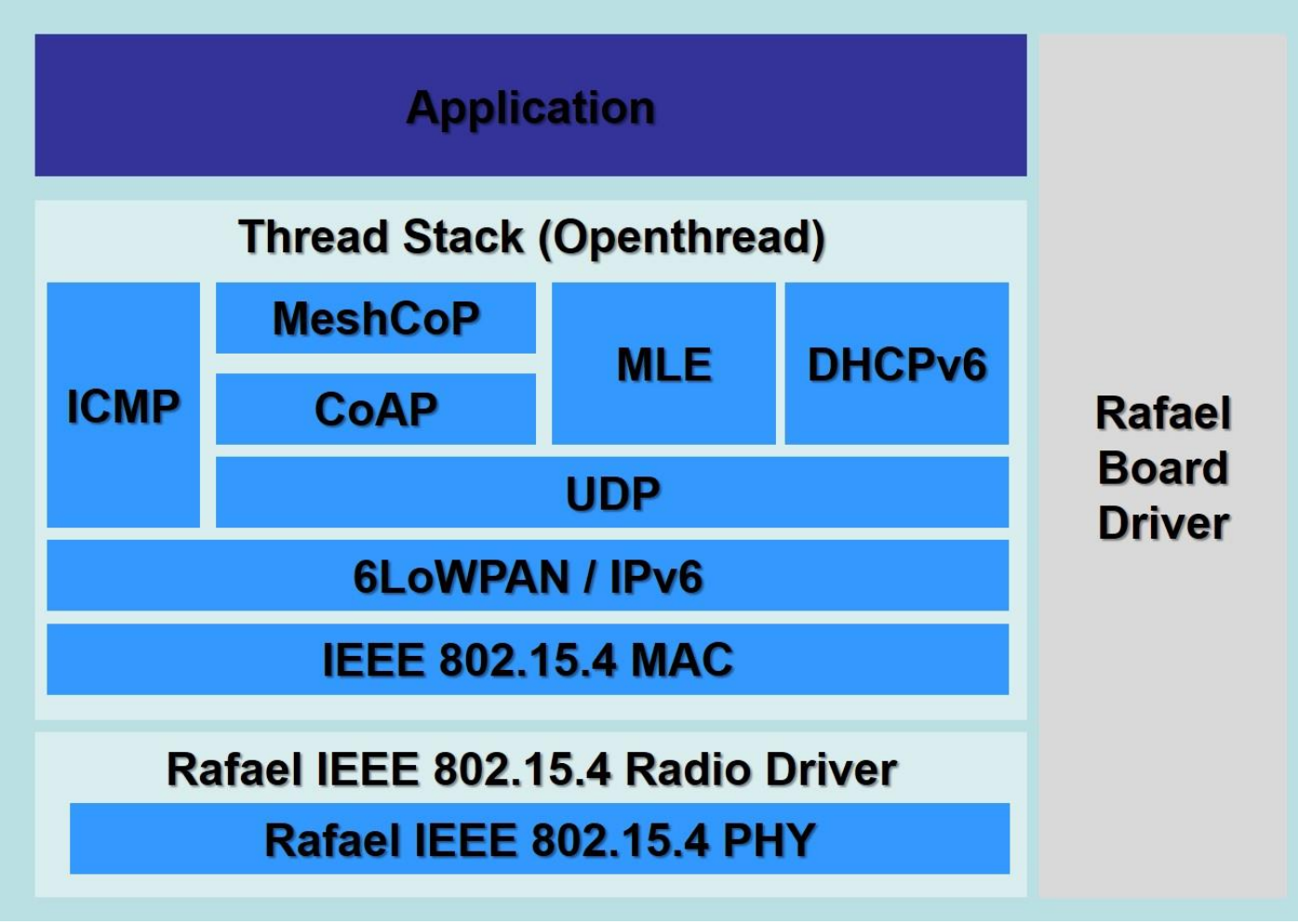
RT58x Thread SDK User Guide V1.3

Table of Contents

1. Introduction

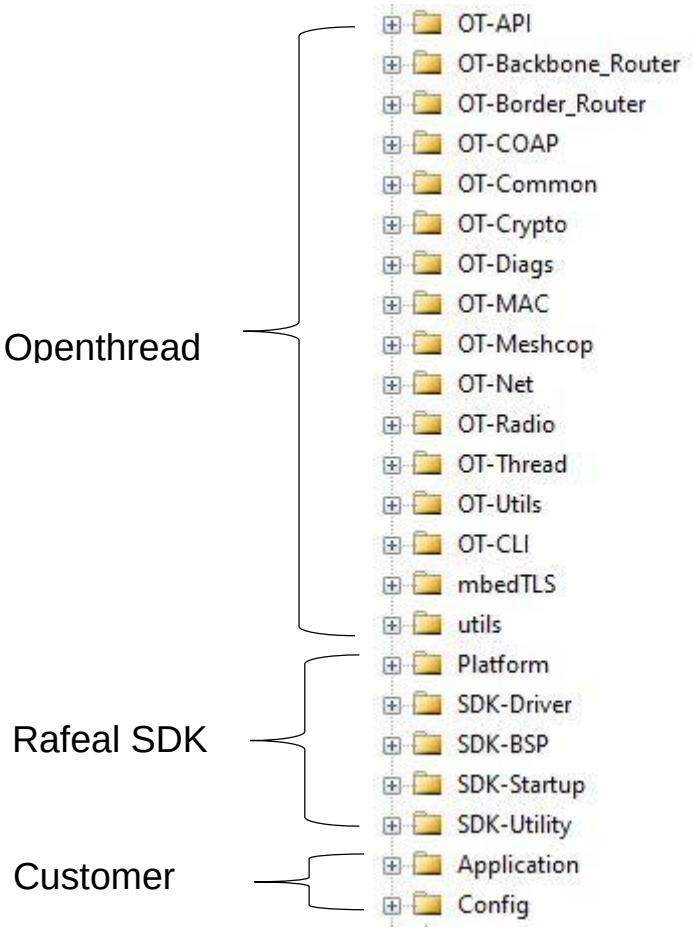
This document describes the interface provided by the Rafael Thread Stack Library. It includes a reference software and supporting library for the Rafael Thread SDK, which combines the open-source project Openthread with Rafael IEEE 802.15.4 and Rafael Driver.
The Rafael Thread Library is easy to integrate with Rafael RT58x series SDK.

