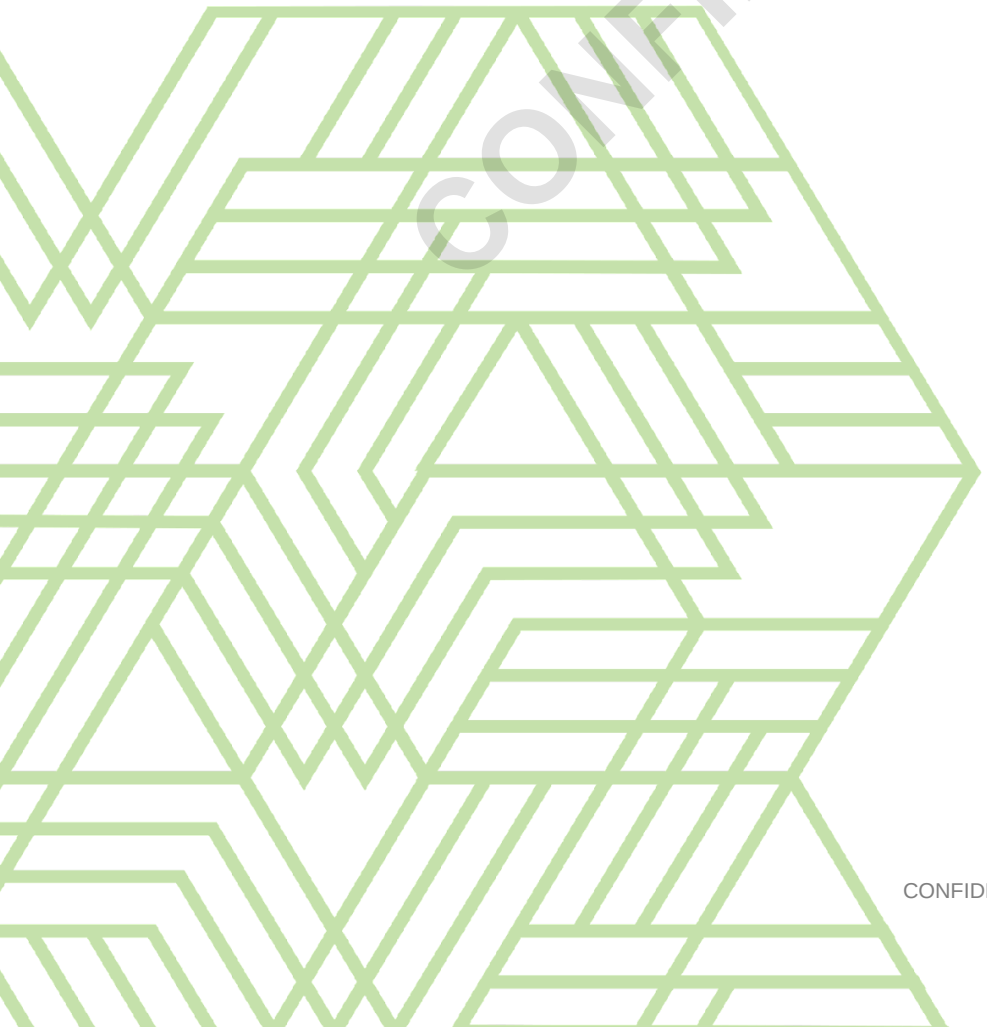
ATM2/ATM3 CT_Tracing Example

User Guide

SUMMARY: This document describes the settings, functionality, and code flow of the CT_Tracing example code running on an Atmosic-based Bluetooth LE system

CONFIDENTIAL



Atmosic™

Table of Contents

CONFIDENTIAL

CONFIDENTIAL

List of Figures

CONFIDENTIAL

CONFIDENTIAL

List of Tables

CONFIDENTIAL

1. Overview

This application note describes the settings, functionality, and code flow of the CT_Tracing example code running on an Atmosic-based Bluetooth LE system such as a contact tracing wristband. This Bluetooth LE system will send advertisements, perform scan, record scan results into flash, and initiate connectable advertisement for a Mobile APP connection. When a connection is made, the Mobile APP can retrieve the beacon logger record using GATT services. Advertising interval, scan duration and period, etc. can be configured using the GATT service interface. Mobile APP can overwrite the default settings.

Note: CT_tracing application uses iBeacon payload format. To test the scanning function, please use any iBeacon device or create iBeacon using Mobile APP. Refer to the Create iBeacon Advertiser section.

1.1 Quick Start

- Install Atmosic SDK x.y.z
- Refer to Pin Setup section
- Go to the CT_Tracing folder of the Atmosic SDK and type “make clean” then “make run_all BOARD=m2202” to program flash. ” Press and hold the button for 5 secs (see MMI event and behavior), the LED will blink and start sending pairing advertisements with “ATM-CTracing” as the device name. Refer to Flash Sector Layout for more details.
  Note: The BOARD setting can be “BOARD=<m2202|m2201|m2221|m2251|m3201|m3221|m3231>
- In this application, the address will be generated randomly during boot up. Refer to Address modes section for detail.

  Random address can be found in the console log as shown in Figure 1.:

```
@07c9c61d [   atm_gap][V]: atm_gap_rand_addr_ind - status = 0
@07c9c6f6 [   atm_gap][V]:  - actv_idx  = 0
@07c9c798 [   atm_gap][V]:  - addr_type = 0x1
@07c9c840 [   atm_gap][V]:  - addr = 0xe9:0x31:0xd5:0x5f:0xf5:0xab
```

Figure 1 - Random Address Message in Console Log

- Console Message for different stages. After booting, see Figure 2:

CONFIDENTIAL

```
@00000050 ATM2xxx-x0x silicon
Stacked 5x5 EXT_FLASH: 4e 56 44 53 11 06 01 04 12 06 01 02 16 06 01 00 ...
@000001e1 Cold reset
@00000272 [   CT_scan][D]: scan_record_list_init - nvds_record[0x200083d0]
@0000037e [   top_mmi][D]: local time: 0 s
@00000426 button_init: CFG_GPIO_MMI_BTN=9
@000004b5 [    CT_ota][V]: CT_ota_check_boot_bank
@00000572 [    CT_ota][N]:  - debug = 0x1000000f, remote AHB=0x100000
@0000066c [    CT_ota][N]:  - active_status   = 1
@00000726 [    CT_ota][N]:  - inactive_status = 1
@000007e1 [    CT_ota][N]:  - bank = 0
@00004a1f [   top_mmi][V]: mmi_enter_hid
```

Figure 2 - Booting Message in Console Log

- MMI on (press and hold the button over 5 secs) and wait for timeout Creating connectable advertisement (timeout is 30 secs), see Figure 3.

CONFIDENTIAL

```
@0002caa9 [   top_mmi][D]: hold 500
@00030aa7 [   top_mmi][D]: hold 1000
@00034aa7 [   top_mmi][D]: hold 1500
@00038aa7 [   top_mmi][D]: hold 2000
@0003caa7 [   top_mmi][D]: hold 2500
@00040aa7 [   top_mmi][D]: hold 3000
@00044aa7 [   top_mmi][D]: hold 3500
@00048aa7 [   top_mmi][D]: hold 4000
@0004caa7 [   top_mmi][D]: hold 4500
@00050aa6 [   top_mmi][D]: hold 5000
@00050b38 [   top_mmi][D]: MMI tag on
@00050c03 [   top_mmi][D]: hold 5000
@00050c97 [   top_mmi][D]: hold 5000
@00050dae [   atm_gap][V]: atm_gap_rand_addr_ind - status = 0
@00050e86 [   atm_gap][V]:  - actv_idx  = 0
@00050f28 [   atm_gap][V]:  - addr_type = 0x1
@00050fd1 [   atm_gap][V]:  - addr = 0xd0:0x12:0x5b:0xbc:0x46:0x1c
@000510c6 [ble_gap_se][N]: BOND MASK : 0
@000512f0 [   atm_gap][W]: Unhandled GAPM msg 0xd1c
@000513cc [   atm_gap][W]: Unhandled GAPM msg 0xd1c
@00051493 +otaps init: hdl:0, app_task:4, sec:4
@00051562 dts: +dts, maxc:5 attc:7 atth:32
@000515ed dts: found port type:1,1:512, att:2
@00051680 dts: found port type:3,1:512, att:5
@00051712 otaps: bulk hdl:2 mbox hdl:5
@0005178c UPGD init mx_wr:2048, mx_len:512
@0005181d UPGD init a_off: 0x0, i_off:0x80000, p_size:0x80000, e_size: 4096
@0005190c UPGD init nv_off: 0x78000, size:32768
@000519a6 UPGD init debug: 0x1000000f
@00051a21 [   atm_gap][W]: Unhandled GAPM msg 0xd1c
@00051ae7 [  atm_prfs][V]: atm_prfs_init:
@00051b8d [   atm_gap][W]: Unhandled GAPM msg 0xd1c
@00051c4a [   top_mmi][V]: ble_init_cfm
@00051ce1 [   CT_scan][V]: scan_init
@00051d6e [   CT_scan][D]:  - entry number(145) for each flash sector
@00051e61 [   CT_scan][D]:  - flash sector total number for record(176) unit: 4KB
@00051f79 [   CT_nvds][E]: Error nvds_init - get record next idx: err=2
@00052072 NVDS: Read: Configuration Parameter
@00052117 [   CT_nvds][D]:  - Adv. Interval      = 0xff (255ms)(unit: ms)
@00052218 [   CT_nvds][D]:  - Enable Encryption  = 0x01
@000522de [   CT_nvds][D]:  - Activation Status  = 0x01
@000523a4 [   CT_nvds][D]:  - Tag Type           = 0
@00052460 [   CT_nvds][D]:  - RSSI Filter Level  = -100dBm
@00052533 [   CT_nvds][D]:  - Proximity Interval = 54 secs
@00052604 [   CT_nvds][D]:  - Scan Period        = 60 secs
@000526d4 [   CT_nvds][D]:  - Scan Duration      = 840 ms
@000527b2 [   top_mmi][D]: Create Connectable Pairing Adv.
@00052881 [   atm_adv][D]: Advertising duration 3000(in unit of 10ms) max_adv_evt 0 (timeout 0ms)
@000529f3 [ble_atmprf][N]: ble_atmprfs_active_svc_db: svc_idx (0), start_hdl (39), attr_num (3)
@00052bb2 [ble_atmprf][N]: ble_atmprfs_active_svc_db: svc_idx (1), start_hdl (42), attr_num (12)
@00052d31 [  app_bass][V]: app_bass_send_lvl_cb: result 2.497V, Capacity 8.9% 9%
@00052e8b [   top_mmi][D]: ATM_ADV_CREATED act_idx=0
@00052f83 [   top_mmi][D]: ATM_ADV_SCANDATA_DONE
@00053051 [   atm_adv][D]: Adv0: ON (0)
@000530ee [   top_mmi][D]: ATM_ADV_ON act_idx=0 entry_idx=0
@000531c4 [   top_mmi][V]: s_working
```

Figure 3 - MMI On Message in Console Log

Figure 4 shows advertising timeout:

```
@000527b2 [   top_mmi][D]: Create Connectable Pairing Adv.
@00052881 [   atm_adv][D]: Advertising duration 3000(in unit of 10ms) max_adv_evt 0 (timeout 0ms)
```

Figure 4 - Connectable Advertisement Timeout Message in Console Log

CT_Tracing configuration setting, see Figure 5.

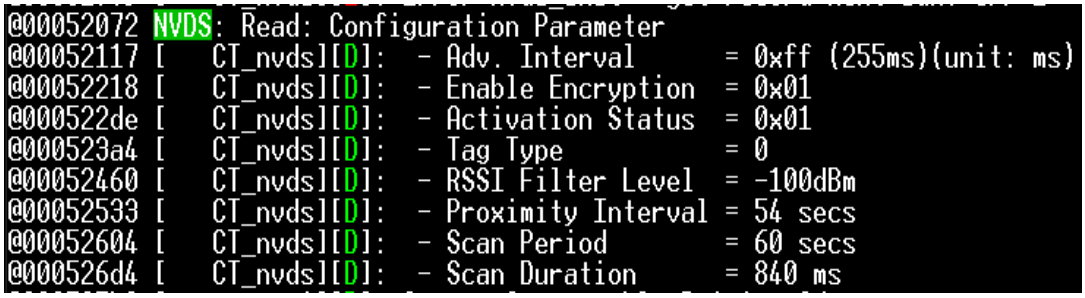


Figure 5 - CT_Tracing Configuration Setting Message in Console Log

Enter the Scan phase and print out the iBeacon device, then save into RAM. See Figure 6.

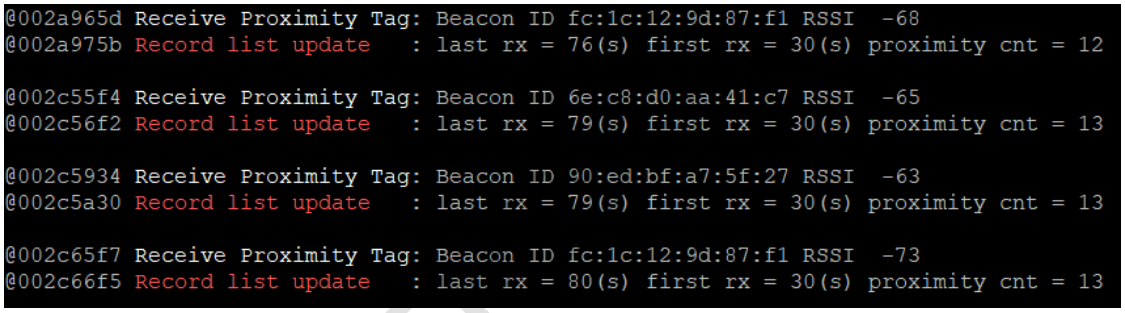


Figure 6 - Enable Scan Message in Console Log

Connectable-advertisement: timeout is 30 secs.
If there is no connection, it will move to scan/iBeacon phase after the advertisement timeout.

Found iBeacon, see Figure 7.

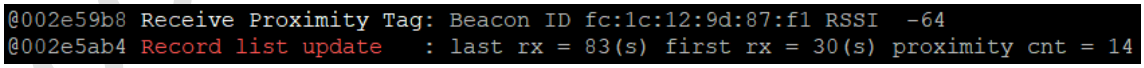


Figure 7 - Scanned iBeacon Message in Console Log

Refer to Scan Device Flow section.

- In the scan phase, the advertisement is also alive by sending an iBeacon and updating the “major” field of iBeacon payload per 500 ms (defined in INTERVAL_UPDATE_IBEACON_PAYLOAD). The default advertisement interval is 255 ms. If you are using Mobile Scan APP to check this iBeacon, you will see the “Major” value increase. Minor field reports current battery level. Refer to Figure 8.

CONFIDENTIAL

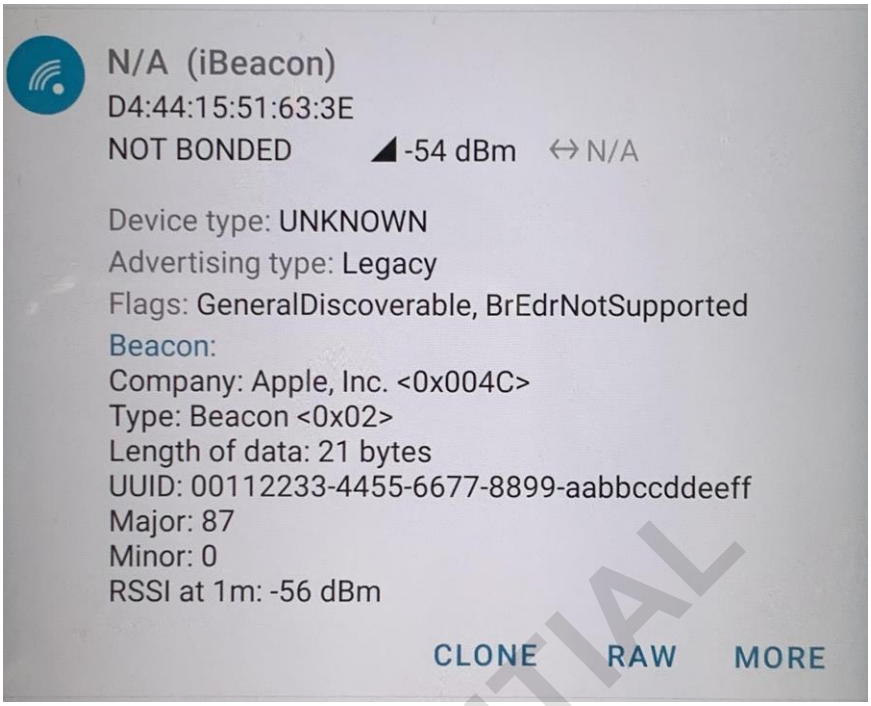


Figure 8 - iBeacon Payload

CONFIDENTIAL

- Connection Phase (press button and hold over 5 secs to enter connectable advertisement, then use Mobile APP to connect). Refer to Connection Parameter Negotiation and Figure 9.

```
@00031c0f s_working
@00082490 [   atmprfs][V]: atmprfs_create: conidx (0)
@0008255d ble_conn_ind
@000825a5 gap_conn_ind
@000825eb  - current peer addr 0x54:0x7E:0xDF:0x0B:0xB1:0xBC
@000826ae [ atm_debug][D]: + Peer (1) 54:7E:DF:0B:B1:BC
@0008277f [ atm_debug][D]: + Connection interval +    36 (unit:1.25ms)
@00082873 [ atm_debug][D]: + Slave latency       +     0
@0008293c [ atm_debug][D]: + Supervision timeout +   500 (unit:  10ms)
@00082a3a gatt_usedcap_update: Used Cap = 0x0
@00082ac9 [ble_atmprf][V]: _ble_atmprfs_connect_ind: conidx (0)
@00082ba6 [   atm_gap][D]: ConnInd idx: 0 role: S
@00082c6c [   atm_adv][D]: Adv0: OFF (0x0)
@00082d0e ATM_ADV_OFF act_idx=0 entry_idx=0
@00088178 [ atm_debug][D]: + Peer (1) 54:7E:DF:0B:B1:BC
@0008824a [ atm_debug][D]: + Connection interval +     6 (unit:1.25ms)
@00088346 [ atm_debug][D]: + Slave latency       +     0
@0008840e [ atm_debug][D]: + Supervision timeout +   500 (unit:  10ms)
@0008bb07 [ atm_debug][D]: + Peer (1) 54:7E:DF:0B:B1:BC
@0008bbd9 [ atm_debug][D]: + Connection interval +    36 (unit:1.25ms)
@0008bccd [ atm_debug][D]: + Slave latency       +     0
@0008bd95 [ atm_debug][D]: + Supervision timeout +   500 (unit:  10ms)
@0009a96e [   atm_gap][V]: target (16 10)
@0009aa0f [   atm_gap][V]: Param NOT good.(36, 0)
@0009e84c [ atm_debug][D]: + Peer (1) 54:7E:DF:0B:B1:BC
@0009e91e [ atm_debug][D]: + Connection interval +    16 (unit:1.25ms)
@0009ea11 [ atm_debug][D]: + Slave latency       +    10
@0009eadb [ atm_debug][D]: + Supervision timeout +   500 (unit:  10ms)
@000b2ab4 [   atm_gap][V]: target (16 10)
@000b2b55 [   atm_gap][V]: Param good(16, 10)
```

Figure 9 - Connection Message in Console Log

Mobile APP can discover Bluetooth LE services as shown in Figure 10. Refer to Bluetooth LE GATT Services for more information.

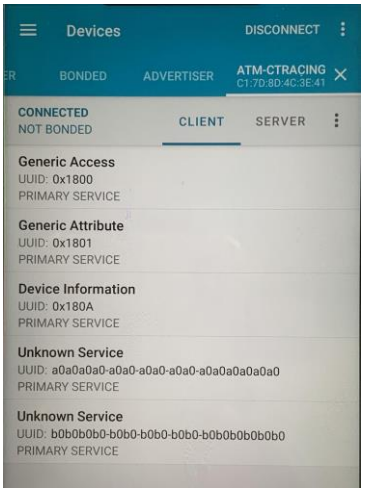


Figure 10 - Bluetooth LE GATT Service in Mobile APP

- Disconnection
  After disconnection, the device will move to scan+iBeacon concurrent mode.

2. Application States

In this application, four MMI (Man Machine Interface) states and nine sub states are defined. The MMI states transition rely on the traversal of sub-states which are triggered by MMI events. See Table 1 for state descriptions.

Table 1 - MMI State Descriptions

MMI State Name		Description
MMI Off		Device is in hibernation mode.
MMI On	CADV	Device is sending connectable advertisements and waiting for connection.
	CONN	Device is connected for setting and data retrieving.
	IBCN	Device is sending iBeacon advertisements and scanning

See Figure 11 for top MMI states transitions.

