BLE Network Helper

Generated by Doxygen 1.9.6

Chapter 1

BLE-Network-Helper 0.1

BLE Mesh Helper APIs for packing and unpacking BLE mesh messages

1.1 Guideline to library usage

1. Download the library
2. Build from source, use your custom build system(for embedded compiler toolchain) or use Cmake(Version 3.1.6 minimum) for Linux/Windows

   if using Cmake

   (a) go to BLE-Network-Helper dir and make a 'build' directory

   (b) go to build directory and run `cmake ..` for static library or `cmake .. -DBUILD_SHARED_LIBS=ON` for shared library

   (c) run `make` followed by `make install`. (for windows, instead of make, it will be something like `cmake --build . --target INSTALL` as a substitute)
3. To uninstall, remove all directories found in `install_manifest.txt` created by cmake

Details of helper functions can be found here

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 backend_prov_data_t Union Reference

`#include <my_custom_models_def.h>`

Public Attributes

- prov_mode_get_t prov_mode_get
- prov_mode_set_t prov_mode_set
- prov_mode_status_t prov_mode_status
- prov_mode_node_reset_t prov_mode_node_reset
- prov_add_netkey_t prov_add_net_key
- prov_add_appkey_t prov_add_app_key
- prov_key_success_t prov_key_success
- prov_key_fail_t prov_key_fail

4.1.1 Member Data Documentation

4.1.1.1 prov_add_app_key

`prov_add_appkey_t backend_prov_data_t::prov_add_app_key`

4.1.1.2 prov_add_net_key

`prov_add_netkey_t backend_prov_data_t::prov_add_net_key`

4.1.1.3 prov_key_fail

```
prov_key_fail_t backend_prov_data_t::prov_key_fail
```

4.1.1.4 prov_key_success

```
prov_key_success_t backend_prov_data_t::prov_key_success
```

4.1.1.5 prov_mode_get

```
prov_mode_get_t backend_prov_data_t::prov_mode_get
```

4.1.1.6 prov_mode_node_reset

```
prov_mode_node_reset_t backend_prov_data_t::prov_mode_node_reset
```

4.1.1.7 prov_mode_set

```
prov_mode_set_t backend_prov_data_t::prov_mode_set
```

4.1.1.8 prov_mode_status

```
prov_mode_status_t backend_prov_data_t::prov_mode_status
```

The documentation for this union was generated from the following file:

- includes/my_custom_models_def.h

4.2 cfg_state_t Union Reference

a BLE Mesh configuration state with parameters defined according to Mesh Specifications

```
#include <network_msg_struct.h>
```

Public Attributes

- esp_ble_mesh_cfg_client_get_state_t get_state
- esp_ble_mesh_cfg_client_set_state_t set_state
- esp_ble_mesh_cfg_client_common_cb_param_t status_state

4.2.1 Detailed Description

a BLE Mesh configuration state with parameters defined according to Mesh Specifications

4.2.2 Member Data Documentation

4.2.2.1 get_state

```
esp_ble_mesh_cfg_client_get_state_t cfg_state_t::get_state
```

4.2.2.2 set_state

```
esp_ble_mesh_cfg_client_set_state_t cfg_state_t::set_state
```

4.2.2.3 status_state

```
esp_ble_mesh_cfg_client_common_cb_param_t cfg_state_t::status_state
```

The documentation for this union was generated from the following file:

- includes/network_msg_struct.h

4.3 prov_add_appkey_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint16_t net_idx
- uint16_t app_idx

4.3.1 Member Data Documentation

4.3.1.1 app_idx

```
uint16_t prov_add_appkey_t::app_idx
```

4.3.1.2 net_idx

```
uint16_t prov_add_appkey_t::net_idx
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.4 prov_add_netkey_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint16_t net_idx

4.4.1 Member Data Documentation

4.4.1.1 net_idx

```
uint16_t prov_add_netkey_t::net_idx
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.5 prov_key_fail_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint32_t opcode

4.5.1 Member Data Documentation

4.5.1.1 opcode

```
uint32_t prov_key_fail_t::opcode
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.6 prov_key_success_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint32_t opcode
- uint16_t net_idx
- uint16_t app_idx

4.6.1 Member Data Documentation

4.6.1.1 app_idx

```
uint16_t prov_key_success_t::app_idx
```

4.6.1.2 net_idx

```
uint16_t prov_key_success_t::net_idx
```

4.6.1.3 opcode

```
uint32_t prov_key_success_t::opcode
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.7 prov_mode_get_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint32_t prov_payload

4.7.1 Member Data Documentation

4.7.1.1 prov_payload

```
uint32_t prov_mode_get_t::prov_payload
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.8 prov_mode_node_reset_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint32_t prov_payload

4.8.1 Member Data Documentation

4.8.1.1 prov_payload

```
uint32_t prov_mode_node_reset_t::prov_payload
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.9 prov_mode_set_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint32_t prov_payload