2. System architecture



3. Thread SDK introduction

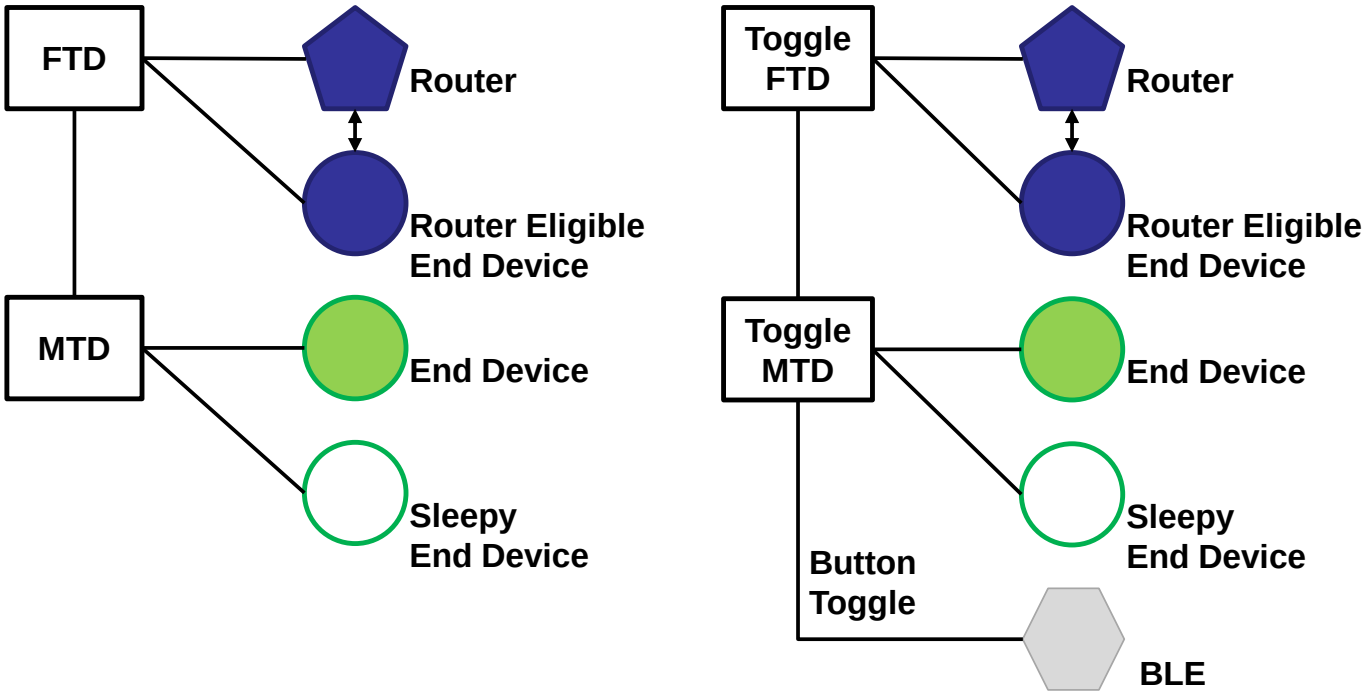
This section provides an overview of the file distribution within the Rafael Thread SDK specifically related to Keil operations.



4. Thread Application introduction

This section introduces the examples and APIs included in the Rafael Thread SDK.

4.1 Introduction to all examples



The following two applications use the 2.4GHz band at the physical layer and the Thread protocol at the network layer, providing an API for user development. The FTD mode can function as a router, while the MTD mode serves as an end device or a sleepy end device.

- Project\Application\Thread\Thread2P4G\Thread2P4G_FTD
- Project\Application\Thread\Thread2P4G\Thread2P4G_MTD

The following four applications use the SubGHz band at the physical layer and the Thread protocol at the network layer, providing an API for user development. The FTD mode can function as a router, while the MTD mode is used for end devices or sleepy end devices. The switch example supports toggling between 2.4GHz BLE and SubGHz over Thread via a button.

- Project\Application\Thread\ThreadSubG\ThreadSubG_FTD
- Project\Application\Thread\ThreadSubG\ThreadSubG_MTD
- Project\Application\Thread\ThreadSubG\ToggleSubGBLE_FTD
- Project\Application\Thread\ThreadSubG\ToggleSubGBLE_MTD

4.2 Initaial

This function initializes the parameters of a thread network.

- app_task_network_configuration_setting

This function initializes the parameters of the sleep node.

- app_sleep_init

This function configures the UDP Sock settings.

- app_sock_init

This function initializes the OTA.

- ota_init

4.3 Command register

This function registers the command

- static const otCliCommand kCommands[] = {name, callback}

1. *name* : registers command name
2. *callback*: command callback

4.4 application process action

This function includes the process of OpenThread tasks and the process of Rafael's driver.

- app_task_process_action
  - otTaskletsProcess
  - otSysProcessDrivers

Customers can independently add their own task-processing procedures.

4.5 App UART API

App uart data send

- int app_uart_data_send(uint8_t u_port, uint8_t *p_data, uint16_t data_len)

1. *port* : Please fill in 1 or 2.
2. *p_data* : user data buffer.
3. *data_len* : user data length.
4. *return* : 1=send fail, 0=send success.

App uart data received

- void app_uart1_recv()

1. Function to read from UART1.

- void app_uart2_recv()

1. Function to read from UART2.

4.6 App UDP API

App UDP data send

- otError app_udp_send(otIp6Address dst_addr, uint8_t *data, uint16_t data_lens)

1. *dst_addr*: please fill destination IPv6.
2. data : user data buffer.
3. data_lens : user data length.
4. *return* : (1~255)=send fail, 0=send success.

App UDP data received

- void void app_udp_receive_handler(void *aContext, otMessage *aMessage, const otMessageInfo *aMessageInfo)

1. Notify when data is received via UDP.

4.7 Software timer (When FreeRTOS is not used)

Software timer create

- sw_timer_t *sw_timer_create(const char *name, uint32_t period, uint32_t auto_reload, uint32_t execute_mode, void *cb_param, sw_timer_cb cb_function)

1. *name* : A human readable text name that is assigned to the timer.
2. *period* : The period of the timer.
3. *auto_reload* : If auto_reload is set to TRUE, then the timer will expire repeatedly with a frequency set by the period parameter. If auto_reload is set to FALSE, then the timer will be a one-shot and enter the dormant state after it expires.
4. *execute_mode* : The execute mode of the timer.
5. *cb_param* : The input parameter of the call back function.
6. *cb_function* : The function to call when the timer expires

Software timer start

- sw_timer_err_t sw_timer_start(sw_timer_t *timer)

1. *timer* : The handle of the timer being started/restarted.

Software timer stop

- sw_timer_err_t sw_timer_stop(sw_timer_t *timer)

1. *timer* : The handle of the timer being stopped.

Software timer reset

- sw_timer_err_t sw_timer_reset(sw_timer_t *timer)

1. *timer* : The handle of the timer being reset/started/restarted.

