SmartThings Device SDK

1.0.0

Generated by Doxygen 1.8.16

1 Overview 1
1.1 Introduction . . . 1
1.2 IoT Core Device Library . . . 1
1.3 Porting Layer . . . 2
1.4 Device Application . . . 3

2 Data Structure Index 3
2.1 Data Structures . . . 3

3 File Index 6
3.1 File List . . . 6

4 Data Structure Documentation 8
4.1 noti_data_raw_t::_quota Struct Reference . . . 8
4.1.1 Field Documentation . . . 8
4.2 noti_data_raw_t::_rate_limit Struct Reference . . . 9
4.2.1 Field Documentation . . . 9
4.3 iot_bsp_fs_handle_t Struct Reference . . . 10
4.3.1 Field Documentation . . . 11
4.4 iot_cap_cmd_data_t Struct Reference . . . 11
4.4.1 Detailed Description . . . 12
4.4.2 Field Documentation . . . 12
4.5 iot_cap_cmd_set Struct Reference . . . 13
4.5.1 Detailed Description . . . 14
4.5.2 Field Documentation . . . 14
4.6 iot_cap_cmd_set_list Struct Reference . . . 14
4.6.1 Detailed Description . . . 16
4.6.2 Field Documentation . . . 16
4.7 iot_cap_evt_data_t Struct Reference . . . 16
4.7.1 Detailed Description . . . 17
4.7.2 Field Documentation . . . 17
4.8 iot_cap_handle Struct Reference . . . 18
4.8.1 Detailed Description . . . 18
4.8.2 Field Documentation . . . 19
4.9 iot_cap_handle_list Struct Reference . . . 20
4.9.1 Detailed Description . . . 20
4.9.2 Field Documentation . . . 20
4.10 iot_cap_unit_t Struct Reference . . . 21
4.10.1 Detailed Description . . . 21
4.10.2 Field Documentation . . . 21
4.11 iot_cap_val_t Struct Reference . . . 22
4.11.1 Detailed Description . . . 22
4.11.2 Field Documentation . . . 23

4.12 iot_cloud_prov_data Struct Reference . . . 24
4.12.1 Detailed Description . . . 24
4.12.2 Field Documentation . . . 24
4.13 iot_command Struct Reference . . . 25
4.13.1 Detailed Description . . . 26
4.13.2 Field Documentation . . . 26
4.14 iot_context Struct Reference . . . 26
4.14.1 Detailed Description . . . 28
4.14.2 Field Documentation . . . 28
4.15 iot_crypto_cipher_info Struct Reference . . . 33
4.15.1 Detailed Description . . . 33
4.15.2 Field Documentation . . . 33
4.16 iot_crypto_ecdh_params Struct Reference . . . 34
4.16.1 Detailed Description . . . 35
4.16.2 Field Documentation . . . 35
4.17 iot_crypto_ed25519_keypair Struct Reference . . . 36
4.17.1 Detailed Description . . . 36
4.17.2 Field Documentation . . . 36
4.18 iot_crypto_keypair Struct Reference . . . 37
4.18.1 Detailed Description . . . 37
4.18.2 Field Documentation . . . 37
4.19 iot_crypto_pk_context Struct Reference . . . 38
4.19.1 Detailed Description . . . 38
4.19.2 Field Documentation . . . 39
4.20 iot_crypto_pk_funcs Struct Reference . . . 39
4.20.1 Detailed Description . . . 40
4.20.2 Field Documentation . . . 40
4.21 iot_crypto_pk_info Struct Reference . . . 40
4.21.1 Detailed Description . . . 41
4.21.2 Field Documentation . . . 41
4.22 iot_devconf_prov_data Struct Reference . . . 42
4.22.1 Detailed Description . . . 43
4.22.2 Field Documentation . . . 43
4.23 iot_device_info Struct Reference . . . 44
4.23.1 Detailed Description . . . 44
4.23.2 Field Documentation . . . 44
4.24 iot_device_prov_data Struct Reference . . . 45
4.24.1 Detailed Description . . . 45
4.24.2 Field Documentation . . . 46
4.25 iot_easysetup_payload Struct Reference . . . 46
4.25.1 Detailed Description . . . 47
4.25.2 Field Documentation . . . 47

4.26 iot_mac Struct Reference . . . 47
4.26.1 Detailed Description . . . 48
4.26.2 Field Documentation . . . 48
4.27 iot_mqtt_ctx Struct Reference . . . 48
4.27.1 Detailed Description . . . 50
4.27.2 Field Documentation . . . 50
4.28 iot_net_socket Struct Reference . . . 51
4.28.1 Detailed Description . . . 52
4.28.2 Field Documentation . . . 52
4.29 iot_noti_data_t Struct Reference . . . 53
4.29.1 Detailed Description . . . 53
4.29.2 Field Documentation . . . 54
4.30 iot_os_mutex Struct Reference . . . 54
4.30.1 Detailed Description . . . 54
4.30.2 Field Documentation . . . 55
4.31 iot_pin_t Struct Reference . . . 55
4.31.1 Detailed Description . . . 55
4.31.2 Field Documentation . . . 55
4.32 iot_registered_data Struct Reference . . . 56
4.32.1 Detailed Description . . . 56
4.32.2 Field Documentation . . . 56
4.33 iot_state_data Struct Reference . . . 57
4.33.1 Detailed Description . . . 58
4.33.2 Field Documentation . . . 58
4.34 iot_uuid Struct Reference . . . 58
4.34.1 Detailed Description . . . 59
4.34.2 Field Documentation . . . 59
4.35 iot_wifi_conf Struct Reference . . . 59
4.35.1 Detailed Description . . . 60
4.35.2 Field Documentation . . . 60
4.36 iot_wifi_prov_data Struct Reference . . . 61
4.36.1 Detailed Description . . . 61
4.36.2 Field Documentation . . . 61
4.37 iot_wifi_scan_result_t Struct Reference . . . 62
4.37.1 Detailed Description . . . 63
4.37.2 Field Documentation . . . 63
4.38 MessageData Struct Reference . . . 64
4.38.1 Field Documentation . . . 65
4.39 MQTTClient::MessageHandlers Struct Reference . . . 65
4.39.1 Field Documentation . . . 65
4.40 MQTTClient Struct Reference . . . 66
4.40.1 Field Documentation . . . 67

4.41 MQTTConnackData Struct Reference . . . 69
4.41.1 Field Documentation . . . 69
4.42 MQTTConnackFlags Union Reference . . . 70
4.42.1 Field Documentation . . . 70
4.43 MQTTConnectFlags Union Reference . . . 71
4.43.1 Field Documentation . . . 71
4.44 MQTTHeader Union Reference . . . 73
4.44.1 Detailed Description . . . 73
4.44.2 Field Documentation . . . 73
4.45 MQTTLenString Struct Reference . . . 74
4.45.1 Field Documentation . . . 74
4.46 MQTTMessage Struct Reference . . . 75
4.46.1 Field Documentation . . . 75
4.47 MQTTPacket_connectData Struct Reference . . . 76
4.47.1 Field Documentation . . . 78
4.48 MQTTPacket_willOptions Struct Reference . . . 79
4.48.1 Detailed Description . . . 80
4.48.2 Field Documentation . . . 80
4.49 MQTTString Struct Reference . . . 81
4.49.1 Field Documentation . . . 81
4.50 MQTTSubackData Struct Reference . . . 82
4.50.1 Field Documentation . . . 82
4.51 MQTTTransport Struct Reference . . . 82
4.51.1 Field Documentation . . . 83
4.52 noti_data_raw_t Union Reference . . . 84
4.52.1 Detailed Description . . . 84
4.52.2 Field Documentation . . . 84
4.53 url_parse_t Struct Reference . . . 85
4.53.1 Detailed Description . . . 85
4.53.2 Field Documentation . . . 85

5 File Documentation **86**
5.1 st-device-sdk-c/doc/mainpage.dox File Reference . . . 86
5.2 st-device-sdk-c/src/crypto/iot_crypto_ed25519.c File Reference . . . 86
5.2.1 Function Documentation . . . 87
5.3 st-device-sdk-c/src/crypto/mbedtls/iot_crypto_mbedtls.c File Reference . . . 89
5.3.1 Function Documentation . . . 90
5.4 st-device-sdk-c/src/crypto/ss/iot_crypto_ss.c File Reference . . . 95
5.4.1 Function Documentation . . . 95
5.5 st-device-sdk-c/src/easysetup/http/iot_easysetup_http.c File Reference . . . 97
5.5.1 Macro Definition Documentation . . . 98
5.5.2 Function Documentation . . . 98

5.5.3 Variable Documentation . . . 102
5.6 st-device-sdk-c/src/easysetup/iot_easysetup_crypto.c File Reference . . . 103
5.6.1 Function Documentation . . . 103
5.7 st-device-sdk-c/src/easysetup/iot_easysetup_d2d.c File Reference . . . 104
5.7.1 Macro Definition Documentation . . . 105
5.7.2 Function Documentation . . . 106
5.8 st-device-sdk-c/src/easysetup/iot_easysetup_st_mqtt.c File Reference . . . 108
5.8.1 Function Documentation . . . 109
5.9 st-device-sdk-c/src/include/bsp/iot_bsp_debug.h File Reference . . . 110
5.9.1 Function Documentation . . . 111
5.10 st-device-sdk-c/src/include/bsp/iot_bsp_fs.h File Reference . . . 112
5.10.1 Enumeration Type Documentation . . . 113
5.10.2 Function Documentation . . . 113
5.11 st-device-sdk-c/src/include/bsp/iot_bsp_nv_data.h File Reference . . . 119
5.11.1 Function Documentation . . . 120
5.12 st-device-sdk-c/src/include/bsp/iot_bsp_random.h File Reference . . . 121
5.12.1 Function Documentation . . . 121
5.13 st-device-sdk-c/src/include/bsp/iot_bsp_system.h File Reference . . . 122
5.13.1 Macro Definition Documentation . . . 123
5.13.2 Function Documentation . . . 123
5.14 st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h File Reference . . . 124
5.14.1 Macro Definition Documentation . . . 126
5.14.2 Enumeration Type Documentation . . . 126
5.14.3 Function Documentation . . . 127
5.15 st-device-sdk-c/src/include/iot_capability.h File Reference . . . 130
5.15.1 Typedef Documentation . . . 131
5.15.2 Enumeration Type Documentation . . . 131
5.16 st-device-sdk-c/src/include/iot_crypto.h File Reference . . . 132
5.16.1 Macro Definition Documentation . . . 135
5.16.2 Typedef Documentation . . . 138
5.16.3 Enumeration Type Documentation . . . 138
5.16.4 Function Documentation . . . 139
5.17 st-device-sdk-c/src/include/iot_crypto_internal.h File Reference . . . 149
5.18 st-device-sdk-c/src/include/iot_debug.h File Reference . . . 150
5.18.1 Macro Definition Documentation . . . 150
5.18.2 Enumeration Type Documentation . . . 152
5.18.3 Function Documentation . . . 153
5.19 st-device-sdk-c/src/include/iot_easysetup.h File Reference . . . 154
5.19.1 Macro Definition Documentation . . . 156
5.19.2 Enumeration Type Documentation . . . 159
5.19.3 Function Documentation . . . 160
5.20 st-device-sdk-c/src/include/iot_error.h File Reference . . . 162

5.20.1 Typedef Documentation . . . 162
5.20.2 Enumeration Type Documentation . . . 163
5.21 st-device-sdk-c/src/include/iot_internal.h File Reference . . . 164
5.21.1 Macro Definition Documentation . . . 166
5.21.2 Function Documentation . . . 167
5.22 st-device-sdk-c/src/include/iot_jwt.h File Reference . . . 182
5.22.1 Macro Definition Documentation . . . 183
5.22.2 Function Documentation . . . 183
5.23 st-device-sdk-c/src/include/iot_main.h File Reference . . . 184
5.23.1 Macro Definition Documentation . . . 186
5.23.2 Typedef Documentation . . . 187
5.23.3 Enumeration Type Documentation . . . 187
5.24 st-device-sdk-c/src/include/iot_nv_data.h File Reference . . . 189
5.24.1 Enumeration Type Documentation . . . 190
5.24.2 Function Documentation . . . 191
5.25 st-device-sdk-c/src/include/iot_util.h File Reference . . . 204
5.25.1 Function Documentation . . . 206
5.26 st-device-sdk-c/src/include/iot_uuid.h File Reference . . . 210
5.26.1 Function Documentation . . . 210
5.27 st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h File Reference . . . 212
5.27.1 Macro Definition Documentation . . . 214
5.27.2 Typedef Documentation . . . 215
5.27.3 Enumeration Type Documentation . . . 215
5.27.4 Function Documentation . . . 216
5.28 st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h File Reference . . . 226
5.28.1 Macro Definition Documentation . . . 227
5.28.2 Enumeration Type Documentation . . . 228
5.28.3 Function Documentation . . . 228
5.29 st-device-sdk-c/src/include/mqtt/iot_mqtt_format.h File Reference . . . 233
5.29.1 Function Documentation . . . 234
5.30 st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h File Reference . . . 236
5.30.1 Macro Definition Documentation . . . 237
5.30.2 Enumeration Type Documentation . . . 238
5.30.3 Function Documentation . . . 238
5.31 st-device-sdk-c/src/include/mqtt/iot_mqtt_publish.h File Reference . . . 246
5.31.1 Macro Definition Documentation . . . 247
5.31.2 Function Documentation . . . 247
5.32 st-device-sdk-c/src/include/mqtt/iot_mqtt_stacktrace.h File Reference . . . 250
5.32.1 Macro Definition Documentation . . . 251
5.33 st-device-sdk-c/src/include/mqtt/iot_mqtt_subscribe.h File Reference . . . 253
5.33.1 Macro Definition Documentation . . . 253
5.33.2 Function Documentation . . . 253

5.34 st-device-sdk-c/src/include/mqtt/iot_mqtt_unsubscribe.h File Reference . . . 257
5.34.1 Macro Definition Documentation . . . 258
5.34.2 Function Documentation . . . 258
5.35 st-device-sdk-c/src/include/os/iot_os_net.h File Reference . . . 261
5.35.1 Typedef Documentation . . . 262
5.35.2 Function Documentation . . . 262
5.36 st-device-sdk-c/src/include/os/iot_os_util.h File Reference . . . 265
5.36.1 Macro Definition Documentation . . . 267
5.36.2 Typedef Documentation . . . 267
5.36.3 Function Documentation . . . 268
5.36.4 Variable Documentation . . . 279
5.37 st-device-sdk-c/src/include/st_dev.h File Reference . . . 279
5.37.1 Macro Definition Documentation . . . 282
5.37.2 Typedef Documentation . . . 282
5.37.3 Enumeration Type Documentation . . . 283
5.37.4 Function Documentation . . . 284
5.38 st-device-sdk-c/src/include/st_dev_version.h File Reference . . . 292
5.38.1 Macro Definition Documentation . . . 292
5.39 st-device-sdk-c/src/iot_api.c File Reference . . . 293
5.39.1 Function Documentation . . . 294
5.40 st-device-sdk-c/src/iot_capability.c File Reference . . . 302
5.40.1 Macro Definition Documentation . . . 303
5.40.2 Function Documentation . . . 303
5.41 st-device-sdk-c/src/iot_crypto.c File Reference . . . 310
5.41.1 Function Documentation . . . 311
5.42 st-device-sdk-c/src/iot_jwt.c File Reference . . . 313
5.42.1 Function Documentation . . . 314
5.43 st-device-sdk-c/src/iot_main.c File Reference . . . 314
5.43.1 Macro Definition Documentation . . . 315
5.43.2 Function Documentation . . . 315
5.44 st-device-sdk-c/src/iot_mqtt.c File Reference . . . 318
5.44.1 Function Documentation . . . 319
5.45 st-device-sdk-c/src/iot_nv_data.c File Reference . . . 324
5.45.1 Macro Definition Documentation . . . 325
5.45.2 Function Documentation . . . 326
5.46 st-device-sdk-c/src/iot_util.c File Reference . . . 340
5.46.1 Function Documentation . . . 341
5.47 st-device-sdk-c/src/iot_uuid.c File Reference . . . 345
5.47.1 Function Documentation . . . 346
5.48 st-device-sdk-c/src/mqtt/client/iot_mqtt_client.c File Reference . . . 348
5.48.1 Function Documentation . . . 349
5.49 st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_client.c File Reference . . . 362

5.49.1 Function Documentation . . . 362
5.50 st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_server.c File Reference . . . 368
5.50.1 Macro Definition Documentation . . . 368
5.50.2 Function Documentation . . . 368
5.51 st-device-sdk-c/src/mqtt/packet/iot_mqtt_deserialize_publish.c File Reference . . . 371
5.51.1 Macro Definition Documentation . . . 371
5.51.2 Function Documentation . . . 371
5.52 st-device-sdk-c/src/mqtt/packet/iot_mqtt_format.c File Reference . . . 372
5.52.1 Function Documentation . . . 373
5.52.2 Variable Documentation . . . 375
5.53 st-device-sdk-c/src/mqtt/packet/iot_mqtt_packet.c File Reference . . . 375
5.53.1 Macro Definition Documentation . . . 376
5.53.2 Function Documentation . . . 376
5.54 st-device-sdk-c/src/mqtt/packet/iot_mqtt_serialize_publish.c File Reference . . . 383
5.54.1 Function Documentation . . . 384
5.55 st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_client.c File Reference . . . 388
5.55.1 Function Documentation . . . 388
5.56 st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_server.c File Reference . . . 391
5.56.1 Function Documentation . . . 391
5.57 st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_client.c File Reference . . . 393
5.57.1 Function Documentation . . . 393
5.58 st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_server.c File Reference . . . 395
5.58.1 Function Documentation . . . 396
5.59 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_debug_esp8266.c File Reference . . . 397
5.59.1 Function Documentation . . . 398
5.59.2 Variable Documentation . . . 399
5.60 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_fs_esp8266.c File Reference . . . 399
5.60.1 Macro Definition Documentation . . . 400
5.60.2 Function Documentation . . . 400
5.61 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_nv_data_esp8266.c File Reference . . . 405
5.61.1 Function Documentation . . . 406
5.62 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_random_esp8266.c File Reference . . . 407
5.62.1 Function Documentation . . . 407
5.63 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_system_esp8266.c File Reference . . . 408
5.63.1 Function Documentation . . . 408
5.64 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_wifi_esp8266.c File Reference . . . 410
5.64.1 Function Documentation . . . 411
5.64.2 Variable Documentation . . . 413
5.65 st-device-sdk-c/src/port/os/freertos/iot_os_net_freertos.c File Reference . . . 414
5.65.1 Function Documentation . . . 415
5.66 st-device-sdk-c/src/port/os/freertos/iot_os_util_freertos.c File Reference . . . 417
5.66.1 Typedef Documentation . . . 418

5.66.2 Function Documentation 419
5.66.3 Variable Documentation 430

Index **433**

1 Overview

1.1 Introduction

SmartThings Device SDK aims to make it easier to develop IoT devices that can be securely connected to the SmartThings cloud server by providing the IoT Core Library. It can also be adopted to the existing original chipset vendor's SDK easily.

For this, the IoT Core Device library and the examples were separated into two git repositories as follows:

- `IoT Core Device Library`
- `Reference`

You can choose to only download the `Reference` repository from git, if you have already ported your project. The IoT core device library is downloaded as a submodule in the `Reference` repository.

1.2 IoT Core Device Library

This library provides APIs to be used in device applications.

These APIs can be classified as follows to facilitate the basic behavior scenario of IoT devices and these have been implemented to make a device application as simple as possible.

For more detail, please click the link of each API.

- connection management
  - st_conn_init()
  - st_conn_set_noti_cb()
  - st_conn_start()
  - st_conn_cleanup()
  - st_conn_ownership_confirm()
- capability management
  - st_cap_handle_init()
  - st_cap_cmd_set_cb()
  - st_cap_attr_create_int()
  - st_cap_attr_create_number()
  - st_cap_attr_create_string()
  - st_cap_attr_create_string_array()
  - st_cap_attr_free()
  - st_cap_attr_send()

1.3 Porting Layer

The IoT core device library has a porting layer to support the use of the same APIs in the device application, even if the platform changes.

In the current STDK version, the platform-dependent directories that must be ported are present in the `src/port/bsp` and `src/port/os` of IoT core device library.

- BSP(Board Support Package)

  The BSP is the well-defined set of functions that the IoT core device library invokes in order to interact with a platform's specific networking, file IO, random number generator, and system feature.

  - File system
    - iot_bsp_fs_close()
    - iot_bsp_fs_deinit()
    - iot_bsp_fs_init()
    - iot_bsp_fs_open()
    - iot_bsp_fs_read()
    - iot_bsp_fs_remove()
    - iot_bsp_fs_write()
  - Non-volatile storage
    - iot_bsp_nv_get_data_path()
  - Random number
    - iot_bsp_random()
  - System
    - iot_bsp_system_get_time_in_sec()
    - iot_bsp_system_get_uniqueid()
    - iot_bsp_system_poweroff()
    - iot_bsp_system_reboot()
    - iot_bsp_system_set_time_in_sec()
  - Wifi
    - iot_bsp_wifi_get_freq()
    - iot_bsp_wifi_get_mac()
    - iot_bsp_wifi_get_scan_result()
    - iot_bsp_wifi_init()
    - iot_bsp_wifi_set_mode()
  - Debug
    - iot_bsp_debug()
    - iot_bsp_debug_check_heap()

- OS

  These APIs are related to operating system.

  - Network
    - iot_os_net_connet()
    - iot_os_net_disconnect()
    - iot_os_net_init()
    - iot_os_net_print_status()
    - iot_os_net_select()
  - Utils

* iot_os_delay()
* iot_os_eventgroup_clear_bits()
* iot_os_eventgroup_create()
* iot_os_eventgroup_delete()
* iot_os_eventgroup_set_bits()
* iot_os_eventgroup_wait_bits()
* iot_os_mutex_init()
* iot_os_mutex_lock()
* iot_os_mutex_unlock()
* iot_os_queue_create()
* iot_os_queue_delete()
* iot_os_queue_receive()
* iot_os_queue_reset()
* iot_os_queue_send()
* iot_os_thread_create()
* iot_os_thread_delete()
* iot_os_thread_yield()
* iot_os_timer_count_ms()
* iot_os_timer_destroy()
* iot_os_timer_init()
* iot_os_timer_isexpired()
* iot_os_timer_left_ms()

1.4 Device Application

A device application is developed using the APIs provided by the IoT Core Device Library. We recommend reuse of the pre-supplied example applications, like st_switch. This allows for rapid development as you begin to develop your new device. Please refer to the API references related to the IoT core device library as shown before

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

noti_data_raw_t::_quota 8

noti_data_raw_t::_rate_limit 9

iot_bsp_fs_handle_t 10

iot_cap_cmd_data_t
Contains data for "command" payload 11

iot_cap_cmd_set
Contains user command callback function data 13

iot_cap_cmd_set_list
Linked list for command callback function data 14

iot_cap_evt_data_t
Contains data for "deviceEvent" payload 16

iot_cap_handle
Contains data for capability handle 18

iot_cap_handle_list
Linked list for capability handle 20

iot_cap_unit_t
Contains a "unit" data 21

iot_cap_val_t
Contains a various type of data which can be int, double, string and string array 22

iot_cloud_prov_data
Contains "cloud provisioning" data 24

iot_command
Contains "internal command" data 25

iot_context
Contains "iot core's main context" data 26

iot_crypto_cipher_info
Contains cipher informations 33

iot_crypto_ecdh_params
Contains parameters to share a key between Things and ST apps 34

iot_crypto_ed25519_keypair
Contains Ed25519 and Curve25519 key pairs 36

iot_crypto_keypair
Contains public and private keys for Ed25519 37

iot_crypto_pk_context
Contains key pair information and public key based function lists 38

iot_crypto_pk_funcs
Contains public key based function lists 39

iot_crypto_pk_info
Contains information of key pair 40

iot_devconf_prov_data
Contains "device configuration" data 42

iot_device_info
Contains "device's information" data 44

iot_device_prov_data
Contains "all device's provisioning" data 45

iot_easysetup_payload
Contains "easy-setup payload" data 46

iot_mac
Contains "wifi mac" data 47

iot_mqtt_ctx
Contains "mqtt handling context" data 48

iot_net_socket
Contains "network management structure" data 51

iot_noti_data_t
Contains data for notification data 53

iot_os_mutex
Contains a mutex data 54

iot_pin_t
Contains a pin values for pin type onboarding process 55

iot_registered_data
Contains "registration message" data 56

iot_state_data
Contains "iot core's main state" data 57

iot_uuid
Contains "uuid" data 58

iot_wifi_conf
Contains a "wifi stack configuration" data 59

iot_wifi_prov_data
Contains "wifi provisioning" data 61

iot_wifi_scan_result_t
Contains a "wifi scan" data 62

MessageData 64

MQTTClient::MessageHandlers 65

MQTTClient 66

MQTTConnackData 69

MQTTConnackFlags 70

MQTTConnectFlags 71

MQTTHeader 73

MQTTLenString 74

MQTTMessage 75

MQTTPacket_connectData 76

MQTTPacket_willOptions 79

MQTTString 81

MQTTSubackData 82

MQTTTransport 82

noti_data_raw_t
Contains data for raw data of each notification 84

url_parse_t
Contains a "url parse" data 85

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

st-device-sdk-c/src/iot_api.c 293

st-device-sdk-c/src/iot_capability.c 302

st-device-sdk-c/src/iot_crypto.c 310

st-device-sdk-c/src/iot_jwt.c 313

st-device-sdk-c/src/iot_main.c 314

st-device-sdk-c/src/iot_mqtt.c 318

st-device-sdk-c/src/iot_nv_data.c 324

st-device-sdk-c/src/iot_util.c 340

st-device-sdk-c/src/iot_uuid.c 345

st-device-sdk-c/src/crypto/iot_crypto_ed25519.c 86

st-device-sdk-c/src/crypto/mbedtls/iot_crypto_mbedtls.c 89

st-device-sdk-c/src/crypto/ss/iot_crypto_ss.c 95

st-device-sdk-c/src/easysetup/iot_easysetup_crypto.c 103

st-device-sdk-c/src/easysetup/iot_easysetup_d2d.c 104

st-device-sdk-c/src/easysetup/iot_easysetup_st_mqtt.c 108

st-device-sdk-c/src/easysetup/http/iot_easysetup_http.c 97

st-device-sdk-c/src/include/iot_capability.h 130

st-device-sdk-c/src/include/iot_crypto.h 132

st-device-sdk-c/src/include/iot_crypto_internal.h 149

st-device-sdk-c/src/include/iot_debug.h 150

st-device-sdk-c/src/include/iot_easysetup.h 154

st-device-sdk-c/src/include/iot_error.h 162

st-device-sdk-c/src/include/iot_internal.h 164

st-device-sdk-c/src/include/iot_jwt.h 182

st-device-sdk-c/src/include/iot_main.h 184

st-device-sdk-c/src/include/iot_nv_data.h 189

st-device-sdk-c/src/include/iot_util.h 204

st-device-sdk-c/src/include/iot_uuid.h 210

st-device-sdk-c/src/include/st_dev.h 279

st-device-sdk-c/src/include/st_dev_version.h 292

st-device-sdk-c/src/include/bsp/iot_bsp_debug.h 110

st-device-sdk-c/src/include/bsp/iot_bsp_fs.h 112

st-device-sdk-c/src/include/bsp/iot_bsp_nv_data.h 119

st-device-sdk-c/src/include/bsp/iot_bsp_random.h 121

st-device-sdk-c/src/include/bsp/iot_bsp_system.h 122

st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h 124

st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h 212

st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h 226

st-device-sdk-c/src/include/mqtt/iot_mqtt_format.h 233

st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h 236

st-device-sdk-c/src/include/mqtt/iot_mqtt_publish.h 246

st-device-sdk-c/src/include/mqtt/iot_mqtt_stacktrace.h 250

st-device-sdk-c/src/include/mqtt/iot_mqtt_subscribe.h 253

st-device-sdk-c/src/include/mqtt/iot_mqtt_unsubscribe.h 257

st-device-sdk-c/src/include/os/iot_os_net.h 261

st-device-sdk-c/src/include/os/iot_os_util.h 265

st-device-sdk-c/src/mqtt/client/iot_mqtt_client.c 348

st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_client.c 362

st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_server.c 368

st-device-sdk-c/src/mqtt/packet/iot_mqtt_deserialize_publish.c 371

st-device-sdk-c/src/mqtt/packet/iot_mqtt_format.c 372

st-device-sdk-c/src/mqtt/packet/iot_mqtt_packet.c 375

st-device-sdk-c/src/mqtt/packet/iot_mqtt_serialize_publish.c 383

st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_client.c 388

st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_server.c 391

st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_client.c 393

st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_server.c 395

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_debug_esp8266.c 397

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_fs_esp8266.c 399

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_nv_data_esp8266.c 405

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_random_esp8266.c 407

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_system_esp8266.c 408

st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_wifi_esp8266.c 410

st-device-sdk-c/src/port/os/freertos/iot_os_net_freertos.c 414

st-device-sdk-c/src/port/os/freertos/iot_os_util_freertos.c 417

4 Data Structure Documentation

4.1 noti_data_raw_t::_quota Struct Reference

```
#include <st_dev.h>
```

Collaboration diagram for noti_data_raw_t::_quota:

Data Fields

- int used

  Current used data usage in bytes.
- int limit

  Current data limit in bytes.

