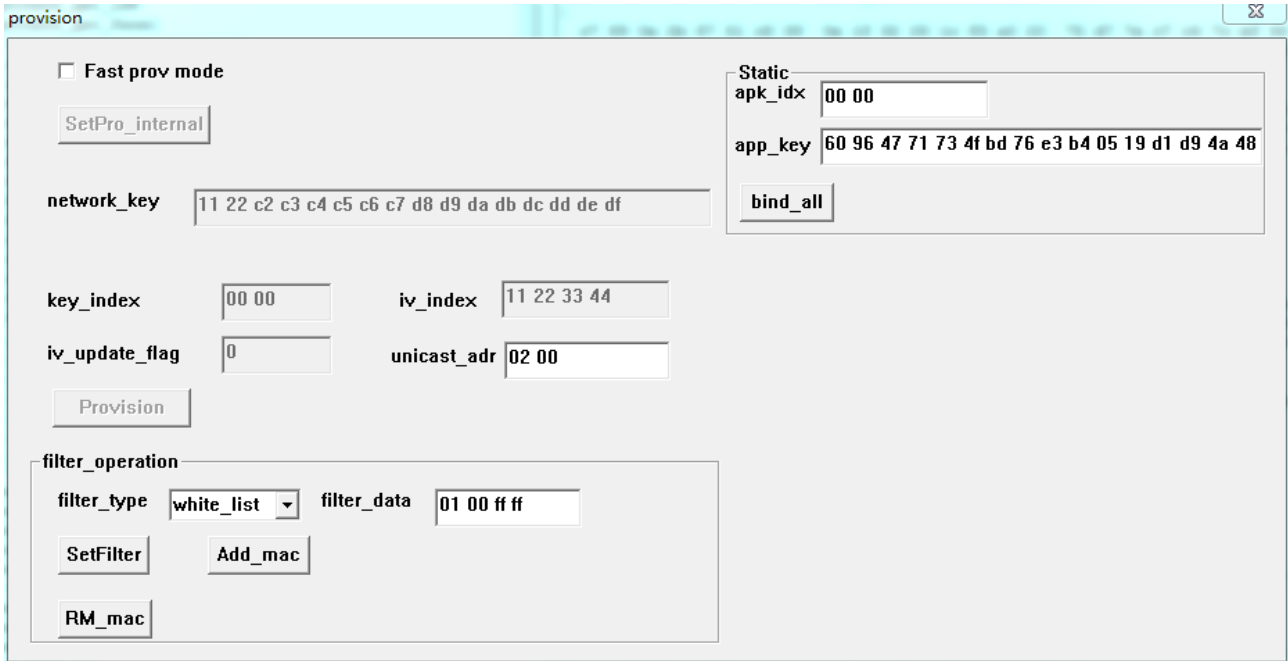


Figure 20.3: Provision parameter setting and device provision

As shown in figure below, the data interaction of Provision is described as following:

- 1) Start provision, provisioner send out data
- 2) public key interaction
- 3) check confirm;
- 4) send provision data (net_key/nkey_index/IV_update_flag/IV_index/unicast_addr);
- 5) Add proxy white list