Software timer change period

- sw_timer_err_t sw_timer_change_period(sw_timer_t *timer, uint32_t period)

1. *timer* : The handle of the timer that is having its period changed.
2. *period* : The new period for the timer.

Software timer change execute mode

- sw_timer_err_t sw_timer_change_execute_mode(sw_timer_t *timer, uint32_t execute_mode)

1. *timer* : The handle of the timer that has its execute mode.
2. *execute_mode*: The new execute mode for the timer.

Software timer delete

- sw_timer_err_t sw_timer_delete(sw_timer_t *timer)

1. *timer* : The handle of the timer being deleted.

Software timer get running

- bool sw_timer_get_running(sw_timer_t *timer)

1. *timer* : The timer beging queried.
2. *return* : ture= is active. false = is dormant.

4.8 Memory Manage

Memory allocation

- void * mem_malloc (uint32_t u32_size)

5. *u32_size* : memory size in bytes.
6. *return* : NULL=allocate fail, memory pointer=allocate success.

Memory Free

- void mem_free (void * ptr)

1. *ptr* : allocate memory pointer.

Memory Copy

- void mem_memcpy (void *dest_ptr, void* src_ptr, uint32_t lens)

1. *dest_ptr* : target copy memory pointer.
2. *src_ptr* : original memory pointer that is being copied.
3. *lens* : copy size.

4.9 Openthread API

Please refer to the OpenThread API.

4.10 FreeRtos API

Please refer to the FreeRtos API.

5. Commonly commands

5.1 Openthread Commands

state

Return state of current state.

```
> state
offline, disabled, detached, child, router or leader
Done
```

channel

Get the IEEE 802.15.4 Channel value.

Note: SUG-G 915M(1-10), 868M(1-14), 433M(1-4), 470M (1-20), 2.4G(11-26)

```
> channel
11
Done
```

SUG-G			
915M	868M	433M	470M
1. 920000 KHz	1. 863000 KHz	1. 433000 KHz	1. 470000 KHz
2. 920500 KHz	2. 863500 KHz	2. 433500 KHz	2. 472000 KHz
3. 921000 KHz	3. 864000 KHz	3. 434000 KHz	3. 474000 KHz
4. 921500 KHz	4. 864500 KHz	4. 434500 KHz	4. 476000 KHz
5. 922000 KHz	5. 865000 KHz		5. 478000 KHz
6. 922500 KHz	6. 865500 KHz		6. 480000 KHz
7. 923000 KHz	7. 866000 KHz		7. 482000 KHz
8. 923500 KHz	8. 866500 KHz		8. 484000 KHz
9. 924000 KHz	9. 867000 KHz		9. 486000 KHz
10. 924500 KHz	10. 867500 KHz		10. 488000 KHz
	11. 868000 KHz		11. 490000 KHz
	12. 868500 KHz		12. 492000 KHz
	13. 869000 KHz		13. 494000 KHz
	14. 869500 KHz		14. 496000 KHz
			15. 498000 KHz
			16. 500000 KHz
			17. 502000 KHz
			18. 504000 KHz
			19. 506000 KHz
			20. 508000 KHz
2.4G			
11. 2405 MHz	15. 2425 MHz	19. 2445 MHz	23. 2465 MHz
12. 2410 MHz	16. 2430 MHz	20. 2450 MHz	24. 2470 MHz
13. 2415 MHz	17. 2435 MHz	21. 2455 MHz	25. 2475 MHz
14. 2420 MHz	18. 2440 MHz	22. 2460 MHz	26. 2480 MHZ

panid

Get the IEEE 802.15.4 PAN ID value.

```
> panid
0xdead
Done
```

networkkey

Get the Thread Network Key value.

```
> networkkey
00112233445566778899aabbccddeeff
Done
```

thread start

Enable Thread protocol operation and attach to a Thread network.

```
> thread start
Done
```

thread stop

Disable Thread protocol operation and detach from a Thread network.

```
> thread stop
Done
```

ipaddr

List all IPv6 addresses assigned to the Thread interface.

```
> ipaddr
fdde:ad00:beef:0:0:ff:fe00:0
fdde:ad00:beef:0:558:f56b:d688:799
fe80:0:0:0:f3d9:2a82:c8d8:fe43
Done
```

ping [async] [-I source] <ipaddr> [size] [count] [interval] [hoplimit] [timeout]

Send an ICMPv6 Echo Request.

async: Use the non-blocking mode. New commands are allowed before the ping process terminates.

source: The source IPv6 address of the echo request.

size: The number of data bytes to be sent ; Limit size: 1280 bytes.

count: The number of ICMPv6 Echo Requests to be sent.
interval: The interval between two consecutive ICMPv6 Echo Requests in seconds. The value may have fractional form, for example 0.5.
hoplimit: The hoplimit of ICMPv6 Echo Request to be sent.
timeout: Time in seconds to wait for the final ICMPv6 Echo Reply after sending out the request. The value may have fractional form.

```
> ping fd00:db8:0:0:76b:6a05:3ae9:a61a
> 16 bytes from fd00:db8:0:0:76b:6a05:3ae9:a61a: icmp_seq=5 hlim=64 time=0ms
1 packets transmitted, 1 packets received. Packet loss = 0.0%. Round-trip min/avg/max = 0/0.0/0 ms.
Done

> ping -I fd00:db8:0:0:76b:6a05:3ae9:a61a ff02::1 100 1 1 1
> 108 bytes from fd00:db8:0:0:f605:fb4b:d429:d59a: icmp_seq=4 hlim=64 time=7ms
1 packets transmitted, 1 packets received. Round-trip min/avg/max = 7/7.0/7 ms.
Done
```

Udp send <ip> <port> <message>
Send a UDP message.
ip: the destination address.
port: the UDP destination port.
message: the message to send ; Limit size: 640 characters.

```
> udp send fdde:ad00:beef:0:bb1:ebd6:ad10:f33 1234 hello
Done
```

For more details about OpenThread commands, please refer.