4.1.1 Field Documentation

4.1.1.1 limit `int limit`

Current data limit in bytes.

4.1.1.2 used `int used`

Current used data usage in bytes.

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

- st-device-sdk-c/src/include/st_dev.h

4.2 noti_data_raw_t::_rate_limit Struct Reference

```
#include <st_dev.h>
```

Collaboration diagram for noti_data_raw_t::_rate_limit:

Data Fields

- int count

  Current rate limit count.
- int threshold

  Current rate limit threshold.
- int remainingTime

  How much time remains for rate limit releasing.
- int sequenceNumber

  Sequence number of event that triggered rate limit.

4.2.1 Field Documentation

4.2.1.1 count `int count`

Current rate limit count.

4.2.1.2 remainingTime `int remainingTime`

How much time remains for rate limit releasing.

4.2.1.3 sequenceNumber `int sequenceNumber`

Sequence number of event that triggered rate limit.

4.2.1.4 threshold `int threshold`

Current rate limit threshold.

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

- st-device-sdk-c/src/include/st_dev.h

4.3 iot_bsp_fs_handle_t Struct Reference

```
#include <iot_bsp_fs.h>
```

Collaboration diagram for iot_bsp_fs_handle_t:

Data Fields

- int fd
- char filename [128]

4.3.1 Field Documentation

4.3.1.1 fd `int fd`

4.3.1.2 filename `char filename[128]`

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

- st-device-sdk-c/src/include/bsp/iot_bsp_fs.h

4.4 iot_cap_cmd_data_t Struct Reference

Contains data for "command" payload.

```
#include <st_dev.h>
```

Collaboration diagram for iot_cap_cmd_data_t:

Data Fields

- uint8_t num_args

  Number of arguments.
- char * args_str [MAX_CAP_ARG]

  Name of each argument.
- iot_cap_val_t cmd_data [MAX_CAP_ARG]

  Value of each arguments.

4.4.1 Detailed Description

Contains data for "command" payload.

4.4.2 Field Documentation

4.4.2.1 args_str `char* args_str[MAX_CAP_ARG]`

Name of each argument.

Note

This is used only if there is more than one argument.

4.4.2.2 cmd_data `iot_cap_val_t cmd_data[MAX_CAP_ARG]`

Value of each arguments.

4.4.2.3 num_args `uint8_t num_args`

Number of arguments.

Note

Usally 1, but if commands type is 'json object', it could be more than 1. (See colorControl capability.)

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

- st-device-sdk-c/src/include/st_dev.h

4.5 iot_cap_cmd_set Struct Reference

Contains user command callback function data.

```
#include <iot_capability.h>
```

Collaboration diagram for iot_cap_cmd_set:

Data Fields

- const char * cmd_type

  NULL-terminated string, which is name of `commands`*.*
- st_cap_cmd_cb cmd_cb

  User callback function.
- void * usr_data

  User data for cmd_cb.

4.5.1 Detailed Description

Contains user command callback function data.

4.5.2 Field Documentation

4.5.2.1 cmd_cb `st_cap_cmd_cb cmd_cb`

User callback function.

4.5.2.2 cmd_type `const char* cmd_type`

NULL-terminated string, which is name of `commands`.

4.5.2.3 usr_data `void* usr_data`

User data for cmd_cb.

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

- st-device-sdk-c/src/include/iot_capability.h

4.6 iot_cap_cmd_set_list Struct Reference

linked list for command callback function data

```
#include <iot_capability.h>
```

Collaboration diagram for iot_cap_cmd_set_list:

Data Fields

- struct iot_cap_cmd_set * command

  a pointer to a command data

- struct iot_cap_cmd_set_list * next

  a pointer to a next list

4.6.1 Detailed Description

linked list for command callback function data

4.6.2 Field Documentation

4.6.2.1 command `struct iot_cap_cmd_set* command`

a pointer to a command data

4.6.2.2 next `struct iot_cap_cmd_set_list* next`

a pointer to a next list

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

- st-device-sdk-c/src/include/iot_capability.h

4.7 iot_cap_evt_data_t Struct Reference

Contains data for "deviceEvent" payload.

```
#include <iot_capability.h>
```

Collaboration diagram for iot_cap_evt_data_t:

Data Fields

- const char * evt_type

  NULL-terminated string, which is name of `attributes`.
- iot_cap_val_t evt_value

  'value' data for deviceEvent.
- iot_cap_unit_t evt_unit

  'unit' data for deviceEvent.

4.7.1 Detailed Description

Contains data for "deviceEvent" payload.

4.7.2 Field Documentation

4.7.2.1 evt_type `const char* evt_type`

NULL-terminated string, which is name of `attributes`.

4.7.2.2 evt_unit `iot_cap_unit_t evt_unit`

'unit' data for deviceEvent.

4.7.2.3 evt_value `iot_cap_val_t evt_value`

'value' data for deviceEvent.

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

- st-device-sdk-c/src/include/iot_capability.h

4.8 iot_cap_handle Struct Reference

Contains data for capability handle.

```
#include <iot_capability.h>
```

Collaboration diagram for iot_cap_handle:

Data Fields

- const char ∗ capability

  NULL-terminated string, which is name of `capability`*.*
- const char ∗ component

  NULL-terminated string, which is name of `component`*.*
- struct iot_cap_cmd_set_list ∗ cmd_list

  List of command data.
- st_cap_init_cb init_cb

  User callback function for init device state.
- void ∗ init_usr_data

  User data for init_cb.
- struct iot_context ∗ ctx

  ctx

4.8.1 Detailed Description

Contains data for capability handle.

4.8.2 Field Documentation

4.8.2.1 capability `const char* capability`

NULL-terminated string, which is name of `capability`.

Use capability id for this variable. e.g. "switchLevel"

4.8.2.2 cmd_list `struct iot_cap_cmd_set_list* cmd_list`

List of command data.

4.8.2.3 component `const char* component`

NULL-terminated string, which is name of `component`.

4.8.2.4 ctx `struct iot_context* ctx`

ctx

4.8.2.5 init_cb `st_cap_init_cb init_cb`

User callback function for init device state.

4.8.2.6 init_usr_data `void* init_usr_data`

User data for init_cb.

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

- st-device-sdk-c/src/include/iot_capability.h

4.9 iot_cap_handle_list Struct Reference

linked list for capability handle

```
#include <iot_capability.h>
```

Collaboration diagram for iot_cap_handle_list:

Data Fields

- struct iot_cap_handle ∗ handle

  a pointer to a capability handle
- struct iot_cap_handle_list ∗ next

  a pointer to a next list

4.9.1 Detailed Description

linked list for capability handle

4.9.2 Field Documentation

4.9.2.1 handle `struct iot_cap_handle* handle`

a pointer to a capability handle

4.9.2.2 next `struct iot_cap_handle_list* next`

a pointer to a next list

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

- st-device-sdk-c/src/include/iot_capability.h

4.10 iot_cap_unit_t Struct Reference

Contains a "unit" data.

`#include <iot_capability.h>`

Collaboration diagram for iot_cap_unit_t:

Data Fields

- uint8_t type

  Unused or string.
- char * string

  NULL-terminated string.

4.10.1 Detailed Description

Contains a "unit" data.

4.10.2 Field Documentation

4.10.2.1 string `char* string`

NULL-terminated string.

4.10.2.2 type `uint8_t type`

Unused or string.

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

- st-device-sdk-c/src/include/iot_capability.h

4.11 iot_cap_val_t Struct Reference

Contains a various type of data which can be int, double, string and string array.

```
#include <st_dev.h>
```

Collaboration diagram for iot_cap_val_t:

Data Fields

- iot_cap_val_type_t type

  Data type to notify valid data.
- uint8_t str_num

  Number of stings. Only used for sting array.
- int integer

  Integer.
- double number

  Float number.
- char * string

  NULL-terminated string.
- char ** strings

  Array of NULL-terminated strings.

4.11.1 Detailed Description

Contains a various type of data which can be int, double, string and string array.

4.11.2 Field Documentation

4.11.2.1 integer `int integer`

Integer.

4.11.2.2 number `double number`

Float number.

4.11.2.3 str_num `uint8_t str_num`

Number of stings. Only used for sting array.

4.11.2.4 string `char* string`

NULL-terminated string.

4.11.2.5 strings `char** strings`

Array of NULL-terminated strings.

4.11.2.6 type `iot_cap_val_type_t type`

Data type to notify valid data.

Note

Even though there are 4 different type of data (integer, number, string, strings) in this structure, only one type of data is used.

Type of capability's data.

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

- st-device-sdk-c/src/include/st_dev.h

4.12 iot_cloud_prov_data Struct Reference

Contains "cloud provisioning" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_cloud_prov_data:

Data Fields

- char * broker_url

  broker url for mqtt
- int broker_port

  broker port num for mqtt
- struct iot_uuid location_id

  location id for ST(server) management
- struct iot_uuid room_id

  room id for ST(server) management
- char * label

  device name, got from the mobile

4.12.1 Detailed Description

Contains "cloud provisioning" data.

4.12.2 Field Documentation

4.12.2.1 broker_port `int broker_port`

broker port num for mqtt

4.12.2.2 broker_url `char* broker_url`

broker url for mqtt

4.12.2.3 label `char* label`

device name, got from the mobile

4.12.2.4 location_id `struct iot_uuid location_id`

location id for ST(server) management

4.12.2.5 room_id `struct iot_uuid room_id`

room id for ST(server) management

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

- st-device-sdk-c/src/include/iot_main.h

4.13 iot_command Struct Reference

Contains "internal command" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_command:

Data Fields

- enum iot_command_type cmd_type

  command type to handle device
- void * param

  additional inform for each command

4.13.1 Detailed Description

Contains "internal command" data.

4.13.2 Field Documentation

4.13.2.1 cmd_type `enum iot_command_type cmd_type`

command type to handle device

4.13.2.2 param `void* param`

additional inform for each command

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

- st-device-sdk-c/src/include/iot_main.h

4.14 iot_context Struct Reference

Contains "iot core's main context" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_context:

Data Fields

- iot_os_queue ∗ cmd_queue

  iot core's internal command queue
- iot_os_queue ∗ pub_queue

  iot core's event publish queue
- iot_os_queue ∗ easysetup_req_queue

  request queue for easy-setup process
- iot_os_queue ∗ easysetup_resp_queue

  response queue for easy-setup process
- iot_state_t curr_state

  reflect current iot_state
- iot_state_t req_state

  reflect requested iot_state
- iot_os_timer state_timer

  state checking timer for each iot_state
- iot_os_eventgroup ∗ usr_events

  User level handling events.
- iot_os_eventgroup ∗ iot_events

  Internal handling events.
- iot_cap_handle_list_t ∗ cap_handle_list

  allocated capability handle lists
- struct iot_mqtt_ctx ∗ client_ctx

  mqtt context ref. for registration

- struct iot_mqtt_ctx * reged_cli

  mqtt context ref. for connection
- struct iot_device_prov_data prov_data

  allocated device provisioning data
- struct iot_devconf_prov_data devconf

  allocated device configuration data
- struct iot_device_info device_info

  allocated device information data
- iot_crypto_cipher_info_t * es_crypto_cipher_info

  cipher context ref. for easy-setup process
- struct iot_registered_data iot_reg_data

  allocated registration data from server
- void * es_httpd_handle

  httpd handler for easy-setup process
- uint16_t scan_num

  number of wifi ap scan result
- iot_wifi_scan_result_t * scan_result

  actual data lists of each wifi ap scan result
- char * lookup_id

  device's lookup id for server & mobile side notification
- st_cap_noti_cb noti_cb

  notification handling callback for each capability
- void * noti_usr_data

  notification handling callback data for user
- st_status_cb status_cb

  iot core status handling callback for user
- iot_status_t status_maps

  iot status callback maps to check it call or not
- unsigned int reported_stat

  iot status callback checking flag to check it reported or not
- void * status_usr_data

  iot core status handling callback data for user
- int curr_otm_feature

  current device's supported onboarding process validation type
- iot_pin_t * pin

  current device's PIN values for PIN type otm
- unsigned int cmd_err

  current command handling error checking value
- unsigned int cmd_status

  current command status
- uint16_t cmd_count [IOT_COMMAND_TYPE_MAX]

  current queued command counts

4.14.1 Detailed Description

Contains "iot core's main context" data.

4.14.2 Field Documentation

4.14.2.1 cap_handle_list `iot_cap_handle_list_t* cap_handle_list`

allocated capability handle lists

4.14.2.2 client_ctx `struct iot_mqtt_ctx* client_ctx`

mqtt context ref. for registration

4.14.2.3 cmd_count `uint16_t cmd_count[IOT_COMMAND_TYPE_MAX]`

current queued command counts

4.14.2.4 cmd_err `unsigned int cmd_err`

current command handling error checking value

4.14.2.5 cmd_queue `iot_os_queue* cmd_queue`

iot core's internal command queue

4.14.2.6 cmd_status `unsigned int cmd_status`

current command status

4.14.2.7 curr_otm_feature `int curr_otm_feature`

current device's supported onboarding process validation type

4.14.2.8 curr_state `iot_state_t curr_state`

reflect current iot_state

4.14.2.9 devconf `struct iot_devconf_prov_data devconf`

allocated device configuration data

4.14.2.10 device_info `struct iot_device_info device_info`

allocated device information data

4.14.2.11 easysetup_req_queue `iot_os_queue* easysetup_req_queue`

request queue for easy-setup process

4.14.2.12 easysetup_resp_queue `iot_os_queue* easysetup_resp_queue`

response queue for easy-setup process

4.14.2.13 es_crypto_cipher_info `iot_crypto_cipher_info_t* es_crypto_cipher_info`

cipher context ref. for easy-setup process

4.14.2.14 es_httpd_handle `void* es_httpd_handle`

httpd handler for easy-setup process

4.14.2.15 iot_events `iot_os_eventgroup* iot_events`

Internal handling events.

4.14.2.16 iot_reg_data `struct iot_registered_data iot_reg_data`

allocated registration data from server

4.14.2.17 lookup_id `char* lookup_id`

device's lookup id for server & mobile side notification

4.14.2.18 noti_cb `st_cap_noti_cb noti_cb`

notification handling callback for each capability

4.14.2.19 noti_usr_data `void* noti_usr_data`

notification handling callback data for user

4.14.2.20 pin `iot_pin_t* pin`

current device's PIN values for PIN type otm

4.14.2.21 prov_data `struct iot_device_prov_data prov_data`

allocated device provisioning data

4.14.2.22 pub_queue `iot_os_queue* pub_queue`

iot core's event publish queue

4.14.2.23 reged_cli `struct iot_mqtt_ctx* reged_cli`

mqtt context ref. for connection

4.14.2.24 reported_stat `unsigned int reported_stat`

iot status callback checking flag to check it reported or not

4.14.2.25 req_state `iot_state_t req_state`

reflect requested iot_state

4.14.2.26 scan_num `uint16_t scan_num`

number of wifi ap scan result

4.14.2.27 scan_result `iot_wifi_scan_result_t* scan_result`

actual data lists of each wifi ap scan result

4.14.2.28 state_timer `iot_os_timer state_timer`

state checking timer for each iot_state

4.14.2.29 status_cb `st_status_cb status_cb`

iot core status handling callback for user

4.14.2.30 status_maps `iot_status_t status_maps`

iot status callback maps to check it call or not

4.14.2.31 status_usr_data `void* status_usr_data`

iot core status handling callback data for user

4.14.2.32 usr_events `iot_os_eventgroup* usr_events`

User level handling events.

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

- st-device-sdk-c/src/include/iot_main.h

4.15 iot_crypto_cipher_info Struct Reference

Contains cipher informations.

```
#include <iot_crypto.h>
```

Collaboration diagram for iot_crypto_cipher_info:

Data Fields

- iot_crypto_cipher_type_t type

  type of cipher algorithm
- iot_crypto_cipher_mode_t mode

  mode of cipher operation
- unsigned char * key

  a pointer to a shared key
- unsigned char * iv

  a pointer to a IV for AES cipher
- size_t key_len

  a length of a shared key
- size_t iv_len

  a length of a IV

4.15.1 Detailed Description

Contains cipher informations.

4.15.2 Field Documentation

4.15.2.1 iv `unsigned char* iv`

a pointer to a IV for AES cipher

4.15.2.2 iv_len `size_t iv_len`

a length of a IV

4.15.2.3 key `unsigned char* key`

a pointer to a shared key

4.15.2.4 key_len `size_t key_len`

a length of a shared key

4.15.2.5 mode `iot_crypto_cipher_mode_t mode`

mode of cipher operation

4.15.2.6 type `iot_crypto_cipher_type_t type`

type of cipher algorithm

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

- st-device-sdk-c/src/include/iot_crypto.h

4.16 iot_crypto_ecdh_params Struct Reference

Contains parameters to share a key between Things and ST apps.

`#include <iot_crypto.h>`

Collaboration diagram for iot_crypto_ecdh_params:

Data Fields

- unsigned char * t_seckey

  a pointer to a things secret key based on curve25519
- unsigned char * s_pubkey

  a pointer to a server public key based on curve25519
- unsigned char * hash_token

  a pointer to a random token as a salt
- size_t hash_token_len

  a length of random token

4.16.1 Detailed Description

Contains parameters to share a key between Things and ST apps.

4.16.2 Field Documentation

4.16.2.1 hash_token `unsigned char* hash_token`

a pointer to a random token as a salt

4.16.2.2 hash_token_len `size_t hash_token_len`

a length of random token

4.16.2.3 s_pubkey `unsigned char* s_pubkey`

a pointer to a server public key based on curve25519

4.16.2.4 t_seckey `unsigned char* t_seckey`

a pointer to a things secret key based on curve25519

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

- st-device-sdk-c/src/include/iot_crypto.h

4.17 iot_crypto_ed25519_keypair Struct Reference

Contains Ed25519 and Curve25519 key pairs.

```
#include <iot_crypto.h>
```

Collaboration diagram for iot_crypto_ed25519_keypair:

Data Fields

- struct iot_crypto_keypair sign

  a pointer to a Ed25519 key pair for signature
- struct iot_crypto_keypair curve

  a pointer to a curve25519 key pair for encryption/decryption

4.17.1 Detailed Description

Contains Ed25519 and Curve25519 key pairs.

4.17.2 Field Documentation

4.17.2.1 curve `struct iot_crypto_keypair curve`

a pointer to a curve25519 key pair for encryption/decryption

4.17.2.2 sign `struct iot_crypto_keypair sign`

a pointer to a Ed25519 key pair for signature

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

- st-device-sdk-c/src/include/iot_crypto.h

4.18 iot_crypto_keypair Struct Reference

Contains public and private keys for Ed25519.

```
#include <iot_crypto.h>
```

Collaboration diagram for iot_crypto_keypair:

Data Fields

- unsigned char * pubkey
- unsigned char * seckey

  a pointer to a public key

4.18.1 Detailed Description

Contains public and private keys for Ed25519.

4.18.2 Field Documentation

4.18.2.1 pubkey `unsigned char* pubkey`

4.18.2.2 seckey `unsigned char* seckey`

a pointer to a public key

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

- st-device-sdk-c/src/include/iot_crypto.h

4.19 iot_crypto_pk_context Struct Reference

Contains key pair information and public key based function lists.

```
#include <iot_crypto.h>
```

Collaboration diagram for iot_crypto_pk_context:

Data Fields

- iot_crypto_pk_info_t * info
- const iot_crypto_pk_funcs_t * fn

  a pointer to a key pair info

4.19.1 Detailed Description

Contains key pair information and public key based function lists.

4.19.2 Field Documentation

4.19.2.1 fn `const iot_crypto_pk_funcs_t* fn`

a pointer to a key pair info

4.19.2.2 info `iot_crypto_pk_info_t* info`

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

- st-device-sdk-c/src/include/iot_crypto.h

4.20 iot_crypto_pk_funcs Struct Reference

Contains public key based function lists.

```
#include <iot_crypto.h>
```

Collaboration diagram for iot_crypto_pk_funcs:

Data Fields

- const char ∗ name
- iot_error_t(∗ sign)(iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size↩_t ∗slen)

  string name to know this
- iot_error_t(∗ verify)(iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size_t slen)

  a pointer to a function to verify a signature

4.20.1 Detailed Description

Contains public key based function lists.

4.20.2 Field Documentation

4.20.2.1 name `const char* name`

4.20.2.2 sign `iot_error_t(* sign) (iot_crypto_pk_context_t *ctx, unsigned char *input, size_t ilen, unsigned char *sig, size_t *slen)`

string name to know this

a pointer to a function to create a signature

4.20.2.3 verify `iot_error_t(* verify) (iot_crypto_pk_context_t *ctx, unsigned char *input, size_t ilen, unsigned char *sig, size_t slen)`

a pointer to a function to verify a signature

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

- st-device-sdk-c/src/include/iot_crypto.h

4.21 iot_crypto_pk_info Struct Reference

Contains information of key pair.

`#include <iot_crypto.h>`

Collaboration diagram for iot_crypto_pk_info:

Data Fields

- iot_crypto_pk_type_t type
- unsigned char * pubkey

  type of key pair
- unsigned char * seckey

  public key of key pair
- size_t pubkey_len

  private key of key pair
- size_t seckey_len

  length of public key

4.21.1 Detailed Description

Contains information of key pair.

4.21.2 Field Documentation

4.21.2.1 pubkey `unsigned char* pubkey`

type of key pair

4.21.2.2 pubkey_len `size_t pubkey_len`

private key of key pair

4.21.2.3 seckey `unsigned char* seckey`

public key of key pair

4.21.2.4 seckey_len `size_t seckey_len`

length of public key

4.21.2.5 type `iot_crypto_pk_type_t type`

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

- st-device-sdk-c/src/include/iot_crypto.h

4.22 iot_devconf_prov_data Struct Reference

Contains "device configuration" data.

`#include <iot_main.h>`

Collaboration diagram for iot_devconf_prov_data:

Data Fields

- char * device_onboarding_id

  onboarding id, determined in devworks
- char * mnid

  mnid, determined in devworks
- char * setupid

  setupid, determined in devworks
- char * device_type

  device_type, determined in devworks
- char * vid

  vid, determined in devworks
- int ownership_validation_type

  onboarding process validation type, JUSTWORKS, BUTTON, PIN, QR
- iot_crypto_pk_type_t pk_type

  Authentication type, determined in devworks.
- char * hashed_sn

  hashed serial, self-generating values during onboarding process

4.22.1 Detailed Description

Contains "device configuration" data.

4.22.2 Field Documentation

4.22.2.1 device_onboarding_id `char* device_onboarding_id`

onboarding id, determined in devworks

4.22.2.2 device_type `char* device_type`

device_type, determined in devworks

4.22.2.3 hashed_sn `char* hashed_sn`

hashed serial, self-generating values during onboarding process

4.22.2.4 mnid `char* mnid`

mnid, determined in devworks

4.22.2.5 ownership_validation_type `int ownership_validation_type`

onboarding process validation type, JUSTWORKS, BUTTON, PIN, QR

4.22.2.6 pk_type `iot_crypto_pk_type_t pk_type`

Authentication type, determined in devworks.

4.22.2.7 setupid `char* setupid`

setupid, determined in devworks

4.22.2.8 vid `char* vid`

vid, determined in devworks

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

- st-device-sdk-c/src/include/iot_main.h

4.23 iot_device_info Struct Reference

Contains "device's information" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_device_info:

Data Fields

- char * firmware_version
  device's binary/firmware version

4.23.1 Detailed Description

Contains "device's information" data.

4.23.2 Field Documentation

4.23.2.1 firmware_version `char* firmware_version`

device's binary/firmware version

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

- st-device-sdk-c/src/include/iot_main.h

4.24 iot_device_prov_data Struct Reference

Contains "all device's provisioning" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_device_prov_data:

Data Fields

- struct iot_wifi_prov_data wifi

  wifi provisionig data, refer to iot_wifi_prov_data struct
- struct iot_cloud_prov_data cloud

  cloud provisionig data, refer to iot_cloud_prov_data struct

4.24.1 Detailed Description

Contains "all device's provisioning" data.

4.24.2 Field Documentation

4.24.2.1 cloud `struct iot_cloud_prov_data cloud`

cloud provisionig data, refer to iot_cloud_prov_data struct

4.24.2.2 wifi `struct iot_wifi_prov_data wifi`

wifi provisionig data, refer to iot_wifi_prov_data struct

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

- st-device-sdk-c/src/include/iot_main.h

4.25 iot_easysetup_payload Struct Reference

Contains "easy-setup payload" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_easysetup_payload:

Data Fields

- enum iot_easysetup_step step

  reflect easy-setup process step
- iot_error_t err

  error status for each step
- char * payload

  actual payload for each step

4.25.1 Detailed Description

Contains "easy-setup payload" data.

4.25.2 Field Documentation

4.25.2.1 err `iot_error_t err`

error status for each step

4.25.2.2 payload `char* payload`

actual payload for each step

4.25.2.3 step `enum iot_easysetup_step step`

reflect easy-setup process step

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

- st-device-sdk-c/src/include/iot_main.h

4.26 iot_mac Struct Reference

Contains "wifi mac" data.

```
#include <iot_bsp_wifi.h>
```

Collaboration diagram for iot_mac:

Data Fields

- unsigned char addr [IOT_WIFI_MAX_BSSID_LEN]

  wifi mac address

4.26.1 Detailed Description

Contains "wifi mac" data.

4.26.2 Field Documentation

4.26.2.1 addr `unsigned char addr[IOT_WIFI_MAX_BSSID_LEN]`

wifi mac address

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

- st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h

4.27 iot_mqtt_ctx Struct Reference

Contains "mqtt handling context" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_mqtt_ctx:

Data Fields

- iot_net_socket net

  network management handle for mqtt
- MQTTClient cli

  mqtt client handle for iot_core
- bool mqtt_connected

mqtt connected status

- unsigned char buf [IOT_BUF_TX_SIZE]

  mqtt buffer for sending
- unsigned char readbuf [IOT_BUF_RX_SIZE]

  mqtt buffer for receiving
- const char * cmd_filter

  mqtt command topic filter string
- const char * noti_filter

  mqtt notification topic filter string
- void * iot_ctx

  iot main context ref. used for mqtt message callback

4.27.1 Detailed Description

Contains "mqtt handling context" data.

4.27.2 Field Documentation

4.27.2.1 buf `unsigned char buf[IOT_BUF_TX_SIZE]`

mqtt buffer for sending

4.27.2.2 cli `MQTTClient cli`

mqtt client handle for iot_core

4.27.2.3 cmd_filter `const char* cmd_filter`

mqtt command topic filter string

4.27.2.4 iot_ctx `void* iot_ctx`

iot main context ref. used for mqtt message callback

4.27.2.5 mqtt_connected `bool mqtt_connected`

mqtt connected status

4.27.2.6 net `iot_net_socket net`

network management handle for mqtt

4.27.2.7 noti_filter `const char* noti_filter`

mqtt notification topic filter string

4.27.2.8 readbuf `unsigned char readbuf[IOT_BUF_RX_SIZE]`

mqtt buffer for receiving

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

- st-device-sdk-c/src/include/iot_main.h

4.28 iot_net_socket Struct Reference

Contains "network management structure" data.

```
#include <iot_os_net.h>
```

Collaboration diagram for iot_net_socket:

Data Fields

- int socket

  socket handle
- int(∗ read)(struct iot_net_socket ∗, unsigned char ∗, unsigned int, iot_os_timer)

  read handle for socket
- int(∗ write)(struct iot_net_socket ∗, unsigned char ∗, unsigned int, iot_os_timer)

  write handle for socket
- int read_count

  number of read data

4.28.1 Detailed Description

Contains "network management structure" data.

4.28.2 Field Documentation

4.28.2.1 read `int(* read) (struct iot_net_socket *, unsigned char *, unsigned int, iot_os_timer)`

read handle for socket

4.28.2.2 read_count `int read_count`

number of read data

4.28.2.3 socket `int socket`

socket handle

4.28.2.4 write `int(* write) (struct iot_net_socket *, unsigned char *, unsigned int, iot_os_timer)`

write handle for socket

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

- st-device-sdk-c/src/include/os/iot_os_net.h

4.29 iot_noti_data_t Struct Reference

Contains data for notification data.

```
#include <st_dev.h>
```

Collaboration diagram for iot_noti_data_t:

Data Fields

- iot_noti_type_t type

  Type of notification's data.
- noti_data_raw_t raw

  Raw data of each notification.

4.29.1 Detailed Description

Contains data for notification data.

4.29.2 Field Documentation

4.29.2.1 raw `noti_data_raw_t raw`

Raw data of each notification.

4.29.2.2 type `iot_noti_type_t type`

Type of notification's data.