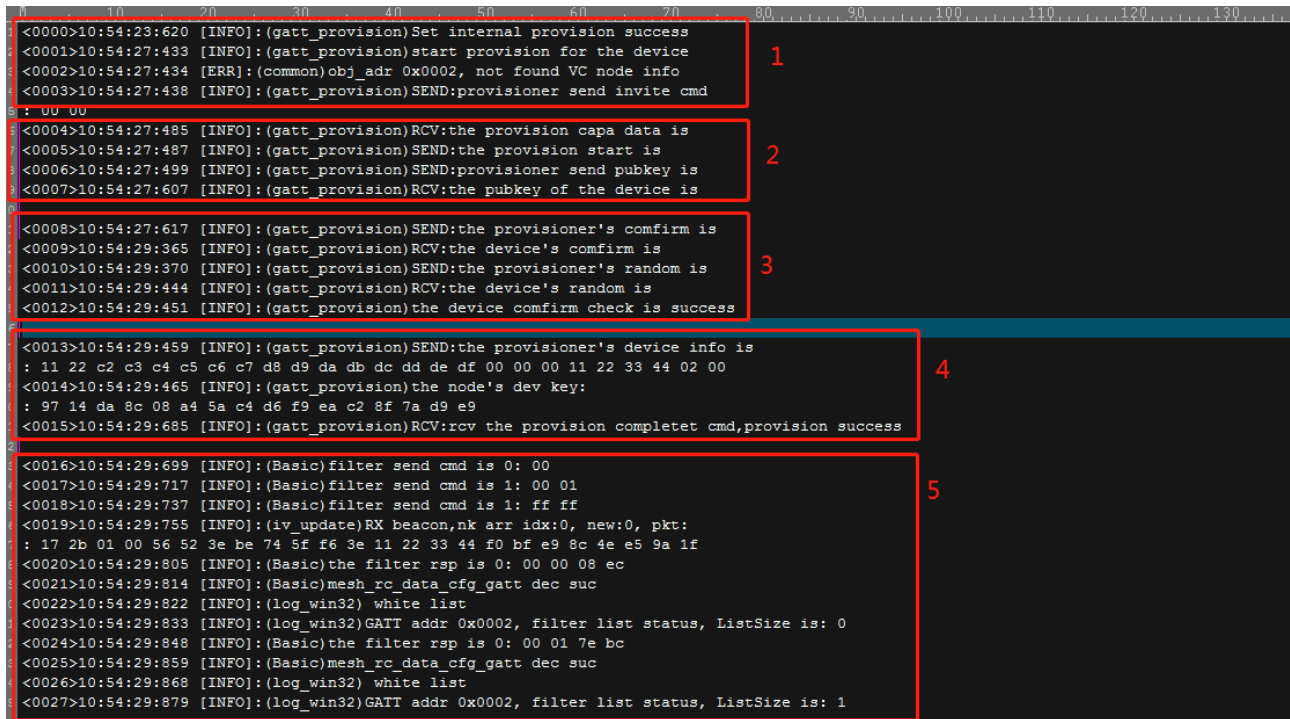


Figure 20.4: The data interaction of Provision

Bind introduction:

Bind is to bind APP key with all device models according to the composition data of the device after the provision. This process will determine the bind time according to the number of models (only 20s). Therefore, bind has been optimized on Tmall Genies and other platforms, and fastbind will be introduced next.

Before Bind, you need to get the composition data of the device first. For detailed analysis format of Composition data, please refer to <4.2.1 Composition Data> of <Mesh_v1.0>.