4.9.1 Member Data Documentation

4.9.1.1 prov_payload

```
uint32_t prov_mode_set_t::prov_payload
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

4.10 prov_mode_status_t Struct Reference

```
#include <my_custom_models_def.h>
```

Public Attributes

- uint8_t prov_is_true

4.10.1 Member Data Documentation

4.10.1.1 prov_is_true

```
uint8_t prov_mode_status_t::prov_is_true
```

The documentation for this struct was generated from the following file:

- includes/my_custom_models_def.h

Chapter 5

File Documentation

5.1 includes/my_crc.h File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Functions

- uint16_t CRCCCITT (uint8_t *data, size_t length_of_interest)

  Runs a CRC-16-CCITT checksum algorithm. It will return and generate 2 bytes of checksum if you are sender. If you are receiver, run the function with the length_of_interest accounting for the last 2 bytes of crc. It should return 0 to indicate no data loss for 99.9984% of the time.

5.1.1 Function Documentation

5.1.1.1 CRCCCITT()

```
uint16_t CRCCCITT (
            uint8_t * data,
            size_t length_of_interest )
```

Runs a CRC-16-CCITT checksum algorithm. It will return and generate 2 bytes of checksum if you are sender. If you are receiver, run the function with the length_of_interest accounting for the last 2 bytes of crc. It should return 0 to indicate no data loss for 99.9984% of the time.

Name : "CRC-16/CITT" Width : 16 Poly : 1021 Init : FFFF RefIn : False RefOut : False XorOut : 0000

Parameters

data	buffer with a size of the message contents + 2 bytes for CRC portion. This buffer's data must be in little endian format regardless of checking or receiving
length_of_interest	number of bytes of interest in message content(exclude CRC part) if generating CRC. If checking CRC, must account for the number of bytes for CRC in length

Returns

uint16_t FOR SENDER: returns crc checksum value. FOR RECEIVER: returns 0 if no data loss

5.2 my_crc.h

Go to the documentation of this file.

```
00001 #ifndef MY_CRC_H
00002 #define MY_CRC_H
00003
00004 #include <stddef.h>
00005 #include <stdint.h>
00026 uint16_t CRCCCITT(uint8_t *data, size_t length_of_interest);
00027 #endif
```

5.3 includes/my_custom_models_def.h File Reference

```
#include <stdint.h>
#include "esp_ble_mesh_defs.h"
```

Classes

- struct prov_mode_get_t
- struct prov_mode_set_t
- struct prov_mode_status_t
- struct prov_mode_node_reset_t
- struct prov_add_netkey_t
- struct prov_add_appkey_t
- struct prov_key_success_t
- struct prov_key_fail_t
- union backend_prov_data_t

Macros

- #define CUSTOM_MODELS_DEF
- #define BACKEND_PROV_OP_3(b0, cid) ((((b0) << 16) | 0xD00000) | (cid))
- #define CID_ESP 0x02E5
- #define ESP_BLE_MESH_VND_MODEL_ID_CLIENT 0x0002
- #define ESP_BLE_MESH_VND_MODEL_ID_SERVER 0x0003
- #define ESP_BLE_MESH_VND_MODEL_OP_GET ESP_BLE_MESH_MODEL_OP_3(0x0A, CID_ESP)
- #define ESP_BLE_MESH_VND_MODEL_OP_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0B, CID_ESP)
- #define ESP_BLE_MESH_VND_MODEL_OP_SET ESP_BLE_MESH_MODEL_OP_3(0x0C, CID_ESP)
- #define ESP_BLE_MESH_VND_MODEL_OP_SET_UNACK ESP_BLE_MESH_MODEL_OP_3(0x0D, CID_ESP)
- #define ESP_BLE_MESH_VND_MODEL_OP_INTR_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0E, CID_ESP)
- #define ESP_BLE_MESH_VND_MODEL_OP_DUMMY_SET ESP_BLE_MESH_MODEL_OP_3(0x0F, CID_ESP)
- #define PROV_MODE_SET BACKEND_PROV_OP_3(0x00, CID_ESP)
- #define PROV_MODE_GET BACKEND_PROV_OP_3(0x01, CID_ESP)
- #define PROV_MODE_STATUS BACKEND_PROV_OP_3(0x02, CID_ESP)

- #define PROV_MODE_NODE_RESET BACKEND_PROV_OP_3(0x03, CID_ESP)
- #define PROV_ADD_NETKEY BACKEND_PROV_OP_3(0x04, CID_ESP)
- #define PROV_ADD_APPKEY BACKEND_PROV_OP_3(0x05, CID_ESP)
- #define PROV_KEY_SUCCESS BACKEND_PROV_OP_3(0x06, CID_ESP)
- #define PROV_KEY_FAIL BACKEND_PROV_OP_3(0x07, CID_ESP)
- #define MESH_CLIENT_TIMEOUT BACKEND_PROV_OP_3(0x08, CID_ESP)
- #define PROV_NODE_INFO BACKEND_PROV_OP_3(0x0F, CID_ESP)
- #define SENSOR_PAYLOAD_BYTES 5
- #define MSG_C_BYTES (SENSOR_PAYLOAD_BYTES+5)
- #define MSG_A_BYTES (MSG_C_BYTES +2)
- #define BLE_NET_CONFIG_PAYLOAD_BYTES 28
- #define MSG_A_NON_NET_PAYLOAD 7
- #define MSG_A_CONFIG_BYTES (BLE_NET_CONFIG_PAYLOAD_BYTES+MSG_A_NON_NET_PAYLOAD)
- #define BACKEND_PROV_DATA_PAYLOAD_BYTES 8
- #define MSG_A_BACKEND_PROV_BYTES (BACKEND_PROV_DATA_PAYLOAD_BYTES+5)
- #define MSG_SEND_TTL 7
- #define MSG_SEND_REL false
- #define MSG_TIMEOUT 0
- #define MSG_ROLE ROLE_NODE

Functions

- struct __attribute__ ((packed))

Variables

- model_sensor_data_t

5.3.1 Macro Definition Documentation

5.3.1.1 BACKEND_PROV_DATA_PAYLOAD_BYTES