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

- st-device-sdk-c/src/include/st_dev.h

4.30 iot_os_mutex Struct Reference

Contains a mutex data.

```
#include <iot_os_util.h>
```

Collaboration diagram for iot_os_mutex:

Data Fields

- iot_os_sem * sem
  semaphore

4.30.1 Detailed Description

Contains a mutex data.

4.30.2 Field Documentation

4.30.2.1 sem `iot_os_sem* sem`

semaphore

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

- st-device-sdk-c/src/include/os/iot_os_util.h

4.31 iot_pin_t Struct Reference

Contains a pin values for pin type onboarding process.

```
#include <st_dev.h>
```

Collaboration diagram for iot_pin_t:

Data Fields

- unsigned char pin [8]

  actual pin values

4.31.1 Detailed Description

Contains a pin values for pin type onboarding process.

4.31.2 Field Documentation

4.31.2.1 pin `unsigned char pin[8]`

actual pin values

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

- st-device-sdk-c/src/include/st_dev.h

4.32 iot_registered_data Struct Reference

Contains "registration message" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_registered_data:

Data Fields

- char deviceId [IOT_REG_UUID_STR_LEN+1]

  device Id, allocated from server
- bool updated

  reflect getting device id
- bool new_reged

  reflect that it is new registration process or not

4.32.1 Detailed Description

Contains "registration message" data.

4.32.2 Field Documentation

4.32.2.1 deviceId `char deviceId[IOT_REG_UUID_STR_LEN+1]`

device Id, allocated from server

4.32.2.2 new_reged `bool new_reged`

reflect that it is new registration process or not

4.32.2.3 updated `bool updated`

reflect getting device id

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

- st-device-sdk-c/src/include/iot_main.h

4.33 iot_state_data Struct Reference

Contains "iot core's main state" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_state_data:

Data Fields

- iot_state_t iot_state

  current iot core's state
- int opt

  additional option for each state

4.33.1 Detailed Description

Contains "iot core's main state" data.

4.33.2 Field Documentation

4.33.2.1 iot_state `iot_state_t iot_state`

current iot core's state

4.33.2.2 opt `int opt`

additional option for each state

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

- st-device-sdk-c/src/include/iot_main.h

4.34 iot_uuid Struct Reference

Contains "uuid" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_uuid:

Data Fields

- unsigned char id [16]

  actual uuid values, 16 octet

4.34.1 Detailed Description

Contains "uuid" data.

4.34.2 Field Documentation

4.34.2.1 id `unsigned char id[16]`

actual uuid values, 16 octet

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

- st-device-sdk-c/src/include/iot_main.h

4.35 iot_wifi_conf Struct Reference

Contains a "wifi stack configuration" data.

```
#include <iot_bsp_wifi.h>
```

Collaboration diagram for iot_wifi_conf:

Data Fields

- iot_wifi_mode_t mode

  wifi operation mode
- char ssid [IOT_WIFI_MAX_SSID_LEN]

  wifi SSID string
- char pass [IOT_WIFI_MAX_PASS_LEN]

  wifi password string
- uint8_t bssid [IOT_WIFI_MAX_BSSID_LEN]

  wifi mac address

4.35.1 Detailed Description

Contains a "wifi stack configuration" data.

4.35.2 Field Documentation

4.35.2.1 bssid `uint8_t bssid[IOT_WIFI_MAX_BSSID_LEN]`

wifi mac address

4.35.2.2 mode `iot_wifi_mode_t mode`

wifi operation mode

4.35.2.3 pass `char pass[IOT_WIFI_MAX_PASS_LEN]`

wifi password string

4.35.2.4 ssid `char ssid[IOT_WIFI_MAX_SSID_LEN]`

wifi SSID string

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

- st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h

4.36 iot_wifi_prov_data Struct Reference

Contains "wifi provisioning" data.

```
#include <iot_main.h>
```

Collaboration diagram for iot_wifi_prov_data:

Data Fields

- char ssid [IOT_WIFI_PROV_SSID_LEN]

  wifi SSID string
- char password [IOT_WIFI_PROV_PASSWORD_LEN]

  wifi password string
- struct iot_mac bssid

  wifi mac addresss struct
- iot_wifi_auth_mode_t security_type

  wifi security type such as WEP, PSK2..

4.36.1 Detailed Description

Contains "wifi provisioning" data.

4.36.2 Field Documentation

4.36.2.1 bssid `struct iot_mac bssid`

wifi mac addresss struct

4.36.2.2 password `char password[IOT_WIFI_PROV_PASSWORD_LEN]`

wifi password string

4.36.2.3 security_type `iot_wifi_auth_mode_t security_type`

wifi security type such as WEP, PSK2..

4.36.2.4 ssid `char ssid[IOT_WIFI_PROV_SSID_LEN]`

wifi SSID string

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

- st-device-sdk-c/src/include/iot_main.h

4.37 iot_wifi_scan_result_t Struct Reference

Contains a "wifi scan" data.

```
#include <iot_bsp_wifi.h>
```

Collaboration diagram for iot_wifi_scan_result_t:

Data Fields

- uint8_t bssid [IOT_WIFI_MAX_BSSID_LEN]

  wifi mac address
- uint8_t ssid [IOT_WIFI_MAX_SSID_LEN+1]

  wifi SSID string
- int8_t rssi

  wifi signal strength
- uint16_t freq

  wifi operation channel
- iot_wifi_auth_mode_t authmode

  wifi authentication mode

4.37.1 Detailed Description

Contains a "wifi scan" data.

4.37.2 Field Documentation

4.37.2.1 authmode `iot_wifi_auth_mode_t authmode`

wifi authentication mode

4.37.2.2 bssid `uint8_t bssid[IOT_WIFI_MAX_BSSID_LEN]`

wifi mac address

4.37.2.3 freq `uint16_t freq`

wifi operation channel

4.37.2.4 rssi `int8_t rssi`

wifi signal strength

4.37.2.5 ssid `uint8_t ssid[IOT_WIFI_MAX_SSID_LEN+1]`

wifi SSID string

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

- st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h

4.38 MessageData Struct Reference

```
#include <iot_mqtt_client.h>
```

Collaboration diagram for MessageData:

Data Fields

- MQTTMessage * message
- MQTTString * topicName

4.38.1 Field Documentation

4.38.1.1 message `MQTTMessage* message`

4.38.1.2 topicName `MQTTString* topicName`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.39 MQTTClient::MessageHandlers Struct Reference

```
#include <iot_mqtt_client.h>
```

Collaboration diagram for MQTTClient::MessageHandlers:

Data Fields

- const char * topicFilter
- void(* fp)(MessageData *, void *)
- void * userData

4.39.1 Field Documentation

4.39.1.1 fp `void(* fp) (MessageData *, void *)`

4.39.1.2 topicFilter `const char* topicFilter`

4.39.1.3 userData `void* userData`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.40 MQTTClient Struct Reference

```
#include <iot_mqtt_client.h>
```

Collaboration diagram for MQTTClient:

Data Structures

- struct MessageHandlers

Data Fields

- unsigned int next_packetid
- unsigned int command_timeout_ms
- size_t buf_size
- size_t readbuf_size
- unsigned char * buf
- unsigned char * readbuf
- unsigned int keepAliveInterval
- char ping_outstanding
- int ping_retry_count
- int isconnected
- int cleansession
- struct MQTTClient::MessageHandlers messageHandlers [MAX_MESSAGE_HANDLERS]
- void(* defaultMessageHandler)(MessageData *, void *)
- void * defaultUserData
- iot_net_socket * ipstack
- iot_os_timer last_sent
- iot_os_timer last_received
- iot_os_timer ping_wait

4.40.1 Field Documentation

4.40.1.1 buf `unsigned char* buf`

4.40.1.2 buf_size `size_t buf_size`

4.40.1.3 cleansession `int cleansession`

4.40.1.4 command_timeout_ms `unsigned int command_timeout_ms`

4.40.1.5 defaultMessageHandler `void(* defaultMessageHandler) (MessageData *, void *)`

4.40.1.6 defaultUserData `void* defaultUserData`

4.40.1.7 ipstack `iot_net_socket* ipstack`

4.40.1.8 isconnected `int isconnected`

4.40.1.9 keepAliveInterval `unsigned int keepAliveInterval`

4.40.1.10 last_received `iot_os_timer last_received`

4.40.1.11 last_sent `iot_os_timer last_sent`

4.40.1.12 messageHandlers `struct MQTTClient::MessageHandlers messageHandlers[MAX_MESSAGE_HANDLERS]`

4.40.1.13 next_packetid `unsigned int next_packetid`

4.40.1.14 ping_outstanding `char ping_outstanding`

4.40.1.15 ping_retry_count `int ping_retry_count`

4.40.1.16 ping_wait `iot_os_timer ping_wait`

4.40.1.17 readbuf `unsigned char * readbuf`

4.40.1.18 readbuf_size `size_t readbuf_size`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.41 MQTTConnackData Struct Reference

```
#include <iot_mqtt_client.h>
```

Collaboration diagram for MQTTConnackData:

Data Fields

- unsigned char rc
- unsigned char sessionPresent

4.41.1 Field Documentation

4.41.1.1 rc `unsigned char rc`

4.41.1.2 sessionPresent `unsigned char sessionPresent`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.42 MQTTConnackFlags Union Reference

```
#include <iot_mqtt_connect.h>
```

Collaboration diagram for MQTTConnackFlags:

Data Fields

- unsigned char all
- struct {
  unsigned int sessionpresent: 1
  unsigned int reserved: 7
  } bits

4.42.1 Field Documentation

4.42.1.1 all `unsigned char all`

all connack flags

4.42.1.2 bits `struct { ... } bits`

4.42.1.3 reserved `unsigned int reserved`

unused

4.42.1.4 sessionpresent `unsigned int sessionpresent`

session present flag

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h

4.43 MQTTConnectFlags Union Reference

```
#include <iot_mqtt_connect.h>
```

Collaboration diagram for MQTTConnectFlags:

Data Fields

- unsigned char all
- struct {
  unsigned int: 1
  unsigned int cleansession: 1
  unsigned int will: 1
  unsigned int willQoS: 2
  unsigned int willRetain: 1
  unsigned int password: 1
  unsigned int username: 1
  } bits

4.43.1 Field Documentation

4.43.1.1 all `unsigned char all`

all connect flags

4.43.1.2 bits `struct { ... } bits`

4.43.1.3 cleansession `unsigned int cleansession`

cleansession flag

4.43.1.4 int `unsigned int`

unused

4.43.1.5 password `unsigned int password`

3.1 password

4.43.1.6 username `unsigned int username`

3.1 user name

4.43.1.7 will `unsigned int will`

will flag

4.43.1.8 willQoS `unsigned int willQoS`

will QoS value

4.43.1.9 willRetain `unsigned int willRetain`

will retain setting

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h

4.44 MQTTHeader Union Reference

```
#include <iot_mqtt_packet.h>
```

Collaboration diagram for MQTTHeader:

Data Fields

- unsigned char byte
- struct {
  unsigned int retain: 1
  unsigned int qos: 2
  unsigned int dup: 1
  unsigned int type: 4
  } bits

4.44.1 Detailed Description

Bitfields for the MQTT header byte.

4.44.2 Field Documentation

4.44.2.1 bits `struct { ... } bits`

4.44.2.2 byte `unsigned char byte`

the whole byte

4.44.2.3 dup `unsigned int dup`

DUP flag bit

4.44.2.4 qos `unsigned int qos`

QoS value, 0, 1 or 2

4.44.2.5 retain `unsigned int retain`

retained flag bit

4.44.2.6 type `unsigned int type`

message type nibble

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h

4.45 MQTTLenString Struct Reference

```
#include <iot_mqtt_packet.h>
```

Collaboration diagram for MQTTLenString:

Data Fields

- int len
- char * data

4.45.1 Field Documentation

4.45.1.1 data `char* data`

4.45.1.2 len `int len`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h

4.46 MQTTMessage Struct Reference

```
#include <iot_mqtt_client.h>
```

Collaboration diagram for MQTTMessage:

Data Fields

- enum QoS qos
- unsigned char retained
- unsigned char dup
- unsigned short id
- void * payload
- size_t payloadlen

4.46.1 Field Documentation

4.46.1.1 dup `unsigned char dup`

4.46.1.2 id `unsigned short id`

4.46.1.3 payload `void* payload`

4.46.1.4 payloadlen `size_t payloadlen`

4.46.1.5 qos `enum QoS qos`

4.46.1.6 retained `unsigned char retained`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.47 MQTTPacket_connectData Struct Reference

```
#include <iot_mqtt_connect.h>
```

Collaboration diagram for MQTTPacket_connectData:

Data Fields

- char struct_id [4]
- int struct_version
- unsigned char MQTTVersion
- MQTTString clientID
- unsigned short keepAliveInterval

- unsigned char cleansession
- unsigned char willFlag
- MQTTPacket_willOptions will
- MQTTString username
- MQTTString password

4.47.1 Field Documentation

4.47.1.1 cleansession `unsigned char cleansession`

4.47.1.2 clientID `MQTTString clientID`

4.47.1.3 keepAliveInterval `unsigned short keepAliveInterval`

4.47.1.4 MQTTVersion `unsigned char MQTTVersion`

Version of MQTT to be used. 3 = 3.1 4 = 3.1.1

4.47.1.5 password `MQTTString password`

4.47.1.6 struct_id `char struct_id[4]`

The eyecatcher for this structure. must be MQTC.

4.47.1.7 struct_version `int struct_version`

The version number of this structure. Must be 0

4.47.1.8 username `MQTTString username`

4.47.1.9 will `MQTTPacket_willOptions will`

4.47.1.10 willFlag `unsigned char willFlag`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h

4.48 MQTTPacket_willOptions Struct Reference

```
#include <iot_mqtt_connect.h>
```

Collaboration diagram for MQTTPacket_willOptions:

Data Fields

- char struct_id [4]
- int struct_version
- MQTTString topicName
- MQTTString message
- unsigned char retained
- char qos

4.48.1 Detailed Description

Defines the MQTT "Last Will and Testament" (LWT) settings for the connect packet.

4.48.2 Field Documentation

4.48.2.1 message `MQTTString message`

The LWT payload.

4.48.2.2 qos `char qos`

The quality of service setting for the LWT message (see MQTTAsync_message.qos and qos).

4.48.2.3 retained `unsigned char retained`

The retained flag for the LWT message (see MQTTAsync_message.retained).

4.48.2.4 struct_id `char struct_id[4]`

The eyecatcher for this structure. must be MQTW.

4.48.2.5 struct_version `int struct_version`

The version number of this structure. Must be 0

4.48.2.6 topicName `MQTTString topicName`

The LWT topic to which the LWT message will be published.

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h

4.49 MQTTString Struct Reference

```
#include <iot_mqtt_packet.h>
```

Collaboration diagram for MQTTString:

Data Fields

- char * cstring
- MQTTLenString lenstring

4.49.1 Field Documentation

4.49.1.1 cstring `char* cstring`

4.49.1.2 lenstring `MQTTLenString lenstring`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h

4.50 MQTTSubackData Struct Reference

`#include <iot_mqtt_client.h>`

Collaboration diagram for MQTTSubackData:

Data Fields

- enum QoS grantedQoS

4.50.1 Field Documentation

4.50.1.1 grantedQoS `enum QoS grantedQoS`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h

4.51 MQTTTransport Struct Reference

`#include <iot_mqtt_packet.h>`

Collaboration diagram for MQTTTransport:

Data Fields

- int(* getfn)(void *, unsigned char *, int)
- void * sck
- int multiplier
- int rem_len
- int len
- char state

4.51.1 Field Documentation

4.51.1.1 getfn `int(* getfn) (void *, unsigned char *, int)`

4.51.1.2 len `int len`

4.51.1.3 multiplier `int multiplier`

4.51.1.4 rem_len `int rem_len`

4.51.1.5 sck `void* sck`

4.51.1.6 state `char state`

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

- st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h

4.52 noti_data_raw_t Union Reference

Contains data for raw data of each notification.

```
#include <st_dev.h>
```

Collaboration diagram for noti_data_raw_t:

Data Structures

- struct _quota
- struct _rate_limit

Data Fields

- struct noti_data_raw_t::_rate_limit rate_limit
- struct noti_data_raw_t::_quota quota

4.52.1 Detailed Description

Contains data for raw data of each notification.

4.52.2 Field Documentation

4.52.2.1 quota `struct noti_data_raw_t::_quota quota`

4.52.2.2 rate_limit `struct noti_data_raw_t::_rate_limit rate_limit`

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

- st-device-sdk-c/src/include/st_dev.h

4.53 url_parse_t Struct Reference

Contains a "url parse" data.

`#include <iot_util.h>`

Collaboration diagram for url_parse_t:

Data Fields

- char * protocol

  broker url's protocol part such as "ssl", "https"
- char * domain

  broker url's domain part such as "test.example.com"
- int port

  broker url's port number part such as 443, 8883'

4.53.1 Detailed Description

Contains a "url parse" data.

4.53.2 Field Documentation

4.53.2.1 domain `char* domain`

broker url's domain part such as "test.example.com"

4.53.2.2 port `int port`

broker url's port number part such as 443, 8883'

4.53.2.3 protocol `char* protocol`

broker url's protocol part such as "ssl", "https"

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

- st-device-sdk-c/src/include/iot_util.h

5 File Documentation

5.1 st-device-sdk-c/doc/mainpage.dox File Reference

5.2 st-device-sdk-c/src/crypto/iot_crypto_ed25519.c File Reference

```
#include <string.h>
#include "iot_main.h"
#include "iot_debug.h"
```

Include dependency graph for iot_crypto_ed25519.c:

Functions

- iot_error_t iot_crypto_ed25519_init_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Prepare a buffer to store the ed25519 and curve25519 keypair.
- void iot_crypto_ed25519_free_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Cleanup the buffer that allocated for keypair.
- iot_error_t iot_crypto_ed25519_convert_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Converts an ed25519 keypair to an x25519 keypair.
- iot_error_t iot_crypto_ed25519_convert_pubkey (unsigned char ∗ed25519_key, unsigned char ∗curve25519↩ _key)

  Converts an ed25519 public key to an x25519 public key.
- iot_error_t iot_crypto_ed25519_convert_seckey (unsigned char ∗ed25519_key, unsigned char ∗curve25519↩ _key)

  Converts an ed25519 secret key to an x25519 secret key.

5.2.1 Function Documentation

5.2.1.1 iot_crypto_ed25519_convert_keypair()

```
iot_error_t iot_crypto_ed25519_convert_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Converts an ed25519 keypair to an x25519 keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair buffers ed25519 keypair should be prepared

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.2.1.2 iot_crypto_ed25519_convert_pubkey()

```
iot_error_t iot_crypto_ed25519_convert_pubkey (
    unsigned char * ed25519_key,
    unsigned char * curve25519_key )
```

Converts an ed25519 public key to an x25519 public key.

Parameters

in	*ed25519_key*	a pointer to a public key buffer
out	*curve25519_key*	a pointer to a buffer to store converted x25519 public key

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.2.1.3 iot_crypto_ed25519_convert_seckey()

```
iot_error_t iot_crypto_ed25519_convert_seckey (
    unsigned char * ed25519_key,
    unsigned char * curve25519_key )
```

Converts an ed25519 secret key to an x25519 secret key.

Parameters

in	*ed25519_key*	a pointer to a secret key buffer
out	*curve25519_key*	a pointer to a buffer to store converted x25519 secret key

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.2.1.4 iot_crypto_ed25519_free_keypair()

```
void iot_crypto_ed25519_free_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Cleanup the buffer that allocated for keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair to cleanup

5.2.1.5 iot_crypto_ed25519_init_keypair()

```
iot_error_t iot_crypto_ed25519_init_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Prepare a buffer to store the ed25519 and curve25519 keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair buffers

Return values

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_MEM_ALLOC	failed to alloc buffer for keypair

5.3 st-device-sdk-c/src/crypto/mbedtls/iot_crypto_mbedtls.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "iot_main.h"
#include "iot_debug.h"
#include "mbedtls/sha256.h"
#include "mbedtls/base64.h"
#include "mbedtls/pk.h"
#include "mbedtls/md.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/entropy.h"
#include "mbedtls/ecp.h"
#include "mbedtls/ecdh.h"
#include "mbedtls/cipher.h"
```

Include dependency graph for iot_crypto_mbedtls.c:

Functions

- iot_error_t iot_crypto_base64_encode (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_↩len, size_t ∗out_len)

  Encode a string as a base64 string.
- iot_error_t iot_crypto_base64_decode (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_↩len, size_t ∗out_len)

  Decode a base64 string as a string.
- iot_error_t iot_crypto_base64_encode_urlsafe (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Encode a string as a urlsafe base64 string.
- iot_error_t iot_crypto_base64_decode_urlsafe (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Decode a urlsafe base64 string as a string.
- iot_error_t iot_crypto_sha256 (unsigned char ∗src, size_t src_len, unsigned char ∗dst)

  Generate a digest by sha256 hash.

- iot_error_t iot_crypto_ecdh_gen_master_secret (unsigned char ∗master, size_t mlen, iot_crypto_ecdh_params_t ∗params)

  Generate a master secret.
- size_t iot_crypto_cipher_get_align_size (iot_crypto_cipher_type_t type, size_t size)

  Calculate the required align size that contains the encrypted data for the input size.
- iot_error_t iot_crypto_cipher_aes (iot_crypto_cipher_info_t ∗info, unsigned char ∗input, size_t ilen, unsigned char ∗out, size_t ∗olen, size_t osize)

  Generic encryption/decryption function.

5.3.1 Function Documentation

5.3.1.1 iot_crypto_base64_decode()

```
iot_error_t iot_crypto_base64_decode (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Decode a base64 string as a string.

Parameters

in	*src*	a pointer to a buffer to decode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully decoded
IOT_ERROR_CRYPTO_BASE64	failed to decode the string

5.3.1.2 iot_crypto_base64_decode_urlsafe()

```
iot_error_t iot_crypto_base64_decode_urlsafe (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Decode a urlsafe base64 string as a string.

Parameters

in	*src*	a pointer to a buffer to decode

Parameters

in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully decoded
IOT_ERROR_CRYPTO_BASE64	failed to decode the string
IOT_ERROR_CRYPTO_BASE64_URLSAFE	failed to encode the string as urlsafe

5.3.1.3 iot_crypto_base64_encode()

```
iot_error_t iot_crypto_base64_encode (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Encode a string as a base64 string.

Parameters

in	*src*	a pointer to a buffer to encode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully encoded
IOT_ERROR_CRYPTO_BASE64	failed to encode the string

5.3.1.4 iot_crypto_base64_encode_urlsafe()

```
iot_error_t iot_crypto_base64_encode_urlsafe (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Encode a string as a urlsafe base64 string.

This function replaces url unsafe characters ('+', '/') to url safe character ('-', '_')

Parameters

in	*src*	a pointer to a buffer to encode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully encoded
IOT_ERROR_CRYPTO_BASE64	failed to encode the string in 3rd lib
IOT_ERROR_CRYPTO_BASE64_URLSAFE	failed to encode the string as urlsafe

Here is the caller graph for this function:

5.3.1.5 iot_crypto_cipher_aes()

```
iot_error_t iot_crypto_cipher_aes (
    iot_crypto_cipher_info_t * info,
    unsigned char * input,
    size_t ilen,
    unsigned char * out,
    size_t * olen,
    size_t osize )
```

Generic encryption/decryption function.

Supported cipher algorithms is AES-256 CBC mode

Parameters

in	*info*	a pointer to a buffer containing cipher context
in	*input*	a pointer to a buffer to encrypt/decrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*out*	a pointer to a buffer to store a signature
out	*olen*	the bytes written to out
in	*osize*	the size of buffer pointed by out in bytes

Return values

IOT_ERROR_NONE	encryption/decryption is a success
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE	not supported cipher is requested.
IOT_ERROR_CRYPTO_CIPHER	failed while encrypting/decrypting

Here is the call graph for this function:

5.3.1.6 iot_crypto_cipher_get_align_size()

```
size_t iot_crypto_cipher_get_align_size (
    iot_crypto_cipher_type_t type,
    size_t size )
```

Calculate the required align size that contains the encrypted data for the input size.

the size of data to encrypt should be aligned when AES is used because AES is operated based on block.

Parameters

in	*type*	cipher type
in	*size*	the size of data to encrypt

Return values

return	the aligned size
return	'0' when error occurs

Here is the caller graph for this function:

5.3.1.7 iot_crypto_ecdh_gen_master_secret()

```
iot_error_t iot_crypto_ecdh_gen_master_secret (
    unsigned char * master,
    size_t mlen,
    iot_crypto_ecdh_params_t * params )
```

Generate a master secret.

This function generates a master secret by combining the shared key and the hashed token. The shared key is generated by ECDH using device's x25519 private key and peer's x25519 public key. The hashed token is shared by peer.

Parameters

out	*master*	a pointer to a buffer to store master secret
in	*mlen*	the size of buffer pointed by master in bytes
in	*params*	a pointer to a buffer containing the private key, public key and hashed token to generate master secret

Return values

IOT_ERROR_NONE	the master secret is sucessfully generated
IOT_ERROR_INVALID_ARGS	params has invalid data
IOT_ERROR_MEM_ALLOC	failed to alloc buffer for shared key
IOT_ERROR_CRYPTO_PK_ECDH	failed in ECDH processing
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.3.1.8 iot_crypto_sha256()

```
iot_error_t iot_crypto_sha256 (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst )
```

Generate a digest by sha256 hash.

Parameters

in	*src*	a pointer to a buffer to generate a digest
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store a digest

Return values

IOT_ERROR_NONE	a digest is sucessfully generated
IOT_ERROR_CRYPTO_SHA256	failed to generate a digest

Here is the caller graph for this function:

5.4 st-device-sdk-c/src/crypto/ss/iot_crypto_ss.c File Reference

```
#include <stdlib.h>
#include "iot_main.h"
```

Include dependency graph for iot_crypto_ss.c:

Functions

- iot_error_t iot_crypto_ss_encrypt (unsigned char ∗input, size_t ilen, unsigned char ∗∗output, size_t ∗olen)

  Encryption of the input data.
- iot_error_t iot_crypto_ss_decrypt (unsigned char ∗input, size_t ilen, unsigned char ∗∗output, size_t ∗olen)

  Decryption of the input data.

5.4.1 Function Documentation

5.4.1.1 iot_crypto_ss_decrypt()

```
iot_error_t iot_crypto_ss_decrypt (
    unsigned char * input,
    size_t ilen,
    unsigned char ** output,
    size_t * olen )
```

Decryption of the input data.

The decryption key is generated from device unique value

Parameters

in	*input*	a pointer to a buffer to decrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*output*	a pointer of pointer to a buffer to store the decrypted data
out	*olen*	the bytes written to output buffer

Return values

IOT_ERROR_NONE	decryption is success
IOT_ERROR_MEM_ALLOC	mem alloc failed for output buffer
IOT_ERROR_CRYPTO_CIPHER_ALIGN	failed to get align size of the input size for output buffer
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE	cipher is requested with not supported algorithm
IOT_ERROR_CRYPTO_CIPHER	cipher operation is failed
IOT_ERROR_CRYPTO_SS_KDF	failed during derivate the key

5.4.1.2 iot_crypto_ss_encrypt()

```
iot_error_t iot_crypto_ss_encrypt (
    unsigned char * input,
    size_t ilen,
    unsigned char ** output,
    size_t * olen )
```

Encryption of the input data.

The encryption key is generated from device unique value

Parameters

in	*input*	a pointer to a buffer to encrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*output*	a pointer of pointer to a buffer to store the encrypted data
out	*olen*	the bytes written to output buffer

Return values

IOT_ERROR_NONE	encryption is success
IOT_ERROR_MEM_ALLOC	mem alloc failed for output buffer
IOT_ERROR_CRYPTO_CIPHER_ALIGN	failed to get align size of the input size for output buffer

Return values

IOT_ERROR_CRYPTO_CIPHER	cipher operation is failed
IOT_ERROR_CRYPTO_SS_KDF	failed during derivate the key

Here is the caller graph for this function:

5.5 st-device-sdk-c/src/easysetup/http/iot_easysetup_http.c File Reference

```
#include <string.h>
#include "lwip_httpd/httpd.h"
#include "lwip_httpd/fs.h"
#include "cJSON.h"
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_debug.h"
#include "iot_easysetup.h"
#include "iot_bsp_wifi.h"
```

Include dependency graph for iot_easysetup_http.c:

Macros

- #define MAX_PAYLOAD_LENGTH 1024
- #define ARRAY_SIZE(x) (int)(sizeof(x)/sizeof(x[0]))
- #define LWIP_HTTPD_POST_MAX_PAYLOAD_LEN 512

Functions

- iot_error_t _iot_easysetup_gen_get_payload (struct iot_context ∗ctx, const char ∗cmd, char ∗∗out_payload)

  http GET method payload handler
- iot_error_t _iot_easysetup_gen_post_payload (struct iot_context ∗ctx, const char ∗cmd, char ∗in_payload, char ∗∗out_payload)

  http POST method payload handler
- err_t httpd_post_begin (void ∗connection, const char ∗uri, const char ∗http_request, u16_t http_request_len, int content_len, char ∗response_uri, u16_t response_uri_len, u8_t ∗post_auto_wnd)
- err_t httpd_post_receive_data (void ∗connection, struct pbuf ∗p)
- void httpd_post_finished (void ∗connection, char ∗response_uri, u16_t response_uri_len)
- void httpd_cgi_handler (const char ∗uri, int iNumParams, char ∗∗pcParam, char ∗∗pcValue, void ∗connection_state)
- int fs_open_custom (struct fs_file ∗file, const char ∗name)
- void fs_close_custom (struct fs_file ∗file)
- void ∗ fs_state_init (struct fs_file ∗file, const char ∗name)
- void fs_state_free (struct fs_file ∗file, void ∗state)
- iot_error_t iot_easysetup_init (struct iot_context ∗ctx)

  Start eayssetup device-to-device sequence.
- void iot_easysetup_deinit (struct iot_context ∗ctx)

  Stop eayssetup device-to-device sequence.