CONFIDENTIAL

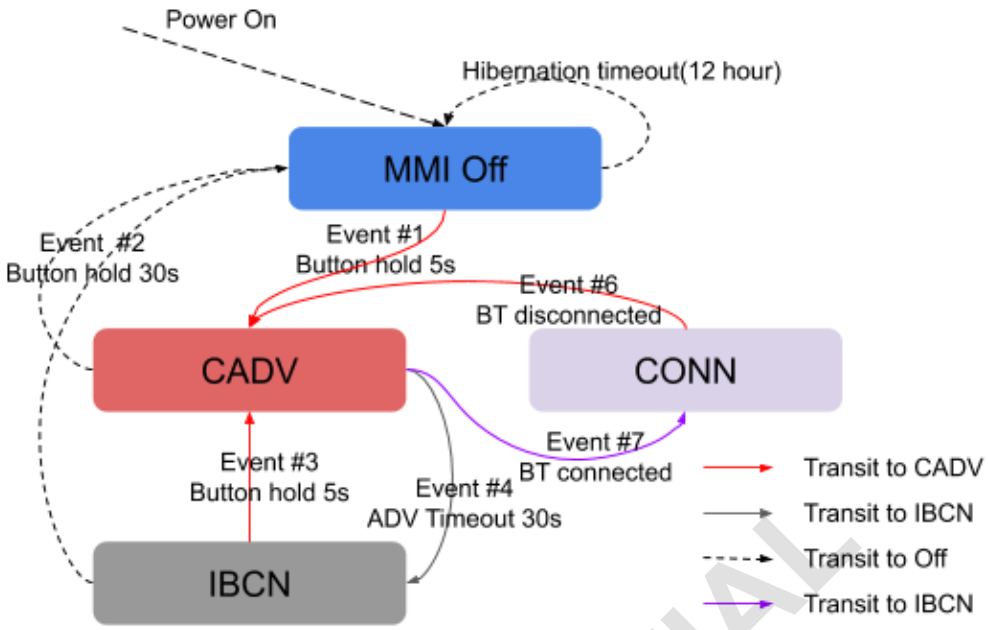


Figure 11 - MMI Transitions

2.1 MMI events and behavior

MMI events trigger state transitions of the application to accomplish user scenarios. The MMI events in this application can originate from GPIO button, ADV timeout, hibernation timeout or BT events.See Table 2 for events and their behavior.

Table 2 - MMI Events and Behavior

Event Number	Event Type	Event Condition	LED Behavior	Transitions/ Description
1	[Button]	Holding for 5 secs. Source default defined: *APP_BTN_POWER_ON_TIME	Blink once every 3 seconds for 30 seconds Source default defined: *APP_LED_TAGON_PERIOD *APP_LED_TAGON_DURATION	[Transition] MMI Off --> CADV
2	[Button]	Holding for 30 secs. Source default defined: *APP_BTN_POWER_OFF_TIME	Blink 2 times (once every 200 ms) Source default defined: *APP_LED_TAGOFF_PERIOD *APP_LED_TAGFF_DURATION	[Transition] CADV --> MMI Off: or [Transition] IBCN --> MMI Off
3	[Button]	Holding for 5 secs. Source default defined:	Blink once every 3 seconds for 30 seconds	[Transition] IBCN --> MMI Off

CONFIDENTIAL

		*APP_BTN_ENTER_CONNECTABLE	Source default defined: *APP_LED_TAGON_PERIOD *APP_LED_TAGON_DURATION	
3-1	[Button]	Short click 3 times Source default defined: *BTN_SHORT_PRESS_TO_BAS_REPORT	Battery level 80 – 100%: LED Blinks 5 times Battery level 60 – 80%: LED Blinks 4 times Battery level 40 – 60%: LED Blinks 3 times Battery level 20 – 40%: LED Blinks 2 times Battery level < 20%: LED Blinks 1 time (once every 200 ms)	1. Battery test limited to 2 tests in 24 hours (mmi_led_quick_blink_times) 2. Only allowed in IBCN state.
3-2	[Button]	Short click 5 times Source default defined: *BTN_SHORT_PRESS_SHOW_RAM_RECORD	N/A	1. Only for debug build FW 2. Suppress scan report console message 3. Every 30 secs to show report in console **Note**: Short click to enable scan report console message (trigger by #2)
3-3	[Button]	Short click 7 times Source default defined: *BTN_SHORT_PRESS_RETREIVE_FLASH_RECORD	N/A	1. Only for debug build FW 2. Suppress scan report console message 3. Read flash sector then print data out 4. Erase flash sector 5. Enter iBeacon+scan finally Note: Short click to enable scan report console message (trigger by #2)
3-4	[Button]	Short click 9 times Source default defined: *BTN_SHORT_PRESS_TO_REBOOT	LED Comes ON for 2 seconds then blinks 2 times (once every 200 ms)	[Transition] ->MMI Off
4	[ADV timeout]	30s after entering CADV Source default defined: *ADV0_START_DURATION	LED off	[Transition] CADV->IBCN
5	[Hibernation timeout]	12 hour after entering MMI Off. Source default defined: *INTERVAL_HIB_SEC	N/A	Update internal second count. [Transition]

CONFIDENTIAL

				MMI Off->MMI Off
6	[BT disconnected]	N/A	If updating new configuration data into flash nvds: LED Blinks 4 times (once every 200 ms)	May update new configuration data into flash nvds. [Transition] CONN --> CADV
			If not updating new configuration data into flash nvds: LED off	
7	[BT connected]	N/A	LED Blinks endless (once every 500 ms) Source default defined: *APP_LED_CONN_PERIOD	[Transition] CADV-->CONN
8	[BT log uploading done]	N/A	LED Blinks 4 times (once every 200 ms)	The data was read out from the BLS (Beacon Logger Service) client.

CONFIDENTIAL

2.2 Sub states

Since BT advertising and scanning utilize many APIs which need sequence controlling the application uses atm_asm module to create sub states for them and use them to accomplish the top MMI states transition. See Figure 12 for the detailed state transitions.

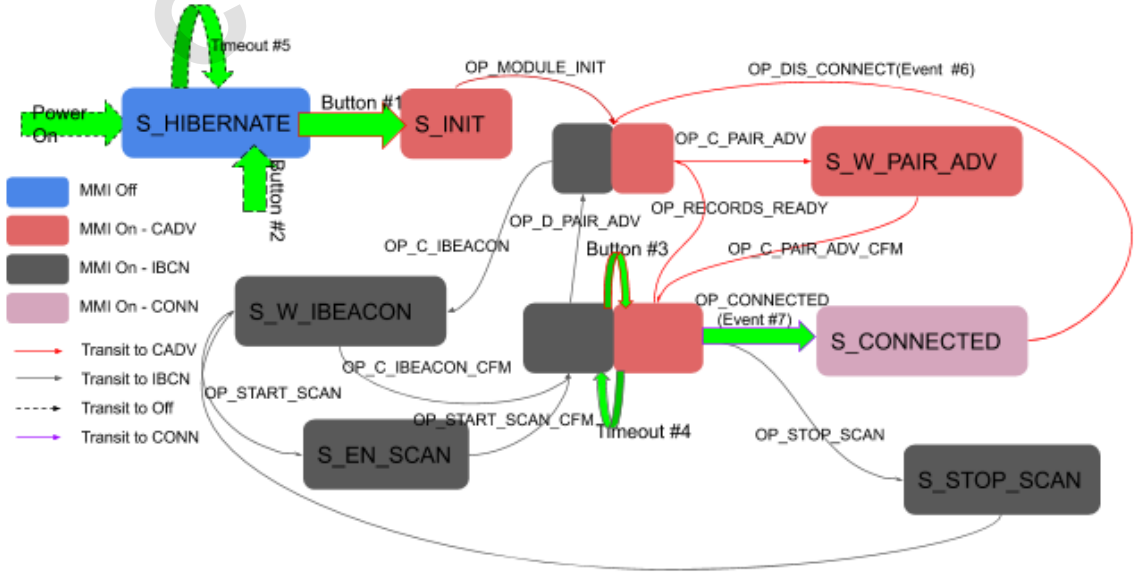


Figure 12 - Detailed State Transitions

2.3 Compile options for initial state

There are two compile options for top_mmi.c (BOOT_TO_HIBERNATION and MMI_ON_TO), see Table 3.

Table 3 - Compile Option for Booting

BOOT_TO_HIBERNATION	MMI_ON_TO	Behavior
Define	0	After power up, enter hibernation. Wait for the button to wake up and move to connectable advertisement (CADV) and will enter iBeacon+scan (IBCN) after the timeout happens. It is the default setting
Define	1	1. After power up, enter hibernation. 2. Wait for the button to wake up and move to iBeacon+scan (IBCN).
Undefine	0	After power up, move to connectable advertisement (CADV) and will enter iBeacon+scan (IBCN) after timeout happens.
Undefine	1	After power up, move to iBeacon+scan (IBCN).

CONFIDENTIAL

3. Software Modules

3.1 Module description

In CT tracing example, nine C source files and thirteen C header files were included. Please refer to Table 4 for the description.

Table 4 - Module Description

Location	File Name	Description
src/bt	CT_adv.c(.h)	Advertisement payload and parameter updating.
	CT_param_adv.h	Advertisement compile configuration.
	CT_gatt.c(.h)	BLS (Beacon Logger Service) GATT operations.
	CT_param_gap.h	GAP compile configuration.
	CT_scan.c(.h)	Scan list and record list handling.

	CT_parm_scan.h	Scan compile configuration.
	CT_ota.c(.h)	Firmware Over The Air (OTA) setting, information and flow.
src/non_bt	CT_button.c(.h)	GPIO button event module.
	CT_cipher.c(.h)	Crypto interface.
	CT_nvds.c(.h)	Record and NVDS interface..
src	top_mmi.c(.h)	Flow and state control.
	top_mmi_input.c(.h)	Button event handler.

3.2 Module hierarchy

In CT tracing example, the top_mmi collaborates all other modules by handling events from BT, button and timer. See Figure 13 for the module hierarchy.

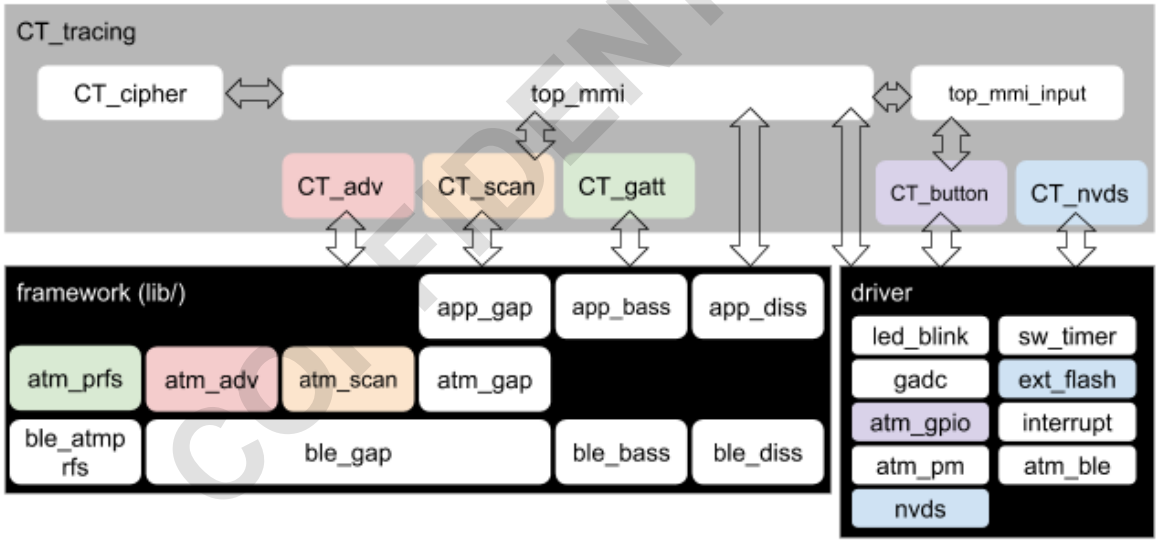


Figure 13 - Module Hierarchy

4. Message Sequence Chart

4.1 Power on, MMI on, and MMI off

After power-on, the Application will enter hibernation. When the button is pressed and held to meet number #1 trigger condition, the application will blink LED and enter “BLE init” phase. When the button is pressed and held to meet number #2 trigger condition behavior, the application will enter the “Hibernation” phase. See Figure 14.

CONFIDENTIAL

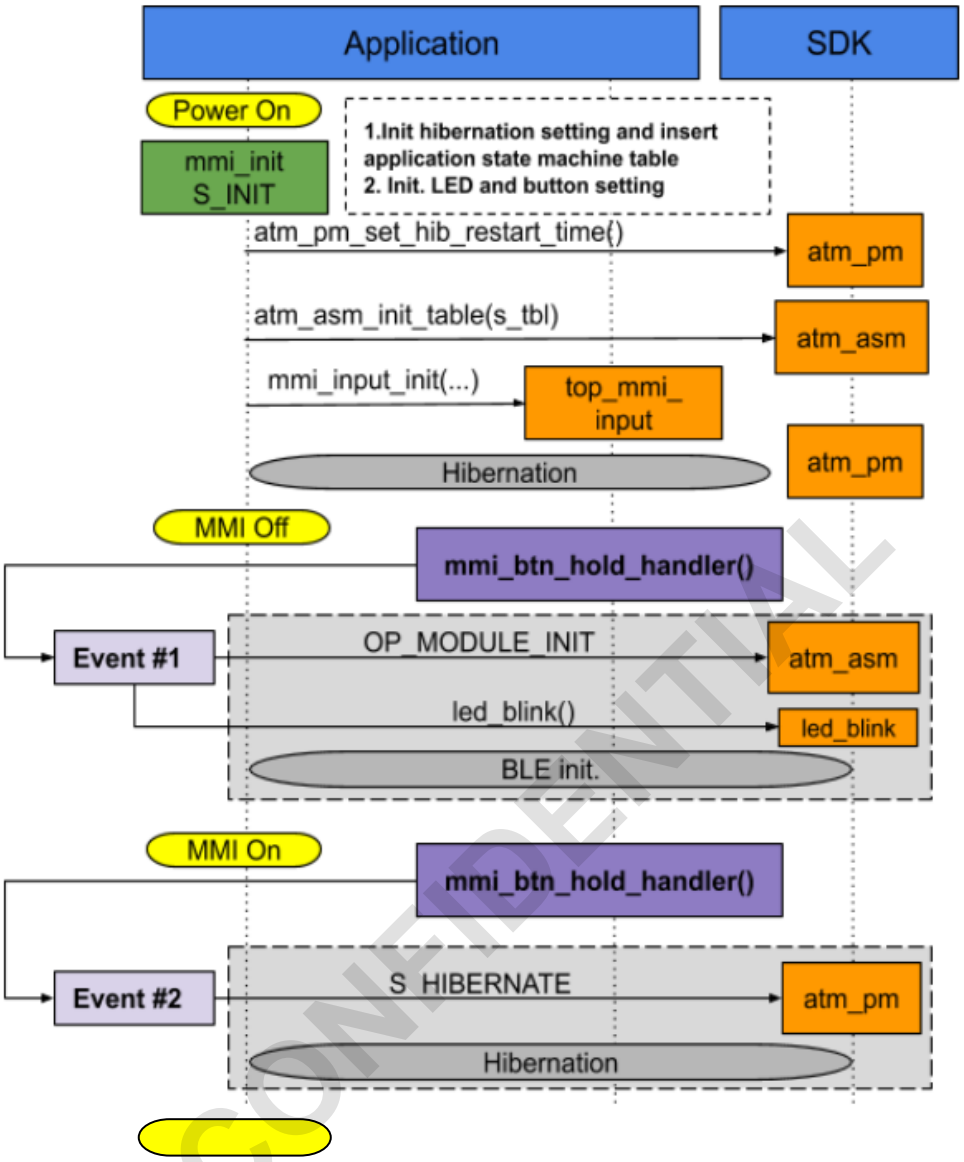


Figure 14 - Power On, MMI On and MMI Off

4.2 Bluetooth LE init. and start connectable pairing advertising (CAVD)

In this phase, the application will prepare to send connectable advertising for connection. The Application needs to use “atm_adv_reg” to register a callback function to SDK Framework. The Application can use “app_nvds” APIs to get Flash NVDS data. Before sending advertising, the Application needs to create advertising activity using the “atm_adv_create” API first. See Figure 15 for the period of Bluetooth LE Initialization and advertising activity creation.

CONFIDENTIAL

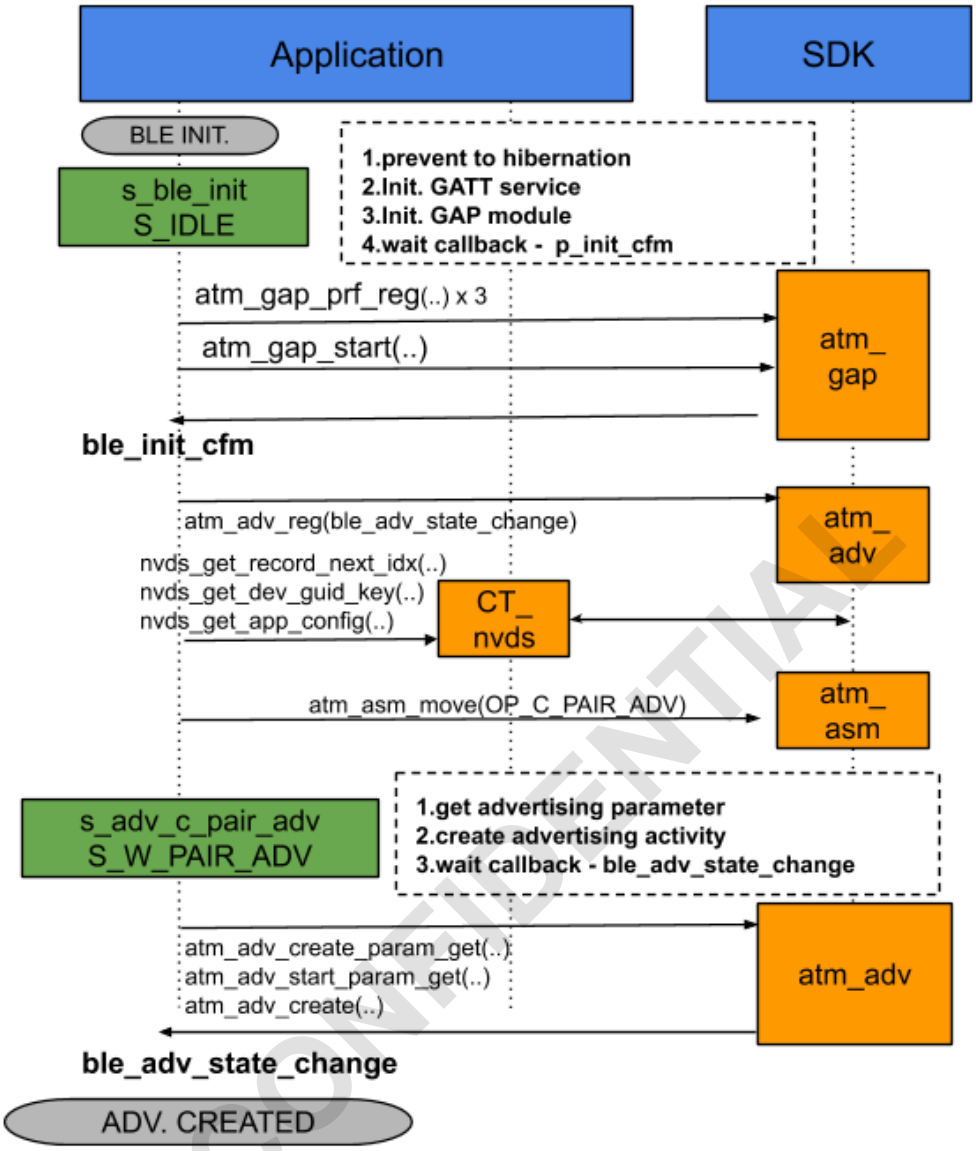


Figure 15 - Bluetooth LE Initialization and Advertising Activity Creation

After calling “atm_adv_create”, the Application will get an advertising activity ID from “ble_adv_state_change” callback function. The Application will use activity ID to configure Bluetooth LE advertising data and Bluetooth LE advertising scan response data, then call atm_adv_start API to enable this advertising activity. See Figure 16 for the period of advertising activity created to to start.