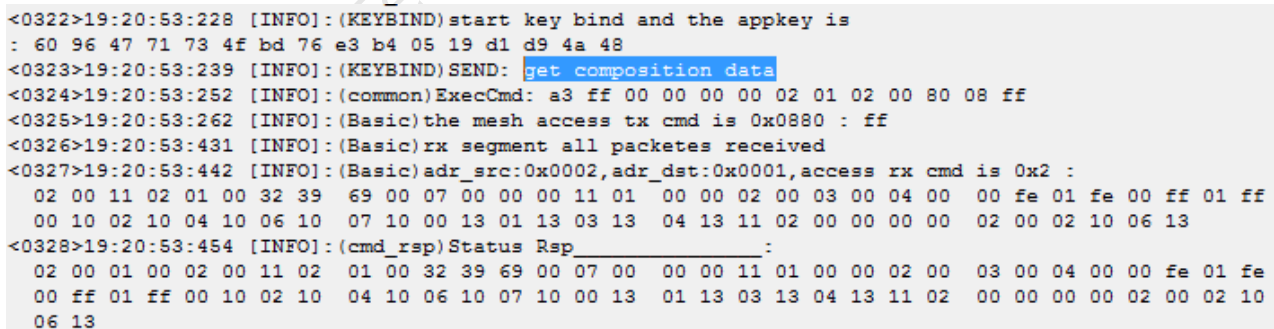


Figure 20.5: Get device composition data

According to the corresponding information of the obtained element and model, the bind command is subsequently issued to bind model by model.

```

<0330>19:20:53:490 [INFO]: (KEYBIND)SEND: appkey add ,0x0 is the appkey index
: 60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48
<0331>19:20:53:502 [INFO]: (common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 00 00 00 00 60 96 47 71 73 4f
<0332>19:20:53:514 [INFO]: (Basic)the mesh access tx cmd is 0x0000 : 00 00 00 60 96 47 71 73 4f bd 76 e
<0333>19:20:53:872 [INFO]: (Basic)rc data layer upper:segment tx success, map in ACK is: 03 00 00 00
<0334>19:20:53:883 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x380 : 80 03 00 00 00
<0335>19:20:53:893 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 03 00 00 00 00
<0336>19:20:53:907 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x0000 rsp_max 1, rsp_cnt 1
<0337>19:20:53:919 [INFO]: (KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0002
<0338>19:20:53:932 [INFO]: (common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 02 00
<0339>19:20:53:945 [INFO]: (Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 02 00
<0340>19:20:54:032 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0341>19:20:54:044 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 3e 00 02 00 00 00 02 00
<0342>19:20:54:063 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0343>19:20:54:073 [INFO]: (KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0003
<0344>19:20:54:085 [INFO]: (common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 03 00
<0345>19:20:54:096 [INFO]: (Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 03 00
<0346>19:20:54:192 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0347>19:20:54:202 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 3e 00 02 00 00 00 03 00
<0348>19:20:54:219 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0349>19:20:54:231 [INFO]: (KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0x0004
<0350>19:20:54:245 [INFO]: (common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 04 00
<0351>19:20:54:258 [INFO]: (Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 04 00
<0352>19:20:54:352 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0353>19:20:54:368 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 3e 00 02 00 00 00 04 00
<0354>19:20:54:384 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0355>19:20:54:396 [INFO]: (KEYBIND)SEND: appkey bind addr: 0x0002,sig model id: 0xfe00
<0356>19:20:54:407 [INFO]: (common)ExecCmd: a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 00 fe
<0357>19:20:54:418 [INFO]: (Basic)the mesh access tx cmd is 0x3d80 : 02 00 00 00 00 fe
<0358>19:20:54:512 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 02 0
<0359>19:20:54:527 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 3e 00 02 00 00 00 00 fe
<0360>19:20:54:547 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1

```

Figure 20.6: Bind

```

<0448>19:20:57:152 [INFO]: (Basic)adr_src:0x0002,adr_dst:0x0001,access rx cmd is 0x3e80 : 80 3e 00 03 0
<0449>19:20:57:166 [INFO]: (cmd_rsp)Status Rsp : 02 00 01 00 80 3e 00 03 00 00 00 06 13
<0450>19:20:57:187 [INFO]: (log_win32)mesh_tx_reliable_stop: op 0x3d80 rsp_max 1, rsp_cnt 1
<0451>19:20:57:218 [INFO]: (KEYBIND)SEND: mesh keybind event success

```

Figure 20.7: Bind

20.2 Configuration Operations

20.2.1 Key add /bind Operation

APPKey add Command format analysis:

CMD-cfg_appkey_add_001= a3 ff 00 00 00 00 02 00 07 00 00 00 00 00 60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48

1	2	3	4	5	7	8	9	10	11	12		
Flag	nk_idx	ak_idx	reliable retry count	reliable rsp_max	dst	op	para0	para1	para2	para3	para4	para5
a3	ff	0000	0000	02	00	0700	00	AppKeyIndex			app key	
											60 96 47 71 73 4f bd 76 e3 b4 05 19 d1 d9 4a 48	

Figure 20.8: APPKey add command

Command format analysis:

- Flag: Defined by Telink, to identify the header packet of USB or UART communication, UART is E8FF, USB is A3FF.
- NK_idx: network key index
- Ak_idx: APP key index
- Reliable retry cnt: Application layer retry (the number of resends if the firmware cannot receive a reply after issuing a command)
- Reliable resp_max: set the number of nodes to reply
- Dst: destination address filling
- Op: standard command code defined by sig mesh specification, please refer to <Mesh_v1.0>, if the command code is not fixed, please refer to <4.3.4 Messages summary> in the document.

Please refer to < 4.3.2.37 Config AppKey Add > in < Mesh_v1.0> for filling data.