Variables

- const char ∗ post_cgi_cmds [ ]
- const char ∗ get_cgi_cmds [ ]

5.5.1 Macro Definition Documentation

5.5.1.1 ARRAY_SIZE

```
#define ARRAY_SIZE(
    x ) (int)(sizeof(x)/sizeof(x[0]))
```

5.5.1.2 LWIP_HTTPD_POST_MAX_PAYLOAD_LEN

```
#define LWIP_HTTPD_POST_MAX_PAYLOAD_LEN 512
```

5.5.1.3 MAX_PAYLOAD_LENGTH

```
#define MAX_PAYLOAD_LENGTH 1024
```

5.5.2 Function Documentation

5.5.2.1 _iot_easysetup_gen_get_payload()

```
iot_error_t _iot_easysetup_gen_get_payload (
    struct iot_context * ctx,
    const char * cmd,
    char ** out_payload )
```

http GET method payload handler

This function handle GET method cgi request

Parameters

in	*ctx*	iot_context handle
in	*cmd*	GET method uri
out	*out_payload*	output payload buffer for GET method. caller has full responsibility to free this memory.

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success

5.5.2.2 _iot_easysetup_gen_post_payload()

```
iot_error_t _iot_easysetup_gen_post_payload (
    struct iot_context * ctx,
    const char * cmd,
    char * in_payload,
    char ** out_payload )
```

http POST method payload handler

This function handle POST method cgi request

Parameters

in	*ctx*	iot_context handle
in	*cmd*	POST method uri
in	*in_payload*	client updated payload via POST method has given to here. this shouldn't be freed inside of this function.
out	*out_payload*	output payload for http response. caller has full responsibility to free this memory

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success

Here is the caller graph for this function:

5.5.2.3 fs_close_custom()

```
void fs_close_custom (
    struct fs_file * file )
```

5.5.2.4 fs_open_custom()

```
int fs_open_custom (
    struct fs_file * file,
    const char * name )
```

5.5.2.5 fs_state_free()

```
void fs_state_free (
    struct fs_file * file,
    void * state )
```

5.5.2.6 fs_state_init()

```
void* fs_state_init (
    struct fs_file * file,
    const char * name )
```

5.5.2.7 httpd_cgi_handler()

```
void httpd_cgi_handler (
    const char * uri,
    int iNumParams,
    char ** pcParam,
    char ** pcValue,
    void * connection_state )
```

Here is the call graph for this function:

5.5.2.8 httpd_post_begin()

```
err_t httpd_post_begin (
    void * connection,
    const char * uri,
    const char * http_request,
    u16_t http_request_len,
    int content_len,
    char * response_uri,
    u16_t response_uri_len,
    u8_t * post_auto_wnd )
```

5.5.2.9 httpd_post_finished()

```
void httpd_post_finished (
    void * connection,
    char * response_uri,
    u16_t response_uri_len )
```

5.5.2.10 httpd_post_receive_data()

```
err_t httpd_post_receive_data (
    void * connection,
    struct pbuf * p )
```

5.5.2.11 iot_easysetup_deinit()

```
void iot_easysetup_deinit (
    struct iot_context * ctx )
```

Stop eayssetup device-to-device sequence.

This function stops httpd working

Parameters

in	*ctx*	iot_context handle

Returns

void

5.5.2.12 iot_easysetup_init()

```
iot_error_t iot_easysetup_init (
    struct iot_context * ctx )
```

Start eayssetup device-to-device sequence.

This function makes wifi mode as soft-ap and starts httpd

Parameters

in	*ctx*	iot_context handle

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_UNINITIALIZED	error

5.5.3 Variable Documentation

5.5.3.1 get_cgi_cmds

```
const char* get_cgi_cmds[]
```

Initial value:

```
=
{
    IOT_ES_URI_GET_DEVICEINFO,
    IOT_ES_URI_GET_WIFISCANINFO,
    IOT_ES_URI_GET_POST_RESPONSE,
    IOT_ES_URI_GET_LOGS_SYSTEMINFO,
    IOT_ES_URI_GET_LOGS_DUMP,
}
```

5.5.3.2 post_cgi_cmds

```
const char* post_cgi_cmds[]
```

Initial value:

```
=
{
    IOT_ES_URI_POST_KEYINFO,
    IOT_ES_URI_POST_CONFIRMINFO,
    IOT_ES_URI_POST_CONFIRM,
    IOT_ES_URI_POST_WIFIPROVISIONINGINFO,
    IOT_ES_URI_POST_SETUPCOMPLETE,
    IOT_ES_URI_POST_LOGS,
}
```

5.6 st-device-sdk-c/src/easysetup/iot_easysetup_crypto.c File Reference

```
#include <string.h>
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_nv_data.h"
#include "iot_debug.h"
```

Include dependency graph for iot_easysetup_crypto.c:

Functions

- void iot_es_crypto_init_pk (iot_crypto_pk_info_t *pk_info, iot_crypto_pk_type_t type)

  initialize the buffer for pubkey information
- iot_error_t iot_es_crypto_load_pk (iot_crypto_pk_info_t *pk_info)

  easy setup crypto to load pubkey
- void iot_es_crypto_free_pk (iot_crypto_pk_info_t *pk_info)

  easy setup crypto to free pubkey

5.6.1 Function Documentation

5.6.1.1 iot_es_crypto_free_pk()

```
void iot_es_crypto_free_pk (
    iot_crypto_pk_info_t * pk_info )
```

easy setup crypto to free pubkey

this function frees the loaded pubkey information

Parameters

in	*pk_info*	loaded pubkey information pointer

5.6.1.2 iot_es_crypto_init_pk()

```
void iot_es_crypto_init_pk (
    iot_crypto_pk_info_t * pk_info,
    iot_crypto_pk_type_t type )
```

initialize the buffer for pubkey information

this function uses to initialize

Parameters

in	*pk_info*	pubkey information structure to initialize
in	*type*	pubkey type, RSA or ED25519

5.6.1.3 iot_es_crypto_load_pk()

```
iot_error_t iot_es_crypto_load_pk (
    iot_crypto_pk_info_t * pk_info )
```

easy setup crypto to load pubkey

this function loads pubkey information

Parameters

in	*pk_info*	pubkey information pointer for loading

Return values

IOT_ERROR_NONE	success.

5.7 st-device-sdk-c/src/easysetup/iot_easysetup_d2d.c File Reference

```
#include <string.h>
#include "cJSON.h"
#include "iot_main.h"
#include "iot_bsp_random.h"
#include "iot_easysetup.h"
#include "iot_internal.h"
#include "iot_nv_data.h"
#include "iot_util.h"
#include "iot_uuid.h"
#include "iot_debug.h"
```

Include dependency graph for iot_easysetup_d2d.c:

Macros

- #define HASH_SIZE (4)
- #define PIN_SIZE 8
- #define MAC_ADDR_BUFFER_SIZE 20
- #define URL_BUFFER_SIZE 64
- #define WIFIINFO_BUFFER_SIZE 20
- #define ES_CONFIRM_MAX_DELAY 10000
- #define JSON_H cJSON

Functions

- iot_error_t iot_easysetup_create_ssid (struct iot_devconf_prov_data ∗devconf, char ∗ssid, size_t ssid_len)

  Create E4 type SSID.
- void st_conn_ownership_confirm (IOT_CTX ∗iot_ctx, bool confirm)

  easysetup user confirm report function
- iot_error_t iot_easysetup_request_handler (struct iot_context ∗ctx, struct iot_easysetup_payload request)

  easysetup cgi request handler

5.7.1 Macro Definition Documentation

5.7.1.1 ES_CONFIRM_MAX_DELAY `#define ES_CONFIRM_MAX_DELAY 10000`

5.7.1.2 HASH_SIZE `#define HASH_SIZE (4)`

5.7.1.3 JSON_H `#define JSON_H cJSON`

5.7.1.4 MAC_ADDR_BUFFER_SIZE `#define MAC_ADDR_BUFFER_SIZE 20`

5.7.1.5 PIN_SIZE `#define PIN_SIZE 8`

5.7.1.6 URL_BUFFER_SIZE `#define URL_BUFFER_SIZE 64`

5.7.1.7 WIFIINFO_BUFFER_SIZE `#define WIFIINFO_BUFFER_SIZE 20`

5.7.2 Function Documentation

5.7.2.1 iot_easysetup_create_ssid()

```
iot_error_t iot_easysetup_create_ssid (
    struct iot_devconf_prov_data * devconf,
    char * ssid,
    size_t ssid_len )
```

Create E4 type SSID.

This function create E4 type soft-ap SSID for this device

Parameters

in	*devconf*	things static information
out	*ssid*	created ssid
in	*ssid_len*	ssid buffer length including null termination

Returns

iot_state_t

Return values

IOT_ERROR_NONE	success

Here is the call graph for this function:

5.7.2.2 iot_easysetup_request_handler()

```
iot_error_t iot_easysetup_request_handler (
    struct iot_context * ctx,
    struct iot_easysetup_payload request )
```

easysetup cgi request handler

This function runs from iot-task by executing actual cgi payload manipulation.
result will be transferred to httpd task (tiT) as easysetup response queue parameter.

Parameters

in	*ctx*	iot_context handle
in	*request*	easysetup payload as input

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success

5.7.2.3 st_conn_ownership_confirm()

```
void st_conn_ownership_confirm (
    IOT_CTX * iot_ctx,
    bool confirm )
```

easysetup user confirm report function

This function reports the user confirmation to easysetup

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*confirm*	user confirmation result

Here is the call graph for this function:

5.8 st-device-sdk-c/src/easysetup/iot_easysetup_st_mqtt.c File Reference

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_util.h"
#include "iot_nv_data.h"
#include "iot_debug.h"
#include "iot_jwt.h"
#include "iot_crypto.h"
#include "iot_os_util.h"
#include "cJSON.h"
```

Include dependency graph for iot_easysetup_st_mqtt.c:

Functions

- iot_error_t iot_es_connect (struct iot_context ∗ctx, int conn_type)

  easy setup connect
- iot_error_t iot_es_disconnect (struct iot_context ∗ctx, int conn_type)

  easy setup disconnect

5.8.1 Function Documentation

5.8.1.1 iot_es_connect()

```
iot_error_t iot_es_connect (
    struct iot_context * ctx,
    int conn_type )
```

easy setup connect

this function tries to connect server for registration or communication process

Parameters

in	*ctx*	iot-core context
in	*conn_type*	set connection type. registration or communication with server

Return values

IOT_ERROR_NONE	success.

5.8.1.2 iot_es_disconnect()

```
iot_error_t iot_es_disconnect (
    struct iot_context * ctx,
    int conn_type )
```

easy setup disconnect

this function tries to disconnect server for registration or communication process

Parameters

in	*ctx*	iot-core context
in	*conn_type*	set connection type. registration or communication with server

Return values

IOT_ERROR_NONE	success.

Here is the caller graph for this function:

5.9 st-device-sdk-c/src/include/bsp/iot_bsp_debug.h File Reference

```
#include <iot_debug.h>
```

Include dependency graph for iot_bsp_debug.h:

This graph shows which files directly or indirectly include this file:

Functions

- void iot_bsp_debug (iot_debug_level_t level, const char *tag, const char *fmt,...)

  Write message into the log.
- void iot_bsp_debug_check_heap (const char *tag, const char *func, const int line, const char *fmt,...)

  Check memory(heap) status.

5.9.1 Function Documentation

5.9.1.1 iot_bsp_debug()

```
void iot_bsp_debug (
    iot_debug_level_t level,
    const char * tag,
    const char * fmt,
    ... )
```

Write message into the log.

This function is not intended to be used directly. Instead, use one of IOT_ERROR, IOT_WARN, IOT_INFO, IOT_↩ DEBUG macros.

Parameters

in	*level*	log level • IOT_DEBUG_LEVEL_NONE • IOT_DEBUG_LEVEL_ERROR • IOT_DEBUG_LEVEL_WARN • IOT_DEBUG_LEVEL_INFO • IOT_DEBUG_LEVEL_DEBUG
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.9.1.2 iot_bsp_debug_check_heap()

```
void iot_bsp_debug_check_heap (
    const char * tag,
    const char * func,
    const int line,
    const char * fmt,
    ... )
```

Check memory(heap) status.

This function check memory(heap) status

Parameters

in	*tag*	tag
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.10 st-device-sdk-c/src/include/bsp/iot_bsp_fs.h File Reference

`#include "iot_error.h"`
Include dependency graph for iot_bsp_fs.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_bsp_fs_handle_t

iot_bsp_fs_open_mode_t

file system open mode.

- enum iot_bsp_fs_open_mode_t { FS_READONLY, FS_READWRITE }
- iot_error_t iot_bsp_fs_init ()

  Initialize a file system.
- iot_error_t iot_bsp_fs_deinit ()

  Deinitialize a file system.
- iot_error_t iot_bsp_fs_open (const char *filename, iot_bsp_fs_open_mode_t mode, iot_bsp_fs_handle_t *handle)

  Open a file.
- iot_error_t iot_bsp_fs_open_from_stnv (const char *filename, iot_bsp_fs_handle_t *handle)

  Open a file from stnv partition.
- iot_error_t iot_bsp_fs_read (iot_bsp_fs_handle_t handle, char *buffer, unsigned int length)

  Read a file.
- iot_error_t iot_bsp_fs_write (iot_bsp_fs_handle_t handle, const char *data, unsigned int length)

  Write a file.
- iot_error_t iot_bsp_fs_close (iot_bsp_fs_handle_t handle)

  Close a file.
- iot_error_t iot_bsp_fs_remove (const char *filename)

  Remove a file.

5.10.1 Enumeration Type Documentation

5.10.1.1 iot_bsp_fs_open_mode_t `enum iot_bsp_fs_open_mode_t`

Enumerator

FS_READONLY	
FS_READWRITE	

5.10.2 Function Documentation

5.10.2.1 iot_bsp_fs_close() `iot_error_t iot_bsp_fs_close ( iot_bsp_fs_handle_t handle )`

Close a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().

Return values

IOT_ERROR_NONE	File close successful.

Here is the caller graph for this function:

5.10.2.2 iot_bsp_fs_deinit() `iot_error_t iot_bsp_fs_deinit ( )`

Deinitialize a file system.

Return values

IOT_ERROR_NONE	File-system deinit successful.
IOT_ERROR_DEINIT_F↩ AIL	File-system deinit failed.

Here is the caller graph for this function:

5.10.2.3 iot_bsp_fs_init() `iot_error_t iot_bsp_fs_init ( )`

Initialize a file system.

This function initializes the file-system of the device. You must call this function before using file-management function (open, read, write, etc.)

Return values

IOT_ERROR_NONE	File-system init successful.
IOT_ERROR_INIT_F↩ AIL	File-system init failed.

Here is the caller graph for this function:

5.10.2.4 iot_bsp_fs_open()

```
iot_error_t iot_bsp_fs_open (
    const char * filename,
    iot_bsp_fs_open_mode_t mode,
    iot_bsp_fs_handle_t * handle )
```

Open a file.

Parameters

in	*filename*	File name.
in	*mode*	File open mode. FS_READONLY or FS_READWRITE.
out	*handle*	A pointer to the file handle.

Return values

IOT_ERROR_NONE	File open successful.
IOT_ERROR_FS_OPEN_F↩ AIL	File open failed.

Here is the caller graph for this function:

5.10.2.5 iot_bsp_fs_open_from_stnv()

```
iot_error_t iot_bsp_fs_open_from_stnv (
    const char * filename,
    iot_bsp_fs_handle_t * handle )
```

Open a file from stnv partition.

This function will return the read-only file-system handle. You can access the stnv partition's file though this handle.

Parameters

in	*filename*	File name.
out	*handle*	A pointer to the file handle.

Return values

IOT_ERROR_NONE	File open successful.
IOT_ERROR_FS_OPEN_FAIL	File open failed.

5.10.2.6 iot_bsp_fs_read()

```
iot_error_t iot_bsp_fs_read (
    iot_bsp_fs_handle_t handle,
    char * buffer,
    unsigned int length )
```

Read a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().

Parameters

out	*buffer*	A pointer to buffer array to store the read data from the file.
in	*length*	The size of buffer.

Return values

IOT_ERROR_NONE	File read successful.
IOT_ERROR_FS_READ_FAIL	File read failed.
IOT_ERROR_FS_NO_FILE	No file.

Here is the caller graph for this function:

5.10.2.7 iot_bsp_fs_remove()

```
iot_error_t iot_bsp_fs_remove (
    const char * filename )
```

Remove a file.

Parameters

in	*filename*	File name.

Return values

IOT_ERROR_NONE	File remove successful.
IOT_ERROR_INVALID_ARGS	Invalid filename
IOT_ERROR_FS_REMOVE_FAIL	File remove failed.

Here is the caller graph for this function:

5.10.2.8 iot_bsp_fs_write()

```
iot_error_t iot_bsp_fs_write (
    iot_bsp_fs_handle_t handle,
    const char * data,
    unsigned int length )
```

Write a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().
in	*data*	A pointer to data array to write to the file.
in	*length*	The size of data.

Return values

IOT_ERROR_NONE	File write successful.
IOT_ERROR_FS_WRITE_FAIL	File write failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.11 st-device-sdk-c/src/include/bsp/iot_bsp_nv_data.h File Reference

```
#include "iot_nv_data.h"
```

Include dependency graph for iot_bsp_nv_data.h:

This graph shows which files directly or indirectly include this file:

Functions

- const char * iot_bsp_nv_get_data_path (iot_nvd_t nv_type)

  Get a nv data path.

5.11.1 Function Documentation

5.11.1.1 iot_bsp_nv_get_data_path()

```
const char* iot_bsp_nv_get_data_path (
    iot_nvd_t nv_type )
```

Get a nv data path.

This function will return the nv data path. The detail of data path depends on the file-system type. ex : espressif nvs-system - nvs key ("nvItemKey") linux file-system - file path ("/stnv/nvItemKey")

Parameters

in	*nv_type*	The type of nv data. declaration is in "iot_nv_data.h"

Returns

nv data path.

See also

iot_nv_data.h

Here is the caller graph for this function:

5.12 st-device-sdk-c/src/include/bsp/iot_bsp_random.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- unsigned int iot_bsp_random ()

  Generate random number.

5.12.1 Function Documentation

5.12.1.1 iot_bsp_random() `unsigned int iot_bsp_random ( )`

Generate random number.

This function generates and returns a random number.

Returns

unsigned int random number

Here is the caller graph for this function:

5.13 st-device-sdk-c/src/include/bsp/iot_bsp_system.h File Reference

```
#include "iot_error.h"
#include <stddef.h>
```

Include dependency graph for iot_bsp_system.h:

This graph shows which files directly or indirectly include this file:

Macros

- #define IOT_REBOOT() iot_bsp_system_reboot()

  Macro to use iot_bsp_system_reboot function.
- #define IOT_POWEROFF() iot_bsp_system_poweroff()

  Macro to use iot_bsp_system_poweroff function.

Functions

- void iot_bsp_system_reboot ()

  Restart system.
- void iot_bsp_system_poweroff ()

  Shutdown system.
- iot_error_t iot_bsp_system_get_time_in_sec (char ∗buf, unsigned int buf_len)

  Get system time in second.
- iot_error_t iot_bsp_system_set_time_in_sec (const char ∗time_in_sec)

  Set system time in second.
- iot_error_t iot_bsp_system_get_uniqueid (unsigned char ∗∗uid, size_t ∗olen)

  Get device unique value.

5.13.1 Macro Definition Documentation

5.13.1.1 IOT_POWEROFF `#define IOT_POWEROFF( ) iot_bsp_system_poweroff()`

Macro to use iot_bsp_system_poweroff function.

5.13.1.2 IOT_REBOOT `#define IOT_REBOOT( ) iot_bsp_system_reboot()`

Macro to use iot_bsp_system_reboot function.

5.13.2 Function Documentation

5.13.2.1 iot_bsp_system_get_time_in_sec()

```
iot_error_t iot_bsp_system_get_time_in_sec (
    char * buf,
    unsigned int buf_len )
```

Get system time in second.

Parameters

out	*buf*	A pointer to data array to store the system time in second.
in	*buf_len*	The length of buffer.

Return values

IOT_ERROR_NONE	Set time successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.

5.13.2.2 iot_bsp_system_get_uniqueid()

```
iot_error_t iot_bsp_system_get_uniqueid (
    unsigned char ** uid,
    size_t * olen )
```

Get device unique value.

The source of unique value

Parameters

out	*uid*	a pointer of pointer to a unique id buffer
out	*olen*	the bytes written to unique id buffer

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_MEM_ALLOC	alloc failed for unique id buffer
IOT_ERROR_NOT_IMPLEMENTED	no way to make unique id

5.13.2.3 iot_bsp_system_poweroff() `void iot_bsp_system_poweroff ( )`

Shutdown system.

This function shuts down system.

5.13.2.4 iot_bsp_system_reboot() `void iot_bsp_system_reboot ( )`

Restart system.

This function restarts system.

5.13.2.5 iot_bsp_system_set_time_in_sec() `iot_error_t iot_bsp_system_set_time_in_sec ( const char * time_in_sec )`

Set system time in second.

Parameters

in	*time_in_sec*	A time value in second from struct timeval's tv_sec (ex : 1546300800)

Return values

IOT_ERROR_NONE	Set time successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.

5.14 st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h File Reference

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

Include dependency graph for iot_bsp_wifi.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_wifi_conf

  Contains a "wifi stack configuration" data.
- struct iot_wifi_scan_result_t

  Contains a "wifi scan" data.
- struct iot_mac

  Contains "wifi mac" data.

Macros

- #define IOT_WIFI_MAX_SSID_LEN (32)
- #define IOT_WIFI_MAX_PASS_LEN (64)
- #define IOT_WIFI_MAX_BSSID_LEN (6)
- #define IOT_WIFI_MAX_SCAN_RESULT (10)
- #define IOT_SOFT_AP_CHANNEL (1)
- #define IOT_WIFI_CMD_TIMEOUT 5000

Enumerations

- enum iot_wifi_mode_t { IOT_WIFI_MODE_OFF = 0, IOT_WIFI_MODE_SCAN, IOT_WIFI_MODE_STATION, IOT_WIFI_MODE_SOFTAP, IOT_WIFI_MODE_P2P, IOT_WIFI_MODE_UNDEFINED = 0x20 }
- enum iot_wifi_freq_t { IOT_WIFI_FREQ_2_4G_ONLY = 0, IOT_WIFI_FREQ_5G_ONLY, IOT_WIFI_FREQ_2_4G_5G_BOTH }
- enum iot_wifi_auth_mode_t { IOT_WIFI_AUTH_OPEN = 0, IOT_WIFI_AUTH_WEP, IOT_WIFI_AUTH_WPA_PSK, IOT_WIFI_AUTH_WPA2_PSK, IOT_WIFI_AUTH_WPA_WPA2_PSK, IOT_WIFI_AUTH_WPA2_ENTERPRISE, IOT_WIFI_AUTH_MAX }

Functions

- iot_error_t iot_bsp_wifi_init ()

  Initialize Wi-Fi function.
- iot_error_t iot_bsp_wifi_set_mode (iot_wifi_conf *conf)

  Set the Wi-Fi mode.
- uint16_t iot_bsp_wifi_get_scan_result (iot_wifi_scan_result_t *scan_result)

  Get the AP scan result.
- iot_error_t iot_bsp_wifi_get_mac (struct iot_mac *wifi_mac)

  Get the Wi-Fi MAC.
- iot_wifi_freq_t iot_bsp_wifi_get_freq (void)

  Get the Wi-Fi Frequency band.

5.14.1 Macro Definition Documentation

5.14.1.1 IOT_SOFT_AP_CHANNEL `#define IOT_SOFT_AP_CHANNEL (1)`

5.14.1.2 IOT_WIFI_CMD_TIMEOUT `#define IOT_WIFI_CMD_TIMEOUT 5000`

5.14.1.3 IOT_WIFI_MAX_BSSID_LEN `#define IOT_WIFI_MAX_BSSID_LEN (6)`

5.14.1.4 IOT_WIFI_MAX_PASS_LEN `#define IOT_WIFI_MAX_PASS_LEN (64)`

5.14.1.5 IOT_WIFI_MAX_SCAN_RESULT `#define IOT_WIFI_MAX_SCAN_RESULT (10)`

5.14.1.6 IOT_WIFI_MAX_SSID_LEN `#define IOT_WIFI_MAX_SSID_LEN (32)`

5.14.2 Enumeration Type Documentation

5.14.2.1 iot_wifi_auth_mode_t `enum iot_wifi_auth_mode_t`

Enumerator

IOT_WIFI_AUTH_OPEN	
IOT_WIFI_AUTH_WEP	
IOT_WIFI_AUTH_WPA_PSK	
IOT_WIFI_AUTH_WPA2_PSK	
IOT_WIFI_AUTH_WPA_WPA2_PSK	
IOT_WIFI_AUTH_WPA2_ENTERPRISE	
IOT_WIFI_AUTH_MAX	

5.14.2.2 iot_wifi_freq_t `enum iot_wifi_freq_t`

Enumerator

IOT_WIFI_FREQ_2_4G_ONLY	
IOT_WIFI_FREQ_5G_ONLY	
IOT_WIFI_FREQ_2_4G_5G_BOTH	

5.14.2.3 iot_wifi_mode_t `enum iot_wifi_mode_t`

Enumerator

IOT_WIFI_MODE_OFF	
IOT_WIFI_MODE_SCAN	
IOT_WIFI_MODE_STATION	
IOT_WIFI_MODE_SOFTAP	
IOT_WIFI_MODE_P2P	
IOT_WIFI_MODE_UNDEFINED	

5.14.3 Function Documentation

5.14.3.1 iot_bsp_wifi_get_freq() `iot_wifi_freq_t iot_bsp_wifi_get_freq ( void )`

Get the Wi-Fi Frequency band.

This function get the Wi-Fi Frequency band

Returns

IOT_WIFI_FREQ_2_4G_ONLY : 2.4GHz only supported IOT_WIFI_FREQ_5G_ONLY : 5GHz only supported IOT_WIFI_FREQ_2_4G_5G_BOTH : 2.4GHz and 5GHz both supported

5.14.3.2 iot_bsp_wifi_get_mac()

```
iot_error_t iot_bsp_wifi_get_mac (
    struct iot_mac * wifi_mac )
```

Get the Wi-Fi MAC.

This function get the Wi-Fi MAC

Parameters

out	*iot_mac*	array to save Wi-Fi MAC

Returns

IOT_ERROR_NONE : Success IOT_ERROR_READ_FAIL

Here is the caller graph for this function:

5.14.3.3 iot_bsp_wifi_get_scan_result()