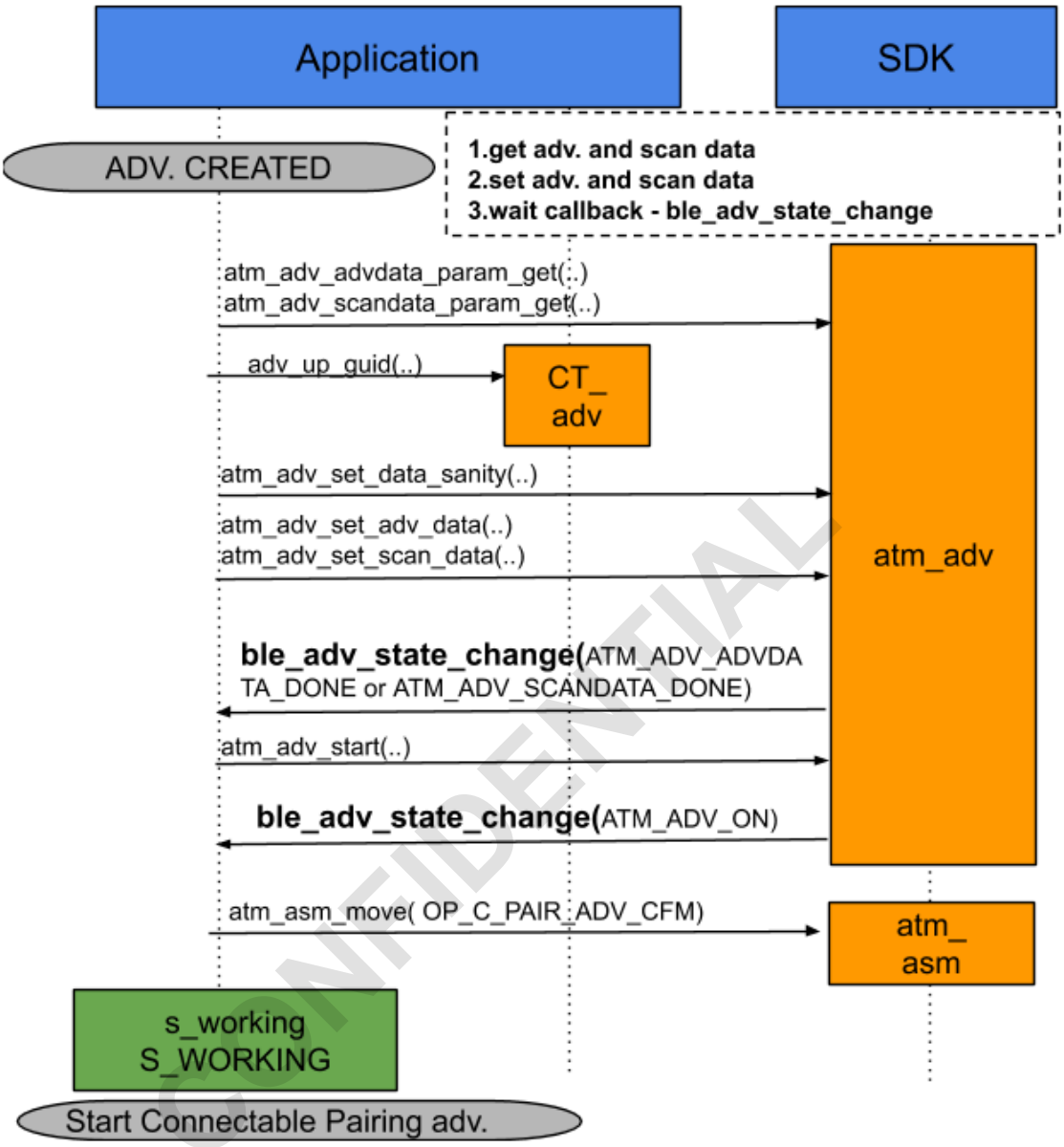


Figure 16 - Advertising Activity Created to Start

4.3 Connectable pairing advertising (CADV) timeout

When advertising is timeout, the Application will receive an “ATM_ADV_OFF” event from the “ble_adv_state_change” callback function. The Application will base on activation status value (default value loaded from Flash NVDS) to create iBeacon advertisement or enter hibernation mode. See Figure 17.

CONFIDENTIAL

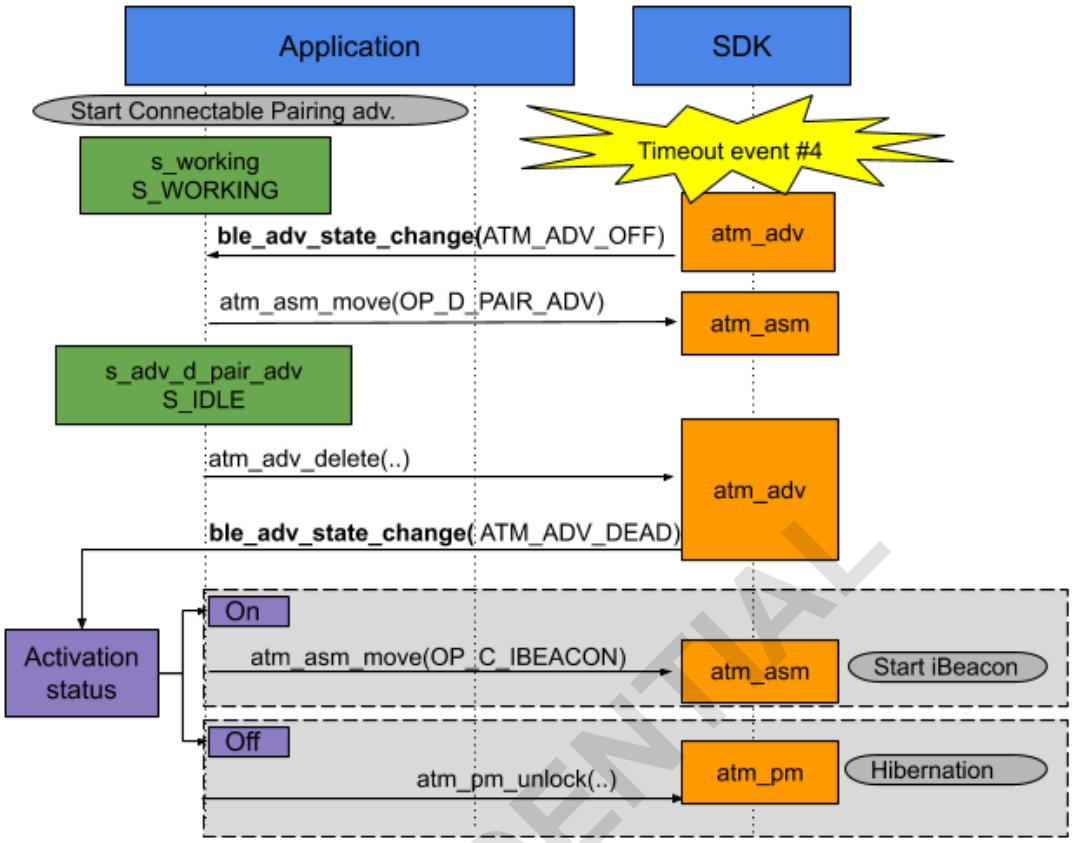


Figure 17 - Connectable Pairing Advertising Timeout

4.4 Start beacon advertising activity

To create and start iBeacon advertising activity is the same as Connectable Pairing Advertising.

The advertising interval of iBeacon can be overwritten by GATT service using Mobile APP. The advertising interval will also be stored in Flash NVDS. The Application also can use the “adv_up_ibeacon_param” API to change the advertising interval. Before updating the advertising payload, the Application can use the “cipher_encrypt” API to encrypt the payload. See Figure 18.

CONFIDENTIAL

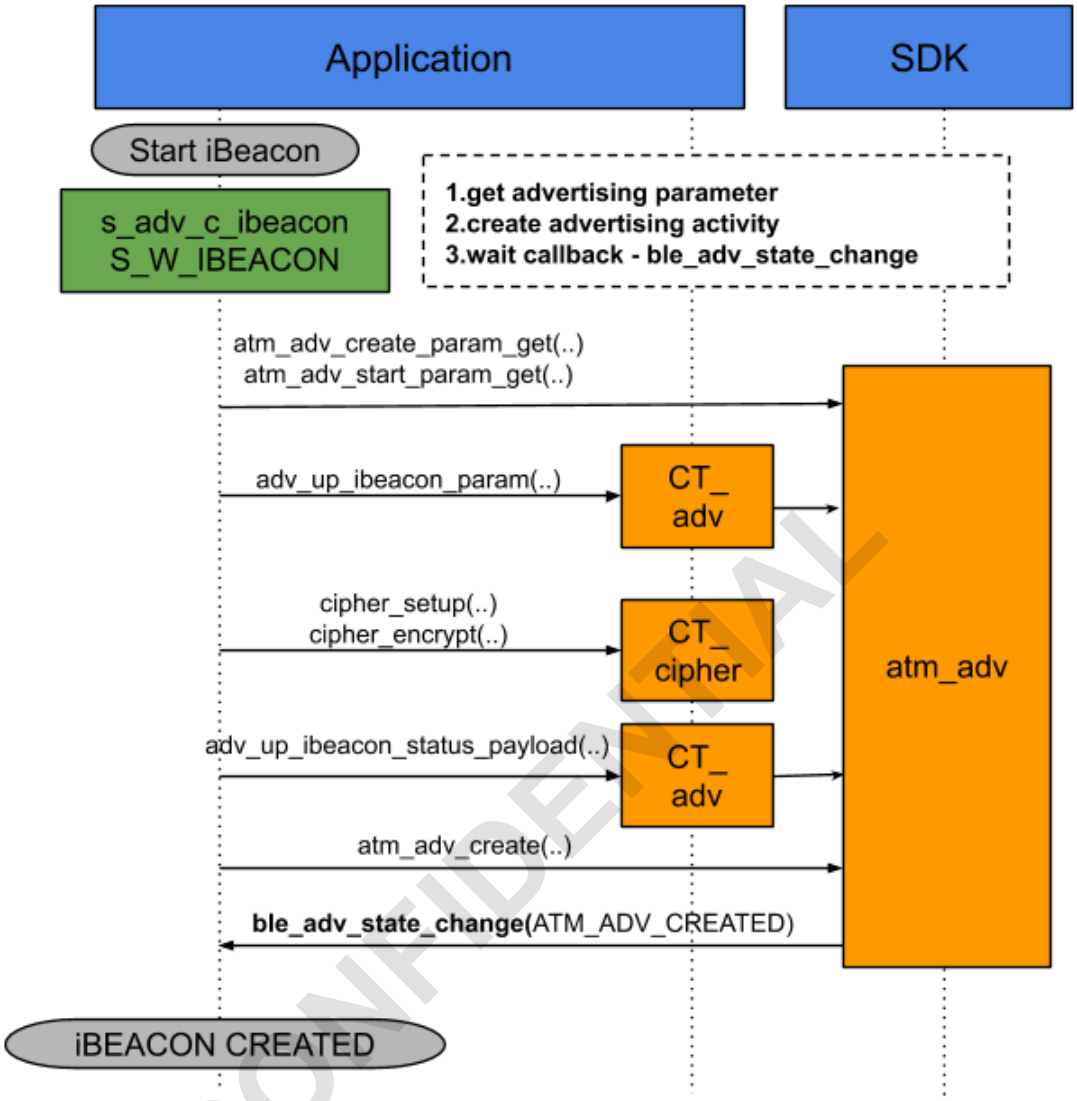


Figure 18 - Start iBeacon and Create Advertising Activity

iBeacon's device UUID can be updated by the "adv_up_guid" API. See Figure 19.

CONFIDENTIAL

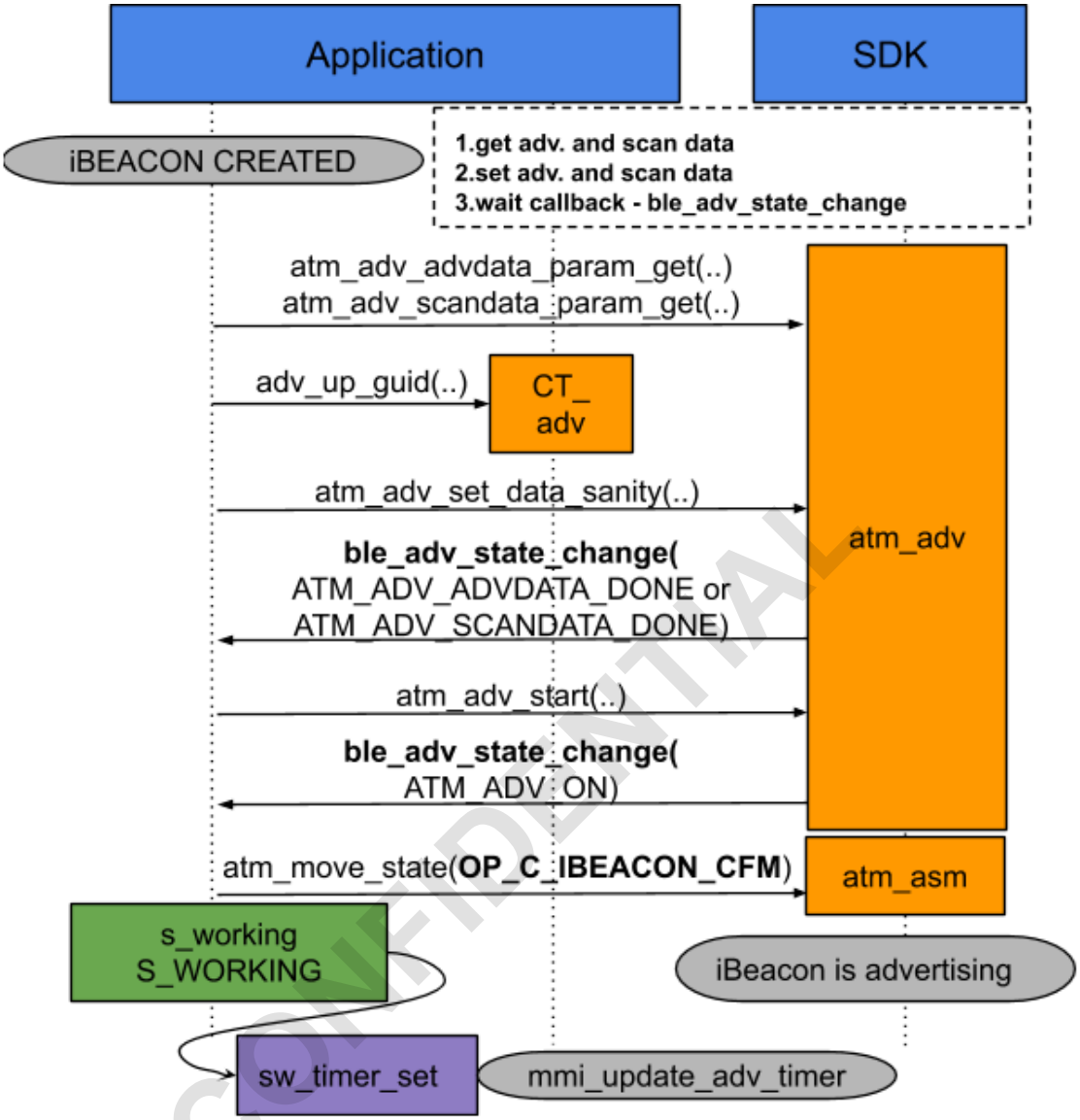


Figure 19 - Start iBeacon and Start Advertising Activity

4.5 Update iBeacon status field of adv. payload

The iBeacon activity is alive so the application can use "adv_up_ibeacon_status_payload" API to update the status field of iBeacon payload. The example will have the "TEST_UPDATE_IBEACON_PAYLOAD" definition, which is defined by default.

It will create one 500 ms timer to update the payload for iBeacon. The "mmi_update_adv_timer" function can explain how to upgrade the payload of iBeacon. See Figure 20.

CONFIDENTIAL

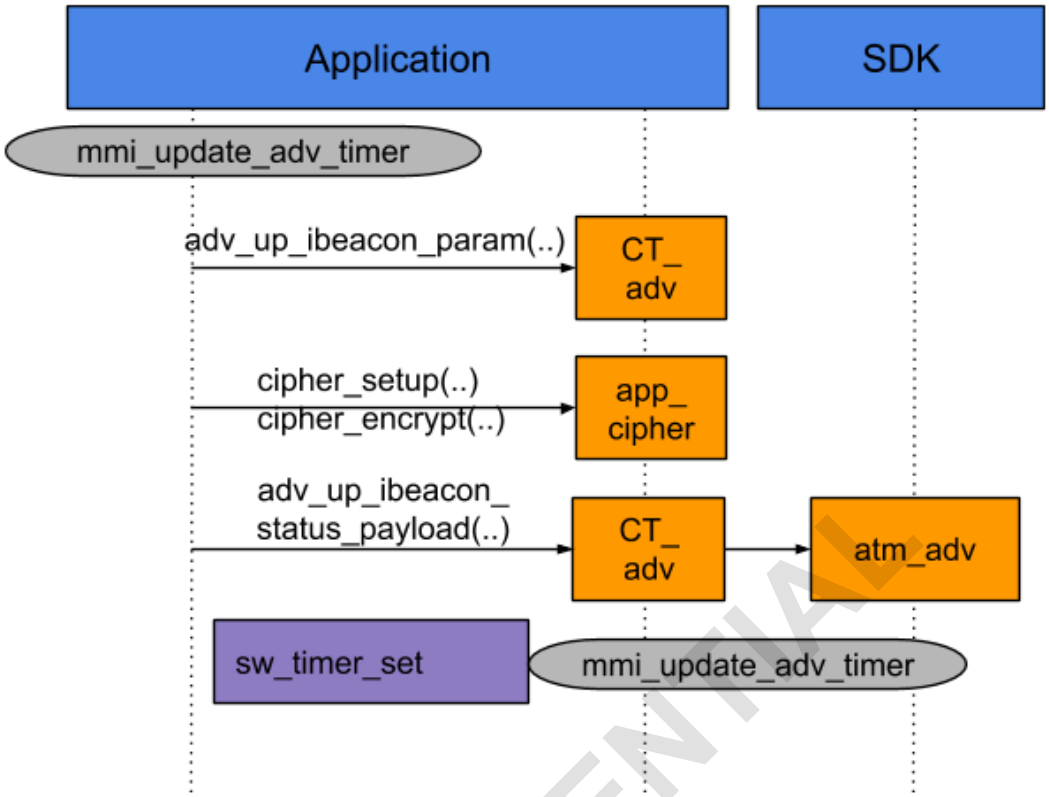


Figure 20 - Update iBeacon Advertising Payload Periodically

4.6 Connection indication

The Application will receive a connection indication event via “p_conn_ind” callback function of GAP callback. The Application will move to the connection state and turn off LED. See Figure 21.

CONFIDENTIAL

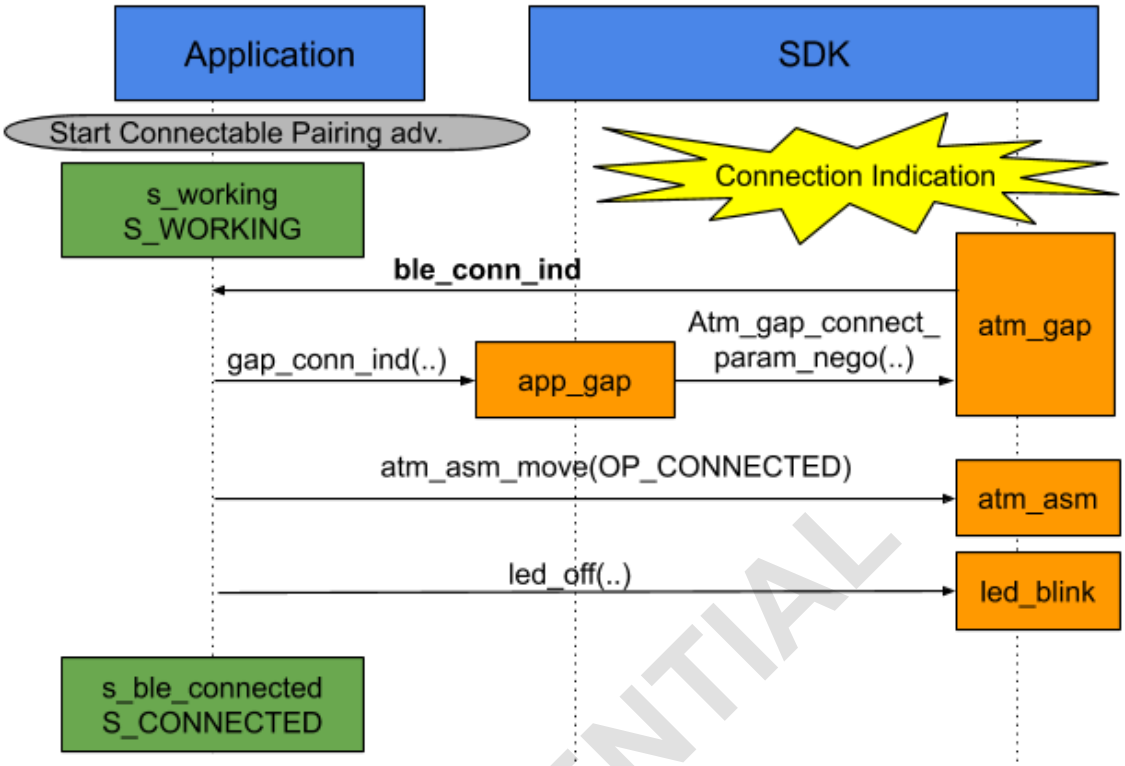


Figure 21 - Connection Indication

CONFIDENTIAL

4.7 Disconnection indication

The Application will receive a disconnection indication event via "ble_adv_state_change" callback function. The Application will record the configuration data into Flash NVDS data if having data update. The Application will check activation status configured using GATT service, then start iBeacon or enter hibernation. See Figure 22.