Field	Size (octets)	Notes
NetKeyIndexAndAppKeyIndex	3	Index of the NetKey and index of the AppKey
AppKey	16	AppKey value

Figure 20.9: Filling data

Key bind Command format analysis:

CMD-cfg_appkey_bind_001 = a3 ff 00 00 00 00 02 00 02 00 80 3d 02 00 00 00 00 10

Command format analysis:

- Flag: Defined by Telink, to identify the header packet of USB or UART communication, UART is E8FF, USB is A3FF.
- NK_idx: network key index
- Ak_idx: APP key index
- Reliable retry cnt: Application layer retry (the number of resends if the firmware cannot receive a reply after issuing a command)
- Reliable resp_max: set the number of nodes to reply
- Dst: destination address filling
- Op: standard command code defined by sig mesh specification, please refer to <Mesh_v1.0>, if the command code is not fixed, please refer to <4.3.4 Messages summary> in the document.

9:12: Transmission parameter part, light HSL control command filling data please refer to< 4.3.2.46 Config Model App Bind >in < Mesh_v1.0>

Transmission parameter format reference:

Field	Size (octets)	Notes
ElementAddress	2	Address of the element
AppKeyIndex	2	Index of the AppKey
ModelIdentifier	2 or 4	SIG Model ID or Vendor Model ID

Figure 20.10: Transmission parameter format reference

20.2.2 Subscription Configuration

CMD-cfg_sub_add = a3 ff 00 00 00 00 00 01 02 00 80 1b 02 00 01 c0 00 10

Or please refer to group index control in 4.4.2.

20.2.3 publish configuration

CMD-cfg_pub_set_sig_2s = a3 ff 00 00 00 00 00 02 00 03 02 00 01 00 00 00 ff 14 15 00 10

Or please refer to GetPub_S control button in 4.4.3

20.2.4 Relay/Friend Function Configuration

Relay: a3 ff 00 00 00 00 02 01 07 00 80 27 01

```
<0000>11:44:52:572 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 27 01
<0001>11:44:52:573 [INFO]:(Basic)the mesh access tx cmd is 0x2780 : 01
<0002>11:44:52:679 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x2880 : 80 28 01 a0
<0003>11:44:52:681 [INFO]:(cmd_rsp)Status Rsp: 0a 00 01 00 80 28 01 a0
<0004>11:44:52:685 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x2780 rsp_max 1, rsp_cnt 1
<0005>11:44:54:880 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 27 00
<0006>11:44:54:882 [INFO]:(Basic)the mesh access tx cmd is 0x2780 : 00
<0007>11:44:54:999 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x2880 : 80 28 00 c4
<0008>11:44:55:001 [INFO]:(cmd_rsp)Status Rsp: 0a 00 01 00 80 28 00 c4
<0009>11:44:55:008 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x2780 rsp_max 1, rsp_cnt 1
```

Figure 20.11: Relay

Friend: a3 ff 00 00 00 00 02 01 07 00 80 10 01

```
<0000>11:44:15:977 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 10 01
<0001>11:44:15:977 [INFO]:(Basic)the mesh access tx cmd is 0x1080 : 01
<0002>11:44:16:080 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1180 : 80 11 01
<0003>11:44:16:081 [INFO]:(cmd_rsp)Status Rsp: 0a 00 01 00 80 11 01
<0004>11:44:16:086 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1080 rsp_max 1, rsp_cnt 1
<0005>11:44:24:487 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 10 00
<0006>11:44:24:488 [INFO]:(Basic)the mesh access tx cmd is 0x1080 : 00
<0007>11:44:24:599 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1180 : 80 11 00
<0008>11:44:24:602 [INFO]:(cmd_rsp)Status Rsp: 0a 00 01 00 80 11 00
<0009>11:44:24:607 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1080 rsp_max 1, rsp_cnt 1
```

Figure 20.12: Friend

Proxy: a3 ff 00 00 00 00 02 01 07 00 80 13 01


```
<0000>11:45:14:247 [INFO]:(common)ExecCmd: a3 ff 00 00 00 00 02 01 0a 00 80 13 01
<0001>11:45:14:248 [INFO]:(Basic)the mesh access tx cmd is 0x1380 : 01
<0002>11:45:14:358 [INFO]:(Basic)adr_src:0x000a,adr_dst:0x0001,access rx cmd is 0x1480 : 80 14 01
<0003>11:45:14:362 [INFO]:(cmd_rsp)Status Rsp: 0a 00 01 00 80 14 01
<0004>11:45:14:367 [INFO]:(log_win32)mesh_tx_reliable_stop: op 0x1380 rsp_max 1, rsp_cnt 1
```

Figure 20.13: Proxy

Or Please refer to “Relay”, “Friend”, “Proxy” control buttons in 4.4.3.