```
uint16_t iot_bsp_wifi_get_scan_result (
    iot_wifi_scan_result_t * scan_result )
```

Get the AP scan result.

This function get the scan result

Parameters

out	*iot_wifi_scan_result_t*	array to save AP list

Returns

number of APs found

Here is the call graph for this function:

5.14.3.4 iot_bsp_wifi_init() `iot_error_t iot_bsp_wifi_init ( )`

Initialize Wi-Fi function.

This function initializes Wi-Fi

Returns

IOT_ERROR_NONE : succeed

Here is the caller graph for this function:

5.14.3.5 iot_bsp_wifi_set_mode() `iot_error_t iot_bsp_wifi_set_mode ( iot_wifi_conf * conf )`

Set the Wi-Fi mode.

This function set the wifi operating mode as scan, station and softap

Parameters

in	*mode*	Wi-Fi operation mode

Returns

IOT_ERROR_NONE : succeed

Here is the caller graph for this function:

5.15 st-device-sdk-c/src/include/iot_capability.h File Reference

`#include "iot_main.h"`
Include dependency graph for iot_capability.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_cap_unit_t

  Contains a "unit" data.
- struct iot_cap_evt_data_t

Contains data for "deviceEvent" payload.

- struct iot_cap_cmd_set

  Contains user command callback function data.
- struct iot_cap_cmd_set_list

  linked list for command callback function data
- struct iot_cap_handle

  Contains data for capability handle.
- struct iot_cap_handle_list

  linked list for capability handle

Typedefs

- typedef struct iot_cap_evt_data_t iot_cap_evt_data_t

  Contains data for "deviceEvent" payload.
- typedef struct iot_cap_cmd_set iot_cap_cmd_set_t

  Contains user command callback function data.
- typedef struct iot_cap_cmd_set_list iot_cap_cmd_set_list_t

  linked list for command callback function data

Enumerations

- enum iot_cap_unit_type { IOT_CAP_UNIT_TYPE_UNUSED, IOT_CAP_UNIT_TYPE_STRING }

5.15.1 Typedef Documentation

5.15.1.1 iot_cap_cmd_set_list_t `typedef struct iot_cap_cmd_set_list iot_cap_cmd_set_list_t`

linked list for command callback function data

5.15.1.2 iot_cap_cmd_set_t `typedef struct iot_cap_cmd_set iot_cap_cmd_set_t`

Contains user command callback function data.

5.15.1.3 iot_cap_evt_data_t `typedef struct iot_cap_evt_data_t iot_cap_evt_data_t`

Contains data for "deviceEvent" payload.

5.15.2 Enumeration Type Documentation

5.15.2.1 iot_cap_unit_type `enum iot_cap_unit_type`

Enumerator

IOT_CAP_UNIT_TYPE_UNUSED	
IOT_CAP_UNIT_TYPE_STRING	

5.16 st-device-sdk-c/src/include/iot_crypto.h File Reference

`#include "sodium.h"`
Include dependency graph for iot_crypto.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_crypto_pk_info

  Contains information of key pair.
- struct iot_crypto_pk_funcs

  Contains public key based function lists.
- struct iot_crypto_pk_context

  Contains key pair information and public key based function lists.
- struct iot_crypto_keypair

  Contains public and private keys for Ed25519.
- struct iot_crypto_ed25519_keypair

  Contains Ed25519 and Curve25519 key pairs.
- struct iot_crypto_ecdh_params

  Contains parameters to share a key between Things and ST apps.
- struct iot_crypto_cipher_info

  Contains cipher informations.

Macros

- #define IOT_ERROR_CRYPTO_BASE64 (IOT_ERROR_CRYPTO_BASE - 1)
- #define IOT_ERROR_CRYPTO_BASE64_URLSAFE (IOT_ERROR_CRYPTO_BASE - 2)
- #define IOT_ERROR_CRYPTO_SHA256 (IOT_ERROR_CRYPTO_BASE - 10)
- #define IOT_ERROR_CRYPTO_PK_SIGN (IOT_ERROR_CRYPTO_BASE - 20)
- #define IOT_ERROR_CRYPTO_PK_VERIFY (IOT_ERROR_CRYPTO_BASE - 21)
- #define IOT_ERROR_CRYPTO_PK_PARSEKEY (IOT_ERROR_CRYPTO_BASE - 22)
- #define IOT_ERROR_CRYPTO_PK_INVALID_ARG (IOT_ERROR_CRYPTO_BASE - 23)
- #define IOT_ERROR_CRYPTO_PK_INVALID_CTX (IOT_ERROR_CRYPTO_BASE - 24)
- #define IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN (IOT_ERROR_CRYPTO_BASE - 25)
- #define IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE (IOT_ERROR_CRYPTO_BASE - 26)
- #define IOT_ERROR_CRYPTO_PK_NULL_FUNC (IOT_ERROR_CRYPTO_BASE - 27)
- #define IOT_ERROR_CRYPTO_PK_ECDH (IOT_ERROR_CRYPTO_BASE - 28)
- #define IOT_ERROR_CRYPTO_ED_KEY_CONVERT (IOT_ERROR_CRYPTO_BASE - 40)
- #define IOT_ERROR_CRYPTO_CIPHER (IOT_ERROR_CRYPTO_BASE - 60)
- #define IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE (IOT_ERROR_CRYPTO_BASE - 61)
- #define IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_MODE (IOT_ERROR_CRYPTO_BASE - 62)
- #define IOT_ERROR_CRYPTO_CIPHER_KEYLEN (IOT_ERROR_CRYPTO_BASE - 63)
- #define IOT_ERROR_CRYPTO_CIPHER_IVLEN (IOT_ERROR_CRYPTO_BASE - 64)
- #define IOT_ERROR_CRYPTO_CIPHER_OUTSIZE (IOT_ERROR_CRYPTO_BASE - 65)
- #define IOT_ERROR_CRYPTO_CIPHER_ALIGN (IOT_ERROR_CRYPTO_BASE - 66)
- #define IOT_ERROR_CRYPTO_SS_KDF (IOT_ERROR_CRYPTO_BASE - 80)
- #define IOT_CRYPTO_PK_TYPE_RSA "RSA"
- #define IOT_CRYPTO_PK_TYPE_ED25519 "ED25519"
- #define IOT_CRYPTO_ED25519_LEN crypto_sign_PUBLICKEYBYTES
- #define IOT_CRYPTO_SECRET_LEN 32
- #define IOT_CRYPTO_IV_LEN 16
- #define IOT_CRYPTO_SHA256_LEN 32
- #define IOT_CRYPTO_SIGNATURE_LEN 256
- #define IOT_CRYPTO_CAL_B64_LEN(x) ((((x) + 2) / 3) ∗ 4 + 1)
- #define IOT_CRYPTO_ALIGN_B64_LEN(x) ((x + 3) & ∼3)

Typedefs

- typedef struct iot_crypto_pk_context iot_crypto_pk_context_t
- typedef struct iot_crypto_pk_info iot_crypto_pk_info_t

  Contains information of key pair.
- typedef struct iot_crypto_pk_funcs iot_crypto_pk_funcs_t

  Contains public key based function lists.
- typedef struct iot_crypto_ed25519_keypair iot_crypto_ed25519_keypair_t

  Contains Ed25519 and Curve25519 key pairs.
- typedef struct iot_crypto_ecdh_params iot_crypto_ecdh_params_t

  Contains parameters to share a key between Things and ST apps.
- typedef struct iot_crypto_cipher_info iot_crypto_cipher_info_t

  Contains cipher informations.

Enumerations

- enum iot_crypto_pk_type_t { IOT_CRYPTO_PK_RSA, IOT_CRYPTO_PK_ED25519, IOT_CRYPTO_PK_UNKNOWN }

  Types of registered public key.
- enum iot_crypto_cipher_type_t { IOT_CRYPTO_CIPHER_AES256, IOT_CRYPTO_CIPHER_UNKNOWN }

  Types of supported cipher algorithms.
- enum iot_crypto_cipher_mode_t { IOT_CRYPTO_CIPHER_DECRYPT, IOT_CRYPTO_CIPHER_ENCRYPT }

  Types of cipher operations.

Functions

- iot_error_t iot_crypto_base64_encode (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Encode a string as a base64 string.
- iot_error_t iot_crypto_base64_decode (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Decode a base64 string as a string.
- iot_error_t iot_crypto_base64_encode_urlsafe (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Encode a string as a urlsafe base64 string.
- iot_error_t iot_crypto_base64_decode_urlsafe (unsigned char ∗src, size_t src_len, unsigned char ∗dst, size_t dst_len, size_t ∗out_len)

  Decode a urlsafe base64 string as a string.
- iot_error_t iot_crypto_sha256 (unsigned char ∗src, size_t src_len, unsigned char ∗dst)

  Generate a digest by sha256 hash.
- iot_error_t iot_crypto_pk_init (iot_crypto_pk_context_t ∗ctx, iot_crypto_pk_info_t ∗info)

  Initialize a context by passed private key data.
- void iot_crypto_pk_free (iot_crypto_pk_context_t ∗ctx)

  Cleanup the context.
- iot_error_t iot_crypto_pk_sign (iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size_t ∗slen)

  Generate a signature.
- iot_error_t iot_crypto_pk_verify (iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size_t slen)

  Verify the signature.
- iot_error_t iot_crypto_ed25519_init_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Prepare a buffer to store the ed25519 and curve25519 keypair.
- void iot_crypto_ed25519_free_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Cleanup the buffer that allocated for keypair.
- iot_error_t iot_crypto_ed25519_convert_keypair (iot_crypto_ed25519_keypair_t ∗kp)

  Converts an ed25519 keypair to an x25519 keypair.
- iot_error_t iot_crypto_ed25519_convert_pubkey (unsigned char ∗ed25519_key, unsigned char ∗curve25519_key)

  Converts an ed25519 public key to an x25519 public key.
- iot_error_t iot_crypto_ed25519_convert_seckey (unsigned char ∗ed25519_key, unsigned char ∗curve25519_key)

  Converts an ed25519 secret key to an x25519 secret key.
- iot_error_t iot_crypto_ecdh_gen_master_secret (unsigned char ∗master, size_t mlen, iot_crypto_ecdh_params_t ∗params)

  Generate a master secret.

- size_t iot_crypto_cipher_get_align_size (iot_crypto_cipher_type_t type, size_t size)

  Calculate the required align size that contains the encrypted data for the input size.
- iot_error_t iot_crypto_cipher_aes (iot_crypto_cipher_info_t ∗info, unsigned char ∗input, size_t ilen, unsigned char ∗out, size_t ∗olen, size_t osize)

  Generic encryption/decryption function.
- iot_error_t iot_crypto_ss_encrypt (unsigned char ∗input, size_t ilen, unsigned char ∗∗output, size_t ∗olen)

  Encryption of the input data.
- iot_error_t iot_crypto_ss_decrypt (unsigned char ∗input, size_t ilen, unsigned char ∗∗output, size_t ∗olen)

  Decryption of the input data.

5.16.1 Macro Definition Documentation

5.16.1.1 IOT_CRYPTO_ALIGN_B64_LEN

```
#define IOT_CRYPTO_ALIGN_B64_LEN(
    x ) ((x + 3) & ~3)
```

5.16.1.2 IOT_CRYPTO_CAL_B64_LEN

```
#define IOT_CRYPTO_CAL_B64_LEN(
    x ) ((((x) + 2) / 3) * 4 + 1)
```

5.16.1.3 IOT_CRYPTO_ED25519_LEN

```
#define IOT_CRYPTO_ED25519_LEN crypto_sign_PUBLICKEYBYTES
```

5.16.1.4 IOT_CRYPTO_IV_LEN

```
#define IOT_CRYPTO_IV_LEN 16
```

5.16.1.5 IOT_CRYPTO_PK_TYPE_ED25519

```
#define IOT_CRYPTO_PK_TYPE_ED25519 "ED25519"
```

5.16.1.6 IOT_CRYPTO_PK_TYPE_RSA

```
#define IOT_CRYPTO_PK_TYPE_RSA "RSA"
```

5.16.1.7 IOT_CRYPTO_SECRET_LEN

```
#define IOT_CRYPTO_SECRET_LEN 32
```

5.16.1.8 IOT_CRYPTO_SHA256_LEN

```
#define IOT_CRYPTO_SHA256_LEN 32
```

5.16.1.9 IOT_CRYPTO_SIGNATURE_LEN

```
#define IOT_CRYPTO_SIGNATURE_LEN 256
```

5.16.1.10 IOT_ERROR_CRYPTO_BASE64

```
#define IOT_ERROR_CRYPTO_BASE64 (IOT_ERROR_CRYPTO_BASE
- 1)
```

5.16.1.11 IOT_ERROR_CRYPTO_BASE64_URLSAFE

```
#define IOT_ERROR_CRYPTO_BASE64_URLSAFE (IOT_ERROR_CRYPTO_BAS
- 2)
```

5.16.1.12 IOT_ERROR_CRYPTO_CIPHER

```
#define IOT_ERROR_CRYPTO_CIPHER (IOT_ERROR_CRYPTO_BASE
- 60)
```

5.16.1.13 IOT_ERROR_CRYPTO_CIPHER_ALIGN

```
#define IOT_ERROR_CRYPTO_CIPHER_ALIGN (IOT_ERROR_CRYPTO_BASE
- 66)
```

5.16.1.14 IOT_ERROR_CRYPTO_CIPHER_IVLEN

```
#define IOT_ERROR_CRYPTO_CIPHER_IVLEN (IOT_ERROR_CRYPTO_BASE
- 64)
```

5.16.1.15 IOT_ERROR_CRYPTO_CIPHER_KEYLEN

```
#define IOT_ERROR_CRYPTO_CIPHER_KEYLEN (IOT_ERROR_CRYPTO_BASE
- 63)
```

5.16.1.16 IOT_ERROR_CRYPTO_CIPHER_OUTSIZE

```
#define IOT_ERROR_CRYPTO_CIPHER_OUTSIZE (IOT_ERROR_CRYPTO_BASE
- 65)
```

5.16.1.17 IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_MODE

```
#define IOT_ERROR_CRYPTO_CIPHER_UNKN↩
OWN_MODE (IOT_ERROR_CRYPTO_BASE - 62)
```

5.16.1.18 IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE

```
#define IOT_ERROR_CRYPTO_CIPHER_UNKNO↩
WN_TYPE (IOT_ERROR_CRYPTO_BASE - 61)
```

5.16.1.19 IOT_ERROR_CRYPTO_ED_KEY_CONVERT

```
#define IOT_ERROR_CRYPTO_ED_KEY_CONVERT (IOT_ERROR_CRYPTO_BA
- 40)
```

5.16.1.20 IOT_ERROR_CRYPTO_PK_ECDH

```
#define IOT_ERROR_CRYPTO_PK_ECDH (IOT_ERROR_CRYPTO_BASE
- 28)
```

5.16.1.21 IOT_ERROR_CRYPTO_PK_INVALID_ARG

```
#define IOT_ERROR_CRYPTO_PK_INVALID_ARG (IOT_ERROR_CRYPTO_BASE
- 23)
```

5.16.1.22 IOT_ERROR_CRYPTO_PK_INVALID_CTX

```
#define IOT_ERROR_CRYPTO_PK_INVALID_CTX (IOT_ERROR_CRYPTO_BASE
- 24)
```

5.16.1.23 IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN

```
#define IOT_ERROR_CRYPTO_PK_INVALID_KEYL↩
EN (IOT_ERROR_CRYPTO_BASE - 25)
```

5.16.1.24 IOT_ERROR_CRYPTO_PK_NULL_FUNC

```
#define IOT_ERROR_CRYPTO_PK_NULL_FUNC (IOT_ERROR_CRYPTO_BASE
- 27)
```

5.16.1.25 IOT_ERROR_CRYPTO_PK_PARSEKEY

```
#define IOT_ERROR_CRYPTO_PK_PARSEKEY (IOT_ERROR_CRYPTO_BASE
- 22)
```

5.16.1.26 IOT_ERROR_CRYPTO_PK_SIGN

```
#define IOT_ERROR_CRYPTO_PK_SIGN (IOT_ERROR_CRYPTO_BASE
- 20)
```

5.16.1.27 IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE

```
#define IOT_ERROR_CRYPTO_PK_UNKNOWN_K↩
EYTYPE (IOT_ERROR_CRYPTO_BASE - 26)
```

5.16.1.28 IOT_ERROR_CRYPTO_PK_VERIFY `#define IOT_ERROR_CRYPTO_PK_VERIFY (IOT_ERROR_CRYPTO_BASE - 21)`

5.16.1.29 IOT_ERROR_CRYPTO_SHA256 `#define IOT_ERROR_CRYPTO_SHA256 (IOT_ERROR_CRYPTO_BASE - 10)`

5.16.1.30 IOT_ERROR_CRYPTO_SS_KDF `#define IOT_ERROR_CRYPTO_SS_KDF (IOT_ERROR_CRYPTO_BASE - 80)`

5.16.2 Typedef Documentation

5.16.2.1 iot_crypto_cipher_info_t `typedef struct iot_crypto_cipher_info iot_crypto_cipher_info_t`

Contains cipher informations.

5.16.2.2 iot_crypto_ecdh_params_t `typedef struct iot_crypto_ecdh_params iot_crypto_ecdh_params_t`

Contains parameters to share a key between Things and ST apps.

5.16.2.3 iot_crypto_ed25519_keypair_t `typedef struct iot_crypto_ed25519_keypair iot_crypto_ed25519_keypair_t`

Contains Ed25519 and Curve25519 key pairs.

5.16.2.4 iot_crypto_pk_context_t `typedef struct iot_crypto_pk_context iot_crypto_pk_context_t`

5.16.2.5 iot_crypto_pk_funcs_t `typedef struct iot_crypto_pk_funcs iot_crypto_pk_funcs_t`

Contains public key based function lists.

5.16.2.6 iot_crypto_pk_info_t `typedef struct iot_crypto_pk_info iot_crypto_pk_info_t`

Contains information of key pair.

5.16.3 Enumeration Type Documentation

5.16.3.1 iot_crypto_cipher_mode_t `enum iot_crypto_cipher_mode_t`

Types of cipher operations.

Enumerator

IOT_CRYPTO_CIPHER_DECRYPT	
IOT_CRYPTO_CIPHER_ENCRYPT	

5.16.3.2 iot_crypto_cipher_type_t `enum iot_crypto_cipher_type_t`

Types of supported cipher algorithms.

Enumerator

IOT_CRYPTO_CIPHER_AES256	
IOT_CRYPTO_CIPHER_UNKNOWN	

5.16.3.3 iot_crypto_pk_type_t `enum iot_crypto_pk_type_t`

Types of registered public key.

Enumerator

IOT_CRYPTO_PK_RSA	
IOT_CRYPTO_PK_ED25519	
IOT_CRYPTO_PK_UNKNOWN	

5.16.4 Function Documentation

5.16.4.1 iot_crypto_base64_decode()

```
iot_error_t iot_crypto_base64_decode (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Decode a base64 string as a string.

Parameters

in	*src*	a pointer to a buffer to decode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully decoded
IOT_ERROR_CRYPTO_BASE64	failed to decode the string

5.16.4.2 iot_crypto_base64_decode_urlsafe()

```
iot_error_t iot_crypto_base64_decode_urlsafe (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Decode a urlsafe base64 string as a string.

Parameters

in	*src*	a pointer to a buffer to decode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully decoded
IOT_ERROR_CRYPTO_BASE64	failed to decode the string
IOT_ERROR_CRYPTO_BASE64_URLSAFE	failed to encode the string as urlsafe

5.16.4.3 iot_crypto_base64_encode()

```
iot_error_t iot_crypto_base64_encode (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Encode a string as a base64 string.

Parameters

in	*src*	a pointer to a buffer to encode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully encoded
IOT_ERROR_CRYPTO_BASE64	failed to encode the string

5.16.4.4 iot_crypto_base64_encode_urlsafe()

```
iot_error_t iot_crypto_base64_encode_urlsafe (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst,
    size_t dst_len,
    size_t * out_len )
```

Encode a string as a urlsafe base64 string.

This function replaces url unsafe characters ('+', '/') to url safe character ('-', '_')

Parameters

in	*src*	a pointer to a buffer to encode
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store base64 string
in	*dst_len*	the size of buffer pointed by dst in bytes
out	*out_len*	the bytes written to dst

Return values

IOT_ERROR_NONE	the string is sucessfully encoded
IOT_ERROR_CRYPTO_BASE64	failed to encode the string in 3rd lib
IOT_ERROR_CRYPTO_BASE64_URLSAFE	failed to encode the string as urlsafe

Here is the caller graph for this function:

5.16.4.5 iot_crypto_cipher_aes()

```
iot_error_t iot_crypto_cipher_aes (
    iot_crypto_cipher_info_t * info,
    unsigned char * input,
```

```
size_t ilen,
unsigned char * out,
size_t * olen,
size_t osize )
```

Generic encryption/decryption function.

Supported cipher algorithms is AES-256 CBC mode

Parameters

in	*info*	a pointer to a buffer containing cipher context
in	*input*	a pointer to a buffer to encrypt/decrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*out*	a pointer to a buffer to store a signature
out	*olen*	the bytes written to out
in	*osize*	the size of buffer pointed by out in bytes

Return values

IOT_ERROR_NONE	encryption/decryption is a success
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE	not supported cipher is requested.
IOT_ERROR_CRYPTO_CIPHER	failed while encrypting/decrypting

Here is the call graph for this function:

5.16.4.6 iot_crypto_cipher_get_align_size()

```
size_t iot_crypto_cipher_get_align_size (
    iot_crypto_cipher_type_t type,
    size_t size )
```

Calculate the required align size that contains the encrypted data for the input size.

the size of data to encrypt should be aligned when AES is used because AES is operated based on block.

Parameters

in	*type*	cipher type
in	*size*	the size of data to encrypt

Return values

return	the aligned size
return	'0' when error occurs

Here is the caller graph for this function:

5.16.4.7 iot_crypto_ecdh_gen_master_secret()

```
iot_error_t iot_crypto_ecdh_gen_master_secret (
    unsigned char * master,
    size_t mlen,
    iot_crypto_ecdh_params_t * params )
```

Generate a master secret.

This function generates a master secret by combining the shared key and the hashed token. The shared key is generated by ECDH using device's x25519 private key and peer's x25519 public key. The hashed token is shared by peer.

Parameters

out	*master*	a pointer to a buffer to store master secret
in	*mlen*	the size of buffer pointed by master in bytes
in	*params*	a pointer to a buffer containing the private key, public key and hashed token to generate master secret

Return values

IOT_ERROR_NONE	the master secret is sucessfully generated
IOT_ERROR_INVALID_ARGS	params has invalid data
IOT_ERROR_MEM_ALLOC	failed to alloc buffer for shared key
IOT_ERROR_CRYPTO_PK_ECDH	failed in ECDH processing
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.16.4.8 iot_crypto_ed25519_convert_keypair()

```
iot_error_t iot_crypto_ed25519_convert_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Converts an ed25519 keypair to an x25519 keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair buffers ed25519 keypair should be prepared

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.16.4.9 iot_crypto_ed25519_convert_pubkey()

```
iot_error_t iot_crypto_ed25519_convert_pubkey (
    unsigned char * ed25519_key,
    unsigned char * curve25519_key )
```

Converts an ed25519 public key to an x25519 public key.

Parameters

in	*ed25519_key*	a pointer to a public key buffer
out	*curve25519_key*	a pointer to a buffer to store converted x25519 public key

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.16.4.10 iot_crypto_ed25519_convert_seckey()

```
iot_error_t iot_crypto_ed25519_convert_seckey (
    unsigned char * ed25519_key,
    unsigned char * curve25519_key )
```

Converts an ed25519 secret key to an x25519 secret key.

Parameters

in	*ed25519_key*	a pointer to a secret key buffer
out	*curve25519_key*	a pointer to a buffer to store converted x25519 secret key

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_CRYPTO_ED_KEY_CONVERT	failed to convert to x25519

5.16.4.11 iot_crypto_ed25519_free_keypair()

```
void iot_crypto_ed25519_free_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Cleanup the buffer that allocated for keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair to cleanup

5.16.4.12 iot_crypto_ed25519_init_keypair()

```
iot_error_t iot_crypto_ed25519_init_keypair (
    iot_crypto_ed25519_keypair_t * kp )
```

Prepare a buffer to store the ed25519 and curve25519 keypair.

Parameters

in	*kp*	a pointer to a structure of the keypair buffers

Return values

IOT_ERROR_NONE	the buffer is sucessfully allocated
IOT_ERROR_MEM_ALLOC	failed to alloc buffer for keypair

5.16.4.13 iot_crypto_pk_free()

```
void iot_crypto_pk_free (
    iot_crypto_pk_context_t * ctx )
```

Cleanup the context.

Parameters

in	*ctx*	a pointer to a buffer to cleanup

5.16.4.14 iot_crypto_pk_init()

```
iot_error_t iot_crypto_pk_init (
    iot_crypto_pk_context_t * ctx,
    iot_crypto_pk_info_t * info )
```

Initialize a context by passed private key data.

Parameters

in	*ctx*	a pointer to a buffer to handle crypto context
in	*info*	a pointer to a buffer containing private key data

Return values

IOT_ERROR_NONE	context is sucessfully initialized
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE	private key data has not supported algorithm

5.16.4.15 iot_crypto_pk_sign()

```
iot_error_t iot_crypto_pk_sign (
    iot_crypto_pk_context_t * ctx,
    unsigned char * input,
    size_t ilen,
    unsigned char * sig,
    size_t * slen )
```

Generate a signature.

Parameters

in	*ctx*	a pointer to a buffer containing crypto context
in	*input*	a pointer to a buffer to generate a signature
in	*ilen*	the size of buffer pointed by input in bytes
out	*sig*	a pointer to a buffer to store a signature
out	*slen*	the bytes written to sig

Return values

IOT_ERROR_NONE	the signature is sucessfully generated
IOT_ERROR_CRYPTO_PK_PARSEKEY	failed to parse device certificate
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN	a length of private key is wrong
IOT_ERROR_CRYPTO_PK_SIGN	failed to generate signature

5.16.4.16 iot_crypto_pk_verify()

```
iot_error_t iot_crypto_pk_verify (
    iot_crypto_pk_context_t * ctx,
    unsigned char * input,
    size_t ilen,
    unsigned char * sig,
    size_t slen )
```

Verify the signature.

Parameters

in	*ctx*	a pointer to a buffer containing crypto context

Parameters

in	*input*	a pointer to a buffer to generate a signature
in	*ilen*	the size of buffer pointed by input in bytes
in	*sig*	a pointer to a buffer containing the signature
in	*slen*	the size of buffer pointed by sig in bytes

Return values

IOT_ERROR_NONE	the signature is sucessfully verified
IOT_ERROR_CRYPTO_PK_NULL_FUNC	verify function is not implemented
IOT_ERROR_CRYPTO_PK_PARSEKEY	failed to parse device certificate
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN	a length of private key is wrong
IOT_ERROR_CRYPTO_PK_VERIFY	the signature is mismatched

5.16.4.17 iot_crypto_sha256()

```
iot_error_t iot_crypto_sha256 (
    unsigned char * src,
    size_t src_len,
    unsigned char * dst )
```

Generate a digest by sha256 hash.

Parameters

in	*src*	a pointer to a buffer to generate a digest
in	*src_len*	the size of buffer pointed by src in bytes
out	*dst*	a pointer to a buffer to store a digest

Return values

IOT_ERROR_NONE	a digest is sucessfully generated
IOT_ERROR_CRYPTO_SHA256	failed to generate a digest

Here is the caller graph for this function:

5.16.4.18 iot_crypto_ss_decrypt()

```
iot_error_t iot_crypto_ss_decrypt (
    unsigned char * input,
    size_t ilen,
    unsigned char ** output,
    size_t * olen )
```

Decryption of the input data.

The decryption key is generated from device unique value

Parameters

in	*input*	a pointer to a buffer to decrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*output*	a pointer of pointer to a buffer to store the decrypted data
out	*olen*	the bytes written to output buffer

Return values

IOT_ERROR_NONE	decryption is success
IOT_ERROR_MEM_ALLOC	mem alloc failed for output buffer
IOT_ERROR_CRYPTO_CIPHER_ALIGN	failed to get align size of the input size for output buffer
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE	cipher is requested with not supported algorithm
IOT_ERROR_CRYPTO_CIPHER	cipher operation is failed
IOT_ERROR_CRYPTO_SS_KDF	failed during derivate the key

5.16.4.19 iot_crypto_ss_encrypt()

```
iot_error_t iot_crypto_ss_encrypt (
    unsigned char * input,
    size_t ilen,
    unsigned char ** output,
    size_t * olen )
```

Encryption of the input data.

The encryption key is generated from device unique value

Parameters

in	*input*	a pointer to a buffer to encrypt
in	*ilen*	the size of buffer pointed by input in bytes
out	*output*	a pointer of pointer to a buffer to store the encrypted data
out	*olen*	the bytes written to output buffer

Return values

IOT_ERROR_NONE	encryption is success
IOT_ERROR_MEM_ALLOC	mem alloc failed for output buffer
IOT_ERROR_CRYPTO_CIPHER_ALIGN	failed to get align size of the input size for output buffer
IOT_ERROR_CRYPTO_CIPHER	cipher operation is failed
IOT_ERROR_CRYPTO_SS_KDF	failed during derivate the key

Here is the caller graph for this function:

5.17 st-device-sdk-c/src/include/iot_crypto_internal.h File Reference

This graph shows which files directly or indirectly include this file:

5.18 st-device-sdk-c/src/include/iot_debug.h File Reference

This graph shows which files directly or indirectly include this file:

iot_debug_level_t

internal debug level.

- #define IOT_DEBUG_PREFIX "[IoT]"
- #define COLOR_CYAN "\033[0;36m"
- #define COLOR_END "\033[0;m"
- #define IOT_ERROR(fmt, args...)

  Error level logging macro.
- #define IOT_WARN(fmt, args...)

  Warning level logging macro.
- #define IOT_INFO(fmt, args...)

  Info level logging macro.
- #define IOT_DEBUG(fmt, args...)

  Debug level logging macro.
- #define HIT()
- #define ENTER()
- #define LEAVE()
- #define IOT_MEM_CHECK(fmt, args...)

  Memory(heap) checking macro.
- #define IOT_ERROR_CHECK(condition, ret, fmt, args...)