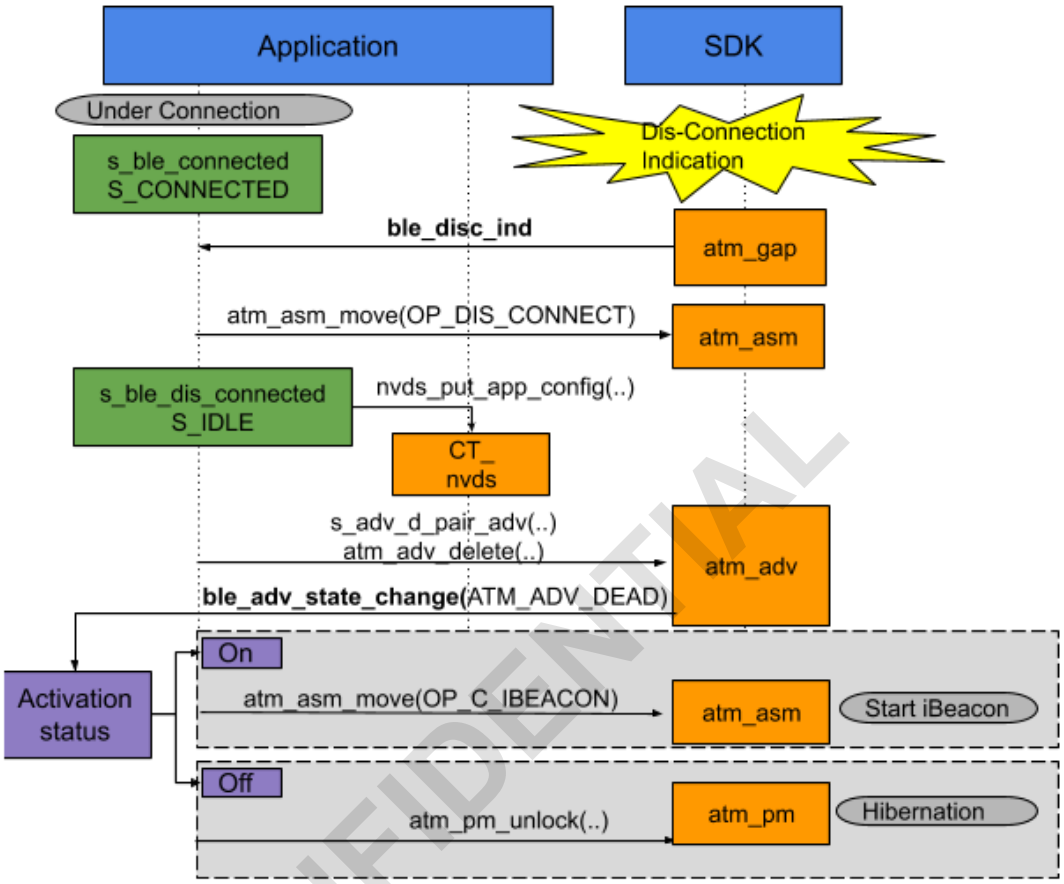


Figure 22 - Disconnection Indication

4.8 GAP Pairing

The top_mmi will receive the pairing request indication event via “p_pair_req_ind” callback function and it will call gap_pair_req_ind function to handle this event. In this example, no bonding is set to complete the whole pairing process. See Figure 23.

CONFIDENTIAL

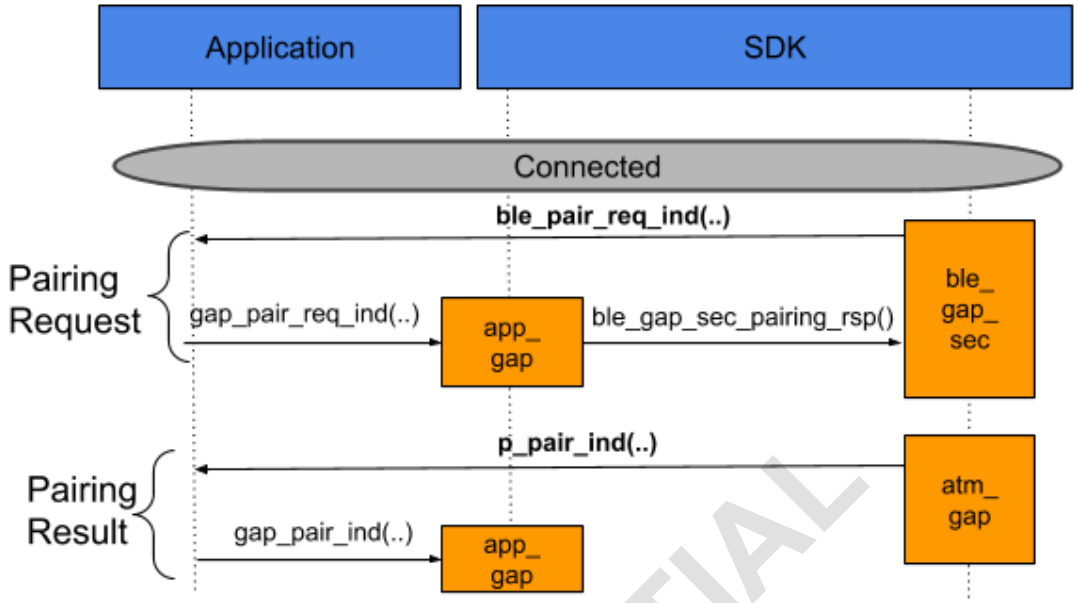


Figure 23 - Pairing Request Indication

5. Hardware Setup

5.1 PIN Setup

Table 5 - PIN Setup

Setting	Reference
Button GPIO setting	PIN_CT_MMI_BTN used in CT_button.c
LED GPIO setting	PIN_LED0 used in led_blink.c

Notes:

1) GPIO settings can be overwritten using PIN_CT_MMI_BTN, PIN_LED0 in build time.
2) PIN_CT_MMI_BTN is active-high logic by default in this example code. For active-low logic, apply the “CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1” on make command..
3) For **ATM2202** EVK (J4: pin 39 as GPIO_9 (button), pin 40 as GPIO_10 (LED)), use
    a) make run_all BOARD=m2202CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1
    b) tie GPIO_9 to GND to simulate button press
4) For **ATM2201** EVK (J4: pin 39 as GPIO_9 (button), pin 40 as GPIO_10 (LED)), use
    a) make run_all BOARD=m2201 CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1

CONFIDENTIAL

b) tie GPIO_9 to GND to simulate button press

5) For **ATM2221** EVK (J4: pin 39 as GPIO_9 (button), pin 37 as GPIO_7(LED)), use
   a) make run_all BOARD=m2221 CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1
   b) tie GPIO_9 to GND simulate button press

5.2 Configure flash layout

For layout customization, the user could modify the USER_SIZE in “makefile” to have a different beacon logger flash sector range. After applying this change, perform “make clean” then “make run_all BOARD=m22022x1>”. If BOARD is not assigned, the default is m2221.

Note: The BOARD setting can be “BOARD=<m2202|m2201|m2221|m2251|m3201|m3221|m3231>

```
ifeq ($(BOARD),m2202)
ifneq (,$(filter OTAPS,$(PROFILES)))
FLASH_SIZE = 0x80000
NVDS_SIZE = 0x8000
ifdef FLASHROM
USER_SIZE = 0x10000
else
USER_SIZE = 0x58000
endif
PMU_CFG := VBAT_GT_1p8V_VDDIO_EXT
else #OTAPS
FLASH_SIZE = 0x100000
NVDS_SIZE = 0x8000
USER_SIZE = 0xD8000
PMU_CFG := VBAT_GT_1p8V_VDDIO_EXT
endif #OTAPS
else ifneq (,$(filter m2201 m2221 m2251 m3201 m3221 m3231
x2xx_emu,$(BOARD)))
ifdef FLASHROM
CFLAGS := $(filter-out -DCFG_OTA,$(CFLAGS))
NVDS_SIZE = 0x8000
USER_SIZE = 0x18000
else ifneq (,$(filter OTAPS,$(PROFILES)))
FLASH_SIZE = 0x40000
NVDS_SIZE = 0x8000
USER_SIZE = 0x18000
else
```

CONFIDENTIAL

```
FLASH_SIZE = 0x80000
NVDS_SIZE = 0x8000
USER_SIZE = 0x58000
endif
else
    $(error "usage: make  $(MAKECMDGOALS)
BOARD=<m2202|m2201|m2221|m2251|m3201|m3221|m3231>")
endif
```

User can confirm flash layout changes in makefile using following command "make layout_info BOARD=m2202". Figure 24 below shows sample output.
Note: The BOARD setting can be “BOARD=<m2202|m2201|m2221|m2251|m3201|m3221|m3231>”

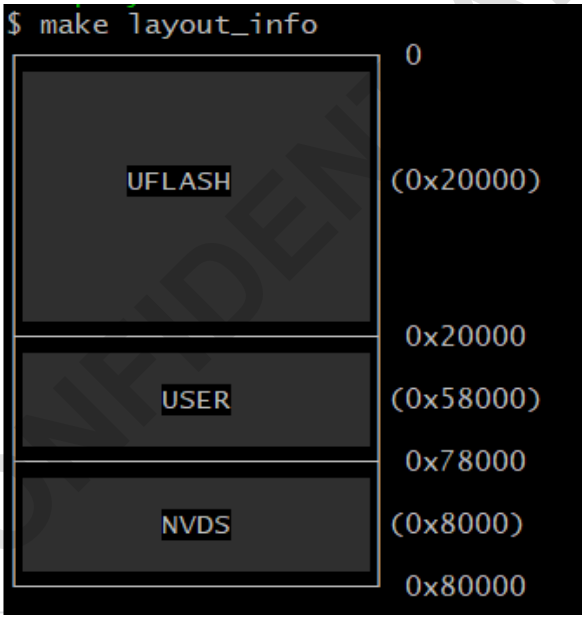


Figure 24 - Flash Layout Message

5.3 Flash sector layout

There are three kinds of makefile targets used to program flash, run_all, run, and push_flash_nvds. The run_all target programs the firmware and Flash NVDS. The run target only program firmware. These two targets both cause the Record sector to be erased. To keep the record sector, please assign PRESERVE_USER 1 on command line. Table 6 shows comparison of targets.

Table 6 - Comparison of Targets

Target	Firmware Program?	NVDSProgram?	Record Erased?
run_all	Y	Y	Y
run	Y	N	Y
run_all RESERVE_USER:=1	Y	Y	N
run PRESERVE_USER:=1	Y	N	N
push_flash_nvds	N	Y	N

To have a clean environment for testing, please use “make run_allct”. The default record sizes of ATM22x1 and ATM2202 W/WO OTA enabled are 0x30000 (0x18000*2), 0x58000, 0xB0000 (0x58000*2) and 0xD8000. Figure 25 shows the default layout of ATM2202.

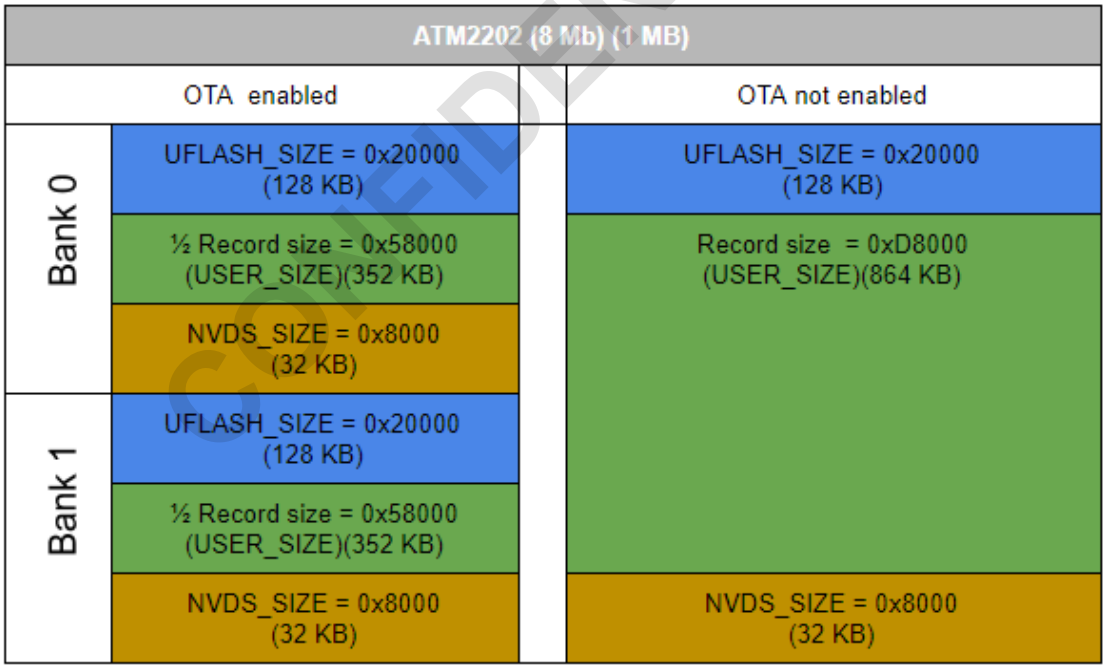


Figure 25 - Default Layout of ATM2202

Figure 26 shows the default layout of ATM22x1.

CONFIDENTIAL

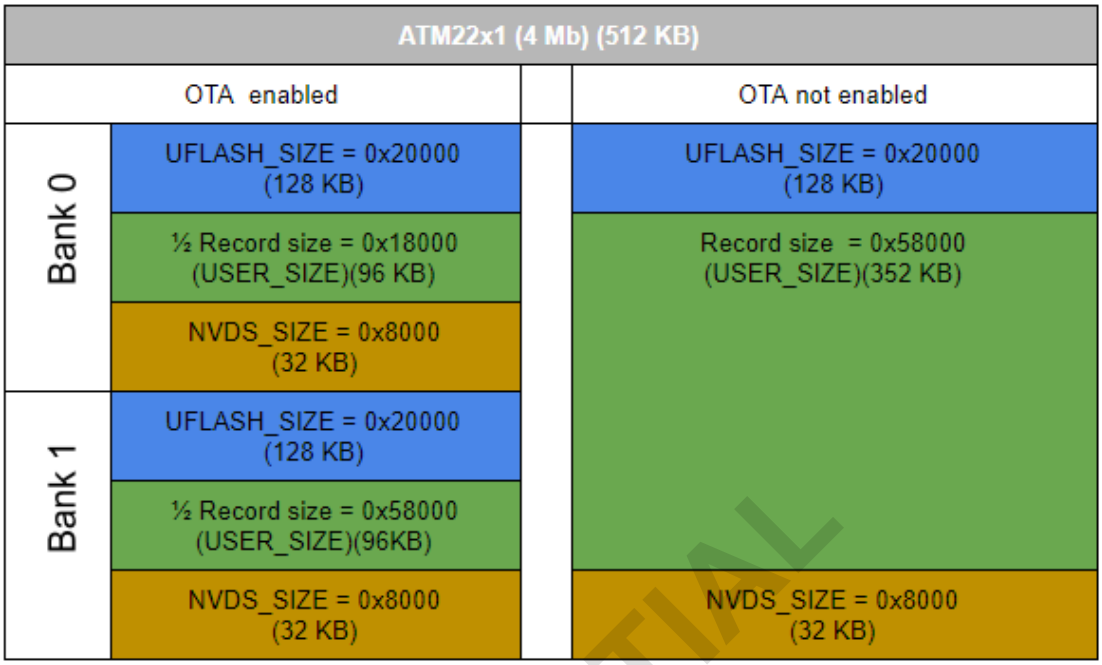


Figure 26 - Default Layout of ATM22x1

5.4 Interface board for console log

Your hardware device may only have a UART console pin (TX and GND pin). You can use the Atmosic interface board as a UART level shifter and connect the USB1 port to the PC using a USB cable. See Figure 27.

Figure 27 - UART Console Pin of Interface Board

1) Device's UART TX pin connects to UART1_TX (JP24 of interface board, right side)
2) Device's GND pin connects to the upper side of JP9.

CONFIDENTIAL

6. Application defined flash NVDS

The following application defined tag data will be used in this example. See Table 7.

Table 7 - Flash NVDS Settings

Tag ID	Description	tds file
0xAA	Device Unique Parameters	\tag_data\AA-APP_DEV_GUID_KEY\default.tds
0xAB	Configuration Parameters	\tag_data\AB-APP_CONFIG\default.tds

6.1 Device unique parameters (Tag ID:0xAA)

The following is the content of the .tds file, whose byte order is LSB first.

```
# GUID
00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
# Pairing Key
01 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
# Device Key
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
```

6.2 Configuration parameters (Tag ID: 0xAB)

The following is the content of the .tds file, whose byte order is LSB first. This parameter will map to nvds_app_config_t structure. Also see Table 8.

```
# Advertising Interval (unit: ms)
FF 00
# Enable Encryption
01
# Activation Status
01
# Tag Type
00
# RSSI Filter Level
9C
# Proximity Interval
36
# Scan Period (unit: sec)
3C
# Scan Duration (unit: 10ms)
54 00
```

CONFIDENTIAL

Table 8 - Tag ID 0xAB - APP_CONFIG Flash NVDS Settings

Description	Byte Size	Unit/Type	Note
Advertising Interval	2	ms	iBeacon advertising interval. Default: 255 (255 ms)
Enable Encryption	1	bool	Payload to be encrypted. Null function for customer to customize. Refer to CT_cihper.c Default: 1
Activation Status	1	bool	Enable activation will let devices to send advertisements and scan after leaving connectable-adv. Stage. Refer to Figure 17 - Connectable Pairing Adv. Timeout Default: 1
Tag Type	1	Advertisement payload type	Advertisement is using iBeacon payload format by default. Customers can extend this. Default: 0
RSSI Filter Level	1	dBm	Scanned beacon's RSSI is larger than this filter level will be logged. Default: -100 dBm
Proximity Interval	1	second	Scanned the same Beacon ID device will be logged (or increase proximity counter) if the receive time is larger than this. Default: 54 (secs)
Scan Period	1	second	Default: 60 (secs)
Scan Duration	1	10 ms	Default: 840 (ms)

CONFIDENTIAL

6.3 Apply the change

Using a text editor tool to open and edit the .tds file. Re-build the example code using "make run_all". The "device unique parameters" and "configuration parameters" will program into the Flash NVDS sector.

6.4 Update device UUID of Advertisement payload

The Application will call "adv_up_uuid" API when creating pairing advertisements or iBeacon advertisements. The Device UUID is from Flash NVDS. See Figure 28.

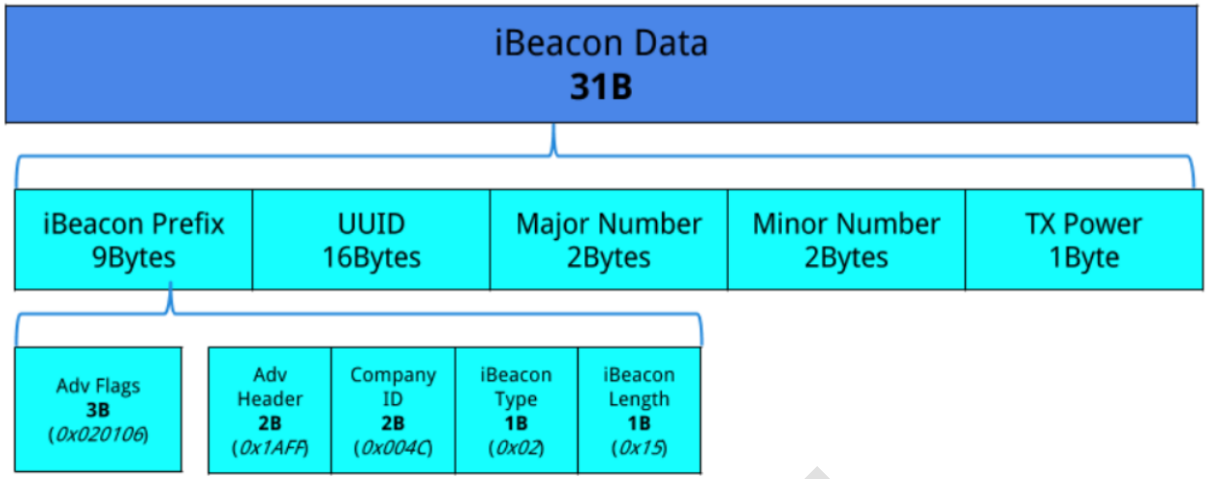


Figure 28 - iBeacon Data Payload

7. Default Parameters

7.1 Advertisements

All the advertisement parameters are defined in “CT_Tracing\src\bt\CT_param_adv.h”. This header can support Keil IDE’s configuration wizard. Open this header file using keil IDE to get a richer configuration interface. See Figure 29. Refer to the “Bluetooth LE Advertisement Example Application Note” document to get more details.