```
#define BACKEND_PROV_DATA_PAYLOAD_BYTES 8
```

5.3.1.2 BACKEND_PROV_OP_3

```
#define BACKEND_PROV_OP_3(
              b0,
              cid ) ((((b0) << 16) | 0xD00000) | (cid))
```

5.3.1.3 BLE_NET_CONFIG_PAYLOAD_BYTES

```
#define BLE_NET_CONFIG_PAYLOAD_BYTES 28
```

5.3.1.4 CID_ESP

```
#define CID_ESP 0x02E5
```

5.3.1.5 CUSTOM_MODELS_DEF

```
#define CUSTOM_MODELS_DEF
```

5.3.1.6 ESP_BLE_MESH_VND_MODEL_ID_CLIENT

```
#define ESP_BLE_MESH_VND_MODEL_ID_CLIENT 0x0002
```

5.3.1.7 ESP_BLE_MESH_VND_MODEL_ID_SERVER

```
#define ESP_BLE_MESH_VND_MODEL_ID_SERVER 0x0003
```

5.3.1.8 ESP_BLE_MESH_VND_MODEL_OP_DUMMY_SET

```
#define ESP_BLE_MESH_VND_MODEL_OP_DUMMY_SET ESP_BLE_MESH_MODEL_OP_3(0x0F, CID_ESP)
```

5.3.1.9 ESP_BLE_MESH_VND_MODEL_OP_GET

```
#define ESP_BLE_MESH_VND_MODEL_OP_GET ESP_BLE_MESH_MODEL_OP_3(0x0A, CID_ESP)
```

5.3.1.10 ESP_BLE_MESH_VND_MODEL_OP_INTR_STATUS

```
#define ESP_BLE_MESH_VND_MODEL_OP_INTR_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0E, CID_ESP)
```

5.3.1.11 ESP_BLE_MESH_VND_MODEL_OP_SET

```
#define ESP_BLE_MESH_VND_MODEL_OP_SET ESP_BLE_MESH_MODEL_OP_3(0x0C, CID_ESP)
```

5.3.1.12 ESP_BLE_MESH_VND_MODEL_OP_SET_UNACK

```
#define ESP_BLE_MESH_VND_MODEL_OP_SET_UNACK ESP_BLE_MESH_MODEL_OP_3(0x0D, CID_ESP)
```

5.3.1.13 ESP_BLE_MESH_VND_MODEL_OP_STATUS

```
#define ESP_BLE_MESH_VND_MODEL_OP_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0B, CID_ESP)
```

5.3.1.14 MESH_CLIENT_TIMEOUT