20.2.5 Heartbeat setting

CMD-cfg_hb_pub_set_sig = a3 ff 00 00 00 00 00 02 00 80 39 01 00 ff 02 01 07 00 00 00

20.3 Control Operations

20.3.1 Control Generic model Demo

Test Preparation

- Provisionner dongle (burn 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn mesh.bin)
- SDK no need to modify

Test Introduction

The test is to achieve the control of the Generic model, mainly to achieve the G_ONOFF_SET command test.

Test and calculate the response time of CMD send and ACK.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#.

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-g_on_03 = e8 ff 00 00 00 00 00 03 00 82 02 01 00

Step 5 The following information will show after firmware send successfully.

```

.. g_on_03
<0011>15:53:49:075 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 01 00
<0012>15:53:49:199 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 00 01 0a
<0013>15:53:49:211 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a
.. g_off_03
<0014>15:54:54:035 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 00 00
<0015>15:54:54:094 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 01 00 0a
<0016>15:54:54:107 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 01 00 0a
.. g_on_03
<0017>15:54:57:898 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 01 00
<0018>15:54:57:955 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 00 01 0a
<0019>15:54:57:969 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a
.. g_off_03
<0020>15:55:01:740 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 03 00 82 02 00 00
<0021>15:55:01:938 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 01 00 0a
<0022>15:55:01:952 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 01 00 0a

```

Figure 20.14: Firmware send successfully

Step 6 As shown in above figure <0011>, the interval between CMD sending time and rsp is $199 - 075 = 124\text{ms}$. Gateway and node are controlled in adv way, the ack reply time is different because of the network.

Step 7 If the destination address of the control is a multicast or broadcast address, the node will add a random delay before replying to the ACK after receiving the command, so that multiple device reply messages are avoided as much as possible. When broadcasting an address as shown in the figure below, the time interval between CMD and rsp is: <0023> $12:390 - 11:752 = 538\text{ ms}$

```

.. g_off
<0023>16:14:11:752 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 02 00 00
<0024>16:14:12:390 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 00
<0025>16:14:12:405 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00
.. g_on
<0026>16:14:14:328 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 02 01 00
<0027>16:14:15:096 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 04 00 01 0a
<0028>16:14:15:129 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 04 00 01 0a

```

Figure 20.15: Broadcast address

20.3.2 CTL model

Test Preparation

- Provisionner dongle (burn 8269_mesh_gw.bin or 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn mesh.bin)
- SDK need to modify #define LIGHT_TYPE_SEL LIGHT_TYPE_CTL

Test Introduction

The test is to achieve the control of the CTL model, mainly to achieve the LIGHT_CTL_SET command test.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#.

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-light_ctl_set = e8 ff 00 00 00 00 00 00 ff ff 82 5e 01 00 20 4e 00 00 00

The following information will show after firmware send successfully

```
.. light_ctl_set
<0000>16:29:46:909 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 5e 01 00 20 4e 00 00 00
<0001>16:29:47:513 [INFO]:(cmd_rsp)Status Rsp_____ : 03 00 02 00 82 60 01 00 20 4e
<0002>16:29:47:521 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 03 00 02 00 82 60 01 00 20 4e
```

Figure 20.16: Firmware send successfully

20.3.3 HSL model

Hsl model will allocate 3 element addresses after provision. The main element is used for lightness, generic model control and configuration model, element2 is for hue control, element3 is for saturation control.

Test Preparation

- Provisionner dongle (burn 8269_mesh_gw.bin or 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn mesh.bin)
- SDK need to modify #define LIGHT_TYPE_SEL LIGHT_TYPE_HSL

Test Introduction:

The test is to achieve the control of the HSL model, mainly to achieve the LIGHT_HSL_SET command test.