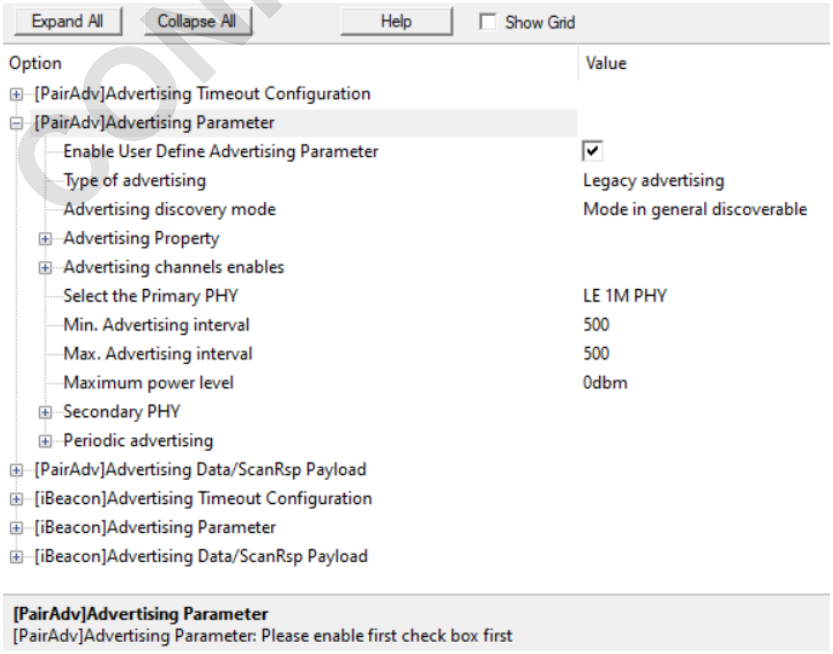


Figure 29 - Keil Configuration Wizard for Advertisement Parameter

CONFIDENTIAL

This example code uses two advertisement sets. One is for the pairing advertisement and the other is for iBeacon advertisement. For the app_env_tag_t structure, CFG_GAP_ADV_MAX_INST is set to 2. The first index is for pairing advertisements and the other is for iBeacon. The connectable pairing advertisement and non-connectable advertising are using the same index (IDX_PAIR_ADV). The example code will use the index to access the array objects and change advertising parameters

```
typedef enum {
    IDX_PAIR_ADV,
    IDX_IBEACON
} adv_set_t;

atm_adv_create_t *create[CFG_GAP_ADV_MAX_INST];
atm_adv_start_t *start[CFG_GAP_ADV_MAX_INST];
atm_adv_data_t *adv_data[CFG_GAP_ADV_MAX_INST];
atm_adv_data_t *scan_data[CFG_GAP_ADV_MAX_INST];
```

The Application uses the index to access parameters and can modify all parameters at runtime (see adv_up_ibeacon_param API as example).

About the payload of advertisement customization, users need to modify “param_gap_adv.h”. See the following CFG_ADV0_DATA_ADV_PAYLOAD and CFG_ADV0_DATA_SCANRSP_PAYLOAD definitions.

```
#define CFG_ADV0_DATA_ADV_PAYLOAD
0x1B,0xFF,0x66,0x66,0x01,0x01,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x00,
0x00,0x00,0x00,0x00,0x0A
// 0x41-'A' 0x54-'T' 0x4D-'M' 0x2D-'-' 0x43-'C' 0x54-'T' 0x72-'r' 0x61-'a' 0x63-'c' 0x69-'i' 0x6E-'n' 0x67-'g'
#define CFG_ADV0_DATA_SCANRSP_PAYLOAD
0x0D,0x09,0x41,0x54,0x4D,0x2D,0x43,0x54,0x72,0x61,0x63,0x69,0x6E,0x67,/*Appearance*/0x03,0x19,0x00,0x02
```

CONFIDENTIAL

7.2 GAP Parameter

It is defined in “CT_Tracing\src\bt\CT_param_gap.h”. This header can support Keil IDE’s configuration wizard. Open this header file using keil IDE to get a richer configuration interface. See Figure 31.

7.2.1 Connection Parameter Negotiation

There are four related parameters: CFG_GAP_CONN_INT_MIN/CFG_GAP_CONN_INT_MAX/CFG_GAP_CONN_TIMEOUT/CFG_GAP_SLAVE_LATENCY. After connecting with Bluetooth LE master, the device will perform connection parameter update negotiation. In app_gap.c -

gap_connect_param_nego(), param_nego is the parameter for connection parameter negotiation. Customers can modify the parameter depending on the application. See Figure 30.

```
const atm_gap_param_nego_t param_nego = {
    .delay = APP_PARAM_NEGO_DELAY,
    .retry_times = APP_PARAM_NEGO_RETRY,
    .check_result = APP_PARAM_NEGO_EACH_TIMEOUT,
    .target = &(struct gapc_conn_param){
        .intv_min = CFG_GAP_CONN_INT_MIN,
        .intv_max = CFG_GAP_CONN_INT_MIN,
        .latency = CFG_GAP_SLAVE_LATENCY,
        .time_out = CFG_GAP_CONN_TIMEOUT,
    }
};
```

Figure 30 - Connection Parameter Setting

7.2.2 Generic Access Device Name

CFG_GAP_DEV_NAME is used to show device name when Mobile APP connects to the device and shows in “Generic Access” service.

7.2.3 Generic Access Appearance

CFG_GAP_APPEARANCE is used to show device name when Mobile APP connects to the device and shows in “Generic Access Appearance” service.

7.2.4 Security Level

The pairing mode and service level security level will use “SEC_PROP” and “SEC_BONDING” for configuration. The default is Unauthenticated no MITM protection.

CONFIDENTIAL

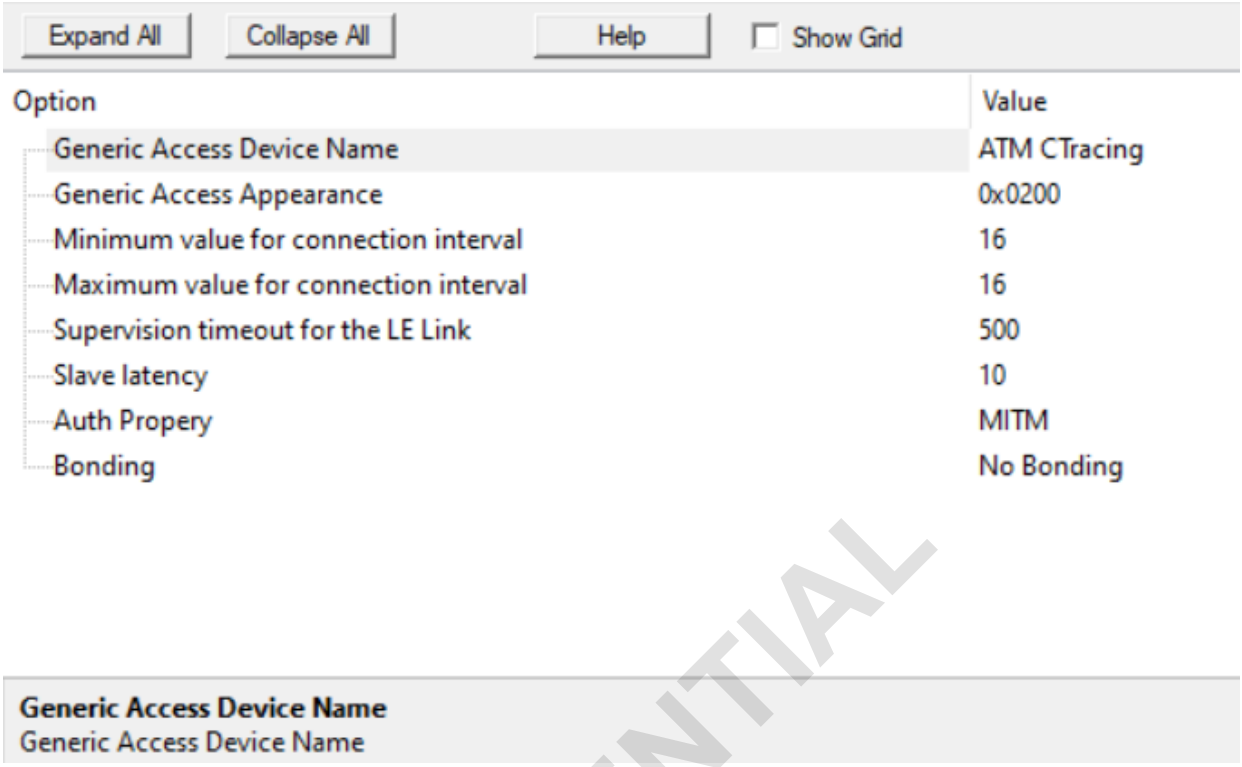


Figure 31 - Keil Configuration Wizard for GAP Parameters

7.3 Scan parameter

It is defined in “CT_Tracing\src\bt\CT_param_scan.h”. This header can support Keil IDE’s configuration wizard. Open this header file using Keil IDE to get a richer configuration interface. See Figure 32.

CONFIDENTIAL

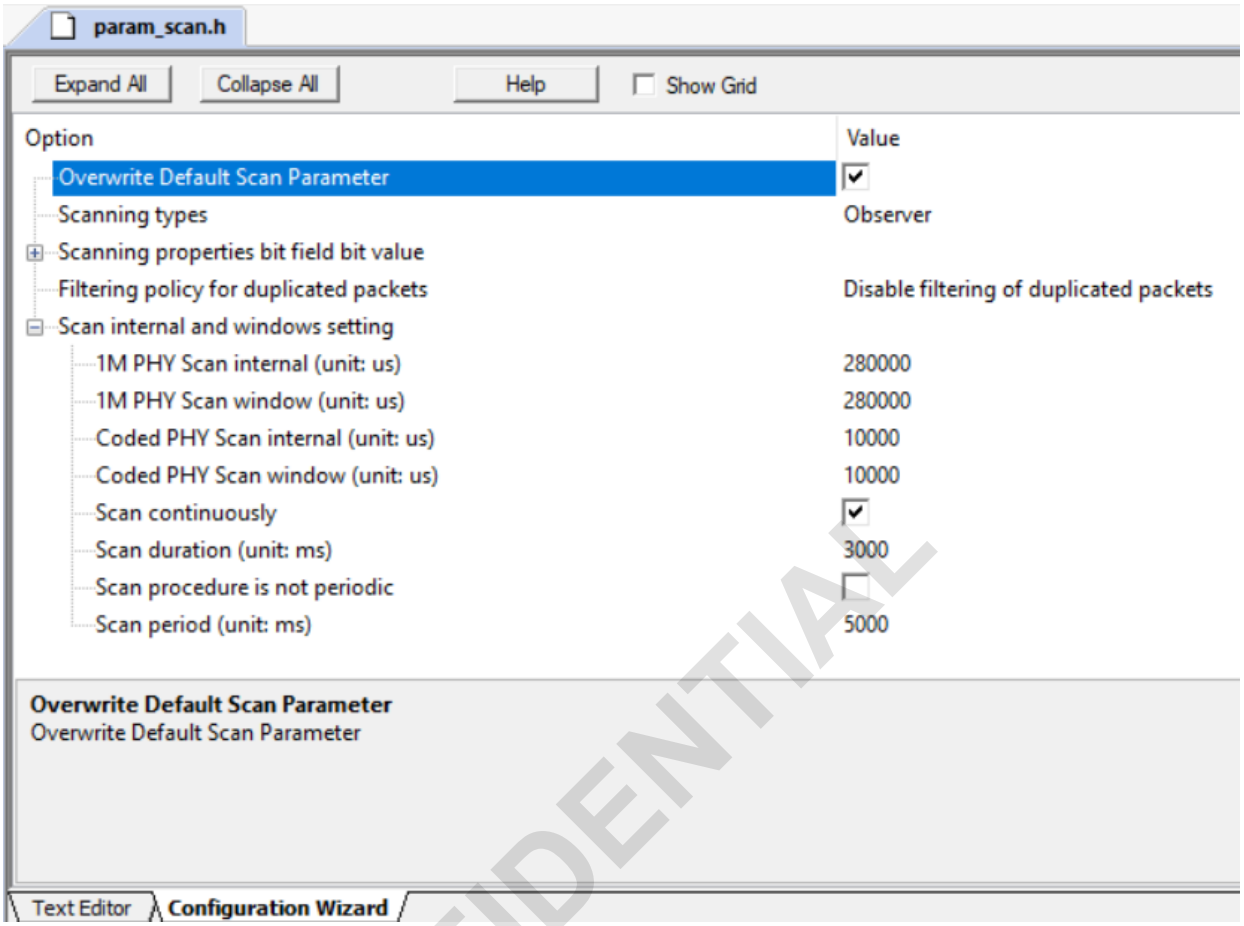


Figure 32 - Keil Configuration Wizard for Scan Parameters

CONFIDENTIAL

8. Button

The button handle is controlled by two modules: top_mmi_input.c and CT_button.c. The CT_button.c responds to the GPIO edge detection and timer measurement for button pressing, release and hold. top_mmi_input.c provides the mechanism of registering high layer user input events (hold, hold release and click) and reports them to top_mmi.c. Figure 33 shows the code pieces about registering input events and its callback function. The definition APP_BUTTON_HOLD_UNIT was defined in CT_button.h to apply as the precision of the timer. It is also used as the qualification of the time of click event.

```
static bool mmi_btn_input_event(mmi_btn_input_t *event)
{
    switch(event->type){
        case mmi_button_hold: {
            DEBUG_TRACE("hold %lu", event->hold_min_ms);
            return mmi_btn_hold_handler(event->hold_min_ms);
        } break;
        case mmi_button_hold_release: {
            DEBUG_TRACE("hold_release %lu", event->hold_min_ms);
            return mmi_btn_click_handler(event->hold_min_ms);
        } break;
        case mmi_button_click: {
            DEBUG_TRACE("click");
        } break;
```

```
const mmi_input_init_t input_config = {
    .cb_event = mmi_btn_input_event,
    .config_count = 6,
    .config = {
        {mmi_button_hold, APP_BTN_ENTER_CONNECTABLE, HOLD_MAX_TIME_6S},
        {mmi_button_hold, APP_BTN_TAP_TIME, HOLD_MAX_TIME_30S},
        {mmi_button_hold, APP_BTN_POWER_ON_TIME, HOLD_MAX_TIME_30S},
        {mmi_button_hold, APP_BTN_POWER_OFF_TIME, HOLD_MAX_TIME_60S},
        {mmi_button_click, 0},
        {mmi_button_quick_click, 0},
    }
};
```

Figure 33 - Register User Input to top_mmi_input.c

Figure 34 shows how top_mmi_input.c detects 1s hold release and 5s hold.

CONFIDENTIAL

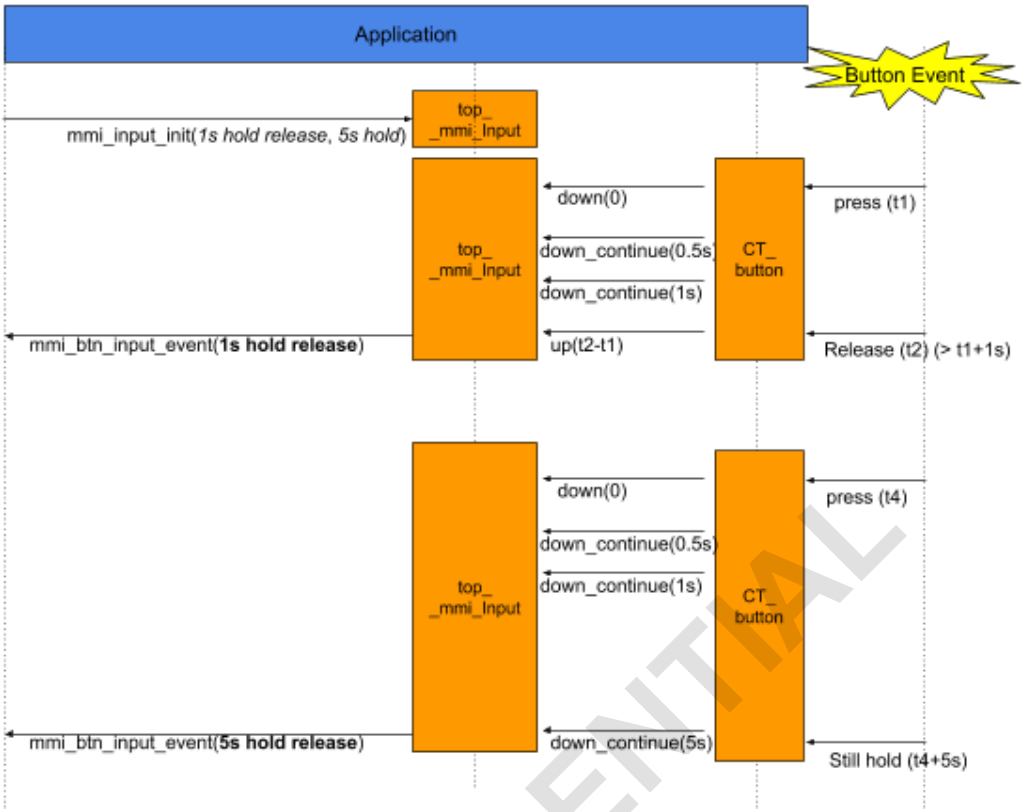


Figure 34 - Sequence Chart of CT_button and top_mmi_input

9. Hibernation Management

In this example code, the atm_pm module's lock scheme is used to manage whether or not to prevent entering a hibernation state. If all modules unlock hibernation, the system will enter hibernation. Two locks were defined in this example code: mmi_lock_hiber and button_lock_hiber. In the source file, `atm_pm_lock()` and `atm_pm_unlock()` are the points where each module decides whether to lock or not. Below is the brief list and description:

- mmi_lock_hiber
  - Used by top_mmi.c
  - Locked when BT resource needed.
- button_lock_hiber
  - Used by CT_button.c
  - Locked when timer needed

CONFIDENTIAL

10. GATT Service Create/Read/Write

This section describes how CT_gatt.c uses ATMPRF (Atmosic Profile) APIs to create the service and response of the GATT operations through the SDK framework. The ATMPRF provides an easy way to help users create their own services quickly that just need to register a few callback functions with the service creation API (ble_atmprfs_add_svc(...)). In this example code, atmprfs_cbs is defined as callback functions, shown in Figure 35.

```
ble_atmprfs_cbs_t const atmprfs_cbs = {
    .read_req = app_att_read_req,
    .write_req = app_att_write_req,
    .att_info_req = app_att_info_req,
};
```

Figure 35 - GATT Callbacks

The following four callback functions are needed:

- g_read_req: Called while peer read characteristics. Use ATMPRF API to provide data here.
- p_write_req: Called while peer write characteristics.
- p_att_info_req: Called when peer use gatt prepare write. Return the right characteristic value length here.

10.1 Create GATT service

There are 5 APIs used to create customization service, see also Figure 36.

- ble_atmprfs_create_add_svc: Add service.
- ble_atmprfs_add_char: Add characteristic.
- ble_atmprfs_add_client_char_cfg: Add client characteristic configuration descriptor.

CONFIDENTIAL

```
void CT_gatt_create_prf(void)
{
    uint8_t const cfg_svc_uuid[ATT_UUID_128_LEN] = {CT_CFG_SVC_UUID};

    uint8_t const cfg_uuid[ATT_UUID_128_LEN] = {CT_CFG_UUID};

    uint8_t const bls_svc_uuid[ATT_UUID_128_LEN] = {CT_BLS_SVC_UUID};

    uint8_t const bls_storage_uuid[ATT_UUID_128_LEN] = {CT_BLS_STORAGE_UUID};

    uint8_t const bls_used_uuid[ATT_UUID_128_LEN] = {CT_BLS_USED_UUID};

    uint8_t const bls_record_size_uuid[ATT_UUID_128_LEN] ={CT_BLS_RECORD_SIZE_UUID};

    uint8_t const bls_cmd_uuid[ATT_UUID_128_LEN] = {CT_BLS_CMD_UUID};

    uint8_t const bls_records_uuid[ATT_UUID_128_LEN] = {CT_BLS_RECORDS_UUID};

    app_att_handle[APP_SVC] = ble_atmprfs_add_svc(cfg_svc_uuid,
        SEC_PROFLE_LEVEL, &atmprfs_cbs);

    app_att_handle[APP_CHAR_DATA] = ble_atmprfs_add_char(cfg_uuid,
        APP_DATA_SEC_PROPERTY, APP_DATA_SIZE);

    app_att_handle[BLS_SVC] = ble_atmprfs_add_svc(bls_svc_uuid,
        SEC_PROFLE_LEVEL, &atmprfs_cbs);

    app_att_handle[BLS_CHAR_STORAGE] = ble_atmprfs_add_char(bls_storage_uuid,
        BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CHAR_STORAGE_CAP);

    app_att_handle[BLS_CHAR_USED] = ble_atmprfs_add_char(bls_used_uuid,
        BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CHAR_USED_CAP);

    app_att_handle[BLS_CHAR_RECORD_SIZE] = ble_atmprfs_add_char(
        bls_record_size_uuid, BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CAHR_RECORDSIZE);

    app_att_handle[BLS_CHAR_CMD] = ble_atmprfs_add_char(bls_cmd_uuid,
        APP_BLS_CHAR_CMD_PROPERTY, SIZE_BLS_CHAR_CMD);

    app_att_handle[BLS_CHAR_RECORDS] = ble_atmprfs_add_char(bls_records_uuid,
        APP_BLS_CHAR_RECORD_PROPERTY, SIZE_BLS_CAHR_RECORDS);

    app_att_handle[BLS_CHAR_RECORDS_CCCD] = ble_atmprfs_add_client_char_cfg();
```

Figure 36 - Service Creation

10.2 Handle ATT Read

When the peer device tries to read data from characteristics, CT_gatt will receive an event from the p_read_req callback. Then CT_gatt will reply to the data by calling ble_atm_prfs_gattc_read_cfm. See Figure 37.