```
#define MESH_CLIENT_TIMEOUT BACKEND_PROV_OP_3(0x08, CID_ESP)
```

5.3.1.15 MSG_A_BACKEND_PROV_BYTES

```
#define MSG_A_BACKEND_PROV_BYTES (BACKEND_PROV_DATA_PAYLOAD_BYTES+5)
```

5.3.1.16 MSG_A_BYTES

```
#define MSG_A_BYTES (MSG_C_BYTES +2)
```

5.3.1.17 MSG_A_CONFIG_BYTES

```
#define MSG_A_CONFIG_BYTES (BLE_NET_CONFIG_PAYLOAD_BYTES+MSG_A_NON_NET_PAYLOAD)
```

5.3.1.18 MSG_A_NON_NET_PAYLOAD

```
#define MSG_A_NON_NET_PAYLOAD 7
```

5.3.1.19 MSG_C_BYTES

```
#define MSG_C_BYTES (SENSOR_PAYLOAD_BYTES+5)
```

5.3.1.20 MSG_ROLE

```
#define MSG_ROLE ROLE_NODE
```

5.3.1.21 MSG_SEND_REL

```
#define MSG_SEND_REL false
```

5.3.1.22 MSG_SEND_TTL

```
#define MSG_SEND_TTL 7
```

5.3.1.23 MSG_TIMEOUT

```
#define MSG_TIMEOUT 0
```

5.3.1.24 PROV_ADD_APPKEY

```
#define PROV_ADD_APPKEY BACKEND_PROV_OP_3(0x05, CID_ESP)
```

5.3.1.25 PROV_ADD_NETKEY

```
#define PROV_ADD_NETKEY BACKEND_PROV_OP_3(0x04, CID_ESP)
```

5.3.1.26 PROV_KEY_FAIL

```
#define PROV_KEY_FAIL BACKEND_PROV_OP_3(0x07, CID_ESP)
```

5.3.1.27 PROV_KEY_SUCCESS

```
#define PROV_KEY_SUCCESS BACKEND_PROV_OP_3(0x06, CID_ESP)
```

5.3.1.28 PROV_MODE_GET

```
#define PROV_MODE_GET BACKEND_PROV_OP_3(0x01, CID_ESP)
```

5.3.1.29 PROV_MODE_NODE_RESET

```
#define PROV_MODE_NODE_RESET BACKEND_PROV_OP_3(0x03, CID_ESP)
```

5.3.1.30 PROV_MODE_SET

```
#define PROV_MODE_SET BACKEND_PROV_OP_3(0x00, CID_ESP)
```

5.3.1.31 PROV_MODE_STATUS

```
#define PROV_MODE_STATUS BACKEND_PROV_OP_3(0x02, CID_ESP)
```

5.3.1.32 PROV_NODE_INFO

```
#define PROV_NODE_INFO BACKEND_PROV_OP_3(0x0F, CID_ESP)
```

5.3.1.33 SENSOR_PAYLOAD_BYTES

```
#define SENSOR_PAYLOAD_BYTES 5
```

5.3.2 Function Documentation

5.3.2.1 __attribute__()

```
struct __attribute__ (
              (packed)  )
```

5.3.3 Variable Documentation

5.3.3.1 model_sensor_data_t

```
model_sensor_data_t
```

5.4 my_custom_models_def.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #ifndef CUSTOM_MODELS_DEF
00003 #define CUSTOM_MODELS_DEF
00004
00005 #include <stdint.h>
00006 #include "esp_ble_mesh_defs.h"
00007
00008 #define BACKEND_PROV_OP_3(b0, cid)    ((((b0) << 16) | 0xD00000) | (cid))
00009 #define CID_ESP 0x02E5 //esp's company id for BLE
00010 #define ESP_BLE_MESH_VND_MODEL_ID_CLIENT 0x0002
00011 #define ESP_BLE_MESH_VND_MODEL_ID_SERVER 0x0003
00012
00013 #define ESP_BLE_MESH_VND_MODEL_OP_GET ESP_BLE_MESH_MODEL_OP_3(0x0A, CID_ESP)
00014 #define ESP_BLE_MESH_VND_MODEL_OP_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0B, CID_ESP)
00015 #define ESP_BLE_MESH_VND_MODEL_OP_SET ESP_BLE_MESH_MODEL_OP_3(0x0C, CID_ESP)
00016 #define ESP_BLE_MESH_VND_MODEL_OP_SET_UNACK ESP_BLE_MESH_MODEL_OP_3(0x0D, CID_ESP)
00017 #define ESP_BLE_MESH_VND_MODEL_OP_INTR_STATUS ESP_BLE_MESH_MODEL_OP_3(0x0E, CID_ESP)
00018 #define ESP_BLE_MESH_VND_MODEL_OP_DUMMY_SET ESP_BLE_MESH_MODEL_OP_3(0x0F, CID_ESP)
00019
00020 #define PROV_MODE_SET BACKEND_PROV_OP_3(0x00, CID_ESP)
00021 #define PROV_MODE_GET BACKEND_PROV_OP_3(0x01, CID_ESP)
00022 #define PROV_MODE_STATUS BACKEND_PROV_OP_3(0x02, CID_ESP)
00023 #define PROV_MODE_NODE_RESET BACKEND_PROV_OP_3(0x03, CID_ESP)
00024 #define PROV_ADD_NETKEY BACKEND_PROV_OP_3(0x04, CID_ESP)
00025 #define PROV_ADD_APPKEY BACKEND_PROV_OP_3(0x05, CID_ESP)
00026 #define PROV_KEY_SUCCESS BACKEND_PROV_OP_3(0x06, CID_ESP)
00027 #define PROV_KEY_FAIL BACKEND_PROV_OP_3(0x07, CID_ESP)
00028 #define MESH_CLIENT_TIMEOUT BACKEND_PROV_OP_3(0x08, CID_ESP)
00029 #define PROV_NODE_INFO BACKEND_PROV_OP_3(0x0F, CID_ESP)
00030
00031 #define SENSOR_PAYLOAD_BYTES 5
00032 #define MSG_C_BYTES (SENSOR_PAYLOAD_BYTES+5)
00033
00034 #define MSG_A_BYTES (MSG_C_BYTES +2) //Msg A sensor format bytes
00035 #define BLE_NET_CONFIG_PAYLOAD_BYTES 28 //bluetooth network config payload for message A(config
      message)
00036 #define MSG_A_NON_NET_PAYLOAD 7  //non bluetooth config payload portions of MsgA config format
00037 #define MSG_A_CONFIG_BYTES (BLE_NET_CONFIG_PAYLOAD_BYTES+MSG_A_NON_NET_PAYLOAD) //MsgA config format
      bytes
00038 #define BACKEND_PROV_DATA_PAYLOAD_BYTES 8 //backend provisioning data payload, does not include
      PROV_NODE_INFO command
00039 #define MSG_A_BACKEND_PROV_BYTES (BACKEND_PROV_DATA_PAYLOAD_BYTES+5) //MsgA backend provision format
      bytes
00040
00041 //for client model usage
00042 #define MSG_SEND_TTL 7
00043 #define MSG_SEND_REL false
00044 #define MSG_TIMEOUT 0
00045 #define MSG_ROLE ROLE_NODE
00046
00047 typedef struct __attribute__((packed))
00048 {
00049   uint8_t byte0;
```

```
00050   uint8_t byte1;
00051   uint8_t byte2;
00052   uint8_t byte3;
00053   uint8_t byte4;
00054 } model_sensor_data_t;
00055
00056 typedef struct {
00057   uint32_t prov_payload;
00058 }prov_mode_get_t;
00059
00060 typedef struct {
00061   uint32_t prov_payload;
00062 }prov_mode_set_t;
00063
00064 typedef struct {
00065   uint8_t prov_is_true;
00066 }prov_mode_status_t;
00067
00068 typedef struct {
00069   uint32_t prov_payload;
00070 }prov_mode_node_reset_t;
00071
00072 typedef struct {
00073   uint16_t net_idx;
00074 }prov_add_netkey_t;
00075
00076 typedef struct {
00077   uint16_t net_idx;
00078   uint16_t app_idx;
00079 }prov_add_appkey_t;
00080
00081 typedef struct {
00082   uint32_t opcode;
00083   uint16_t net_idx;
00084   uint16_t app_idx;
00085 }prov_key_success_t;
00086
00087 typedef struct {
00088   uint32_t opcode;
00089 }prov_key_fail_t;
00090
00091 typedef union {
00092   prov_mode_get_t prov_mode_get;
00093   prov_mode_set_t prov_mode_set;
00094   prov_mode_status_t prov_mode_status;
00095   prov_mode_node_reset_t prov_mode_node_reset;
00096   prov_add_netkey_t prov_add_net_key;
00097   prov_add_appkey_t prov_add_app_key;
00098   prov_key_success_t prov_key_success;
00099   prov_key_fail_t prov_key_fail;
00100 } backend_prov_data_t;
00101
00102 #endif //CUSTOM_MODELS_DEF
```