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time

Step 2 Burn mesh node bin in dongle_2#

Step 3 Firmware add mesh node into the network

Step 4 Send, or double click in firmware int CMD bar the following command:

CMD-light_hsl_set = a3 ff 00 00 00 00 00 00 ff ff 82 76 01 00 00 50 00 80 00

The following information will show after firmware send successfully

```
<0000>15:57:25:186 [INFO]:(common)ExecCmd: e8 ff 00 00 00 00 02 00 ff ff 82 76 00 20 00 50 00 80 00 00
<0001>15:57:25:210 [INFO]:(GATEWAY) gateway mesh cmd sendback src:0001 dst:ffff,opcode is 7682: 00 20
<0002>15:57:25:506 [INFO]:(cmd_rsp)Status Rsp: 02 00 01 00 82 78 00 20 00 50 00 80
<0003>15:57:25:515 [INFO]:(GATEWAY)HCI_GATEWAY_RSP_OP_CODE
: 91 81 02 00 01 00 82 78 00 20 00 50 00 80
```

Figure 20.17: Firmware send successfully

20.34 Vendor model

Self-defined OP operation

Test Preparation

- Provisionner dongle (burn 8269_mesh_master_dongle.bin)
- Firmware tool(sig_mesh_tool.exe)
- Dongle_2# (burn 8258_mesh.bin)
- SDK need to modify, please refer to test introduction

SDK Modify Introduction

- 1) Configure in vendor_model.h:

```
#define VD_USER_ONOFF_GET          0xE1
#define VD_USER__ONOFF_SET        0xE2
#define VD_USER__ONOFF_SET_NOACK  0xE3
#define VD_USER__ONOFF_STATUS     0xE4
```

- 2) Declair in vendor_model.c

```
{VD_USER_ONOFF_SET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set,
↪ VD_USER_ONOFF_STATUS},
{VD_USER_ONOFF_GET, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_get,
↪ VD_USER_ONOFF_STATUS},
{VD_USER_ONOFF_SET_NOACK, 0, VENDOR_MD_LIGHT_C, VENDOR_MD_LIGHT_S, cb_vd_light_onoff_set,
↪ STATUS_NONE},
{VD_USER_ONOFF_STATUS, 1, VENDOR_MD_LIGHT_S, VENDOR_MD_LIGHT_C, cb_vd_light_onoff_status,
↪ STATUS_NONE},
```

- 3) Add to main_loop.

Test Introduction

Telink SDK has the following limitation based on customer usage:

- 1) A friend node can have 16 LPN at most, the default value is 2, user may modify MAX_LPN_NUM to 16.
- 2) The data of 1 LPN node cached by the Friend node can support long packets, and the maximum length of a long packet is 41 bytes. (When the APP sends data, the maximum parameter transmission is 41 bytes.)

20.3.5 Gateway Transmit Long Packet to LPN

Test Preparation

- Provisioner dongle (burn 8258_mesh_gw.bin)
- Firmware tool(sig_mesh_tool.exe), select tl_node_gateway.ini
- Dongle_2# (burn LPN.bin)
- SDK need to modify, please refer to test introduction

Test Introduction

Telink SDK has the following limitation based on customer usage:

- 1) A friend node can have 16 LPN at most, the default value is 2, user may modify MAX_LPN_NUM to 16.
- 2) The data of 1 LPN node cached by the Friend node can support long packets, and the maximum length of a long packet is 41 bytes. (When the APP sends data, the maximum parameter transmission is 41 bytes.)

The SDK is modified as following:

- 1) Set DEBUG_SUSPEND =1 (no Low Power mode);
- 2) Add the following information in cb_vd_light_onoff_set().

```
u8 debug_fn_reciver_data[64];
u8 debug_fn_cnt;
memset(debug_fn_reciver_data,0,sizeof(debug_fn_reciver_data));
memcpy(debug_fn_reciver_data,par,par_len);

: u8 debug_fn_reciver_data[64];
: u8 debug_fn_cnt;
: int cb_vd_light_onoff_set(u8 *par, int par_len, mesh_cb_fun_par_t *cb_par)
: {
:     debug_fn_cnt++;
:     int err = -1;
:     int pub_flag = 0;
:     //model_g_light_s_t *p_model = (model_g_light_s_t *)cb_par->model;
:     vd_light_onoff_set_t *p_set = (vd_light_onoff_set_t *)par;
:
:     memset(debug_fn_reciver_data,0,sizeof(debug_fn_reciver_data));
:     memcpy(debug_fn_reciver_data,par,par_len);
: }
```

Figure 20.18: cb_vd_light_onoff_set

Test Step

Step 1 After burning gateway bin into dongle_1 #, insert it into the computer and open the firmware software sig_mesh_tool.exe at the same time.