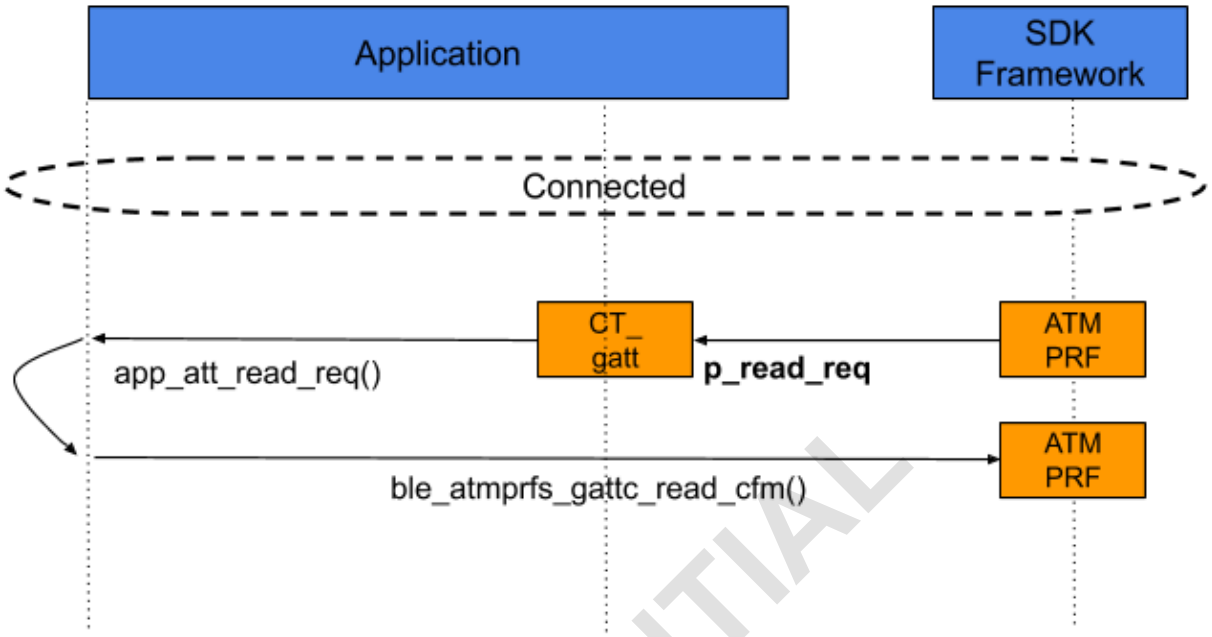


Figure 37 - GATT Read

10.3 Handle ATT Write

When the peer device tries to write data to characteristics, CT_gatt will receive an event from the p_write_req callback, and it will inform top_mmi by calling mmi_cfg_data_handler or mmi_bls_cmd_handler function. See Figure 38.

CONFIDENTIAL

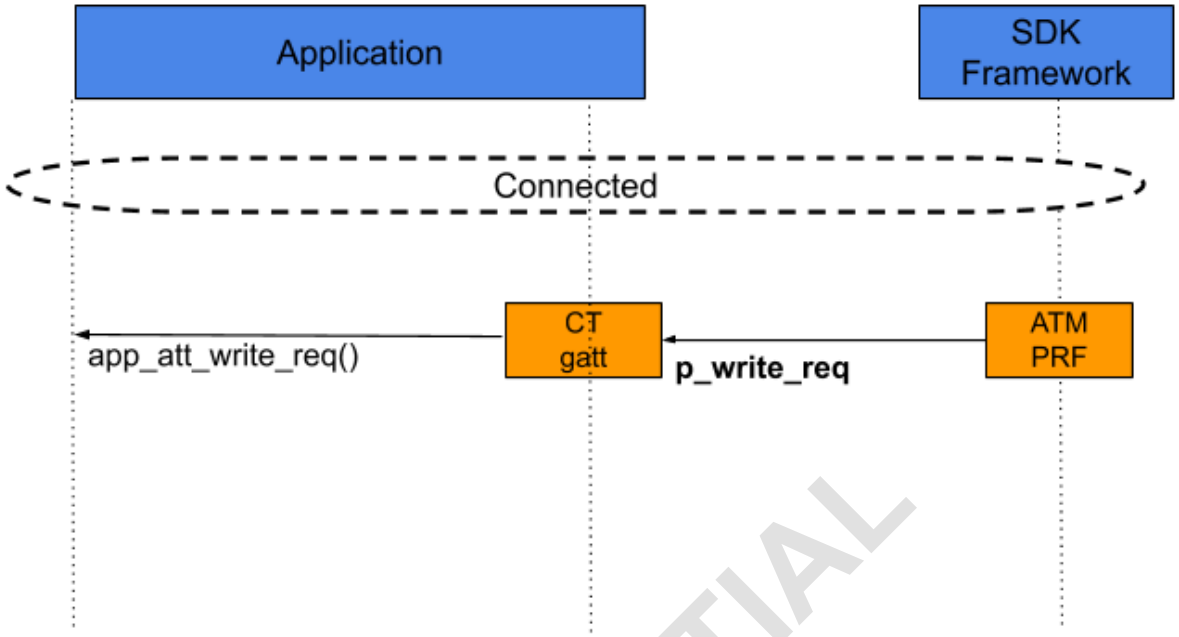


Figure 38 - GATT Write

Except for the normal operation, users can respond errors to the peer if the data is not valid or the state is not allowed to write. In the example code, some wrong operations with wrong data length, wrong data or wrong command are banned. Please check the source file for more information.

11. Address Modes

There are four address modes in this example code. “Generate new Random Address while Booting.” is default mode.

1. Generate new Random Address while Booting

In `CT_param_adv.h`:

```
#define CFG_ADV0_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
#define CFG_ADV1_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
```

In `top_mmi.c - s_ble_init()`

```
atm_gap_gen_rand_addr(GAP_STATIC_ADDR);
```

In CT_param_gap.h:

```
/*
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR5 0xC0
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR4 0x11
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR3 0x22
```

CONFIDENTIAL

```
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR2 0x33
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR1 0x44
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR0 0x55
*/
```

2. Public Device Address

Example code will follow up Flash NVDS tag's setting.

In `param_gap_adv.h`:

```
#define CFG_ADV0_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
#define CFG_ADV1_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
```

In `top_mmi.c - s_ble_init()`

```
//atm_gap_gen_rand_addr(GAP_STATIC_ADDR);
```

In `param_gap.h`:

```
/*
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR5 0xC0
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR4 0x11
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR3 0x22
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR2 0x33
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR1 0x44
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR0 0x55
*/
```

3. Fixed Random Address

In `param_gap_adv.h`:

```
#define CFG_ADV0_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
#define CFG_ADV1_OWNER_ADDR_TYPE GAPM_STATIC_ADDR
```

In `top_mmi.c - s_ble_init()`

```
//atm_gap_gen_rand_addr(GAP_STATIC_ADDR);
```

In `param_gap.h`:

```
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR5 0xC0
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR4 0x11
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR3 0x22
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR2 0x33
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR1 0x44
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR0 0x55
```

4. Random Address Rotate

Re-new address timeout is 15 mins by default.

CONFIDENTIAL

In `param_gap_adv.h`:

```
#define CFG_ADV0_OWNER_ADDR_TYPE GAPM_GEN_NON_RSLV_ADDR
#define CFG_ADV1_OWNER_ADDR_TYPE GAPM_GEN_NON_RSLV_ADDR
```

In `top_mmi.c - s_ble_init()`

```
//atm_gap_gen_rand_addr(GAP_STATIC_ADDR);
```

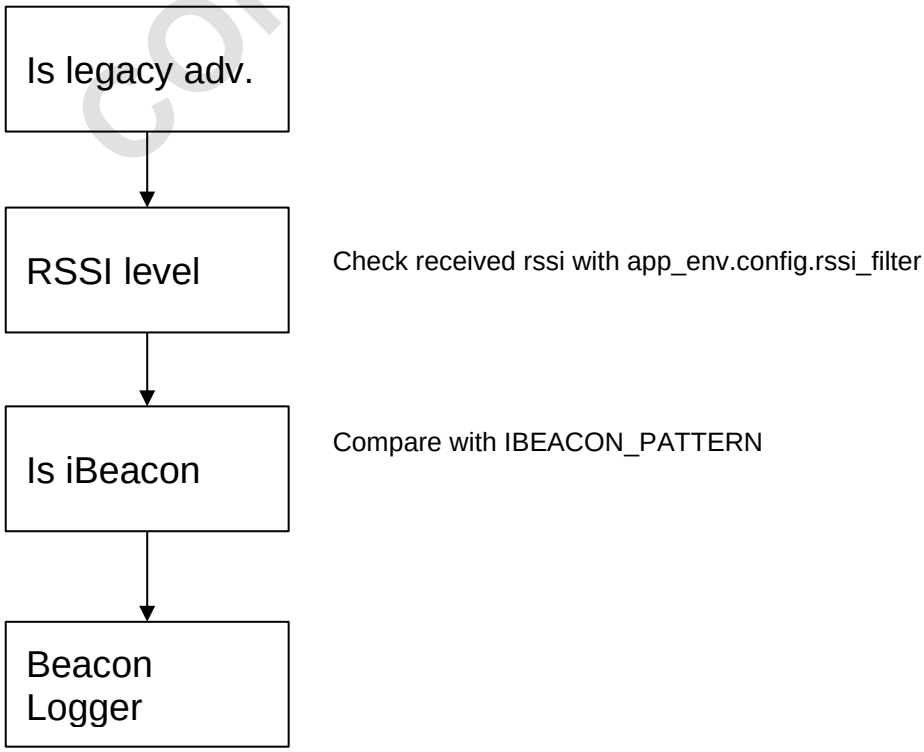
In `param_gap.h`:

```
/*
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR5 0xC0
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR4 0x11
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR3 0x22
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR2 0x33
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR1 0x44
#define CFG_GAP_OWN_STATIC_RANDOM_ADDR0 0x55
*/
```

12. Scan Device Flow

The default scan parameter is using 1 Mbps PHY and passive scan (refer to param_scan.h).

Scan device report filter policy is In CT_scan.c (scan_report_ind). See Figure 39.



CONFIDENTIAL

Figure 39 - Scan Device Report Filter Policy

12.1 Create iBeacon Advertiser

Clone Method:
Scan an iBeacon device in the scan list, clone it, modify UUID, then enable. See Figure 40.

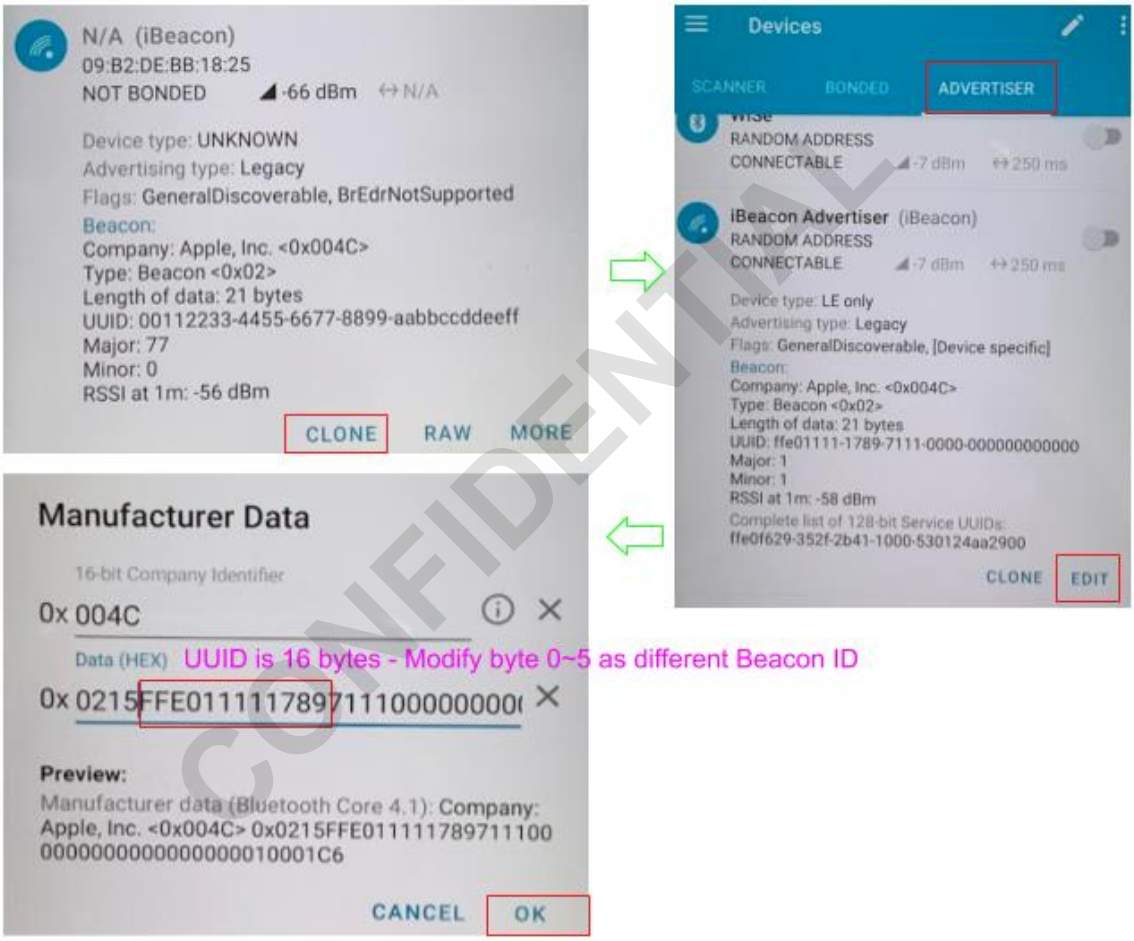


Figure 40 - iBeacon Advertiser Clone Method

Create New Method:
Follow iBeacon payload format to create a new iBeacon advertiser, then enable. See Figure 41.

CONFIDENTIAL

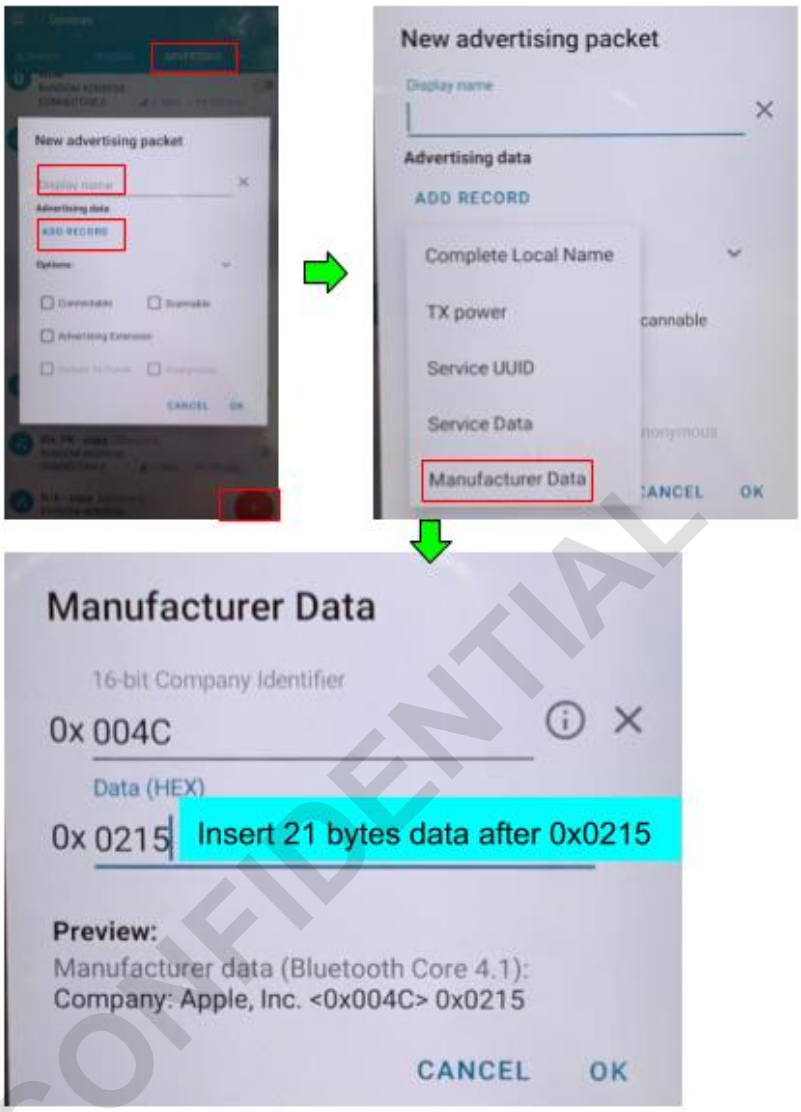


Figure 41 - iBeacon Advertiser Create New Method

CONFIDENTIAL

Note: 0x004C is a company ID defined in iBeacon payload format. In this example, synchronizing the setting with FW source code for the filter policy is needed (refer to CT_scan.c : IBEACON_PATTERN).

12.2 Beacon ID

This Application will NOT use BT ADDR to identify the peer device instead of Beacon ID. Beacon ID is from the UUID field of iBeacon payload. It will take byte 0~5 of UUID as Beacon.

12.3 Beacon Logger Process

In Beacon Logger Process, each received iBeacon will keep/update the data field defined in device_info_t structure, see Figure 42 and Table 9.

All the data will be kept in the RAM and use linked lists (scan_list and record_list) to manage the buffer usage. For each received iBeacon, the Application will search through the linked list (scan_list) using Beacon ID.

If this is a new Beacon ID, the Application will create a new entry into scan_list.

If there is already a Beacon ID in scan_list and duration is longer than the proximity interval, it will be removed from scan_list and inserted into records_list.

If there is already a Beacon ID in scan_list and record_list and duration is longer than proximity interval, the proximity counter in records_list will be updated.

If there is already a Beacon ID in scan_list and duration is longer than nearby timeout, it will be removed from scan_list and inserted into records_list.

Once the size of records_list is more than 4 Kbyte (a flash sector), it will be flushed to flash. In this application, the oldest data in flash will be overwritten when it is full. Please check bls_scan_report_ram_to_flash() in top_mmi.c for more detail about flash record write operations See Figure 43 for the flows.

```
typedef struct device_info {
    co_list_hdr_t hdr;
    uint32_t start_time;
    uint32_t rx_time;
    uint16_t proxi_cnt;
    int8_t max_rssi;
    uint8_t beacon_type;
    uint8_t beacon_ID[6];
} device_info_t;
```

Figure 42 - device_info_t Structure for Scan

CONFIDENTIAL

Table 9 - device_info_t Structure for Scan

Data Field	Description
start_time	First received time (unit: second)
rx_time	Last received time (unit: second)
proxi_cnt	Counter: Received the same Beacon ID in proximity internal.
max_rssi	Maximum received RSSI value
beacon_type	Is TAG_IBEACON or TAG_MY_BEACON
beacon_ID	Beacon ID

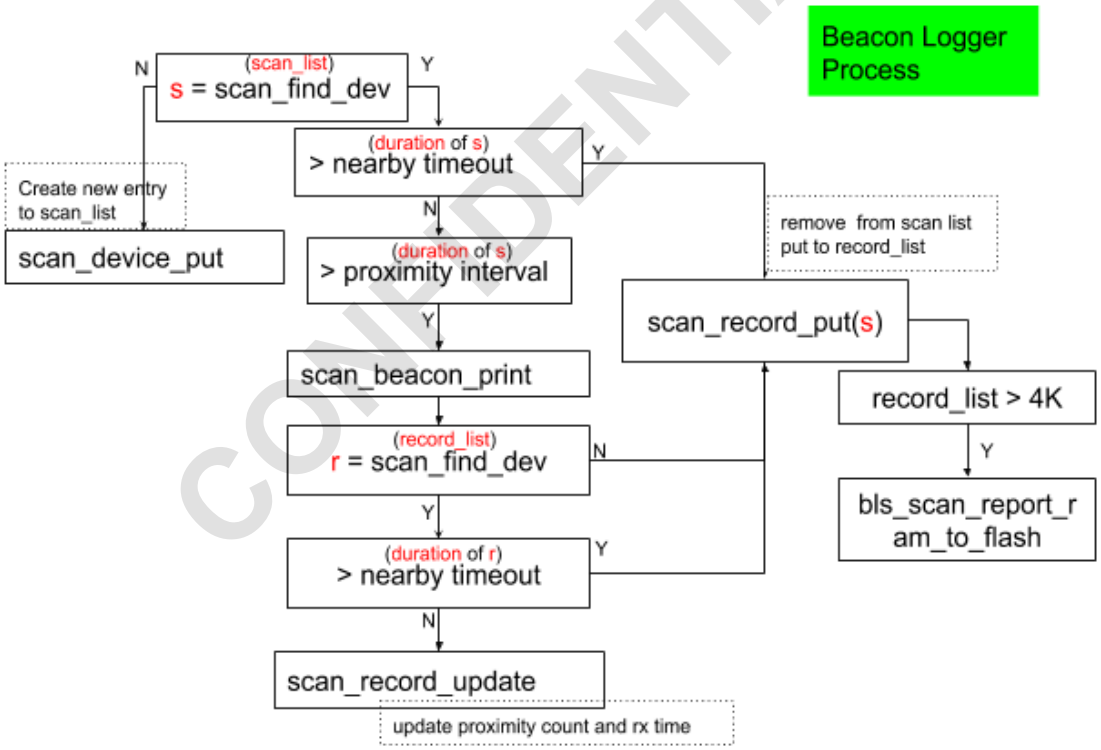


Figure 43 - Beacon Logger Process

CONFIDENTIAL

13. Bluetooth LE GATT Services

Generic Access and Generic Attribute service will be created by the SDK framework automatically. Standard DIS and customized service will be added in `top_mmi.c` (`s_ble_init`). See below:

```
atm_gap_prf_reg(BLE_DISS, app_bass_param());
CT_gatt_create_prf();
atm_gap_prf_reg(BLE_ATMPRFS, NULL);
```

Table 10 - Bluetooth LE GATT Service UUID

Service Name	UUID	Note
Generic Access	0x1800	
Generic Attribute	0x1801	
Device Information	0x180A	
Customized service - Activation and Adv. Parameter Setting Service	a0a0a0a0...	Refer to cfg_svc_uuid of CT_gatt.c
Customized service - Beacon logger	b0b0b0b0...	Refer to bls_svc_uuid of CT_gatt.c

See Figure 44 for Activation and Advertising Parameter Service.