5.5 includes/network_msg_struct.h File Reference

helper functions to pack/unpack BLE Mesh messages(format A, format C)

```
#include "esp_ble_mesh_defs.h"
#include "esp_ble_mesh_config_model_api.h"
#include "my_custom_models_def.h"
```

Classes

- union cfg_state_t

  a BLE Mesh configuration state with parameters defined according to Mesh Specifications

Macros

- #define NET_MSG_STRUCT

Functions

- uint32_t extract_opcode (uint8_t *buf)

  extract opcode segment from any message.
- uint16_t extract_addr (uint8_t *buf)

  extract address segment from message A.
- int32_t extract_sensor_data_msgA (uint8_t *buf, model_sensor_data_t *sensor_buf)

  extract sensor data payload from buf to store in sensor_buf
- int32_t extract_bt_data_msgA (uint8_t *buf, cfg_state_t *state)

  extract bluetooth config data payload from buf to store in cfg_state_t state
- int32_t extract_backend_prov_data_msgA (uint8_t *buf, backend_prov_data_t *prov_data)

  extract backend prov data payload from buf to store in backend_prov_data_t prov_data
- int32_t extract_node_data_msgA (uint8_t *buf, esp_ble_mesh_node_t *node_info)

  extract node information when msgA of PROV_NODE_INFO opcode is sent to backend. Note buf is dynamically allocated
- void free_node_data (esp_ble_mesh_node_t *node_info)

  free dynamically allocated pointer esp_ble_mesh_node_t node_info after "extract_back_end_prov_data_msgA()"*
- uint8_t * set_sensor_data_msgA (uint32_t opcode, uint16_t addr, model_sensor_data_t *sensor_buf)

  Set the buffer containing sensor message A , which includes generating and setting the crc in message A.
- uint8_t * set_bt_data_msgA (uint32_t opcode, uint16_t addr, cfg_state_t *state)

  Set the buffer containing bt message A , which includes generating and setting the crc in message A.
- uint8_t * set_backend_prov_data_msgA (uint32_t opcode, backend_prov_data_t *prov_data)

  Set the buffer containing backend prov message A , which includes generating and setting the crc in message A.
- int32_t extract_sensor_data_msgC (uint8_t *buf, model_sensor_data_t *sensor_buf)

  extract sensor data payload from buf to store in sensor_buf
- uint8_t * set_sensor_data_msgC (uint32_t opcode, model_sensor_data_t *sensor_buf)

  Set the buffer containing message C , which includes generating and setting the crc in message C.