5.2 Rafael User Commands

ota

- Retrieve current OTA information.

```
> ota
ota state : idle
ota image version : 0x12345678
ota image size : 0x493e0
ota image crc : 0xabcdabcd
Done
```

ota start

- Start the network-wide update.

```
> ota start
Done
```

ota send <ipv6>

- Perform OTA update on a specific device.

```
> ota send fd00:db8:0:0:5038:3233:3245:4d37
Done
```

ota stop

- Stop OTA data transmission.

```
> ota stop
Done
```

ota debug

- Enable OTA debug logging: 0 to disable, 1 to enable.

```
> ota debug 1
Done
```

mem

- Display dynamic memory management information.

```
> mem
+------------+-------+-------------------------
|   Pointer      | Size    | Func-Line(Task)
+------------+-------+-------------------------
| 0x20011850 | 00032 | sw_timer_create-351(deadbeef)
| 0x20011878 | 00032 | sw_timer_create-351(deadbeef)
+------------+-------+-------------------------
Free Size : 0x0027A8(10152), Min 0x002708(9992)
Done
```

nwk (Supports only subg_ftd leader)

- reads network management information.

```
> index role parent rloc extaddr rssi
================================================
[1] child 0400 0401 5038323332454D37 -5 0
================================================
total num 1
Done
```

appudp <ipv6> <data>

- Application-side UDP transmission command.

```
> appudp fd00:db8:0:0:5038:3233:3245:4d37 123456
Done
```

6. Start Thread network

1. Prepare two boards with the flashed Example.
2. Open a terminal (Tera Term)
3. Connect to the used COM port with the following direct UART settings:
   - Baud rate: 115200
   - 8 data bits
   - 1 stop bit
   - No parity
   - Flow control: none
4. Check the board one role

```
> state
leader
Done
```

5. Check the board two role

```
> state
child
Done
```

6. Check the board two IP

```
> ipaddr
fdde:ad00:beef:0:0:ff:fe00:0
fdde:ad00:beef:0:558:f56b:d688:799
fe80:0:0:0:f3d9:2a82:c8d8:fe43
Done
```

7. Use board one to ping board two.

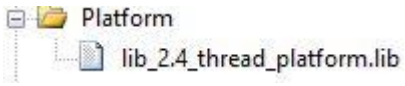
```
> ping fdde:ad00:beef:0:558:f56b:d688:799
> 16 bytes from ping fdde:ad00:beef:0:558:f56b:d688:799: icmp_seq=5 hlim=64
time=0ms
1 packets transmitted, 1 packets received. Packet loss = 0.0%. Round-trip
min/avg/max = 0/0.0/0 ms.
Done
```

8. To modify Thread network parameters, you can either use code to programmatically modify the parameters or use commands through the command line or configuration interface.

7. Configuration

7.1 Project Configuration

- PLAFFORM_CONFIG_ENABLE_SUBG
  If defined, enable SUG-G; otherwise, default to using 2.4GHz.
  Note: Use the corresponding library for the platform.



7.2 Main Configuration

- RFB_DATA_RATE
  Set SUG-G data rate value; supported Value: [FSK_50K; FSK_100K; FSK_150K; FSK_200K; FSK_300K].

- RFB_CCA_THRESHOLD
  Set clear channel assessment (CCA) threshold value; Default: 75 (-75 dBm)

- RFB_SUBG_FREQUENCY_BAND
  Set SubG frequency band value; Default: 0, (0: 915M, 1: 868M, 2: 433M, 3: 470M)

8. Sniffer

8.1 Download

- Python

```
https://www.python.org/downloads/release/python-379/
```

- Wireshark

```
https://www.wireshark.org/download.html
```

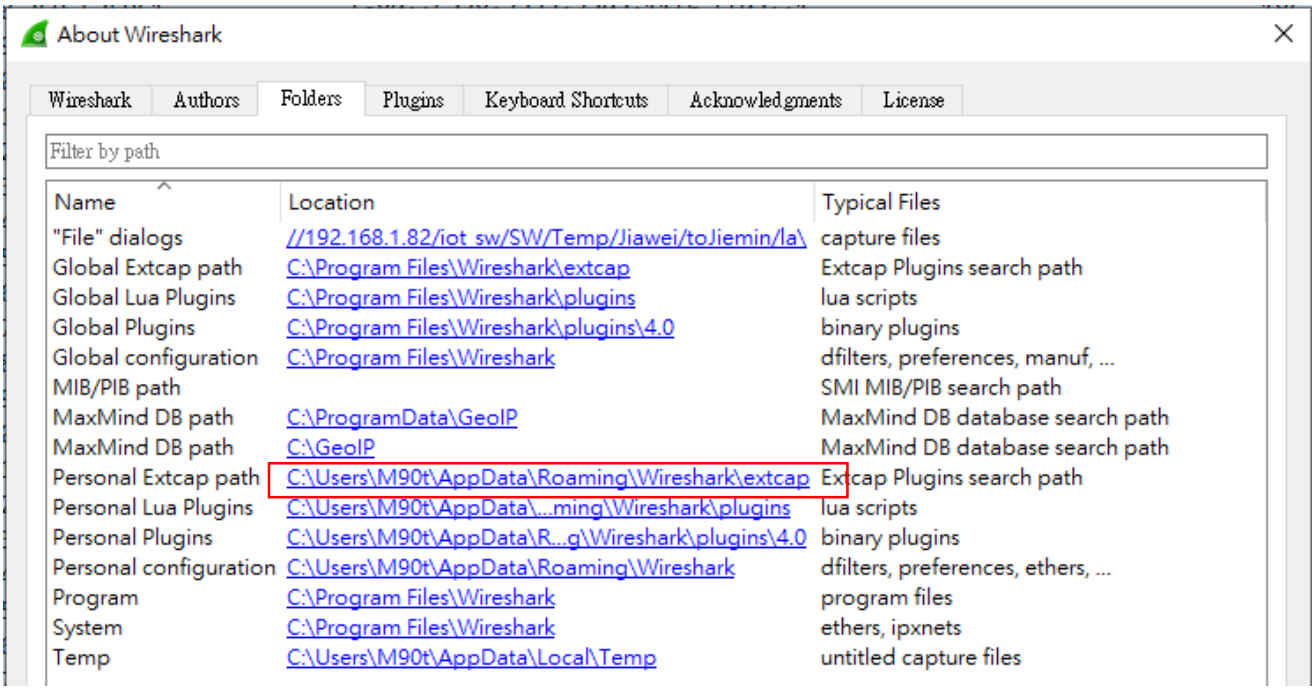
8.2 Download sniffer bin

The bin file in the path “pyspinel /RafaelMicroSinfferBin”.