  Condition checking macro.
- #define IOT_WARN_CHECK(condition, ret, fmt, args...)
- #define IOT_DEBUG_CHECK(condition, ret, fmt, args...)
- enum iot_debug_level_t { IOT_DEBUG_LEVEL_NONE = 0, IOT_DEBUG_LEVEL_ERROR, IOT_DEBUG_LEVEL_WARN, IOT_DEBUG_LEVEL_INFO, IOT_DEBUG_LEVEL_DEBUG, IOT_DEBUG_LEVEL_MAX }
- void iot_bsp_debug (iot_debug_level_t level, const char *tag, const char *fmt,...)

  Write message into the log.
- void iot_bsp_debug_check_heap (const char *tag, const char *func, const int line, const char *fmt,...)

  Check memory(heap) status.

5.18.1 Macro Definition Documentation

5.18.1.1 COLOR_CYAN `#define COLOR_CYAN "\033[0;36m"`

5.18.1.2 COLOR_END

```
#define COLOR_END "\033[0;m"
```

5.18.1.3 ENTER

```
#define ENTER( )
```

5.18.1.4 HIT

```
#define HIT( )
```

5.18.1.5 IOT_DEBUG

```
#define IOT_DEBUG(
            fmt,
            args...  )
```

Debug level logging macro.

Macro to use log function

5.18.1.6 IOT_DEBUG_CHECK

```
#define IOT_DEBUG_CHECK(
            condition,
            ret,
            fmt,
            args...  )
```

Value:

```
do { \
        if ((condition)) { \
            IOT_DEBUG(fmt, ##args); \
            return (ret); \
        } \
} while (0)
```

5.18.1.7 IOT_DEBUG_PREFIX

```
#define IOT_DEBUG_PREFIX "[IoT]"
```

5.18.1.8 IOT_ERROR

```
#define IOT_ERROR(
            fmt,
            args...  )
```

Error level logging macro.

Macro to use log function

5.18.1.9 IOT_ERROR_CHECK

```
#define IOT_ERROR_CHECK(
            condition,
            ret,
            fmt,
            args...  )
```

Value:

```
do { \
        if ((condition)) { \
            IOT_ERROR(fmt, ##args); \
            return (ret); \
        } \
} while (0)
```

Condition checking macro.

Macro to check condition

5.18.1.10 IOT_INFO

```
#define IOT_INFO(
            fmt,
            args...  )
```

Info level logging macro.

Macro to use log function

5.18.1.11 IOT_MEM_CHECK

```
#define IOT_MEM_CHECK(
            fmt,
            args...  )
```

Memory(heap) checking macro.

Macro to check memory(heap)

5.18.1.12 IOT_WARN

```
#define IOT_WARN(
            fmt,
            args...  )
```

Warning level logging macro.

Macro to use log function

5.18.1.13 IOT_WARN_CHECK

```
#define IOT_WARN_CHECK(
            condition,
            ret,
            fmt,
            args...  )
```

Value:

```
do { \
        if ((condition)) { \
            IOT_WARN(fmt, ##args); \
            return (ret); \
        } \
} while (0)
```

5.18.1.14 LEAVE

```
#define LEAVE( )
```

5.18.2 Enumeration Type Documentation

5.18.2.1 iot_debug_level_t

```
enum iot_debug_level_t
```

Enumerator

IOT_DEBUG_LEVEL_NONE	
IOT_DEBUG_LEVEL_ERROR	
IOT_DEBUG_LEVEL_WARN	
IOT_DEBUG_LEVEL_INFO	
IOT_DEBUG_LEVEL_DEBUG	
IOT_DEBUG_LEVEL_MAX	

5.18.3 Function Documentation

5.18.3.1 iot_bsp_debug()

```
void iot_bsp_debug (
    iot_debug_level_t level,
    const char * tag,
    const char * fmt,
    ... )
```

Write message into the log.

This function is not intended to be used directly. Instead, use one of IOT_ERROR, IOT_WARN, IOT_INFO, IOT_↩ DEBUG macros.

Parameters

in	*level*	log level • IOT_DEBUG_LEVEL_NONE • IOT_DEBUG_LEVEL_ERROR • IOT_DEBUG_LEVEL_WARN • IOT_DEBUG_LEVEL_INFO • IOT_DEBUG_LEVEL_DEBUG
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.18.3.2 iot_bsp_debug_check_heap()

```
void iot_bsp_debug_check_heap (
    const char * tag,
    const char * func,
    const int line,
    const char * fmt,
    ... )
```

Check memory(heap) status.

This function check memory(heap) status

Parameters

in	*tag*	tag
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.19 st-device-sdk-c/src/include/iot_easysetup.h File Reference

```
#include "iot_error.h"
#include "iot_main.h"
```

Include dependency graph for iot_easysetup.h:

This graph shows which files directly or indirectly include this file:

Macros

- #define IOT_OVF_TYPE_JUSTWORKS (1 << OVF_BIT_JUSTWORKS)
- #define IOT_OVF_TYPE_BUTTON (1 << OVF_BIT_BUTTON)
- #define IOT_OVF_TYPE_PIN (1 << OVF_BIT_PIN)
- #define IOT_OVF_TYPE_QR (1 << OVF_BIT_QR)
- #define IOT_ES_URI_POST_KEYINFO "keyinfo"
- #define IOT_ES_URI_POST_CONFIRMINFO "confirminfo"

- #define IOT_ES_URI_POST_CONFIRM "confirm"
- #define IOT_ES_URI_POST_WIFIPROVISIONINGINFO "wifiprovisioninginfo"
- #define IOT_ES_URI_POST_SETUPCOMPLETE "setupcomplete"
- #define IOT_ES_URI_POST_LOGS "logs"
- #define IOT_ES_URI_GET_DEVICEINFO "/deviceinfo"
- #define IOT_ES_URI_GET_WIFISCANINFO "/wifiscaninfo"
- #define IOT_ES_URI_GET_POST_RESPONSE "/post_response"
- #define IOT_ES_URI_GET_LOGS_SYSTEMINFO "/logs/systeminfo"
- #define IOT_ES_URI_GET_LOGS_DUMP "/logs/dump"
- #define IOT_ERROR_EASYSETUP_INVALID_REQUEST (IOT_ERROR_EASYSETUP_BASE - 1)
- #define IOT_ERROR_EASYSETUP_INVALID_SEQUENCE (IOT_ERROR_EASYSETUP_BASE - 2)
- #define IOT_ERROR_EASYSETUP_NOT_SUPPORTED (IOT_ERROR_EASYSETUP_BASE - 3)
- #define IOT_ERROR_EASYSETUP_INVALID_PIN (IOT_ERROR_EASYSETUP_BASE - 4)
- #define IOT_ERROR_EASYSETUP_INVALID_QR (IOT_ERROR_EASYSETUP_BASE - 5)
- #define IOT_ERROR_EASYSETUP_INVALID_BROKER_URL (IOT_ERROR_EASYSETUP_BASE - 6)
- #define IOT_ERROR_EASYSETUP_INVALID_PAYLOAD (IOT_ERROR_EASYSETUP_BASE - 7)
- #define IOT_ERROR_EASYSETUP_INTERNAL_SERVER_ERROR (IOT_ERROR_EASYSETUP_BASE - 50)
- #define IOT_ERROR_EASYSETUP_WIFI_SCANLIST_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 51)
- #define IOT_ERROR_EASYSETUP_SERIAL_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 52)
- #define IOT_ERROR_EASYSETUP_CONFIRM_TIMEOUT (IOT_ERROR_EASYSETUP_BASE - 53)
- #define IOT_ERROR_EASYSETUP_CONFIRM_DENIED (IOT_ERROR_EASYSETUP_BASE - 54)
- #define IOT_ERROR_EASYSETUP_SHARED_KEY_CREATION_FAIL (IOT_ERROR_EASYSETUP_BASE - 55)
- #define IOT_ERROR_EASYSETUP_RPK_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 56)
- #define IOT_ERROR_EASYSETUP_CERTIFICATE_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 57)
- #define IOT_ERROR_EASYSETUP_PIN_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 58)
- #define IOT_ERROR_EASYSETUP_SPUB_DECODE_ERROR (IOT_ERROR_EASYSETUP_BASE - 59)
- #define IOT_ERROR_EASYSETUP_RAND_DECODE_ERROR (IOT_ERROR_EASYSETUP_BASE - 60)
- #define IOT_ERROR_EASYSETUP_INTERNAL_WARNING (IOT_ERROR_EASYSETUP_BASE - 97)
- #define IOT_ERROR_EASYSETUP_INTERNAL_CRITICAL_ERROR (IOT_ERROR_EASYSETUP_BASE - 98)
- #define IOT_ERROR_EASYSETUP_400_BASE IOT_ERROR_EASYSETUP_INVALID_REQUEST
- #define IOT_ERROR_EASYSETUP_500_BASE IOT_ERROR_EASYSETUP_INTERNAL_SERVER_ERROR
- #define IOT_ERROR_EASYSETUP_MAX (IOT_ERROR_EASYSETUP_BASE - 99)

Enumerations

- enum ownership_validation_feature {
  OVF_BIT_JUSTWORKS = 0, OVF_BIT_QR, OVF_BIT_BUTTON, OVF_BIT_PIN,
  OVF_BIT_MAX_FEATURE }

Functions

- iot_error_t iot_easysetup_request_handler (struct iot_context *ctx, struct iot_easysetup_payload request)

  easysetup cgi request handler
- iot_error_t iot_easysetup_create_ssid (struct iot_devconf_prov_data *devconf, char *ssid, size_t ssid_len)

  Create E4 type SSID.
- iot_error_t iot_easysetup_init (struct iot_context *ctx)

  Start eayssetup device-to-device sequence.
- void iot_easysetup_deinit (struct iot_context *ctx)

  Stop eayssetup device-to-device sequence.

5.19.1 Macro Definition Documentation

5.19.1.1 IOT_ERROR_EASYSETUP_400_BASE

```
#define IOT_ERROR_EASYSETUP_400_BASE IOT_ERROR_EASYSETUP_INVALID_R
```

5.19.1.2 IOT_ERROR_EASYSETUP_500_BASE

```
#define IOT_ERROR_EASYSETUP_500_BASE IOT_ERROR_EASYSETUP_INTERNAL_
```

5.19.1.3 IOT_ERROR_EASYSETUP_CERTIFICATE_NOT_FOUND

```
#define IOT_ERROR_EASYSETUP_CERTI↩
FICATE_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 57)
```

5.19.1.4 IOT_ERROR_EASYSETUP_CONFIRM_DENIED

```
#define IOT_ERROR_EASYSETUP_CONFIRM_DENI↩
ED (IOT_ERROR_EASYSETUP_BASE - 54)
```

5.19.1.5 IOT_ERROR_EASYSETUP_CONFIRM_TIMEOUT

```
#define IOT_ERROR_EASYSETUP_CONFIRM_TIME↩
OUT (IOT_ERROR_EASYSETUP_BASE - 53)
```

5.19.1.6 IOT_ERROR_EASYSETUP_INTERNAL_CRITICAL_ERROR

```
#define IOT_ERROR_EASYSETUP_INT↩
ERNAL_CRITICAL_ERROR (IOT_ERROR_EASYSETUP_BASE - 98)
```

5.19.1.7 IOT_ERROR_EASYSETUP_INTERNAL_SERVER_ERROR

```
#define IOT_ERROR_EASYSETUP_INTE↩
RNAL_SERVER_ERROR (IOT_ERROR_EASYSETUP_BASE - 50)
```

5.19.1.8 IOT_ERROR_EASYSETUP_INTERNAL_WARNING

```
#define IOT_ERROR_EASYSETUP_INTERNAL_W↩
ARNING (IOT_ERROR_EASYSETUP_BASE - 97)
```

5.19.1.9 IOT_ERROR_EASYSETUP_INVALID_BROKER_URL

```
#define IOT_ERROR_EASYSETUP_INVALID_B↩
ROKER_URL (IOT_ERROR_EASYSETUP_BASE - 6)
```

5.19.1.10 IOT_ERROR_EASYSETUP_INVALID_PAYLOAD

```
#define IOT_ERROR_EASYSETUP_INVALID_PAYLOAD (IOT_ERROR_EASYSETUP_BASE - 7)
```

5.19.1.11 IOT_ERROR_EASYSETUP_INVALID_PIN

```
#define IOT_ERROR_EASYSETUP_INVALID_PIN (IOT_ERROR_EASYSETUP_BA
- 4)
```

5.19.1.12 IOT_ERROR_EASYSETUP_INVALID_QR

```
#define IOT_ERROR_EASYSETUP_INVALID_QR (IOT_ERROR_EASYSETUP_BAS
- 5)
```

5.19.1.13 IOT_ERROR_EASYSETUP_INVALID_REQUEST

```
#define IOT_ERROR_EASYSETUP_INVALID_REQUEST (IOT_ERROR_EASYSETUP_BASE - 1)
```

5.19.1.14 IOT_ERROR_EASYSETUP_INVALID_SEQUENCE

```
#define IOT_ERROR_EASYSETUP_INVALID_SEQUENCE (IOT_ERROR_EASYSETUP_BASE - 2)
```

5.19.1.15 IOT_ERROR_EASYSETUP_MAX

```
#define IOT_ERROR_EASYSETUP_MAX (IOT_ERROR_EASYSETUP_BASE
- 99)
```

5.19.1.16 IOT_ERROR_EASYSETUP_NOT_SUPPORTED

```
#define IOT_ERROR_EASYSETUP_NOT_SUPPORTED (IOT_ERROR_EASYSETUP_BASE - 3)
```

5.19.1.17 IOT_ERROR_EASYSETUP_PIN_NOT_FOUND

```
#define IOT_ERROR_EASYSETUP_PIN_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 58)
```

5.19.1.18 IOT_ERROR_EASYSETUP_RAND_DECODE_ERROR

```
#define IOT_ERROR_EASYSETUP_RAND_DECODE_ERROR (IOT_ERROR_EASYSETUP_BASE - 60)
```

5.19.1.19 IOT_ERROR_EASYSETUP_RPK_NOT_FOUND

```
#define IOT_ERROR_EASYSETUP_RPK_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 56)
```

5.19.1.20 IOT_ERROR_EASYSETUP_SERIAL_NOT_FOUND

```
#define IOT_ERROR_EASYSETUP_SERIAL_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 52)
```

5.19.1.21 IOT_ERROR_EASYSETUP_SHARED_KEY_CREATION_FAIL

```
#define IOT_ERROR_EASYSETUP_SHARED_KEY_CREATION_FAIL (IOT_ERROR_EASYSETUP_BASE - 55)
```

5.19.1.22 IOT_ERROR_EASYSETUP_SPUB_DECODE_ERROR

```
#define IOT_ERROR_EASYSETUP_SPUB_DECODE_ERROR (IOT_ERROR_EASYSETUP_BASE - 59)
```

5.19.1.23 IOT_ERROR_EASYSETUP_WIFI_SCANLIST_NOT_FOUND

```
#define IOT_ERROR_EASYSETUP_WIFI_SCANLIST_NOT_FOUND (IOT_ERROR_EASYSETUP_BASE - 51)
```

5.19.1.24 IOT_ES_URI_GET_DEVICEINFO

```
#define IOT_ES_URI_GET_DEVICEINFO "/deviceinfo"
```

5.19.1.25 IOT_ES_URI_GET_LOGS_DUMP

```
#define IOT_ES_URI_GET_LOGS_DUMP "/logs/dump"
```

5.19.1.26 IOT_ES_URI_GET_LOGS_SYSTEMINFO

```
#define IOT_ES_URI_GET_LOGS_SYSTEMINFO "/logs/systeminfo"
```

5.19.1.27 IOT_ES_URI_GET_POST_RESPONSE

```
#define IOT_ES_URI_GET_POST_RESPONSE "/post_response"
```

5.19.1.28 IOT_ES_URI_GET_WIFISCANINFO

```
#define IOT_ES_URI_GET_WIFISCANINFO "/wifiscaninfo"
```

5.19.1.29 IOT_ES_URI_POST_CONFIRM

```
#define IOT_ES_URI_POST_CONFIRM "confirm"
```

5.19.1.30 IOT_ES_URI_POST_CONFIRMINFO

```
#define IOT_ES_URI_POST_CONFIRMINFO "confirminfo"
```

5.19.1.31 IOT_ES_URI_POST_KEYINFO

```
#define IOT_ES_URI_POST_KEYINFO "keyinfo"
```

5.19.1.32 IOT_ES_URI_POST_LOGS

```
#define IOT_ES_URI_POST_LOGS "logs"
```

5.19.1.33 IOT_ES_URI_POST_SETUPCOMPLETE

```
#define IOT_ES_URI_POST_SETUPCOMPLETE "setupcomplete"
```

5.19.1.34 IOT_ES_URI_POST_WIFIPROVISIONINGINFO

```
#define IOT_ES_URI_POST_WIFIPROVISIONINGI↩
NFO "wifiprovisioninginfo"
```

5.19.1.35 IOT_OVF_TYPE_BUTTON

```
#define IOT_OVF_TYPE_BUTTON (1 << OVF_BIT_BUTTON)
```

5.19.1.36 IOT_OVF_TYPE_JUSTWORKS

```
#define IOT_OVF_TYPE_JUSTWORKS (1 << OVF_BIT_JUSTWORKS)
```

5.19.1.37 IOT_OVF_TYPE_PIN

```
#define IOT_OVF_TYPE_PIN (1 << OVF_BIT_PIN)
```

5.19.1.38 IOT_OVF_TYPE_QR

```
#define IOT_OVF_TYPE_QR (1 << OVF_BIT_QR)
```

5.19.2 Enumeration Type Documentation

5.19.2.1 ownership_validation_feature

```
enum ownership_validation_feature
```

Enumerator

OVF_BIT_JUSTWORKS	
OVF_BIT_QR	
OVF_BIT_BUTTON	
OVF_BIT_PIN	
OVF_BIT_MAX_FEATURE	

5.19.3 Function Documentation

5.19.3.1 iot_easysetup_create_ssid()

```
iot_error_t iot_easysetup_create_ssid (
    struct iot_devconf_prov_data * devconf,
    char * ssid,
    size_t ssid_len )
```

Create E4 type SSID.

This function create E4 type soft-ap SSID for this device

Parameters

in	*devconf*	things static information
out	*ssid*	created ssid
in	*ssid_len*	ssid buffer length including null termination

Returns

iot_state_t

Return values

IOT_ERROR_NONE	success

Here is the call graph for this function:

5.19.3.2 iot_easysetup_deinit()

```
void iot_easysetup_deinit (
    struct iot_context * ctx )
```

Stop eayssetup device-to-device sequence.

This function stops httpd working

Parameters

in	*ctx*	iot_context handle

Returns

void

5.19.3.3 iot_easysetup_init()

```
iot_error_t iot_easysetup_init (
    struct iot_context * ctx )
```

Start eayssetup device-to-device sequence.

This function makes wifi mode as soft-ap and starts httpd

Parameters

in	*ctx*	iot_context handle

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_UNINITIALIZED	error

5.19.3.4 iot_easysetup_request_handler()

```
iot_error_t iot_easysetup_request_handler (
    struct iot_context * ctx,
    struct iot_easysetup_payload request )
```

easysetup cgi request handler

This function runs from iot-task by executing actual cgi payload manipulation.
result will be transferred to httpd task (tiT) as easysetup response queue parameter.

Parameters

in	*ctx*	iot_context handle
in	*request*	easysetup payload as input

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success

5.20 st-device-sdk-c/src/include/iot_error.h File Reference

This graph shows which files directly or indirectly include this file:

iot_error_t

internal error codes.

- enum iot_error_t {
  IOT_ERROR_NONE = 0, IOT_ERROR_BAD_REQ = -1, IOT_ERROR_INVALID_ARGS = -2, IOT_ERROR_WRITE_FAIL = -3,
  IOT_ERROR_READ_FAIL = -4, IOT_ERROR_UNINITIALIZED = -5, IOT_ERROR_MEM_ALLOC = -6, IOT_ERROR_TIMEOUT = -7,
  IOT_ERROR_REG_UPDATED = -10, IOT_ERROR_NOT_IMPLEMENTED = -11, IOT_ERROR_INIT_FAIL = -12, IOT_ERROR_DEINIT_FAIL = -13,
  IOT_ERROR_NV_DATA_ERROR = -14, IOT_ERROR_NV_DATA_NOT_EXIST = -15, IOT_ERROR_FS_OPEN_FAIL = -16, IOT_ERROR_FS_READ_FAIL = -17,
  IOT_ERROR_FS_WRITE_FAIL = -18, IOT_ERROR_FS_REMOVE_FAIL = -19, IOT_ERROR_FS_CLOSE_FAIL = -20, IOT_ERROR_FS_NO_FILE = -21,
  IOT_ERROR_FS_ENCRYPT_INIT = -22, IOT_ERROR_FS_ENCRYPT_FAIL = -23, IOT_ERROR_FS_DECRYPT_FAIL = -24, IOT_ERROR_UUID_FAIL = -25,
  IOT_ERROR_MQTT_NETCONN_FAIL = -200, IOT_ERROR_MQTT_CONNECT_FAIL = -201, IOT_ERROR_MQTT_SERVER = -202, IOT_ERROR_MQTT_PUBLISH_FAIL = -203,
  IOT_ERROR_MQTT_REJECT_CONNECT = -204, IOT_ERROR_CRYPTO_BASE = -1000, IOT_ERROR_JWT_BASE = -2000, IOT_ERROR_PROV_FAIL = -2998,
  IOT_ERROR_CONNECT_FAIL = -2999, IOT_ERROR_EASYSETUP_BASE = -3000 }
- typedef enum iot_error_t iot_error_t

5.20.1 Typedef Documentation

5.20.1.1 iot_error_t `typedef enum iot_error_t iot_error_t`

5.20.2 Enumeration Type Documentation

5.20.2.1 iot_error_t `enum iot_error_t`

Enumerator

IOT_ERROR_NONE	
IOT_ERROR_BAD_REQ	
IOT_ERROR_INVALID_ARGS	
IOT_ERROR_WRITE_FAIL	
IOT_ERROR_READ_FAIL	
IOT_ERROR_UNINITIALIZED	
IOT_ERROR_MEM_ALLOC	
IOT_ERROR_TIMEOUT	
IOT_ERROR_REG_UPDATED	
IOT_ERROR_NOT_IMPLEMENTED	
IOT_ERROR_INIT_FAIL	
IOT_ERROR_DEINIT_FAIL	
IOT_ERROR_NV_DATA_ERROR	
IOT_ERROR_NV_DATA_NOT_EXIST	
IOT_ERROR_FS_OPEN_FAIL	
IOT_ERROR_FS_READ_FAIL	
IOT_ERROR_FS_WRITE_FAIL	
IOT_ERROR_FS_REMOVE_FAIL	
IOT_ERROR_FS_CLOSE_FAIL	
IOT_ERROR_FS_NO_FILE	
IOT_ERROR_FS_ENCRYPT_INIT	
IOT_ERROR_FS_ENCRYPT_FAIL	
IOT_ERROR_FS_DECRYPT_FAIL	
IOT_ERROR_UUID_FAIL	
IOT_ERROR_MQTT_NETCONN_FAIL	
IOT_ERROR_MQTT_CONNECT_FAIL	
IOT_ERROR_MQTT_SERVER_UNAVAIL	
IOT_ERROR_MQTT_PUBLISH_FAIL	
IOT_ERROR_MQTT_REJECT_CONNECT	
IOT_ERROR_CRYPTO_BASE	
IOT_ERROR_JWT_BASE	
IOT_ERROR_PROV_FAIL	
IOT_ERROR_CONNECT_FAIL	
IOT_ERROR_EASYSETUP_BASE	

5.21 st-device-sdk-c/src/include/iot_internal.h File Reference

```
#include "iot_capability.h"
#include "iot_crypto.h"
```

Include dependency graph for iot_internal.h:

This graph shows which files directly or indirectly include this file:

Macros

- #define IOT_TASK_NAME "iot-task"
- #define IOT_TASK_STACK_SIZE (1024∗5)
- #define IOT_TASK_PRIORITY (4)
- #define IOT_QUEUE_LENGTH (10)
- #define IOT_PUB_QUEUE_LENGTH (10)
- #define IOT_TOPIC_SIZE (100)
- #define IOT_PAYLOAD_SIZE (1024)
- #define IOT_PUB_TOPIC_REGISTRATION "/v1/registrations"
- #define IOT_SUB_TOPIC_REGISTRATION "/v1/registrations/notification/%s"
- #define IOT_PUB_TOPIC_EVENT "/v1/deviceEvents/%s"
- #define IOT_SUB_TOPIC_COMMAND "/v1/commands/%s"
- #define IOT_SUB_TOPIC_NOTIFICATION "/v1/notifications/%s"
- #define IOT_DEFAULT_TIMEOUT 12000 /∗ milli-seconds ∗/
- #define IOT_MQTT_KEEPALIVE_INTERVAL 120 /∗ seconds ∗/

Functions

- iot_error_t iot_command_send (struct iot_context *ctx, enum iot_command_type cmd_type, const void *param, int param_size)

  send command to iot main task
- iot_error_t iot_state_update (struct iot_context *ctx, iot_state_t new_state, int need_interact)

  update iot state
- iot_error_t iot_easysetup_request (struct iot_context *ctx, enum iot_easysetup_step step, const void *payload)

  send easysetup cgi payload manipulation request
- iot_error_t iot_api_onboarding_config_load (unsigned char *onboarding_config, unsigned int onboarding_↩ config_len, struct iot_devconf_prov_data *devconf)

  load "onboarding_config.json" from application source directory
- iot_error_t iot_api_device_info_load (unsigned char *device_info, unsigned int device_info_len, struct iot_device_info *info)

  load "device_info.json" from application source directory
- void iot_api_onboarding_config_mem_free (struct iot_devconf_prov_data *devconf)

  free onboarding config memory
- void iot_api_device_info_mem_free (struct iot_device_info *info)

  free device info memory
- void iot_api_prov_data_mem_free (struct iot_device_prov_data *prov)

  free prov data memory
- iot_error_t iot_api_read_device_identity (unsigned char *device_nv_info, unsigned int device_nv_info_len, const char *object, char **nv_data)

  Extract required data from "device_info.json" which is located in application source directory.
- iot_error_t iot_device_cleanup (struct iot_context *ctx)

  device cleanup
- iot_error_t iot_es_connect (struct iot_context *ctx, int conn_type)

  easy setup connect
- iot_error_t iot_es_disconnect (struct iot_context *ctx, int conn_type)

  easy setup disconnect
- void iot_es_crypto_init_pk (iot_crypto_pk_info_t *pk_info, iot_crypto_pk_type_t type)

  initialize the buffer for pubkey information
- iot_error_t iot_es_crypto_load_pk (iot_crypto_pk_info_t *pk_info)

  easy setup crypto to load pubkey
- void iot_es_crypto_free_pk (iot_crypto_pk_info_t *pk_info)

  easy setup crypto to free pubkey
- iot_error_t iot_mqtt_connect (struct iot_mqtt_ctx *target_cli, char *username, char *sign_data)

  mqtt connect
- iot_error_t iot_mqtt_publish (struct iot_context *ctx, void *payload)

  pyblish mqtt message
- iot_error_t iot_mqtt_subscribe (struct iot_mqtt_ctx *mqtt_ctx)

  subscribe mqtt topic
- iot_error_t iot_mqtt_unsubscribe (struct iot_mqtt_ctx *mqtt_ctx)

  unsubscribe mqtt topic
- void iot_mqtt_disconnect (struct iot_mqtt_ctx *target_cli)

  mqtt disconnect
- iot_error_t iot_mqtt_registration (struct iot_mqtt_ctx *mqtt_ctx)

  register device to ST server
- void iot_cap_sub_cb (iot_cap_handle_list_t *cap_handle_list, char *payload)

  callback for mqtt command msg
- void iot_noti_sub_cb (struct iot_context *ctx, char *payload)

callback for mqtt noti msg

- void iot_cap_call_init_cb (iot_cap_handle_list_t *cap_handle_list)

  call init callback

- iot_error_t iot_get_time_in_sec (char *buf, size_t buf_len)

  get time data by sec

- iot_error_t iot_get_time_in_ms (char *buf, size_t buf_len)

  get time data in msec

5.21.1 Macro Definition Documentation

5.21.1.1 IOT_DEFAULT_TIMEOUT

```
#define IOT_DEFAULT_TIMEOUT 12000 /* milli-seconds */
```

5.21.1.2 IOT_MQTT_KEEPALIVE_INTERVAL

```
#define IOT_MQTT_KEEPALIVE_INTERVAL 120 /* seconds */
```

5.21.1.3 IOT_PAYLOAD_SIZE

```
#define IOT_PAYLOAD_SIZE (1024)
```

5.21.1.4 IOT_PUB_QUEUE_LENGTH

```
#define IOT_PUB_QUEUE_LENGTH (10)
```

5.21.1.5 IOT_PUB_TOPIC_EVENT

```
#define IOT_PUB_TOPIC_EVENT "/v1/deviceEvents/%s"
```

5.21.1.6 IOT_PUB_TOPIC_REGISTRATION

```
#define IOT_PUB_TOPIC_REGISTRATION "/v1/registrations"
```

5.21.1.7 IOT_QUEUE_LENGTH

```
#define IOT_QUEUE_LENGTH (10)
```

5.21.1.8 IOT_SUB_TOPIC_COMMAND

```
#define IOT_SUB_TOPIC_COMMAND "/v1/commands/%s"
```

5.21.1.9 IOT_SUB_TOPIC_NOTIFICATION `#define IOT_SUB_TOPIC_NOTIFICATION "/v1/notifications/%s"`

5.21.1.10 IOT_SUB_TOPIC_REGISTRATION `#define IOT_SUB_TOPIC_REGISTRATION "/v1/registrations/notification/%s`

5.21.1.11 IOT_TASK_NAME `#define IOT_TASK_NAME "iot-task"`

5.21.1.12 IOT_TASK_PRIORITY `#define IOT_TASK_PRIORITY (4)`

5.21.1.13 IOT_TASK_STACK_SIZE `#define IOT_TASK_STACK_SIZE (1024*5)`

5.21.1.14 IOT_TOPIC_SIZE `#define IOT_TOPIC_SIZE (100)`

5.21.2 Function Documentation

5.21.2.1 iot_api_device_info_load()

```
iot_error_t iot_api_device_info_load (
    unsigned char * device_info,
    unsigned int device_info_len,
    struct iot_device_info * info )
```

load "device_info.json" from application source directory

"device_info.json" should be updated by application developer
This function parses downloaded "device_info.json" to be used for EasySetup
Only firmwareVersion will be parsed by this api. others are handled by another api

Parameters

in	*device_info*	start pointer of json data
in	*device_info_len*	json data length
out	*info*	"device_info.json" will be parsed and mapped to this internal structure

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.