5.5.1 Detailed Description

helper functions to pack/unpack BLE Mesh messages(format A, format C)

Author

Shane

Version

0.1

Date

2023-02-22

Copyright

5.5.2 Macro Definition Documentation

5.5.2.1 NET_MSG_STRUCT

```
#define NET_MSG_STRUCT
```

5.5.3 Function Documentation

5.5.3.1 extract_addr()

```
uint16_t extract_addr (
            uint8_t * buf )
```

extract address segment from message A.

Parameters

buf	pointer to message A

Returns

uint16_t unicast addr or group addr, 0 if fail

5.5.3.2 extract_backend_prov_data_msgA()

```
int32_t extract_backend_prov_data_msgA (
            uint8_t * buf,
            backend_prov_data_t * prov_data )
```

extract backend prov data payload from buf to store in backend_prov_data_t prov_data

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A backend prov msg format
prov_data	

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.5.3.3 extract_bt_data_msgA()

```
int32_t extract_bt_data_msgA (
            uint8_t * buf,
            cfg_state_t * state )
```

extract bluetooth config data payload from buf to store in cfg_state_t state

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A bluetooth config msg format
state	empty buffer to store bt config data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.5.3.4 extract_node_data_msgA()

```
int32_t extract_node_data_msgA (
            uint8_t * buf,
            esp_ble_mesh_node_t * node_info )
```

extract node information when msgA of PROV_NODE_INFO opcode is sent to backend. Note buf is dynamically allocated

Parameters

buf	msgA format for PROV_NODE_INFO special opcode
node_info	pointer that will be initialised to node information after function call

Returns

number of nodes provisioned, -1 if fail

5.5.3.5 extract_opcode()

```
uint32_t extract_opcode (
            uint8_t * buf )
```

extract opcode segment from any message.

Parameters

buf	pointer to message

Returns

uint32_t opcode, 0 if fail

5.5.3.6 extract_sensor_data_msgA()

```
int32_t extract_sensor_data_msgA (
            uint8_t * buf,
            model_sensor_data_t * sensor_buf )
```

extract sensor data payload from buf to store in sensor_buf

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A sensor msg format
sensor_buf	empty buffer to store sensor data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.5.3.7 extract_sensor_data_msgC()

```
int32_t extract_sensor_data_msgC (
            uint8_t * buf,
            model_sensor_data_t * sensor_buf )
```

extract sensor data payload from buf to store in sensor_buf

Parameters

buf	buf should have enough data allocated and user should ensure it contains message C msg format
sensor_buf	empty buffer to store sensor data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.5.3.8 free_node_data()

```
void free_node_data (
            esp_ble_mesh_node_t * node_info )
```

free dynamically allocated pointer esp_ble_mesh_node_t* node_info after "extract_back_end_prov_data_msgA()"

Parameters

node_info	

5.5.3.9 set_backend_prov_data_msgA()

```
uint8_t * set_backend_prov_data_msgA (
            uint32_t opcode,
            backend_prov_data_t * prov_data )
```

Set the buffer containing backend prov message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
prov_data	valid backend prov data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless prov_data contains correct data

5.5.3.10 set_bt_data_msgA()

```
uint8_t * set_bt_data_msgA (
            uint32_t opcode,
            uint16_t addr,
            cfg_state_t * state )
```

Set the buffer containing bt message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
addr	destination address
state	valid bt config data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless state contains correct data

5.5.3.11 set_sensor_data_msgA()

```
uint8_t * set_sensor_data_msgA (
            uint32_t opcode,
            uint16_t addr,
            model_sensor_data_t * sensor_buf )
```

Set the buffer containing sensor message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
addr	destination address
sensor_buf	valid sensor data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless sensor_buf contains correct data

5.5.3.12 set_sensor_data_msgC()

```
uint8_t * set_sensor_data_msgC (
            uint32_t opcode,
            model_sensor_data_t * sensor_buf )
```

Set the buffer containing message C , which includes generating and setting the crc in message C.