8.3 Install pyspinel package

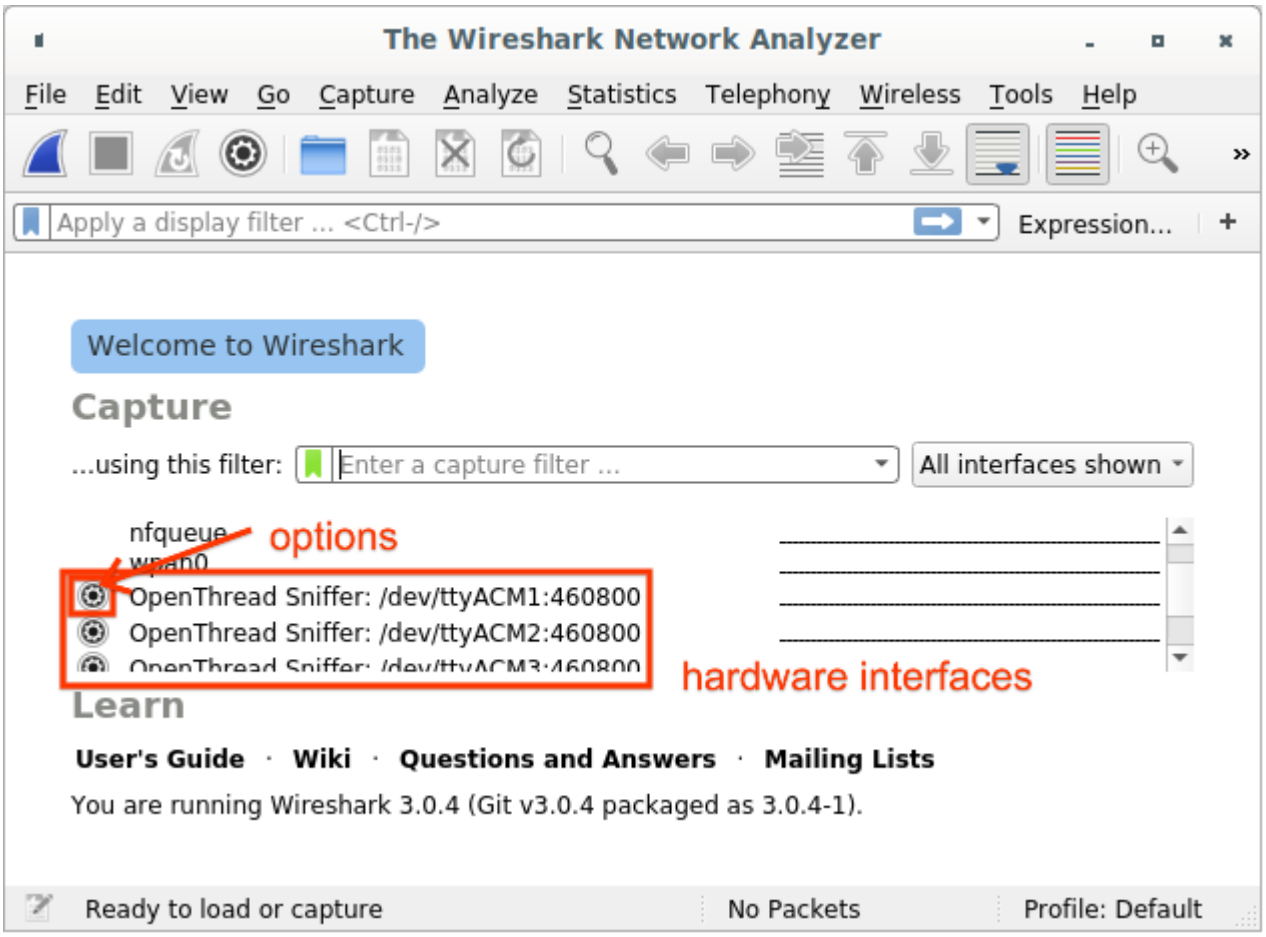
```
$ pip3 install pyspinel
```

Copy the provided **extcap_ot.py** and **extcap_ot.bat** to the extcap directory.



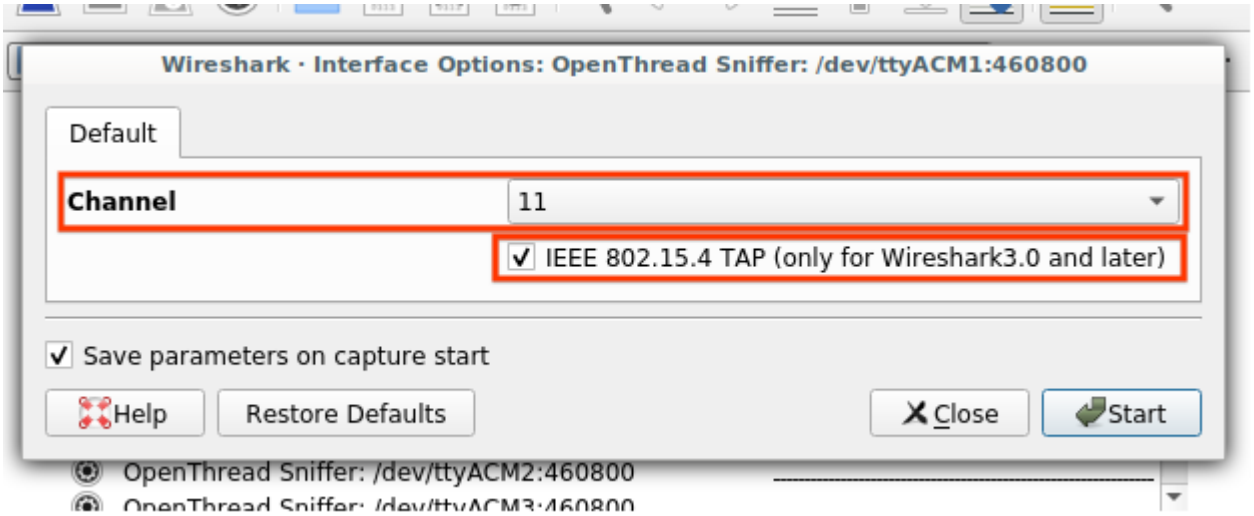
8.4 Interface in wireshark

If this is your first time using an interface, click the **Options** button to the left of the interface. Otherwise, “capture”->“refresh interfaces” to find the interface.