Return values

IOT_ERROR_MEM_ALLOC	memory allocation failure.

example

```
{
    "deviceInfo": {
        "firmwareVersion": "FwVer0011A",
    ...
    }
}
```

Here is the caller graph for this function:

5.21.2.2 iot_api_device_info_mem_free()

```
void iot_api_device_info_mem_free (
    struct iot_device_info * info )
```

free device info memory

this function frees the loaded device's information

Parameters

in	*info*	loaded device's information

5.21.2.3 iot_api_onboarding_config_load()

```
iot_error_t iot_api_onboarding_config_load (
    unsigned char * onboarding_config,
    unsigned int onboarding_config_len,
    struct iot_devconf_prov_data * devconf )
```

load "onboarding_config.json" from application source directory

"onboarding_config.json" can be downloaded from SmartThings Developer Workspace
This function parses downloaded "onboarding_config.json" to be used for EasySetup

Parameters

in	*onboarding_config*	start pointer of json data
in	*onboarding_config_len*	json data length
out	*devconf*	"onboarding_config.json" will be parsed and mapped to this internal structure

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.
IOT_ERROR_MEM_ALLOC	memory allocation failure.
IOT_ERROR_CRYPTO_SHA256	sha256 error.
IOT_ERROR_CRYPTO_BASE64	base64 error.
IOT_ERROR_CRYPTO_BASE64_URLSAFE	base64 urlsafe error.

example

```
{
   "onboardingConfig": {
       "deviceOnboardingID": "NAME", // max. 13 character. this will be prefix of soft-ap ssid.
       "mnId": "MNID", // mnId for developer and/or manufacturer. "MNID" shouldn't be used.
       "setupId": "999", // 3-digit Device onboarding ID for this device.
       "vid": "VID", // VID(Vendor ID) for this profile.
       "deviceTypeId": "TYPE", // Device type which is selected from Developer Workspace.
       "ownershipValidationType": [ "JUSTWORKS", "BUTTON", "PIN", "QR" ],
           // "JUSTWORKS" for confirming without user interaction.
           // "BUTTON" for confirming by pressing builtin button.
           // "PIN" for confirming by matching 8-digit number PIN
           // "QR" for confirming by scanning a QR code by SmartThings app.
       "identityType": "ED25519 or CERTIFICATE" // ED25519 or X.509 CERTIFICATE
    }
}
```

Here is the caller graph for this function:

5.21.2.4 iot_api_onboarding_config_mem_free()

```
void iot_api_onboarding_config_mem_free (
    struct iot_devconf_prov_data * devconf )
```

free onboarding config memory

this function frees the loaded onboarding configuration

Parameters

in	*devconf*	loaded onboarding configuration

5.21.2.5 iot_api_prov_data_mem_free()

```
void iot_api_prov_data_mem_free (
```

```
struct iot_device_prov_data * prov )
```

free prov data memory

this function frees the loaded provisioning data

Parameters

in	*prov*	loaded provisioning data

Here is the caller graph for this function:

5.21.2.6 iot_api_read_device_identity()

```
iot_error_t iot_api_read_device_identity (
    unsigned char * device_nv_info,
    unsigned int device_nv_info_len,
    const char * object,
    char ** nv_data )
```

Extract required data from "device_info.json" which is located in application source directory.

"device_info.json" should be updated by application developer
This function parses downloaded "device_info.json" to be used for EasySetup

Parameters

in	*device_nv_info*	starting pointer of json data
in	*device_nv_info_len*	json data length
in	*object*	object name for searching json data.
out	*nv_data*	"device_info.json" will be parsed by "object" and mapped to this pointer

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.
IOT_ERROR_MEM_ALLOC	memory allocation failure.

example

```
{
 "nvProfile": {
    "privateKey": "privateKey", // Client (= Device) Private key
    "publicKey": "publicKey", // Client (= Device) Public key
    "serialNumber": "serialNumber" // Device Serial Number
 }
}
```

Here is the caller graph for this function:

5.21.2.7 iot_cap_call_init_cb()

```
void iot_cap_call_init_cb (
    iot_cap_handle_list_t * cap_handle_list )
```

call init callback

this function is used to call all allocated capability callbacks when target is connected

Parameters

in	*cap_handle_list*	allocated capability handle list

5.21.2.8 iot_cap_sub_cb()

```
void iot_cap_sub_cb (
    iot_cap_handle_list_t * cap_handle_list,
    char * payload )
```

callback for mqtt command msg

this function is used to handle command message from server

Parameters

in	*cap_handle_list*	allocated capability handle list
in	*payload*	received raw message from server

Here is the caller graph for this function:

5.21.2.9 iot_command_send()

```
iot_error_t iot_command_send (
    struct iot_context * ctx,
    enum iot_command_type cmd_type,
    const void * param,
    int param_size )
```

send command to iot main task

this function sends specific command to iot-task via queue

Parameters

in	*ctx*	iot-core context
in	*cmd_type*	actual specific command type
in	*param*	additional parameter data for each command
in	*param_size*	additional parameter size

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_MEM_ALLOC	memory allocation failed
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

Here is the caller graph for this function:

5.21.2.10 iot_device_cleanup()

```
iot_error_t iot_device_cleanup (
    struct iot_context * ctx )
```

device cleanup

this function triggers clean-up process. All registered data will be removed

Parameters

in	*ctx*	iot-core context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.11 iot_easysetup_request()

```
iot_error_t iot_easysetup_request (
    struct iot_context * ctx,
    enum iot_easysetup_step step,
    const void * payload )
```

send easysetup cgi payload manipulation request

easysetup cgi payload manipulation should be done at iot-task. This function sends payload to iot-task via queue

Parameters

in	*ctx*	iot-core context
in	*step*	indicates which uri(command) is dealing with
in	*payload*	payload data - mostly json data

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

5.21.2.12 iot_es_connect()

```
iot_error_t iot_es_connect (
    struct iot_context * ctx,
    int conn_type )
```

easy setup connect

this function tries to connect server for registration or communication process

Parameters

in	*ctx*	iot-core context
in	*conn_type*	set connection type. registration or communication with server

Return values

IOT_ERROR_NONE	success.

5.21.2.13 iot_es_crypto_free_pk()

```
void iot_es_crypto_free_pk (
    iot_crypto_pk_info_t * pk_info )
```

easy setup crypto to free pubkey

this function frees the loaded pubkey information

Parameters

in	*pk_info*	loaded pubkey information pointer

5.21.2.14 iot_es_crypto_init_pk()

```
void iot_es_crypto_init_pk (
    iot_crypto_pk_info_t * pk_info,
    iot_crypto_pk_type_t type )
```

initialize the buffer for pubkey information

this function uses to initialize

Parameters

in	*pk_info*	pubkey information structure to initialize
in	*type*	pubkey type, RSA or ED25519

5.21.2.15 iot_es_crypto_load_pk()

```
iot_error_t iot_es_crypto_load_pk (
    iot_crypto_pk_info_t * pk_info )
```

easy setup crypto to load pubkey

this function loads pubkey information

Parameters

in	*pk_info*	pubkey information pointer for loading

Return values

IOT_ERROR_NONE	success.

5.21.2.16 iot_es_disconnect()

```
iot_error_t iot_es_disconnect (
    struct iot_context * ctx,
    int conn_type )
```

easy setup disconnect

this function tries to disconnect server for registration or communication process

Parameters

in	*ctx*	iot-core context
in	*conn_type*	set connection type. registration or communication with server

Return values

IOT_ERROR_NONE	success.

Here is the caller graph for this function:

5.21.2.17 iot_get_time_in_ms()

```
iot_error_t iot_get_time_in_ms (
    char * buf,
    size_t buf_len )
```

get time data in msec

this function tries to get time value in millisecond by string

Parameters

in	*buf*	buffer point to contain millisecond based string value
in	*buf_len*	size of allocated buffer for string

Return values

IOT_ERROR_NONE	success.

5.21.2.18 iot_get_time_in_sec()

```
iot_error_t iot_get_time_in_sec (
    char * buf,
    size_t buf_len )
```

get time data by sec

this function tries to get time value in second by string

Parameters

in	*buf*	buffer point to contain second based string value
in	*buf_len*	size of allocated buffer for string

Return values

IOT_ERROR_NONE	success.

5.21.2.19 iot_mqtt_connect()

```
iot_error_t iot_mqtt_connect (
    struct iot_mqtt_ctx * target_cli,
    char * username,
    char * sign_data )
```

mqtt connect

this function connects ST server by mqtt

Parameters

in	*target_cli*	mqtt handling context
in	*username*	username to connect ST server based on mqtt protocol
in	*sign_data*	specific password that was jwt token-based to connect ST server

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.20 iot_mqtt_disconnect()

```
void iot_mqtt_disconnect (
    struct iot_mqtt_ctx * target_cli )
```

mqtt disconnect

this function is used to disconnect from server

Parameters

in	*target_cli*	mqtt handling context

Here is the call graph for this function:

5.21.2.21 iot_mqtt_publish()

```
iot_error_t iot_mqtt_publish (
    struct iot_context * ctx,
    void * payload )
```

pyblish mqtt message

this function is used to publish command & notification message

Parameters

in	*ctx*	iot-core context
in	*payload*	raw message sending to server

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.22 iot_mqtt_registration()

```
iot_error_t iot_mqtt_registration (
    struct iot_mqtt_ctx * mqtt_ctx )
```

register device to ST server

this function is used to handle the registration process

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.23 iot_mqtt_subscribe()

```
iot_error_t iot_mqtt_subscribe (
    struct iot_mqtt_ctx * mqtt_ctx )
```

subscribe mqtt topic

this function is used to subscribe command & notification topics

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.24 iot_mqtt_unsubscribe()

```
iot_error_t iot_mqtt_unsubscribe (
    struct iot_mqtt_ctx * mqtt_ctx )
```

unsubscribe mqtt topic

this function is used to unsubscribe command & notification topics

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.21.2.25 iot_noti_sub_cb()

```
void iot_noti_sub_cb (
    struct iot_context * ctx,
    char * payload )
```

callback for mqtt noti msg

this function is used to handle notification message from server

Parameters

in	*ctx*	iot-core context
in	*payload*	received raw message from server

Here is the caller graph for this function:

5.21.2.26 iot_state_update()

```
iot_error_t iot_state_update (
    struct iot_context * ctx,
    iot_state_t new_state,
    int need_interact )
```

update iot state

this function tries to update iot-state using iot_command_send internally

Parameters

in	*ctx*	iot-core context
in	*new_state*	new iot-state to update
in	*need_interact*	additional parameter data for each command

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_MEM_ALLOC	memory allocation failed
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

5.22 st-device-sdk-c/src/include/iot_jwt.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define IOT_ERROR_JWT_INVALID_ARG (IOT_ERROR_JWT_BASE - 0)
- #define IOT_ERROR_JWT_MALLOC (IOT_ERROR_JWT_BASE - 1)
- #define IOT_ERROR_JWT_CJSON (IOT_ERROR_JWT_BASE - 2)

Functions

- iot_error_t iot_jwt_create (char **token, const char *sn, iot_crypto_pk_info_t *pk_info)

  Create a JWT as proof of the device's identity.

5.22.1 Macro Definition Documentation

5.22.1.1 IOT_ERROR_JWT_CJSON

```
#define IOT_ERROR_JWT_CJSON (IOT_ERROR_JWT_BASE - 2)
```

5.22.1.2 IOT_ERROR_JWT_INVALID_ARG

```
#define IOT_ERROR_JWT_INVALID_ARG (IOT_ERROR_JWT_BASE - 0)
```

5.22.1.3 IOT_ERROR_JWT_MALLOC

```
#define IOT_ERROR_JWT_MALLOC (IOT_ERROR_JWT_BASE - 1)
```

5.22.2 Function Documentation

5.22.2.1 iot_jwt_create()

```
iot_error_t iot_jwt_create (
    char ** token,
    const char * sn,
    iot_crypto_pk_info_t * pk_info )
```

Create a JWT as proof of the device's identity.

This function makes a JWT string to connect to ST Cloud. Pass the JWT as password. Supported key types are RS256 and ED25519.

Parameters

out	*token*	a pointer of buffer to store a formatted and signed string
in	*sn*	device serial number as user name
in	*pk_info*	private key data

Return values

IOT_ERROR_NONE	JWT is sucessfully generated
IOT_ERROR_JWT_MALLOC	no more available heap memory
IOT_ERROR_JWT_CJSON	failed to make json

5.23 st-device-sdk-c/src/include/iot_main.h File Reference

```
#include "st_dev.h"
#include "iot_mqtt_client.h"
#include "iot_error.h"
#include "iot_bsp_wifi.h"
#include "iot_os_util.h"
#include "iot_crypto.h"
#include "iot_os_net.h"
```

Include dependency graph for iot_main.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_uuid

  Contains "uuid" data.
- struct iot_wifi_prov_data

  Contains "wifi provisioning" data.
- struct iot_cloud_prov_data

  Contains "cloud provisioning" data.

- struct iot_devconf_prov_data

  Contains "device configuration" data.
- struct iot_device_prov_data

  Contains "all device's provisioning" data.
- struct iot_command

  Contains "internal command" data.
- struct iot_easysetup_payload

  Contains "easy-setup payload" data.
- struct iot_registered_data

  Contains "registration message" data.
- struct iot_mqtt_ctx

  Contains "mqtt handling context" data.
- struct iot_device_info

  Contains "device's information" data.
- struct iot_state_data

  Contains "iot core's main state" data.
- struct iot_context

  Contains "iot core's main context" data.

Macros

- #define IOT_WIFI_PROV_SSID_LEN (31 + 1)
- #define IOT_WIFI_PROV_PASSWORD_LEN (63 + 1)
- #define IOT_BUF_TX_SIZE 3500
- #define IOT_BUF_RX_SIZE 512
- #define IOT_EVENT_BIT_COMMAND (1 << 0)
- #define IOT_EVENT_BIT_CAPABILITY (1 << 1)
- #define IOT_EVENT_BIT_EASYSETUP_REQ (1 << 2)
- #define IOT_EVENT_BIT_EASYSETUP_RESP (1 << 3)
- #define IOT_EVENT_BIT_EASYSETUP_CONFIRM (1 << 4)
- #define IOT_EVENT_BIT_ALL (IOT_EVENT_BIT_COMMAND | IOT_EVENT_BIT_CAPABILITY | IOT_EVENT_BIT_EASYSETUP_REQ)
- #define IOT_REG_UUID_STR_LEN (36)

Typedefs

- typedef enum iot_state_type iot_state_t
- typedef struct iot_cap_handle_list iot_cap_handle_list_t

Enumerations

- enum _iot_noti_type {
  _IOT_NOTI_TYPE_UNKNOWN = IOT_NOTI_TYPE_UNKNOWN, _IOT_NOTI_TYPE_DEV_DELETED = IOT_NOTI_TYPE_DEV_DELETED, _IOT_NOTI_TYPE_RATE_LIMIT = IOT_NOTI_TYPE_RATE_LIMIT, _IOT_NOTI_TYPE_QUOTA_REACHED = IOT_NOTI_TYPE_QUOTA_REACHED, _IOT_NOTI_TYPE_JWT_EXPIRED }
- enum iot_command_type {
  IOT_COMMAND_READY_TO_CTL, IOT_COMMAND_NETWORK_MODE, IOT_COMMAND_CHECK_PROV_STATUS, IOT_COMMAND_SELF_CLEANUP, IOT_COMMAND_CHECK_CLOUD_STATE, IOT_COMMAND_CLOUD_REGISTERING, IOT_COMMAND_CLOUD_REGISTE IOT_COMMAND_CLOUD_CONNECTING, IOT_COMMAND_NOTIFICATION_RECEIVED, IOT_COMMAND_TYPE_MAX, IOT_CMD_STATE_HANDLE }

- enum iot_easysetup_step { IOT_EASYSETUP_STEP_DEVICEINFO, IOT_EASYSETUP_STEP_KEYINFO, IOT_EASYSETUP_STEP_CONFIRMINFO, IOT_EASYSETUP_STEP_CONFIRM, IOT_EASYSETUP_STEP_WIFISCANINFO, IOT_EASYSETUP_STEP_WIFIPROVIONINGINFO, IOT_EASYSETUP_STEP_S IOT_EASYSETUP_STEP_LOG_SYSTEMINFO, IOT_EASYSETUP_STEP_LOG_CREATE_DUMP, IOT_EASYSETUP_STEP_LOG_GET_DUMP }
- enum iot_connect_type { IOT_CONNECT_TYPE_REGISTRATION, IOT_CONNECT_TYPE_COMMUNICATION }
- enum iot_state_type { IOT_STATE_CHANGE_FAILED = -2, IOT_STATE_UNKNOWN = -1, IOT_STATE_INITIALIZED = 0, IOT_STATE_PROV_ENTER, IOT_STATE_PROV_CONFIRMING, IOT_STATE_PROV_DONE, IOT_STATE_CLOUD_DISCONNECTED, IOT_STATE_CLOUD_REGISTERING, IOT_STATE_CLOUD_CONNECTING, IOT_STATE_CLOUD_CONNECTED }
- enum iot_state_opt { IOT_STATE_OPT_NONE, IOT_STATE_OPT_NEED_INTERACT }

5.23.1 Macro Definition Documentation

5.23.1.1 IOT_BUF_RX_SIZE `#define IOT_BUF_RX_SIZE 512`

5.23.1.2 IOT_BUF_TX_SIZE `#define IOT_BUF_TX_SIZE 3500`

5.23.1.3 IOT_EVENT_BIT_ALL `#define IOT_EVENT_BIT_ALL (IOT_EVENT_BIT_COMMAND | IOT_EVENT_BIT_CAPABILITY | IOT_EVENT_BIT_EASYSETUP_REQ)`

5.23.1.4 IOT_EVENT_BIT_CAPABILITY `#define IOT_EVENT_BIT_CAPABILITY (1 << 1)`

5.23.1.5 IOT_EVENT_BIT_COMMAND `#define IOT_EVENT_BIT_COMMAND (1 << 0)`

5.23.1.6 IOT_EVENT_BIT_EASYSETUP_CONFIRM `#define IOT_EVENT_BIT_EASYSETUP_CONFIRM (1 << 4)`

5.23.1.7 IOT_EVENT_BIT_EASYSETUP_REQ `#define IOT_EVENT_BIT_EASYSETUP_REQ (1 << 2)`

5.23.1.8 IOT_EVENT_BIT_EASYSETUP_RESP `#define IOT_EVENT_BIT_EASYSETUP_RESP (1 << 3)`

5.23.1.9 IOT_REG_UUID_STR_LEN `#define IOT_REG_UUID_STR_LEN (36)`

5.23.1.10 IOT_WIFI_PROV_PASSWORD_LEN `#define IOT_WIFI_PROV_PASSWORD_LEN (63 + 1)`

5.23.1.11 IOT_WIFI_PROV_SSID_LEN `#define IOT_WIFI_PROV_SSID_LEN (31 + 1)`

5.23.2 Typedef Documentation

5.23.2.1 iot_cap_handle_list_t `typedef struct iot_cap_handle_list iot_cap_handle_list_t`

5.23.2.2 iot_state_t `typedef enum iot_state_type iot_state_t`

5.23.3 Enumeration Type Documentation

5.23.3.1 _iot_noti_type `enum _iot_noti_type`

Enumerator

_IOT_NOTI_TYPE_UNKNOWN	
_IOT_NOTI_TYPE_DEV_DELETED	
_IOT_NOTI_TYPE_RATE_LIMIT	
_IOT_NOTI_TYPE_QUOTA_REACHED	
_IOT_NOTI_TYPE_JWT_EXPIRED	

5.23.3.2 iot_command_type `enum iot_command_type`

Enumerator

IOT_COMMAND_READY_TO_CTL	
IOT_COMMAND_NETWORK_MODE	
IOT_COMMAND_CHECK_PROV_STATUS	
IOT_COMMAND_SELF_CLEANUP	
IOT_COMMAND_CHECK_CLOUD_STATE	
IOT_COMMAND_CLOUD_REGISTERING	
IOT_COMMAND_CLOUD_REGISTERED	
IOT_COMMAND_CLOUD_CONNECTING	
IOT_COMMAND_NOTIFICATION_RECEIVED	
IOT_COMMAND_TYPE_MAX	
IOT_CMD_STATE_HANDLE	

5.23.3.3 iot_connect_type `enum iot_connect_type`

Enumerator

IOT_CONNECT_TYPE_REGISTRATION	
IOT_CONNECT_TYPE_COMMUNICATION	

5.23.3.4 iot_easysetup_step `enum iot_easysetup_step`

Enumerator

IOT_EASYSETUP_STEP_DEVICEINFO	
IOT_EASYSETUP_STEP_KEYINFO	
IOT_EASYSETUP_STEP_CONFIRMINFO	
IOT_EASYSETUP_STEP_CONFIRM	
IOT_EASYSETUP_STEP_WIFISCANINFO	
IOT_EASYSETUP_STEP_WIFIPROVIONINGINFO	
IOT_EASYSETUP_STEP_SETUPCOMPLETE	
IOT_EASYSETUP_STEP_LOG_SYSTEMINFO	
IOT_EASYSETUP_STEP_LOG_CREATE_DUMP	
IOT_EASYSETUP_STEP_LOG_GET_DUMP	

5.23.3.5 iot_state_opt `enum iot_state_opt`

Enumerator

IOT_STATE_OPT_NONE	
IOT_STATE_OPT_NEED_INTERACT	

5.23.3.6 iot_state_type `enum iot_state_type`

Enumerator

IOT_STATE_CHANGE_FAILED	
IOT_STATE_UNKNOWN	
IOT_STATE_INITIALIZED	
IOT_STATE_PROV_ENTER	
IOT_STATE_PROV_CONFIRMING	
IOT_STATE_PROV_DONE	
IOT_STATE_CLOUD_DISCONNECTED	
IOT_STATE_CLOUD_REGISTERING	
IOT_STATE_CLOUD_CONNECTING	
IOT_STATE_CLOUD_CONNECTED	

5.24 st-device-sdk-c/src/include/iot_nv_data.h File Reference

```
#include "iot_error.h"
#include "iot_main.h"
```

Include dependency graph for iot_nv_data.h:

This graph shows which files directly or indirectly include this file:

iot_nvd_t

internal nv data codes.

- enum iot_nvd_t {
  IOT_NVD_WIFI_PROV_STATUS = 0, IOT_NVD_AP_SSID, IOT_NVD_AP_PASS, IOT_NVD_AP_BSSID, IOT_NVD_AP_AUTH_TYPE, IOT_NVD_CLOUD_PROV_STATUS, IOT_NVD_SERVER_URL, IOT_NVD_SERVER_PORT, IOT_NVD_LOCATION_ID, IOT_NVD_ROOM_ID, IOT_NVD_LABEL, IOT_NVD_DEVICE_ID, IOT_NVD_PRIVATE_KEY, IOT_NVD_PUBLIC_KEY, IOT_NVD_CA_CERT, IOT_NVD_SUB_CERT, IOT_NVD_SERIAL_NUM, IOT_NVD_MAX }
- iot_error_t iot_nv_init (unsigned char *device_info, unsigned int device_info_len)

  Initialize a nv file-system.
- iot_error_t iot_nv_deinit ()

  Deinitialize a nv file-system.
- iot_error_t iot_nv_get_prov_data (struct iot_device_prov_data *prov_data)

  Get provisioning data from the nv file-system.
- iot_error_t iot_nv_set_prov_data (struct iot_device_prov_data *prov_data)

  Set provisioning data to the nv file-system.
- iot_error_t iot_nv_erase_prov_data ()

  Erase wifi/cloud provisioning data.
- iot_error_t iot_nv_get_wifi_prov_data (struct iot_wifi_prov_data *wifi_prov)

  Get wifi provisioning data from the nv file-system.
- iot_error_t iot_nv_set_wifi_prov_data (struct iot_wifi_prov_data *wifi_prov)

  Set wifi provisioning data to the nv file-system.
- iot_error_t iot_nv_get_cloud_prov_data (struct iot_cloud_prov_data *cloud_prov)

  Get cloud provisioning data from the nv file-system.
- iot_error_t iot_nv_set_cloud_prov_data (struct iot_cloud_prov_data *cloud_prov)

  Set cloud provisioning data to the nv file-system.
- iot_error_t iot_nv_get_private_key (char **key, unsigned int *len)

  Get a private key from the nv file-system.
- iot_error_t iot_nv_get_public_key (char **key, unsigned int *len)

  Get a public key from the nv file-system.
- iot_error_t iot_nv_get_root_certificate (char **cert, unsigned int *len)

  Get a root cert from the nv file-system.
- iot_error_t iot_nv_get_client_certificate (char **cert, unsigned int *len)

  Get a client cert from the nv file-system.
- iot_error_t iot_nv_get_device_id (char **device_id, unsigned int *len)

  Get a device id from the nv file-system.
- iot_error_t iot_nv_set_device_id (const char *device_id)

  Set a device id to the nv file-system.
- iot_error_t iot_nv_get_serial_number (char **sn, unsigned int *len)

  Get a serial number from the nv file-system.
- iot_error_t iot_nv_erase (iot_nvd_t nv_type)

  Erase a nv data.

5.24.1 Enumeration Type Documentation

5.24.1.1 iot_nvd_t `enum iot_nvd_t`

Enumerator

IOT_NVD_WIFI_PROV_STATUS	
IOT_NVD_AP_SSID	
IOT_NVD_AP_PASS	
IOT_NVD_AP_BSSID	
IOT_NVD_AP_AUTH_TYPE	
IOT_NVD_CLOUD_PROV_STATUS	
IOT_NVD_SERVER_URL	
IOT_NVD_SERVER_PORT	
IOT_NVD_LOCATION_ID	
IOT_NVD_ROOM_ID	
IOT_NVD_LABEL	
IOT_NVD_DEVICE_ID	
IOT_NVD_PRIVATE_KEY	
IOT_NVD_PUBLIC_KEY	
IOT_NVD_CA_CERT	
IOT_NVD_SUB_CERT	
IOT_NVD_SERIAL_NUM	
IOT_NVD_MAX	

5.24.2 Function Documentation

5.24.2.1 iot_nv_deinit() `iot_error_t iot_nv_deinit ( )`

Deinitialize a nv file-system.

Return values

IOT_ERROR_NONE	NV file-system deinit successful.
IOT_ERROR_DEINIT_FAIL	NV file-system deinit failed.

See also

iot_bsp_fs_deinit

Here is the call graph for this function:

5.24.2.2 iot_nv_erase()

```
iot_error_t iot_nv_erase (
    iot_nvd_t nv_type )
```

Erase a nv data.

This function erase the nv data completely in File-system.

Parameters

in	*nv_type*	The type of nv data to erase.

Return values

IOT_ERROR_NONE	NV data erase successful.
IOT_ERROR_INVALID_ARGS	Invalid nv type.
IOT_ERROR_NV_DATA_ERROR	NV data erase failed.
IOT_ERROR_NV_DATA_NOT_EXIST	NV data does not exist.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.3 iot_nv_erase_prov_data() `iot_error_t iot_nv_erase_prov_data ( )`

Erase wifi/cloud provisioning data.

This function erases the wifi/cloud provisioning data in File-system.

Return values

IOT_ERROR_NONE	NV data erase successful.
IOT_ERROR_NV_DATA_ERROR	NV data erase failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.4 iot_nv_get_client_certificate() `iot_error_t iot_nv_get_client_certificate ( char ** cert, unsigned int * len )`

Get a client cert from the nv file-system.