Parameters

opcode	opcode of command
addr	destination address
sensor_buf	valid sensor data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message C if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless sensor_buf contains correct data

5.6 network_msg_struct.h

Go to the documentation of this file.

```
00001
00012 #pragma once
00013 #ifndef NET_MSG_STRUCT
00014 #define NET_MSG_STRUCT
00015
00016 #if defined (__GLIBC__)
00017 # include <endian.h>
00018 #endif
00019 # if (__BYTE_ORDER == __LITTLE_ENDIAN)
00020 #else
00021 #error "invalid endianness"
00022 #endif
00023
00024 #include "esp_ble_mesh_defs.h"
00025 #include "esp_ble_mesh_config_model_api.h"
00026 #include "my_custom_models_def.h"
00027
00028
00029 #ifdef __cplusplus
00030 extern "C" {
00031 #endif
00032
00033 #pragma pack(1) //ensure no padding in cfg_state_t
00038 typedef union
00039 {
00040   esp_ble_mesh_cfg_client_get_state_t get_state;
00041   esp_ble_mesh_cfg_client_set_state_t set_state;
00042   esp_ble_mesh_cfg_client_common_cb_param_t status_state;
00043 }cfg_state_t;
00044 #pragma pack()
00045
00052 uint32_t extract_opcode(uint8_t *buf);
00053
00060 uint16_t extract_addr(uint8_t *buf);
00061
00069 int32_t extract_sensor_data_msgA(uint8_t *buf, model_sensor_data_t *sensor_buf);
00070
00078 int32_t extract_bt_data_msgA(uint8_t* buf, cfg_state_t* state);
00079
00087 int32_t extract_backend_prov_data_msgA(uint8_t *buf, backend_prov_data_t *prov_data);
00088
00096 int32_t extract_node_data_msgA(uint8_t *buf, esp_ble_mesh_node_t *node_info);
00097
00103 void free_node_data(esp_ble_mesh_node_t *node_info);
00104
00113 uint8_t* set_sensor_data_msgA(uint32_t opcode, uint16_t addr, model_sensor_data_t *sensor_buf);
00114
00123 uint8_t* set_bt_data_msgA(uint32_t opcode, uint16_t addr, cfg_state_t *state);
00124
00132 uint8_t *set_backend_prov_data_msgA(uint32_t opcode, backend_prov_data_t *prov_data);
00133
00141 int32_t extract_sensor_data_msgC(uint8_t *buf, model_sensor_data_t* sensor_buf);
00142
00151 uint8_t* set_sensor_data_msgC(uint32_t opcode, model_sensor_data_t *sensor_buf);
00152
00153
00154 #ifdef __cplusplus
00155 }
00156 #endif
00157
00158 #endif /* NET_MSG_STRUCT */
```

5.7 README.md File Reference

5.8 srcs/my_crc.c File Reference

```
#include <stddef.h>
#include <stdint.h>
```

Functions

- uint16_t CRCCCITT (uint8_t *data, size_t length_of_interest)

  Runs a CRC-16-CCITT checksum algorithm. It will return and generate 2 bytes of checksum if you are sender. If you are receiver, run the function with the length_of_interest accounting for the last 2 bytes of crc. It should return 0 to indicate no data loss for 99.9984% of the time.

5.8.1 Function Documentation

5.8.1.1 CRCCCITT()

```
uint16_t CRCCCITT (
            uint8_t * data,
            size_t length_of_interest )
```

Runs a CRC-16-CCITT checksum algorithm. It will return and generate 2 bytes of checksum if you are sender. If you are receiver, run the function with the length_of_interest accounting for the last 2 bytes of crc. It should return 0 to indicate no data loss for 99.9984% of the time.

Name : "CRC-16/CITT" Width : 16 Poly : 1021 Init : FFFF RefIn : False RefOut : False XorOut : 0000

Parameters

data	buffer with a size of the message contents + 2 bytes for CRC portion. This buffer's data must be in little endian format regardless of checking or receiving
length_of_interest	number of bytes of interest in message content(exclude CRC part) if generating CRC. If checking CRC, must account for the number of bytes for CRC in length

Returns

uint16_t FOR SENDER: returns crc checksum value. FOR RECEIVER: returns 0 if no data loss

5.9 srcs/network_msg_struct.c File Reference

```
#include "network_msg_struct.h"
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <endian.h>
#include "my_crc.h"
```

Functions

- uint32_t extract_opcode (uint8_t *buf)

  extract opcode segment from any message.
- uint16_t extract_addr (uint8_t *buf)

  extract address segment from message A.
- int32_t extract_sensor_data_msgA (uint8_t *buf, model_sensor_data_t *sensor_buf)

  extract sensor data payload from buf to store in sensor_buf
- int32_t extract_bt_data_msgA (uint8_t *buf, cfg_state_t *state)

  extract bluetooth config data payload from buf to store in cfg_state_t state
- int32_t extract_backend_prov_data_msgA (uint8_t *buf, backend_prov_data_t *prov_data)

  extract backend prov data payload from buf to store in backend_prov_data_t prov_data
- int32_t extract_node_data_msgA (uint8_t *buf, esp_ble_mesh_node_t *node_info)

  extract node information when msgA of PROV_NODE_INFO opcode is sent to backend. Note buf is dynamically allocated
- void free_node_data (esp_ble_mesh_node_t *node_info)

  free dynamically allocated pointer esp_ble_mesh_node_t node_info after "extract_back_end_prov_data_msgA()"*
- uint8_t * set_sensor_data_msgA (uint32_t opcode, uint16_t addr, model_sensor_data_t *sensor_buf)