8.5 Options setting in wireshark

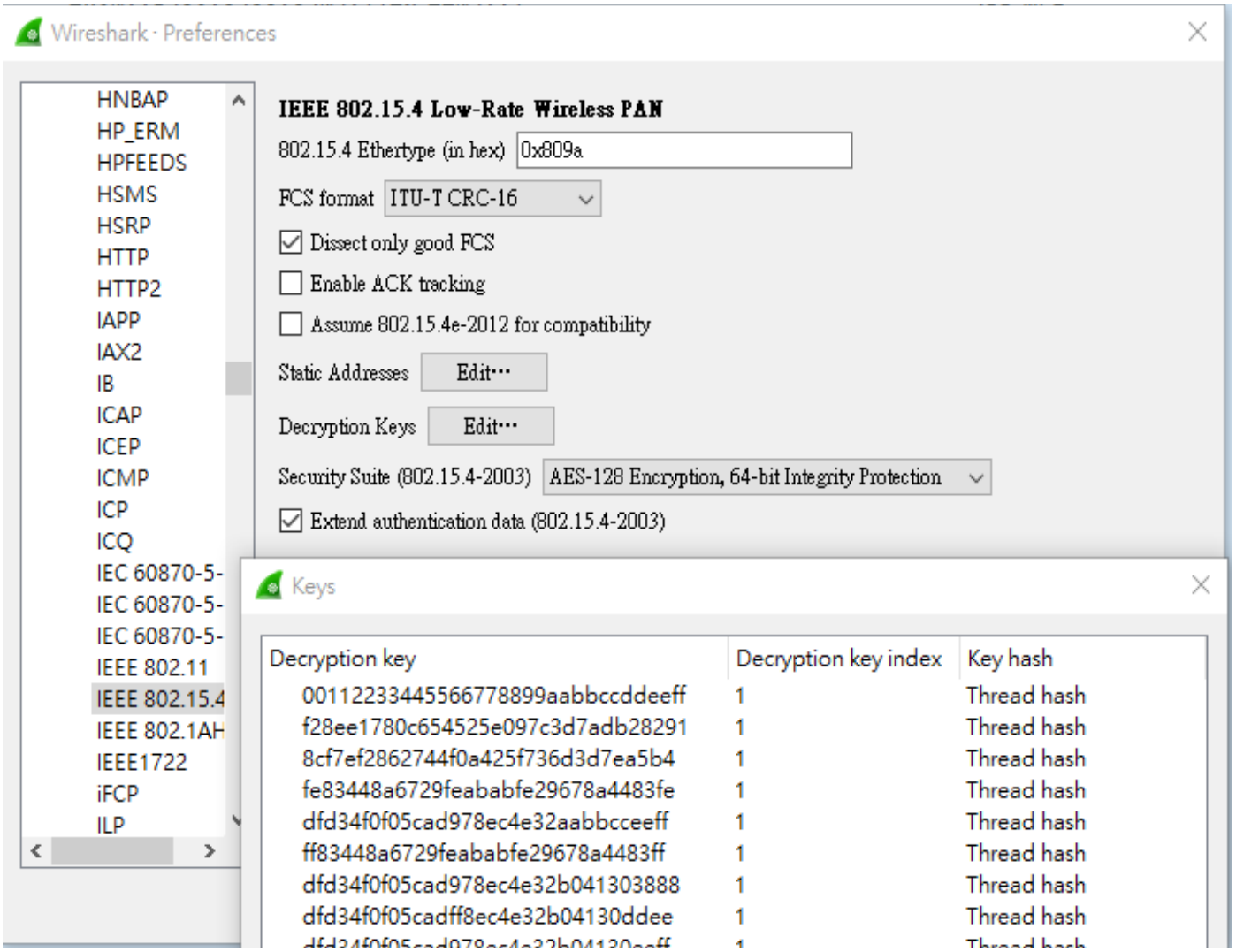
- Set the **Channel** to the desired value.
- Check **IEEE 802.15.4 TAP** to ensure that the channel information is included in the pcap output and can be displayed in the Wireshark GUI.
- Check **Save parameters on capture start** to ensure that these parameters are saved after the start of the capture, to avoid having to set it again the next time you use the interface (unless you need to change the channel).
- Click **Start**.



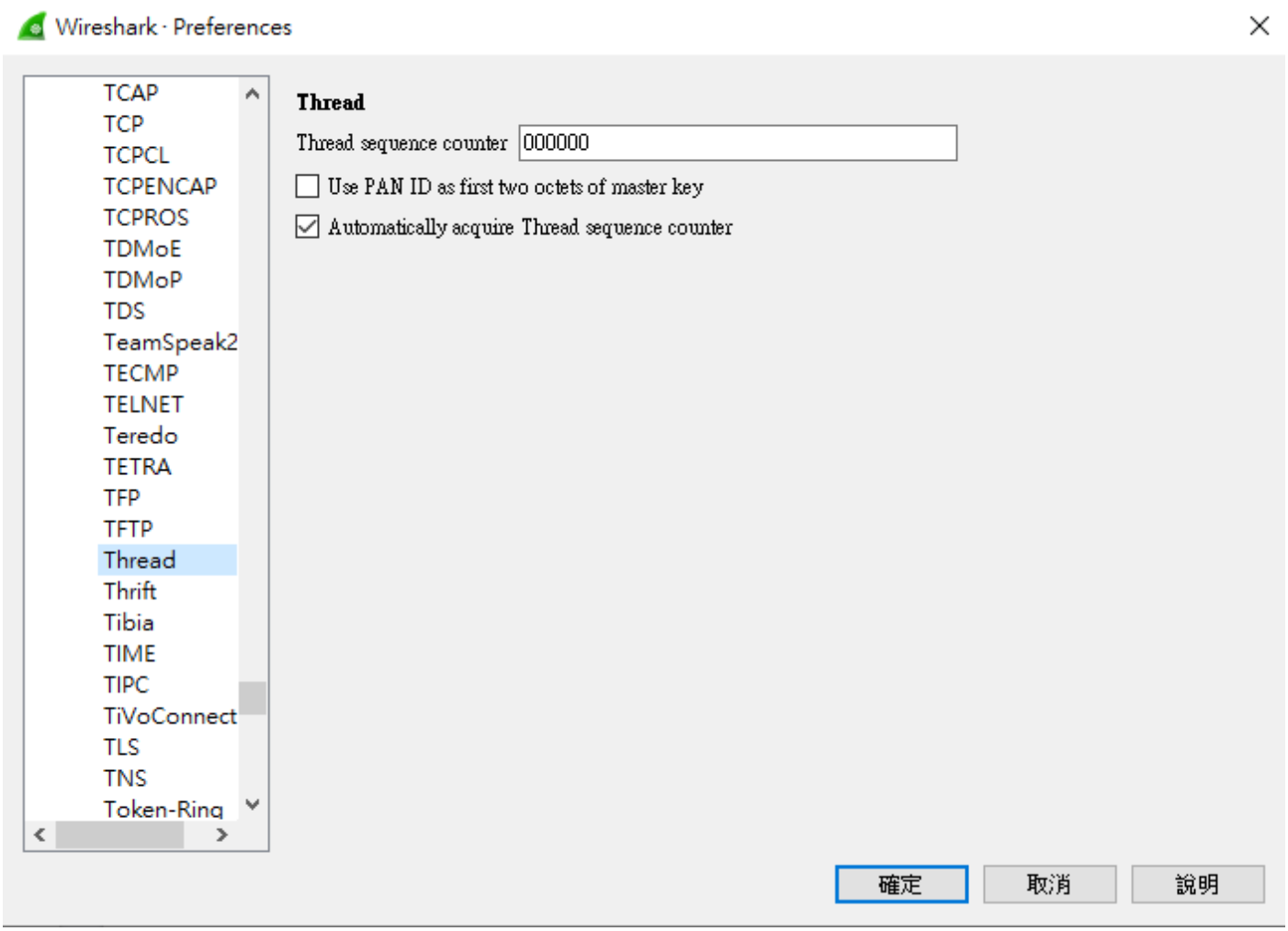
8.6 Thread configure setting in wireshark

To configure protocols, select Preferences... in Wireshark and expand the Protocols section.