Step 2 Burn LPN node bin in dongle_2#

Step 3 Firmware add LPN into the network.

Step 4 Control LPN node by existing vendor_on/vendor_off command, LPN can be controlled normally.

Step 5 Lengthen vendor_on command transmission parameter to 41 byte, then send

CMD-vendor_on = a3 ff 00 00 00 00 00 00 ff ff c2 11 02 c4 02 01 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

Step 6 Check debug_fn_reciver_data[64] with tdebug tool after firmware send successfully.

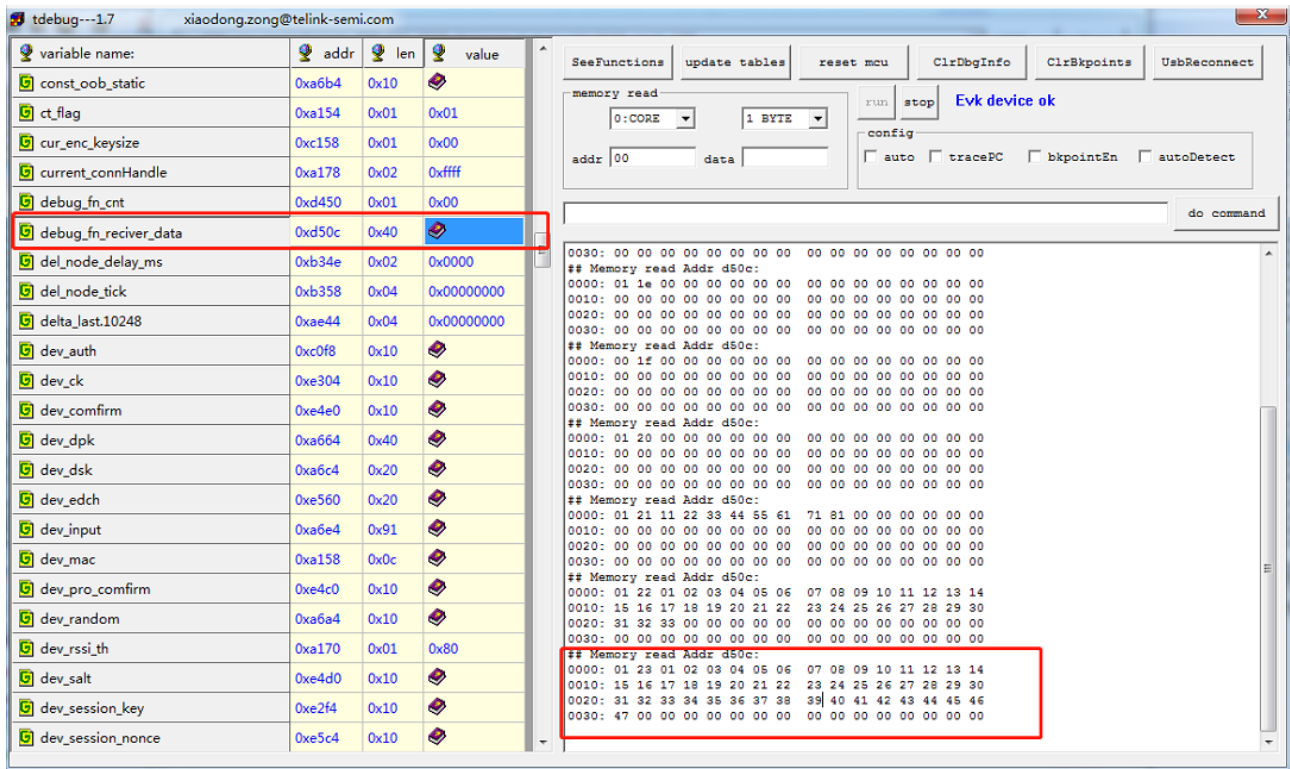


Figure 20.19: tdebug tool