  Set the buffer containing sensor message A , which includes generating and setting the crc in message A.
- uint8_t * set_bt_data_msgA (uint32_t opcode, uint16_t addr, cfg_state_t *state)

  Set the buffer containing bt message A , which includes generating and setting the crc in message A.
- uint8_t * set_backend_prov_data_msgA (uint32_t opcode, backend_prov_data_t *prov_data)

  Set the buffer containing backend prov message A , which includes generating and setting the crc in message A.
- int32_t extract_sensor_data_msgC (uint8_t *buf, model_sensor_data_t *sensor_buf)

  extract sensor data payload from buf to store in sensor_buf
- uint8_t * set_sensor_data_msgC (uint32_t opcode, model_sensor_data_t *sensor_buf)

  Set the buffer containing message C , which includes generating and setting the crc in message C.

5.9.1 Function Documentation

5.9.1.1 extract_addr()

```
uint16_t extract_addr (
            uint8_t * buf )
```

extract address segment from message A.

Parameters

buf	pointer to message A

Returns

uint16_t unicast addr or group addr, 0 if fail

5.9.1.2 extract_backend_prov_data_msgA()

```
int32_t extract_backend_prov_data_msgA (
            uint8_t * buf,
            backend_prov_data_t * prov_data )
```

extract backend prov data payload from buf to store in backend_prov_data_t prov_data

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A backend prov msg format
prov_data	

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.9.1.3 extract_bt_data_msgA()

```
int32_t extract_bt_data_msgA (
            uint8_t * buf,
            cfg_state_t * state )
```

extract bluetooth config data payload from buf to store in cfg_state_t state

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A bluetooth config msg format
state	empty buffer to store bt config data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.9.1.4 extract_node_data_msgA()

```
int32_t extract_node_data_msgA (
            uint8_t * buf,
            esp_ble_mesh_node_t * node_info )
```

extract node information when msgA of PROV_NODE_INFO opcode is sent to backend. Note buf is dynamically allocated

Parameters

buf	msgA format for PROV_NODE_INFO special opcode
node_info	pointer that will be initialised to node information after function call

Returns

number of nodes provisioned, -1 if fail

5.9.1.5 extract_opcode()

```
uint32_t extract_opcode (
            uint8_t * buf )
```

extract opcode segment from any message.

Parameters

buf	pointer to message

Returns

uint32_t opcode, 0 if fail

5.9.1.6 extract_sensor_data_msgA()

```
int32_t extract_sensor_data_msgA (
            uint8_t * buf,
            model_sensor_data_t * sensor_buf )
```

extract sensor data payload from buf to store in sensor_buf

Parameters

buf	buf should have enough data allocated and user should ensure it contains message A sensor msg format
sensor_buf	empty buffer to store sensor data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.9.1.7 extract_sensor_data_msgC()

```
int32_t extract_sensor_data_msgC (
            uint8_t * buf,
            model_sensor_data_t * sensor_buf )
```

extract sensor data payload from buf to store in sensor_buf

Parameters

buf	buf should have enough data allocated and user should ensure it contains message C msg format
sensor_buf	empty buffer to store sensor data, must be initialised first

Returns

returns 0 if successful, -1 if fail. Note: returning 0 does not guarantee succcess unless buf contains correct data

5.9.1.8 free_node_data()

```
void free_node_data (
            esp_ble_mesh_node_t * node_info )
```

free dynamically allocated pointer esp_ble_mesh_node_t* node_info after "extract_back_end_prov_data_msgA()"

Parameters

node_info	

5.9.1.9 set_backend_prov_data_msgA()

```
uint8_t * set_backend_prov_data_msgA (
            uint32_t opcode,
            backend_prov_data_t * prov_data )
```

Set the buffer containing backend prov message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
prov_data	valid backend prov data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless prov_data contains correct data

5.9.1.10 set_bt_data_msgA()

```
uint8_t * set_bt_data_msgA (
            uint32_t opcode,
            uint16_t addr,
            cfg_state_t * state )
```

Set the buffer containing bt message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
addr	destination address
state	valid bt config data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless state contains correct data

5.9.1.11 set_sensor_data_msgA()

```
uint8_t * set_sensor_data_msgA (
            uint32_t opcode,
            uint16_t addr,
            model_sensor_data_t * sensor_buf )
```

Set the buffer containing sensor message A , which includes generating and setting the crc in message A.

Parameters

opcode	opcode of command
addr	destination address
sensor_buf	valid sensor data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message A if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless sensor_buf contains correct data

5.9.1.12 set_sensor_data_msgC()

```
uint8_t * set_sensor_data_msgC (
            uint32_t opcode,
            model_sensor_data_t * sensor_buf )
```

Set the buffer containing message C , which includes generating and setting the crc in message C.

Parameters

opcode	opcode of command
addr	destination address
sensor_buf	valid sensor data pointer

Returns

returns pointer to buffer(static define, not thread-safe) containing message C if successful, NULL if fail. Note: returning valid pointer does not guarantee succcess unless sensor_buf contains correct data

Index