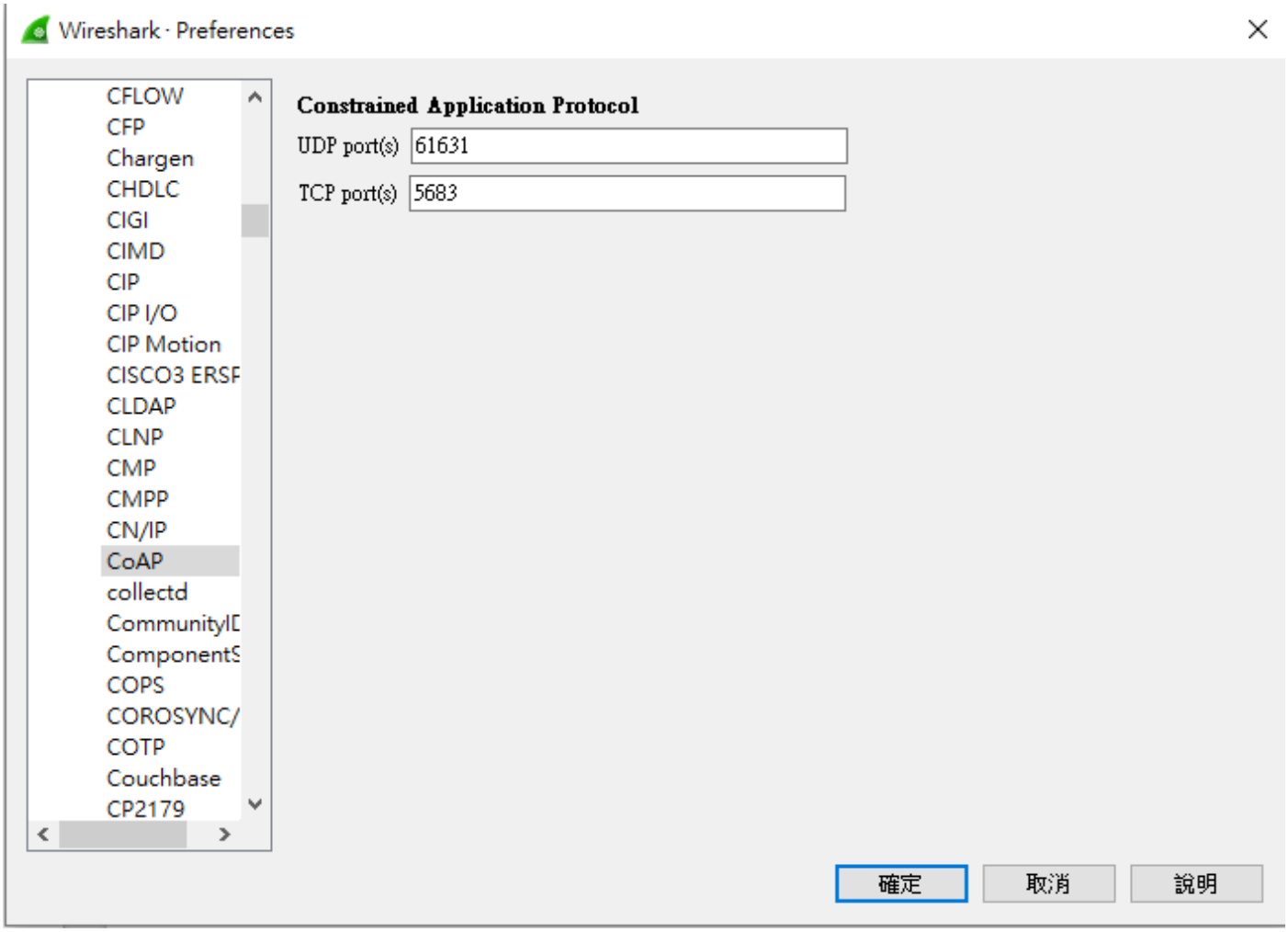
- IEEE 802.15.4
  1. Click + to add a **Decryption key**.
  2. Enter the Thread network Master Key into the **Decryption key** column.
  3. Enter "1" as the **Decryption key index**.
  4. Select Thread hash from the **Key hash** column listbox.



- Thread
  1. Enter "00000000" for the Thread sequence counter.
  2. Uncheck Use PAN ID as first two octets of master key.
  3. Check Automatically acquire Thread sequence counter.



- CoAP
  1. Enter UDP port “61631”.
  2. TCP port “5683”.