CONFIDENTIAL

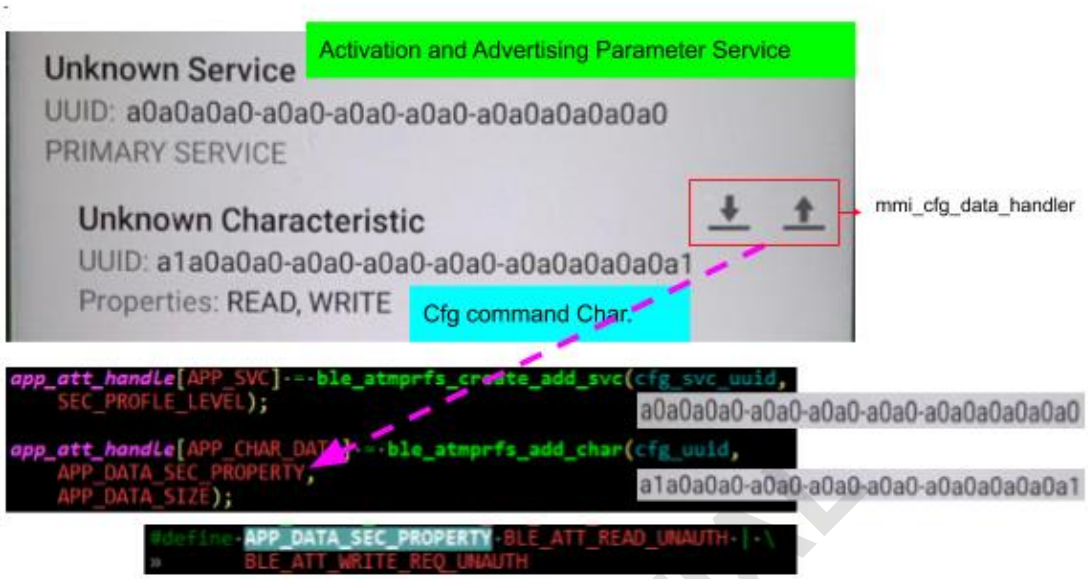
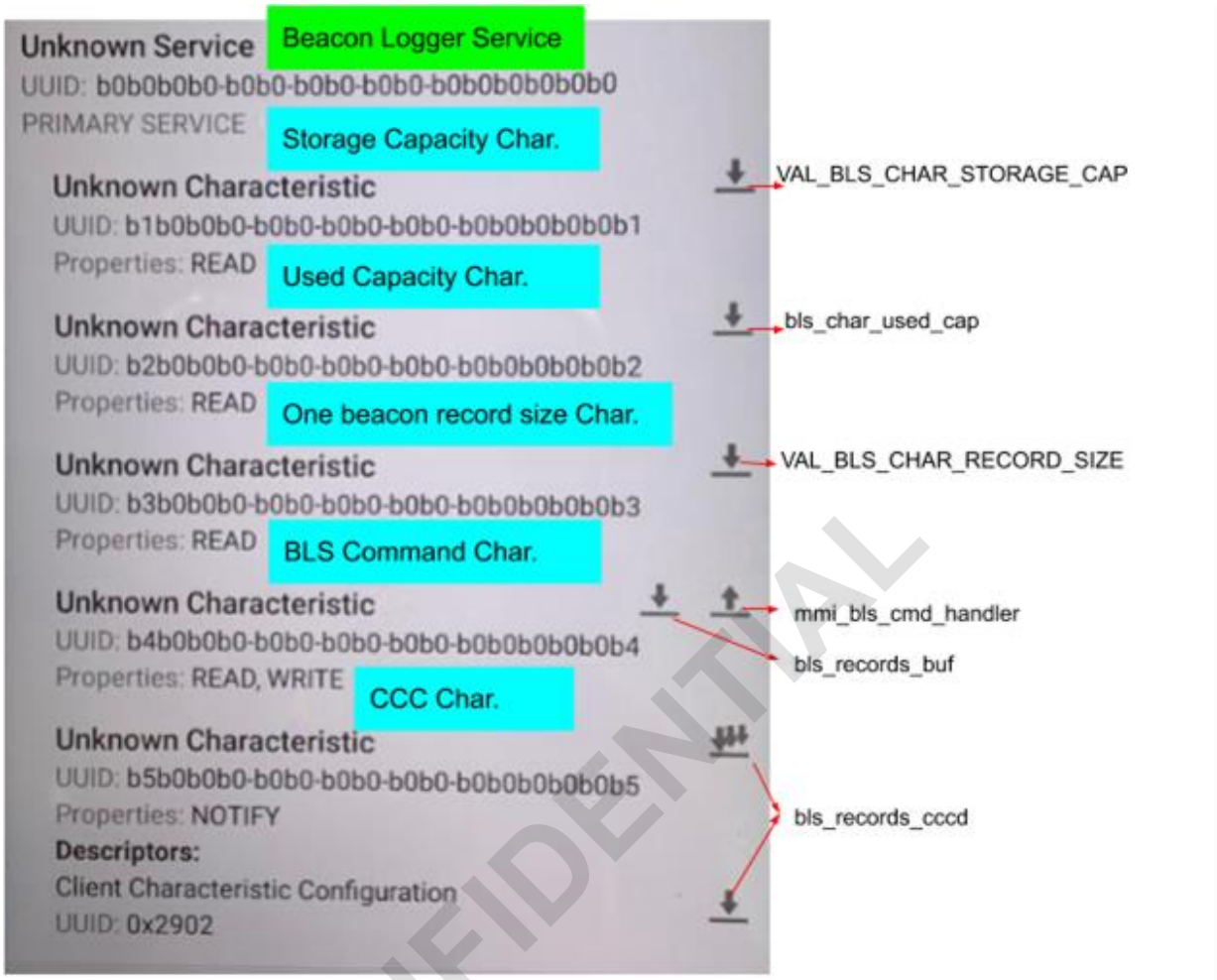


Figure 44 - Activation and Advertising Parameter Service

See Figure 45 for Beacon Logger Service.

CONFIDENTIAL



CONFIDENTIAL

```
app_att_handle[BLS_SVC] = ble_atmprfs_create_add_svc(bls_svc_uuid,[]
    SEC_PROFLE_LEVEL);

app_att_handle[BLS_CHAR_STORAGE] = ble_atmprfs_add_char(bls_storage_uuid,
    BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CHAR_STORAGE_CAP);

app_att_handle[BLS_CHAR_USED] = ble_atmprfs_add_char(bls_used_uuid,
    BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CHAR_USED_CAP);

app_att_handle[BLS_CHAR_RECORD_SIZE] =
    ble_atmprfs_add_char(bls_record_size_uuid,
    BLE_ATT_READ_NO_SECURITY, SIZE_BLS_CAHR_RECORDSIZE);

app_att_handle[BLS_CHAR_CMD] = ble_atmprfs_add_char(bls_cmd_uuid,
    APP_BLS_CHAR_CMD_PROPERTY, SIZE_BLS_CHAR_CMD);

app_att_handle[BLS_CHAR_RECORDS] = ble_atmprfs_add_char(bls_records_uuid,
    APP_BLS_CHAR_RECORD_PROPERTY, SIZE_BLS_CAHR_RECORDS);

app_att_handle[BLS_CHAR_RECORDS_CCCD] = ble_atmprfs_add_client_char_cfg();
```

Figure 45 - Beacon Logger Service

13.1 Command handler

Below are two characteristics to handle attribute write requests, see Figure 46 and Figure 47. The data buffer is 8 bytes defined in “APP_DATA_SIZE”. mmi_cfg_sub_command() API will handle the write request from Mobile APP.

BLS Command Characteristic:

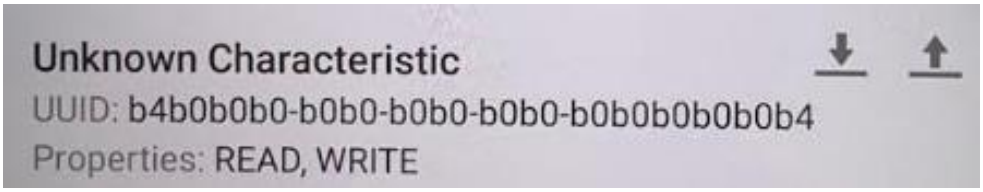


Figure 46 - BLS Command Characteristic

Cfg. Command Characteristic:

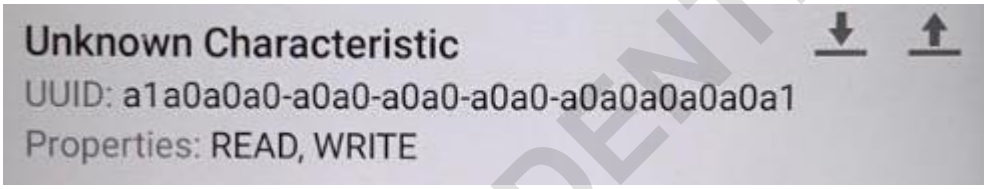


Figure 47 - Cfg. Command Characteristic

Use the “sub_cmd_payload” data structure to pack the command. See Figure 48.

```
//·Command·Payload
struct·sub_cmd_payload·{
····uint8_t·command;
····union·{
»       struct·tag_cfg_field·cfg_tag;
»       struct·beacon_log_cfg_field·cfg_b_logger;
»       struct·up_reftime_cmd_field·cmd_up_ref_time;
»       struct·retrieve_cmd_field·cmd_retrieve;
····};
····sub_cmd_vale·sub_cmd;
}·__PACKED;
```

Figure 48 - Common Command Payload Format

The command field is always “0x00” defined in “APP_TAG_CFG” and uses a sub_cmd field to identify the command purpose. See Figure 49 and Table 11.

CONFIDENTIAL

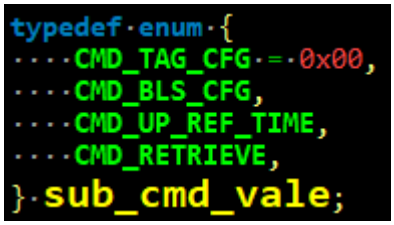


Figure 49 - Sub Command Value

CMD_TAG_CFG is for Cfg Command Characteristic and CMD_BLS_CFG, CMD_UP_REF_TIME and CMD_RETRIEVE are for BLS Command Characteristic.

Table 11 - Sub Command Table

Sub Command	Purpose	CMD Raw Buffer Example
CMD_TAG_CFG	Let Mobile APP to overwrite: app_env.config.act_status app_env.config.en_enc app_env.config.adv_interval app_env.config.tag_type Refer to Table 8 - Tag ID 0xAB - APP_CONFIG Flash NVDS Settings	66 0B 00 // advertising interval unit: 100ms 00 // beacon type 30 // enable activation and encryption 00 00 // reserve 00 // sub-cmd - CMD_TAG_CFG
CMD_BLS_CFG	Let Mobile APP to overwrite: app_env.config.rssi_filter - app_env.config.proximity_interval app_env.config.scan_period app_env.config.scan_duration Refer to Table 8 - Tag ID 0xAB - APP_CONFIG Flash NVDS Settings	66 98 // rssi filter 05 // proximity interval 0A // scan period 0A 00 // scan duration 00 // reserve 01 // sub-cmd - CMD_BLS_CFG
CMD_UP_REF_TIME	Let Mobile APP to update system time	66 Xx xx xx xx // new system time 00 00 // reserve 02 // sub-cmd - CMD_UP_REF_TIME
CMD_RETRIEVE	Let Mobile APP to retrieve beacon logger recorded in flash sector	66 00 00 00 00 00 00 // reserve 03 // sub-cmd - CMD_RETRIEVE

CONFIDENTIAL

The command raw buffer can be saved as a profile in nRF connect APP for testing. See Figure 50.

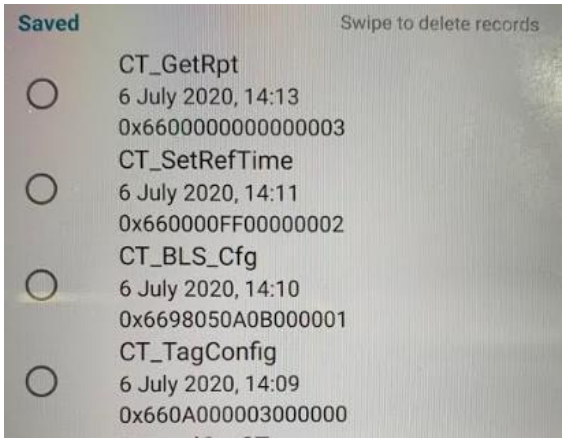


Figure 50 - Command Profile

CONFIDENTIAL

13.2 Software Real Time Clock

The Application will have one second timer and counter update into sec_cnt of CT_scan.c.

When the device enters hibernation, the device will retain one second counter and current system clock time. The Application will restore them after waking up from hibernation. The Application can adjust the sec_cnt to compensate for the duration of hibernation. Twelve hours (defined in INTERVAL_HIB_SEC) is the default value that the device will wake up automatically in hibernation mode to update the one second counter. Mobile APP can use GATT service (submit CMD_UP_REF_TIME) to update absolute time to device, then the start_time of beacon logger will use this absolute time base. See Figure 51.

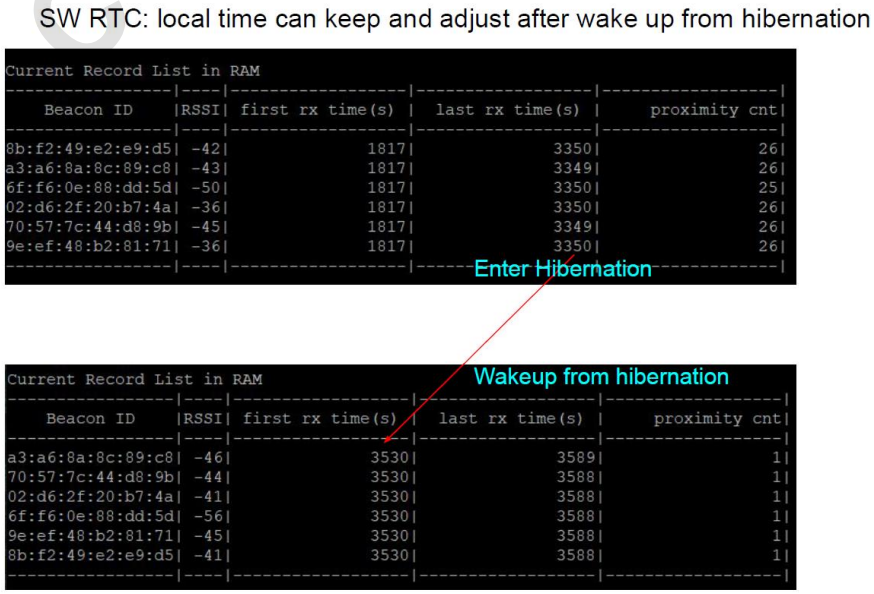


Figure 51 - Software Real Time Clock

13.3 Retrieve Beacon Logger

Beacon logger data saved in the flash sector can be retrieved by Mobile APP. Mobile APP needs to enable the notification property, then send CMD_RETRIEVE command. See Figure 52.

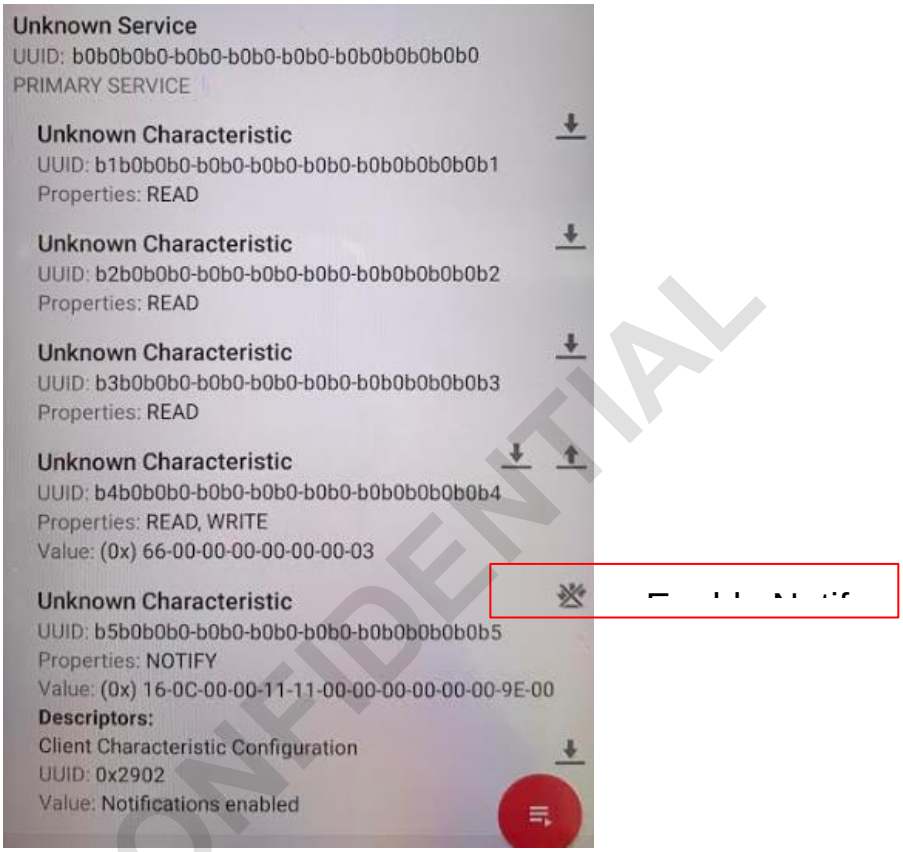


Figure 52 - Enable Notification Property

13.4 MTU size

To have better performance, Mobile APP can perform MTU exchange before sending CMD_RETRIEVE command. The MTU size is set to 259 defined in CFG_GAP_MAX_LL_MTU (param_gap.h). The MTU size should be larger than the buffer size used to send notification packets defined in SIZE_BLS_CAHR_RECORDS. To increase SIZE_BLS_CAHR_RECORDS, please also increase CFG_GAP_MAX_LL_MTU to gain better transmit performance. See Figure 53.