9. OTA upgrade

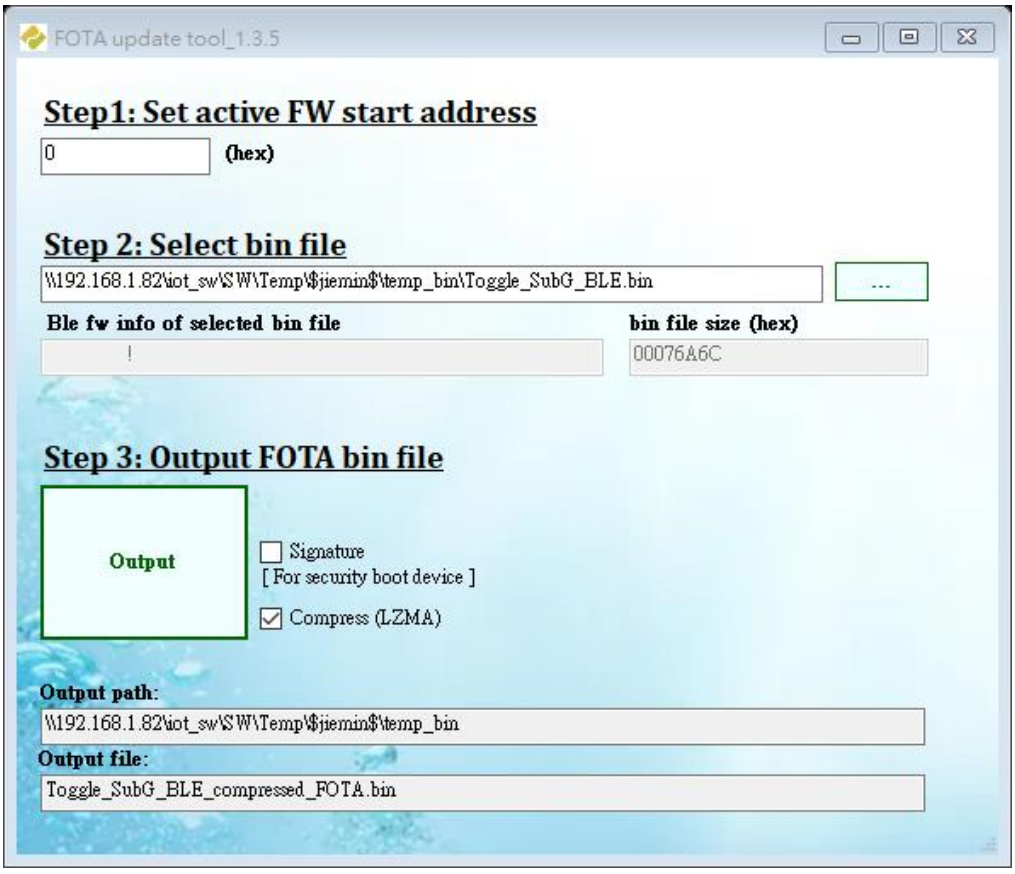
9.1 Process

1. Use the IoT_EVALUATION_TOOL tool to compress the binary file.
2. Use the OTA download tool to transmit the updated binary file (compressed Bin file) to the RT58x via UART1.
3. Once the tool update is complete, use the OTA command to initiate the update process.
4. After the module update is finished, the RT58x will automatically reboot and load the new bin file.

9.2 IoT_EVALUATION_TOOL tool

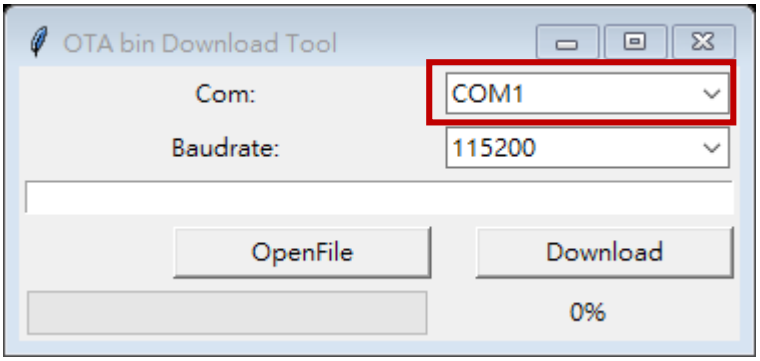
1.Open this tool and select “FOTA” from the options.

2. Set the FW start address to 0 and click the “Output” button to get the compressed Bin file.

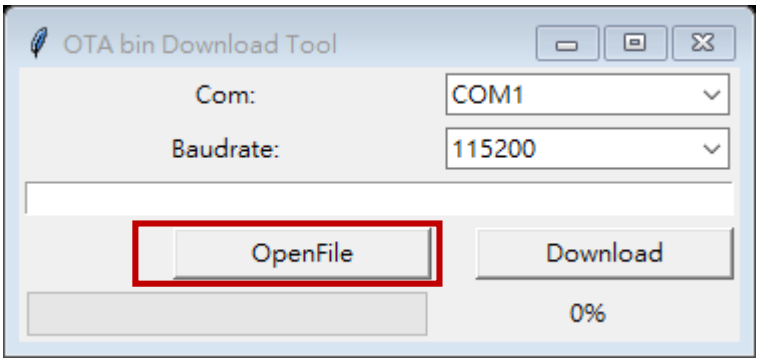


9.3 OTA download tool

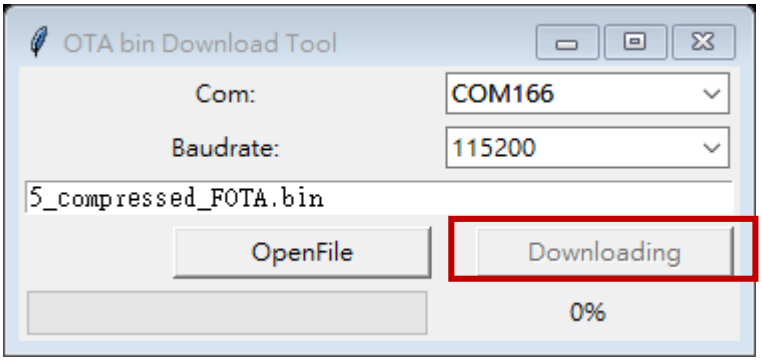
1. Select UART 1 comport.



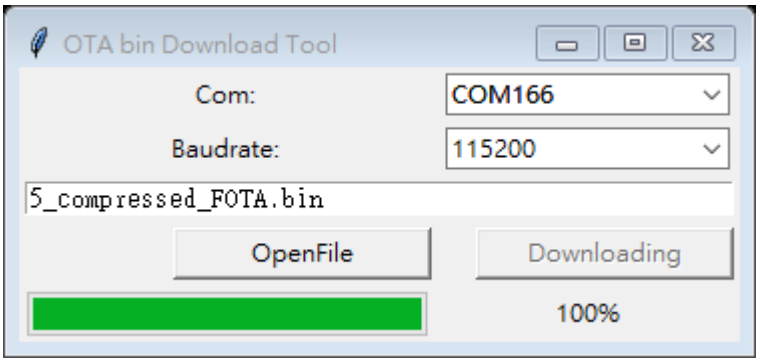
2. Select the bin file compressed using FOTA tool.



3. Start downloading bin file



4. Download to complete.



9.4 OTA Command

1. ota information command

```
> ota
ota image code : 1234
ota image version : 12121212
ota image size : XXXX
ota image crc : XXXX
Done
```

2. ota start command (For 1 to many)

- segments_size : suggest using a 255-byte payload.
- interval: suggest a 2000- millisecond interval.

```
> ota start <segments_size> <intervel>
Done
```

3. ota send command (For 1 to 1)

- ipaddr : destination IP address.

```
> ota send <ipaddr>
Done
```

Revision History

Revision	Description	Owner	Date
V1.0	Initial version	Jiemin	2023/06/26
V1.1	Add SubG frequency band config	Jiemin	2024/02/20
V1.2	Add example API explain	Jiemin	2024/07/29
V1.3	Add OTA download tool explain	Jiemin	2025/02/04