CONFIDENTIAL

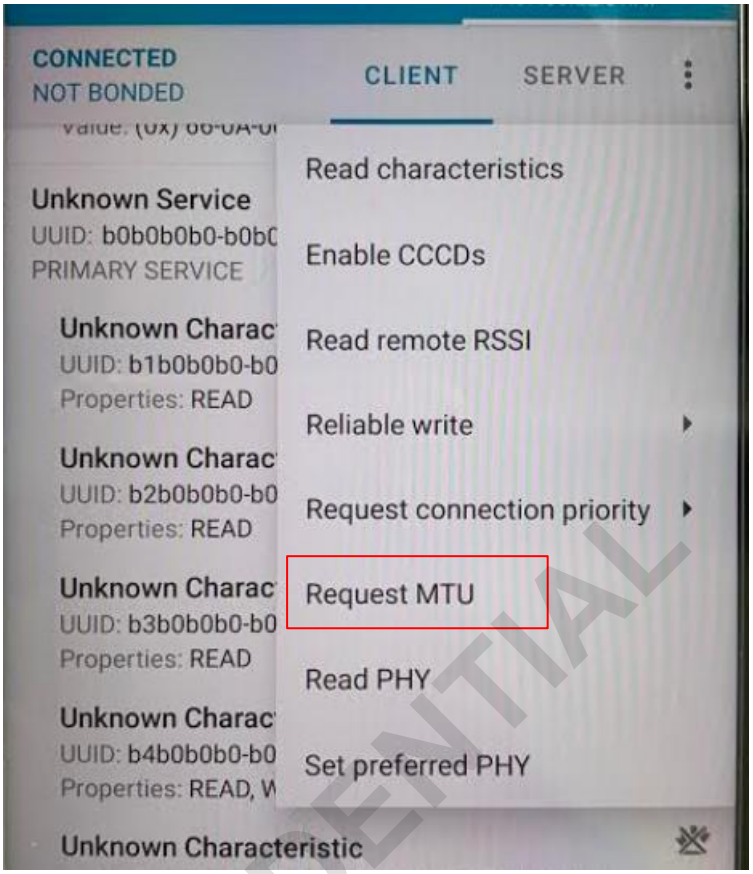


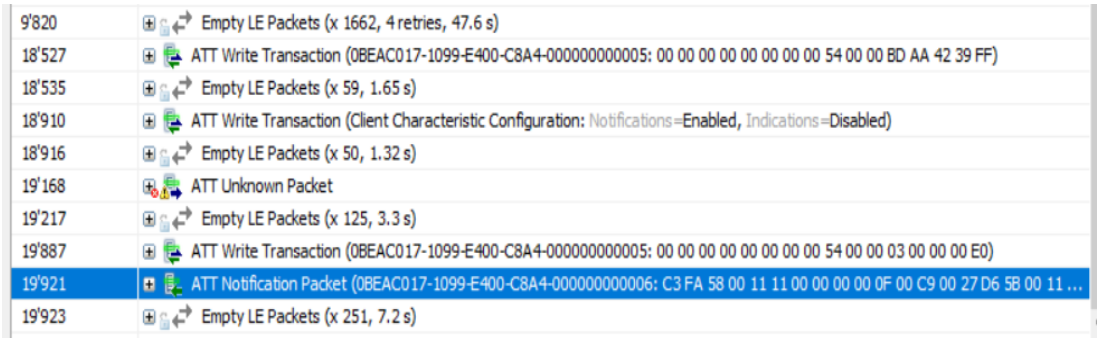
Figure 53 - Request MTU

CONFIDENTIAL

13.5 Notify Packet Format to Report Beacon Logger

`mmi_mv_flash_data_to_list()` API will unpack beacon logger data from the flash sector then pack into the notification packet. record_info structure is the format reporting beacon logger to Mobile APP using GATT notification, see Figure 54. The upper layer of example code will use 256 as buffer size defined in SIZE_BLS_CAHR_RECORDS and concatenate each beacon information, then call gatt_records_send() API to send notification packet.

Figure 54 is the notification packet buffer parser example:



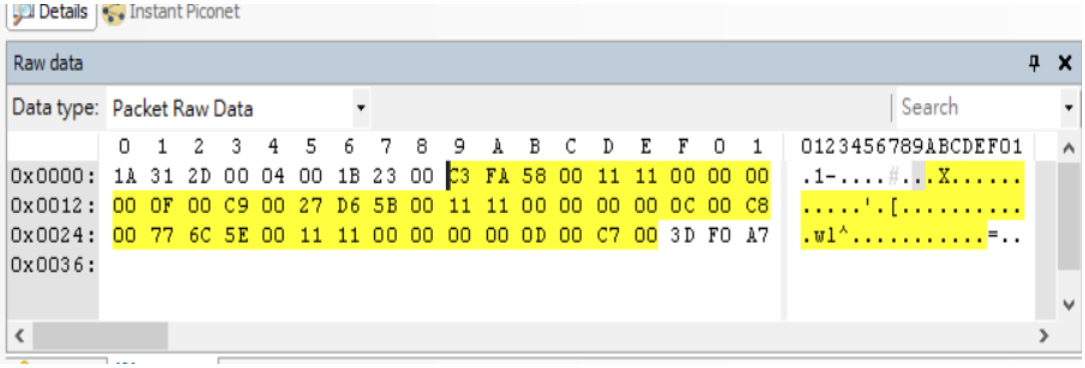


Figure 54 - Notification Beacon Report

The raw data in the notification buffer is “C3 FA 58 00 11 11 00 00 00 00 0F 00 C9 00 27 D6 5B 00 11 11 00 00 00 00 0C 00 C8 00 77 6C 5E 00 11 11 00 00 00 00 0D 00 C7 00”.

The Mobile APP can base on “Beacon report format” (Figure 55) to parse. Table 12 shows the beacon report parameters.

```
struct record_info {
    uint32_t start_time;
    uint8_t beacon_ID[6];
    uint16_t proxi_cnt;
    int8_t max_rssi;
    uint8_t beacon_type;
} __PACKED;
```

Figure 55 - Beacon Report Format

Table 12 - Beacon Report Parameters

Received time	Beacon ID	Proximity Count	Max. RSSI	Tag Type
C3 FA 58 00	11 11 00 00 00 00	0F 00	C9	00
27 D6 5B 00	11 11 00 00 00 00	0C 00	C8	00
77 6C 5E 00	11 11 00 00 00 00	0D 00	C7	00

Mobile APP can use proximity counter and proximity interval to calculate how long the user is nearby with Beacon ID owner.

14. Console Log for Beacon Record Dump

CONFIDENTIAL

Refer to MMI event and behavior (event #3-2 and #3-3) to trigger the dump process. There will be four "\n" characteristics at the front of the dump table, and two "\n" characteristics at the end of the dump table.

14.1 Show Current Record List in RAM

Print out the record list per 30s until the user triggers the event #3-2 (short click) to stop this periodic report. See Figure 56.

```
Current Record List in RAM
-----------------|----|------------------|------------------|------------------|
    Beacon ID    |RSSI| first rx time(s) |  last rx time(s) |     proximity cnt|
-----------------|----|------------------|------------------|------------------|
ff:e0:11:11:11:1a| -46|               167|               220|                 4|
ff:e8:81:44:77:89| -46|               140|               229|                10|
ff:e0:11:11:11:1a| -48|               140|               149|                 1|
ff:e0:11:11:11:14| -56|               140|               229|                10|
ff:e0:11:44:77:89| -47|               140|               229|                10|
00:90:11:44:77:89| -48|               139|               221|                 9|
-----------------|----|------------------|------------------|------------------|
```

Figure 56 - Show Current Record List in RAM

14.2 Retrieve Flash Record Beacon List

1) Each 4 k flash sector can store ~150 beacon log entries.
2) For testing, the user can use the event #3-3 to trigger flushing record beacon flash sectors (enter connectable advertising stage). After performing this flash record dump, it will erase flash sectors that are used by beacon loggers.
3) proximity cnt: Indicates how long will stay near the proximity tag (unit is scan period, in Figure 57 scan period is 10 seconds.)

CONFIDENTIAL

Retrieve Flash Record Beacon List

Flash Sector Offset	Beacon ID	RSSI	first rx time(s)	last rx time(s)	proximity cnt
0x28000	ff:e0:11:11:11:1a	-48	256	256	0
0x28000	00:90:11:44:77:89	-48	274	274	0
0x28000	ff:e0:11:44:77:89	-48	274	274	0
0x28000	ff:e8:81:44:77:89	-48	273	273	0
0x28000	ff:e0:11:11:11:14	-57	274	274	0
0x28000	ff:e0:11:11:11:1a	-48	50	256	20
0x28000	ff:e0:11:44:77:89	-48	32	274	25
0x28000	ff:e8:81:44:77:89	-48	32	273	26
0x28000	ff:e0:11:11:11:14	-58	32	274	26
0x28000	00:90:11:44:77:89	-48	31	274	25
0x29000	ff:e0:11:11:11:1a	-53	429	429	0
0x29000	00:90:11:44:77:89	-53	411	411	0
0x29000	ff:e0:11:44:77:89	-52	429	429	0
0x29000	ff:e8:81:44:77:89	-52	429	429	0
0x29000	ff:e0:11:11:11:14	-60	429	429	0
0x29000	ff:e0:11:11:11:1a	-53	322	429	8
0x29000	ff:e0:11:44:77:89	-49	312	429	12
0x29000	00:90:11:44:77:89	-51	313	411	10
0x29000	ff:e8:81:44:77:89	-50	312	429	13
0x29000	ff:e0:11:11:11:14	-58	312	429	13
0x2a000	ff:e0:11:11:11:1a	-50	567	567	0
0x2a000	00:90:11:44:77:89	-50	567	567	0
0x2a000	ff:e0:11:11:11:14	-60	567	567	0
0x2a000	ff:e8:81:44:77:89	-50	567	567	0
0x2a000	ff:e0:11:44:77:89	-51	567	567	0
0x2a000	ff:e0:11:11:11:1a	-50	468	567	8
0x2a000	00:90:11:44:77:89	-50	468	567	9
0x2a000	ff:e0:11:11:11:14	-61	468	567	10
0x2a000	ff:e8:81:44:77:89	-51	469	567	11
0x2a000	ff:e0:11:44:77:89	-54	469	567	10
0x2b000	ff:e0:11:11:11:1a	-49	681	681	0
0x2b000	00:90:11:44:77:89	-49	672	672	0
0x2b000	ff:e8:81:44:77:89	-49	681	681	0
0x2b000	ff:e0:11:44:77:89	-50	680	680	0
0x2b000	ff:e0:11:11:11:14	-58	680	680	0
0x2b000	ff:e0:11:11:11:1a	-49	609	681	4
0x2b000	ff:e8:81:44:77:89	-49	600	681	9
0x2b000	ff:e0:11:44:77:89	-50	599	680	8
0x2b000	ff:e0:11:11:11:14	-60	600	680	9
0x2b000	00:90:11:44:77:89	-49	600	672	8
0x2c000	00:90:11:44:77:89	-48	856	856	0
0x2c000	ff:e0:11:44:77:89	-50	855	855	0
0x2c000	ff:e0:11:11:11:14	-56	855	855	0
0x2c000	ff:e8:81:44:77:89	-48	855	855	0
0x2c000	00:90:11:44:77:89	-48	713	856	14
0x2c000	ff:e0:11:11:11:1a	-48	712	820	7
0x2c000	ff:e0:11:44:77:89	-50	712	855	16
0x2c000	ff:e0:11:11:11:14	-58	712	855	16
0x2c000	ff:e8:81:44:77:89	-48	712	855	16
0x2d000	ff:e0:11:11:11:1a	-49	1061	1061	0
0x2d000	00:90:11:44:77:89	-48	1061	1061	0
0x2d000	ff:e0:11:11:11:14	-58	1060	1060	0
0x2d000	ff:e0:11:44:77:89	-50	1061	1061	0
0x2d000	ff:e8:81:44:77:89	-49	1060	1060	0
0x2d000	ff:e0:11:11:11:14	-59	899	1060	18
0x2d000	ff:e0:11:11:11:1a	-49	891	1061	14
0x2d000	00:90:11:44:77:89	-48	891	1061	17
0x2d000	ff:e0:11:44:77:89	-52	890	1061	19
0x2d000	ff:e8:81:44:77:89	-49	890	1060	19
0x2e000	ff:e0:11:11:11:1a	-54	1094	1103	1
0x2e000	ff:e8:81:44:77:89	-53	1094	1103	1
0x2e000	ff:e0:11:11:11:14	-61	1094	1103	1
0x2e000	ff:e0:11:44:77:89	-53	1094	1103	1
0x2e000	00:90:11:44:77:89	-52	1094	1103	1

Figure 57 - Retrieve Flash Record Beacon List

14.3 Leave and Return - Nearby Timeout Case

Beacon ID[9e:ef:48] leave and return after Nearby Timeout (2*scan period = 2*60 = 120 secs), which will create new record entry for Beacon ID[9e:ef:48]

Nearby timeout configuration: (in CT_scan.c)
#define NEARBY_TIME_MUL (2) //unit: scan period
See Figure 58.

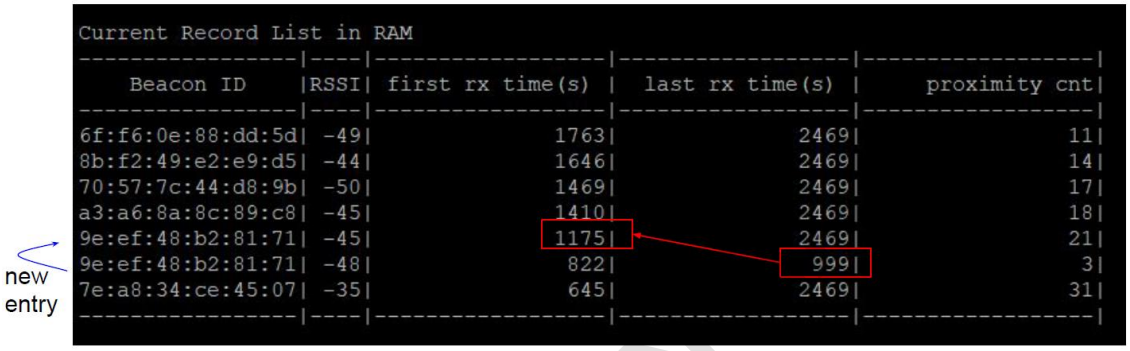


Figure 58 - Leave and Return - Nearby Timeout Case

14.4 Leave and Return - Still In Nearby Timeout

Beacon ID[6f:f6:0e] leave and return quickly within nearby timeout. It will not create new record entry. It's proximity counter will be less than others. See Figure 59.

Figure 59 - Leave and Return - Still in Nearby Timeout Case

15. OTA

15.1 Enable Atmosic OTA Service

- Add OTAPS module into PROFILES of the makefile.
- Add ble_otps module into FRAMEWORK_MODULES of the makefile.
- PROFILES += DISS BASS OTAPS
- FRAMEWORK_MODULES += app_gap … ble_otaps

15.2 ATM2202 Flash Layout

The makefile will check if enabling “Atmosic OTA Service” then switch to use the flash layout for OTA. See Figure 60.

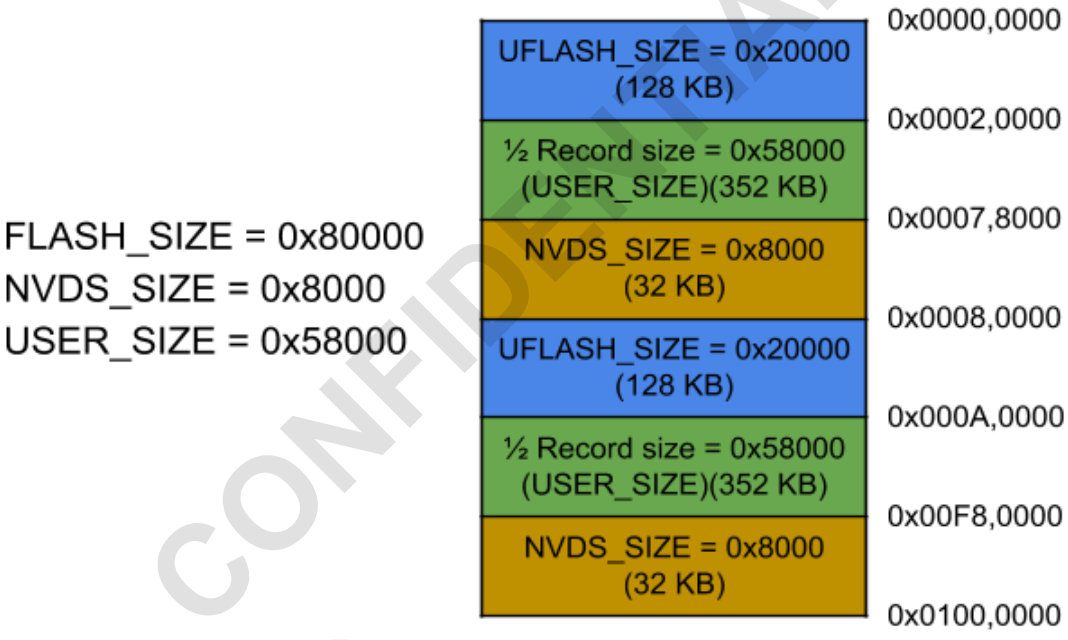


Figure 60 - Flash Layout for OTA

CONFIDENTIAL

15.3 Build Firmware for EVK

Build and download OTA-enabled firmware:

make run_all ERASE_UPGRADE_DATA=1 BOARD=m2202
CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1

Note: Will have the following console message - “Erasing upgd sector …”

```
Erasing upgd sector at 0x00080000 to 0x00081000
```

Build OTA Image for APP:

make clean
make build_flash_nvds
make build_archive BOARD=m2202 CFG_GPIO_MMI_BTN_ACTIVE_LOW:=1

Note: CFG_xxx environment variables depend on your hardware board

15.4 SW Virtual Record Pool for OTA

See Figure 61.

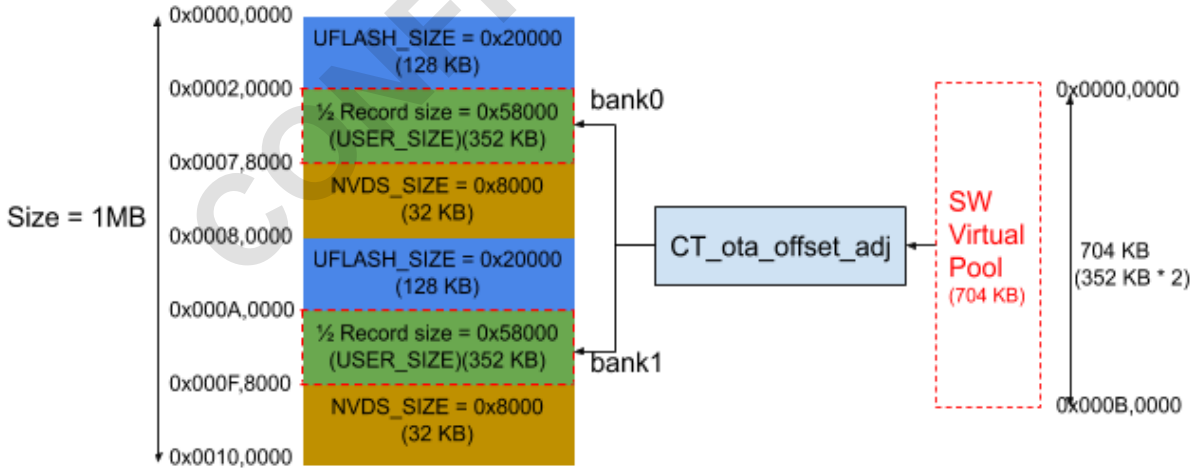


Figure 61 - SW Virtual Record Pool for OTA

CONFIDENTIAL

Revision History

Date	Version	Description
May 13, 2022	0.60	Updated for SDK 5.1.0 release.
April 14, 2021	0.54	Undated format, no content change.
March 29, 2021	0.53	Undated Quick Start section.
December 2, 2020	0.52	Corrected typos.
November 27, 2020	0.51	Corrected typos.
November 23, 2020	0.50	Initial version created.

CONFIDENTIAL

ATMOSIC TECHNOLOGIES – DISCLAIMER

This product document is intended to be a general informational aid and not a substitute for any literature or labeling accompanying your purchase of the Atmosic product. Atmosic reserves the right to amend its product literature at any time without notice and for any reason, including to improve product design or function. While Atmosic strives to make its documents accurate and current, Atmosic makes no warranty or representation that the information contained in this document is completely accurate, and Atmosic hereby disclaims (i) any and all liability for any errors or inaccuracies contained in any document or in any other product literature and any damages or lost profits resulting therefrom; (ii) any and all liability and responsibility for any action you take or fail to take based on the information contained in this document; and (iii) any and all implied warranties which may attach to this document, including warranties of fitness for particular purpose, non-infringement and merchantability. Consequently, you assume all risk in your use of this document, the Atmosic product, and in any action you take or fail to take based upon the information in this document. Any statements in this document in regard to the suitability of an Atmosic product for certain types of applications are based on Atmosic's general knowledge of typical requirements in generic applications and are not binding statements about the suitability of Atmosic products for any particular application. It is your responsibility as the customer to validate that a particular Atmosic product is suitable for use in a particular application. All content in this document is proprietary, copyrighted, and owned or licensed by Atmosic, and any unauthorized use of content or trademarks contained herein is strictly prohibited.

Copyright ©2020 - 2022 by Atmosic Technologies. All rights reserved. Atmosic logo is a registered trademark of Atmosic Technologies Inc. All other trademarks are the properties of their respective holders.

CONFIDENTIAL

Atmosic™

Atmosic Technologies | 2105 S. Bascom Ave. | Campbell CA, 95008
www.atmosic.com