Parameters

out	*cert*	A pointer to data array to store the client cert from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.

Return values

IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.24.2.5 iot_nv_get_cloud_prov_data()

```
iot_error_t iot_nv_get_cloud_prov_data (
    struct iot_cloud_prov_data * cloud_prov )
```

Get cloud provisioning data from the nv file-system.

Parameters

out	*cloud_prov*	A pointer to data structure to store the cloud provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.6 iot_nv_get_device_id()

```
iot_error_t iot_nv_get_device_id (
    char ** device_id,
    unsigned int * len )
```

Get a device id from the nv file-system.

Parameters

out	*device_id*	A pointer to data array to store the device id from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.24.2.7 iot_nv_get_private_key()

```
iot_error_t iot_nv_get_private_key (
    char ** key,
    unsigned int * len )
```

Get a private key from the nv file-system.

Parameters

out	*key*	A pointer to data array to store the private key from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.24.2.8 iot_nv_get_prov_data()

```
iot_error_t iot_nv_get_prov_data (
    struct iot_device_prov_data * prov_data )
```

Get provisioning data from the nv file-system.

Parameters

out	*prov_data*	A pointer to data structure to store the provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

See also

iot_nv_get_wifi_prov_data

iot_nv_get_cloud_prov_data

Here is the call graph for this function:

5.24.2.9 iot_nv_get_public_key()

```
iot_error_t iot_nv_get_public_key (
    char ** key,
    unsigned int * len )
```

Get a public key from the nv file-system.

Parameters

out	*key*	A pointer to data array to store the public key from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.24.2.10 iot_nv_get_root_certificate()

```
iot_error_t iot_nv_get_root_certificate (
    char ** cert,
    unsigned int * len )
```

Get a root cert from the nv file-system.

Parameters

out	*cert*	A pointer to data array to store the root cert from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

5.24.2.11 iot_nv_get_serial_number()

```
iot_error_t iot_nv_get_serial_number (
    char ** sn,
    unsigned int * len )
```

Get a serial number from the nv file-system.

Parameters

out	*sn*	A pointer to data array to store the serial number from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.12 iot_nv_get_wifi_prov_data()

```
iot_error_t iot_nv_get_wifi_prov_data (
    struct iot_wifi_prov_data * wifi_prov )
```

Get wifi provisioning data from the nv file-system.

Parameters

out	*wifi_prov*	A pointer to data structure to store the wifi provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.13 iot_nv_init()

```
iot_error_t iot_nv_init (
    unsigned char * device_info,
    unsigned int device_info_len )
```

Initialize a nv file-system.

This function initializes the nv file-system of the device. You must call this function before using nv management function.

Return values

IOT_ERROR_NONE	NV file-system init successful.
IOT_ERROR_INIT_F↩ AIL	NV file-system init failed.

See also

iot_bsp_fs_init

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.14 iot_nv_set_cloud_prov_data()

```
iot_error_t iot_nv_set_cloud_prov_data (
    struct iot_cloud_prov_data * cloud_prov )
```

Set cloud provisioning data to the nv file-system.

Parameters

in	*cloud_prov*	A pointer to cloud provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.24.2.15 iot_nv_set_device_id()

```
iot_error_t iot_nv_set_device_id (
    const char * device_id )
```

Set a device id to the nv file-system.

Parameters

in	*device_id*	The string of device id from MQTT Server.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

5.24.2.16 iot_nv_set_prov_data()

```
iot_error_t iot_nv_set_prov_data (
    struct iot_device_prov_data * prov_data )
```

Set provisioning data to the nv file-system.

Parameters

in	*prov_data*	A pointer to provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

See also

iot_nv_set_wifi_prov_data
iot_nv_set_cloud_prov_data

Here is the call graph for this function:

5.24.2.17 iot_nv_set_wifi_prov_data()

```
iot_error_t iot_nv_set_wifi_prov_data (
    struct iot_wifi_prov_data * wifi_prov )
```

Set wifi provisioning data to the nv file-system.

Parameters

in	*wifi_prov*	A pointer to wifi provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.25 st-device-sdk-c/src/include/iot_util.h File Reference

```
#include "iot_error.h"
#include "iot_main.h"
```

Include dependency graph for iot_util.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct url_parse_t

  Contains a "url parse" data.

Functions

- iot_error_t iot_util_url_parse (char ∗url, url_parse_t ∗output)

  parse url with protocol, domain, port number parts for st-iot-core
- iot_error_t iot_util_convert_str_uuid (const char ∗str, struct iot_uuid ∗uuid)

  uuid type string to iot_uuid struct converting function for st-iot-core
- iot_error_t iot_util_convert_uuid_str (struct iot_uuid ∗uuid, char ∗str, int max_sz)

  iot_uuid struct based value to uuid type string converting function for st-iot-core
- iot_error_t iot_util_get_random_uuid (struct iot_uuid ∗uuid)

  To get random uuid based on iot_uuid struct.
- iot_error_t iot_util_convert_str_mac (char ∗str, struct iot_mac ∗mac)

  To convert WIFI mac string into iot_mac struct value.
- iot_error_t iot_util_convert_mac_str (struct iot_mac ∗mac, char ∗str, int max_sz)

  To convert iot_mac value intto WIFI mac string.
- uint16_t iot_util_convert_channel_freq (uint8_t channel)

  To convert Wi-Fi channel into frequency value.

5.25.1 Function Documentation

5.25.1.1 iot_util_convert_channel_freq()

```
uint16_t iot_util_convert_channel_freq (
    uint8_t channel )
```

To convert Wi-Fi channel into frequency value.

This function tries to convert from the channel to frequency

Parameters

in	*Wi-Fi*	channel

Returns

Wi-Fi frequency

Here is the caller graph for this function:

5.25.1.2 iot_util_convert_mac_str()

```
iot_error_t iot_util_convert_mac_str (
    struct iot_mac * mac,
    char * str,
    int max_sz )
```

To convert iot_mac value intto WIFI mac string.

This function tries to convert from the iot_mac struct value to string

Parameters

in	*mac*	converting wanted iot_mac struct value pointer
in	*str*	allocated memory pointer for converted WIFI mac string
in	*max_sz*	max size of allocated memory pointer

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_INVALID_ARGS	invalid arguments

5.25.1.3 iot_util_convert_str_mac()

```
iot_error_t iot_util_convert_str_mac (
    char * str,
    struct iot_mac * mac )
```

To convert WIFI mac string into iot_mac struct value.

This function tries to convert from the string to iot_mac struct value

Parameters

in	*str*	WIFI mac string pointer such as 'xx:xx:xx:xx:xx:xx'
in	*mac*	allocated iot_mac struct pointer to get iot_mac value from str

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_INVALID_ARGS	invalid arguments

5.25.1.4 iot_util_convert_str_uuid()

```
iot_error_t iot_util_convert_str_uuid (
    const char * str,
    struct iot_uuid * uuid )
```

uuid type string to iot_uuid struct converting function for st-iot-core

This function tries to convert from uuid type string to iot_uuid struct

Parameters

in	*str*	uuid type string pointer such as 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
in	*uuid*	allocated iot_uuid struct pointer to get iot_uuid value from str

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the caller graph for this function:

5.25.1.5 iot_util_convert_uuid_str()

```
iot_error_t iot_util_convert_uuid_str (
    struct iot_uuid * uuid,
    char * str,
    int max_sz )
```

iot_uuid struct based value to uuid type string converting function for st-iot-core

This function tries to convert from iot_uuid struct based value to uuid type string

Parameters

in	*uuid*	converting wanted iot_uuid struct value pointer
in	*str*	allocated memory pointer for converted uuid type string
in	*max_sz*	max size of allocated memory pointer

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the caller graph for this function:

5.25.1.6 iot_util_get_random_uuid()

```
iot_error_t iot_util_get_random_uuid (
    struct iot_uuid * uuid )
```

To get random uuid based on iot_uuid struct.

This function tries to make random iot_uuid values

Parameters

in	*uuid*	allocated iot_uuid struct pointer to get random iot_uuid value

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the call graph for this function:

5.25.1.7 iot_util_url_parse()

```
iot_error_t iot_util_url_parse (
    char * url,
    url_parse_t * output )
```

parse url with protocol, domain, port number parts for st-iot-core

This function parse give url protocol, domain, port number parts

Parameters

in	*url*	null-terminated url string like "https://example.sample.com:1234"
out	*output*	parsed output with url_parse_t type

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

5.26 st-device-sdk-c/src/include/iot_uuid.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- iot_error_t iot_random_uuid_from_mac (struct iot_uuid ∗uuid)

  Change mac to random uuid This function changes mac to random uuid.
- iot_error_t iot_uuid_from_mac (struct iot_uuid ∗uuid)

  Change mac to uuid This function changes mac to the uuid.

5.26.1 Function Documentation

5.26.1.1 iot_random_uuid_from_mac()

```
iot_error_t iot_random_uuid_from_mac (
    struct iot_uuid * uuid )
```

Change mac to random uuid This function changes mac to random uuid.

Parameters

out	*uuid*	created random uuid

Returns

IOT_ERROR_NONE on success

Here is the call graph for this function:

Here is the caller graph for this function:

5.26.1.2 iot_uuid_from_mac()

```
iot_error_t iot_uuid_from_mac (
    struct iot_uuid * uuid )
```

Change mac to uuid This function changes mac to the uuid.

Parameters

out	*uuid*	created uuid

Returns

IOT_ERROR_NONE on success

Here is the call graph for this function:

5.27 st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h File Reference

```
#include <stdbool.h>
#include "iot_mqtt_packet.h"
#include "iot_os_net.h"
#include "iot_os_util.h"
```

Include dependency graph for iot_mqtt_client.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct MQTTMessage
- struct MessageData
- struct MQTTConnackData
- struct MQTTSubackData
- struct MQTTClient
- struct MQTTClient::MessageHandlers

Macros

- #define DLLImport
- #define DLLExport
- #define MAX_PACKET_ID 65535 /∗ according to the MQTT specification - do not change! ∗/
- #define IOT_MQTT_STACK_SIZE 2048
- #define IOT_MQTT_PRIORITY 4
- #define CONFIG_STDK_MQTT_SEND_BUFFER 2048
- #define CONFIG_STDK_MQTT_RECV_BUFFER 2048
- #define CONFIG_STDK_MQTT_SEND_CYCLE 30000
- #define CONFIG_STDK_MQTT_PUBLISH_RETRY 3
- #define CONFIG_STDK_MQTT_PING_RETRY 3
- #define CONFIG_STDK_MQTT_RECV_CYCLE 100
- #define CONFIG_STDK_MQTT_DYNAMIC_BUFFER 1
- #define MAX_MESSAGE_HANDLERS 5 /∗ redefinable - how many subscriptions do you want? ∗/
- #define DefaultClient {0, 0, 0, 0, NULL, NULL, 0, 0, 0}

Typedefs

- typedef struct MQTTMessage MQTTMessage
- typedef struct MQTTClient MQTTClient
- typedef struct MessageData MessageData
- typedef struct MQTTConnackData MQTTConnackData
- typedef struct MQTTSubackData MQTTSubackData
- typedef void(∗ messageHandler) (MessageData ∗, void ∗)

Enumerations

- enum QoS { QOS0, QOS1, QOS2, SUBFAIL = 0x80 }
- enum returnCode { MQTT_DISCONNECTED = -3, MQTT_BUFFER_OVERFLOW = -2, MQTT_FAILURE = -1, MQTT_SUCCESS = 0 }

Functions

- DLLExport bool MQTTClientInit (MQTTClient ∗client, iot_net_socket ∗network, unsigned int command_timeout_ms, unsigned char ∗sendbuf, size_t sendbuf_size, unsigned char ∗readbuf, size_t readbuf_size)
- DLLExport int MQTTConnectWithResults (MQTTClient ∗client, MQTTPacket_connectData ∗options, MQTTConnackData ∗data)
- DLLExport int MQTTConnect (MQTTClient ∗client, MQTTPacket_connectData ∗options)
- DLLExport int MQTTPublish (MQTTClient ∗client, const char ∗, MQTTMessage ∗)
- DLLExport int MQTTSetMessageHandler (MQTTClient ∗c, const char ∗topicFilter, messageHandler messageHandler, void ∗userData)
- DLLExport int MQTTSubscribe (MQTTClient ∗client, const char ∗topicFilter, enum QoS, messageHandler, void ∗userData)
- DLLExport int MQTTSubscribeWithResults (MQTTClient ∗client, const char ∗topicFilter, enum QoS, messageHandler, MQTTSubackData ∗data, void ∗userData)
- DLLExport int MQTTUnsubscribe (MQTTClient ∗client, const char ∗topicFilter)
- DLLExport int MQTTDisconnect (MQTTClient ∗client)
- DLLExport int MQTTYield (MQTTClient ∗client, int time)

5.27.1 Macro Definition Documentation

5.27.1.1 CONFIG_STDK_MQTT_DYNAMIC_BUFFER `#define CONFIG_STDK_MQTT_DYNAMIC_BUFFER 1`

5.27.1.2 CONFIG_STDK_MQTT_PING_RETRY `#define CONFIG_STDK_MQTT_PING_RETRY 3`

5.27.1.3 CONFIG_STDK_MQTT_PUBLISH_RETRY `#define CONFIG_STDK_MQTT_PUBLISH_RETRY 3`

5.27.1.4 CONFIG_STDK_MQTT_RECV_BUFFER `#define CONFIG_STDK_MQTT_RECV_BUFFER 2048`

5.27.1.5 CONFIG_STDK_MQTT_RECV_CYCLE `#define CONFIG_STDK_MQTT_RECV_CYCLE 100`

5.27.1.6 CONFIG_STDK_MQTT_SEND_BUFFER `#define CONFIG_STDK_MQTT_SEND_BUFFER 2048`

5.27.1.7 CONFIG_STDK_MQTT_SEND_CYCLE `#define CONFIG_STDK_MQTT_SEND_CYCLE 30000`

5.27.1.8 DefaultClient `#define DefaultClient {0, 0, 0, 0, NULL, NULL, 0, 0, 0}`

5.27.1.9 DLLExport `#define DLLExport`

5.27.1.10 DLLImport `#define DLLImport`

5.27.1.11 IOT_MQTT_PRIORITY

```
#define IOT_MQTT_PRIORITY 4
```

5.27.1.12 IOT_MQTT_STACK_SIZE

```
#define IOT_MQTT_STACK_SIZE 2048
```

5.27.1.13 MAX_MESSAGE_HANDLERS

```
#define MAX_MESSAGE_HANDLERS 5 /* redefinable - how many subscriptions do you want? */
```

5.27.1.14 MAX_PACKET_ID

```
#define MAX_PACKET_ID 65535 /* according to the MQTT specification - do not change! */
```

5.27.2 Typedef Documentation

5.27.2.1 MessageData

```
typedef struct MessageData MessageData
```

5.27.2.2 messageHandler

```
typedef void(* messageHandler) (MessageData *, void *)
```

5.27.2.3 MQTTClient

```
typedef struct MQTTClient MQTTClient
```

5.27.2.4 MQTTConnackData

```
typedef struct MQTTConnackData MQTTConnackData
```

5.27.2.5 MQTTMessage

```
typedef struct MQTTMessage MQTTMessage
```

5.27.2.6 MQTTSubackData

```
typedef struct MQTTSubackData MQTTSubackData
```

5.27.3 Enumeration Type Documentation

5.27.3.1 QoS

```
enum QoS
```

Enumerator

QOS0	
QOS1	
QOS2	
SUBFAIL	

5.27.3.2 returnCode `enum returnCode`

Enumerator

MQTT_DISCONNECTED	
MQTT_BUFFER_OVERFLOW	
MQTT_FAILURE	
MQTT_SUCCESS	

5.27.4 Function Documentation

5.27.4.1 MQTTClientInit()

```
DLLExport bool MQTTClientInit (
    MQTTClient * client,
    iot_net_socket * network,
    unsigned int command_timeout_ms,
    unsigned char * sendbuf,
    size_t sendbuf_size,
    unsigned char * readbuf,
    size_t readbuf_size )
```

Create an MQTT client object

Parameters

client	
network	
command_timeout_ms	

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.2 MQTTConnect()

```
DLLExport int MQTTConnect (
    MQTTClient * client,
    MQTTPacket_connectData * options )
```

MQTT Connect - send an MQTT connect packet down the network and wait for a Connack The nework object must be connected to the network endpoint before calling this

Parameters

options	- connect options

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.3 MQTTConnectWithResults()

```
DLLExport int MQTTConnectWithResults (
    MQTTClient * client,
    MQTTPacket_connectData * options,
    MQTTConnackData * data )
```

MQTT Connect - send an MQTT connect packet down the network and wait for a Connack The nework object must be connected to the network endpoint before calling this

Parameters

options	- connect options

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.4 MQTTDisconnect()

```
DLLExport int MQTTDisconnect (
    MQTTClient * client )
```

MQTT Disconnect - send an MQTT disconnect packet and close the connection

Parameters

client	- the client object to use

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.5 MQTTPublish()

```
DLLExport int MQTTPublish (
    MQTTClient * client,
    const char * ,
    MQTTMessage *  )
```

MQTT Publish - send an MQTT publish packet and wait for all acks to complete for all QoSs

Parameters

client	- the client object to use
topic	- the topic to publish to
message	- the message to send

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.6 MQTTSetMessageHandler()

```
DLLExport int MQTTSetMessageHandler (
    MQTTClient * c,
    const char * topicFilter,
    messageHandler messageHandler,
    void * userData )
```

MQTT SetMessageHandler - set or remove a per topic message handler

Parameters

client	- the client object to use
topicFilter	- the topic filter set the message handler for
messageHandler	- pointer to the message handler function or NULL to remove

Returns

success code

5.27.4.7 MQTTSubscribe()

```
DLLExport int MQTTSubscribe (
    MQTTClient * client,
    const char * topicFilter,
    enum QoS,
    messageHandler ,
    void * userData )
```

MQTT Subscribe - send an MQTT subscribe packet and wait for suback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to subscribe to
message	- the message to send

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.8 MQTTSubscribeWithResults()

```
DLLExport int MQTTSubscribeWithResults (
    MQTTClient * client,
    const char * topicFilter,
    enum QoS,
    messageHandler ,
    MQTTSubackData * data,
    void * userData )
```

MQTT Subscribe - send an MQTT subscribe packet and wait for suback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to subscribe to
message	- the message to send
data	- suback granted QoS returned

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.9 MQTTUnsubscribe()

```
DLLExport int MQTTUnsubscribe (
    MQTTClient * client,
    const char * topicFilter )
```

MQTT Subscribe - send an MQTT unsubscribe packet and wait for unsuback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to unsubscribe from

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.27.4.10 MQTTYield()

```
DLLExport int MQTTYield (
    MQTTClient * client,
    int time )
```

MQTT Yield - MQTT background

Parameters

client	- the client object to use
time	- the time, in milliseconds, to yield for

Returns

success code

Here is the call graph for this function:

5.28 st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- union MQTTConnectFlags
- struct MQTTPacket_willOptions
- struct MQTTPacket_connectData
- union MQTTConnackFlags

Macros

- #define DLLImport
- #define DLLExport
- #define MQTTPacket_willOptions_initializer { {'M', 'Q', 'T', 'W'}, 0, {NULL, {0, NULL}}, {NULL, {0, NULL}}, 0, 0 }
- #define MQTTPacket_connectData_initializer

Enumerations

- enum connack_return_codes { MQTT_CONNECTION_ACCEPTED = 0, MQTT_UNNACCEPTABLE_PROTOCOL = 1, MQTT_CLIENTID_REJECTED = 2, MQTT_SERVER_UNAVAILABLE = 3, MQTT_BAD_USERNAME_OR_PASSWORD = 4, MQTT_NOT_AUTHORIZED = 5 }

Functions

- DLLExport int MQTTSerialize_connect (unsigned char ∗buf, int buflen, MQTTPacket_connectData ∗options)
- DLLExport int MQTTDeserialize_connect (MQTTPacket_connectData ∗data, unsigned char ∗buf, int len)
- DLLExport int MQTTSerialize_connect_size (MQTTPacket_connectData ∗options)
- DLLExport int MQTTSerialize_connack (unsigned char ∗buf, int buflen, unsigned char connack_rc, unsigned char sessionPresent)
- DLLExport int MQTTDeserialize_connack (unsigned char ∗sessionPresent, unsigned char ∗connack_rc, unsigned char ∗buf, int buflen)
- DLLExport int MQTTSerialize_disconnect (unsigned char ∗buf, int buflen)
- DLLExport int MQTTSerialize_pingreq (unsigned char ∗buf, int buflen)
- DLLExport int MQTTSerialize_disconnect_size ()
- DLLExport int MQTTSerialize_pingreq_size ()

5.28.1 Macro Definition Documentation

5.28.1.1 DLLExport `#define DLLExport`

5.28.1.2 DLLImport `#define DLLImport`

5.28.1.3 MQTTPacket_connectData_initializer `#define MQTTPacket_connectData_initializer`

Value:

```
{ {'M', 'Q', 'T', 'C'}, 0, 4, {NULL, {0, NULL}}, 60, 1, 0, \
        MQTTPacket_willOptions_initializer, {NULL, {0, NULL}}, {NULL, {0, NULL}} }
```

5.28.1.4 MQTTPacket_willOptions_initializer `#define MQTTPacket_willOptions_initializer { {'M', 'Q', 'T', 'W'}, 0, {NULL, {0, NULL}}, {NULL, {0, NULL}}, 0, 0 }`

5.28.2 Enumeration Type Documentation

5.28.2.1 connack_return_codes `enum connack_return_codes`

Enumerator

MQTT_CONNECTION_ACCEPTED	
MQTT_UNNACCEPTABLE_PROTOCOL	
MQTT_CLIENTID_REJECTED	
MQTT_SERVER_UNAVAILABLE	
MQTT_BAD_USERNAME_OR_PASSWORD	
MQTT_NOT_AUTHORIZED	

5.28.3 Function Documentation

5.28.3.1 MQTTDeserialize_connack()

```
DLLExport int MQTTDeserialize_connack (
    unsigned char * sessionPresent,
    unsigned char * connack_rc,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into connack data - return code

Parameters

sessionPresent	the session present flag returned (only for MQTT 3.1.1)
connack_rc	returned integer value of the connack return code
buf	the raw buffer data, of the correct length determined by the remaining length field
len	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.28.3.2 MQTTDeserialize_connect()

```
DLLExport int MQTTDeserialize_connect (
    MQTTPacket_connectData * data,
    unsigned char * buf,
    int len )
```

Deserializes the supplied (wire) buffer into connect data structure

Parameters

data	the connect data structure to be filled out
buf	the raw buffer data, of the correct length determined by the remaining length field
len	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.28.3.3 MQTTSerialize_connack()

```
DLLExport int MQTTSerialize_connack (
    unsigned char * buf,
    int buflen,
    unsigned char connack_rc,
    unsigned char sessionPresent )
```

Serializes the connack packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
connack_rc	the integer connack return code to be used
sessionPresent	the MQTT 3.1.1 sessionPresent flag

Returns

serialized length, or error if 0

5.28.3.4 MQTTSerialize_connect()

```
DLLExport int MQTTSerialize_connect (
    unsigned char * buf,
```

```
int buflen,
MQTTPacket_connectData * options )
```

Serializes the connect options into the buffer.

Parameters

buf	the buffer into which the packet will be serialized
len	the length in bytes of the supplied buffer
options	the options to be used to build the connect packet

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.28.3.5 MQTTSerialize_connect_size()

```
DLLExport int MQTTSerialize_connect_size (
    MQTTPacket_connectData * options )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.28.3.6 MQTTSerialize_disconnect()

```
DLLExport int MQTTSerialize_disconnect (
    unsigned char * buf,
    int buflen )
```

Serializes a disconnect packet into the supplied buffer, ready for writing to a socket

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer, to avoid overruns

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.28.3.7 MQTTSerialize_disconnect_size() `DLLExport int MQTTSerialize_disconnect_size ( )`

Here is the caller graph for this function:

5.28.3.8 MQTTSerialize_pingreq()

```
DLLExport int MQTTSerialize_pingreq (
    unsigned char * buf,
    int buflen )
```

Serializes a disconnect packet into the supplied buffer, ready for writing to a socket

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer, to avoid overruns

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.28.3.9 MQTTSerialize_pingreq_size() `DLLExport int MQTTSerialize_pingreq_size ( )`

Here is the caller graph for this function:

5.29 st-device-sdk-c/src/include/mqtt/iot_mqtt_format.h File Reference

```
#include "iot_mqtt_stacktrace.h"
#include "iot_mqtt_packet.h"
```

Include dependency graph for iot_mqtt_format.h:

This graph shows which files directly or indirectly include this file:

Functions

- const char ∗ MQTTPacket_getName (unsigned short packetid)
- int MQTTStringFormat_connect (char ∗strbuf, int strbuflen, MQTTPacket_connectData ∗data)
- int MQTTStringFormat_connack (char ∗strbuf, int strbuflen, unsigned char connack_rc, unsigned char sessionPresent)
- int MQTTStringFormat_publish (char ∗strbuf, int strbuflen, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, unsigned char ∗payload, int payloadlen)
- int MQTTStringFormat_ack (char ∗strbuf, int strbuflen, unsigned char packettype, unsigned char dup, unsigned short packetid)
- int MQTTStringFormat_subscribe (char ∗strbuf, int strbuflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ], int requestedQoSs[ ])
- int MQTTStringFormat_suback (char ∗strbuf, int strbuflen, unsigned short packetid, int count, int ∗granted↩ QoSs)
- int MQTTStringFormat_unsubscribe (char ∗strbuf, int strbuflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ])
- char ∗ MQTTFormat_toClientString (char ∗strbuf, int strbuflen, unsigned char ∗buf, int buflen)
- char ∗ MQTTFormat_toServerString (char ∗strbuf, int strbuflen, unsigned char ∗buf, int buflen)

5.29.1 Function Documentation

5.29.1.1 MQTTFormat_toClientString()

```
char* MQTTFormat_toClientString (
    char * strbuf,
    int strbuflen,
    unsigned char * buf,
    int buflen )
```

5.29.1.2 MQTTFormat_toServerString()

```
char* MQTTFormat_toServerString (
    char * strbuf,
    int strbuflen,
    unsigned char * buf,
    int buflen )
```

5.29.1.3 MQTTPacket_getName()

```
const char* MQTTPacket_getName (
    unsigned short packetid )
```

5.29.1.4 MQTTStringFormat_ack()

```
int MQTTStringFormat_ack (
    char * strbuf,
    int strbuflen,
    unsigned char packettype,
    unsigned char dup,
    unsigned short packetid )
```

5.29.1.5 MQTTStringFormat_connack()

```
int MQTTStringFormat_connack (
    char * strbuf,
    int strbuflen,
    unsigned char connack_rc,
    unsigned char sessionPresent )
```

5.29.1.6 MQTTStringFormat_connect()

```
int MQTTStringFormat_connect (
    char * strbuf,
    int strbuflen,
    MQTTPacket_connectData * data )
```

5.29.1.7 MQTTStringFormat_publish()

```
int MQTTStringFormat_publish (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    int qos,
    unsigned char retained,
    unsigned short packetid,
    MQTTString topicName,
    unsigned char * payload,
    int payloadlen )
```

5.29.1.8 MQTTStringFormat_suback()

```
int MQTTStringFormat_suback (
    char * strbuf,
    int strbuflen,
    unsigned short packetid,
    int count,
    int * grantedQoSs )
```

5.29.1.9 MQTTStringFormat_subscribe()

```
int MQTTStringFormat_subscribe (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[],
    int requestedQoSs[] )
```

5.29.1.10 MQTTStringFormat_unsubscribe()

```
int MQTTStringFormat_unsubscribe (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[] )
```

5.30 st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h File Reference

```
#include "iot_mqtt_connect.h"
#include "iot_mqtt_publish.h"
#include "iot_mqtt_subscribe.h"
#include "iot_mqtt_unsubscribe.h"
#include "iot_mqtt_format.h"
```

Include dependency graph for iot_mqtt_packet.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- union MQTTHeader
- struct MQTTLenString
- struct MQTTString
- struct MQTTTransport

Macros

- #define DLLImport
- #define DLLExport
- #define MQTTString_initializer {NULL, {0, NULL}}

Enumerations

- enum errors { MQTTPACKET_BUFFER_TOO_SHORT = -2, MQTTPACKET_READ_ERROR = -1, MQTTPACKET_READ_COMPLETE }
- enum msgTypes {
  CONNECT = 1, CONNACK, PUBLISH, PUBACK,
  PUBREC, PUBREL, PUBCOMP, SUBSCRIBE,
  SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ,
  PINGRESP, DISCONNECT }

Functions

- int MQTTstrlen (MQTTString mqttstring)
- DLLExport int MQTTSerialize_ack (unsigned char ∗buf, int buflen, unsigned char type, unsigned char dup, unsigned short packetid)
- DLLExport int MQTTDeserialize_ack (unsigned char ∗packettype, unsigned char ∗dup, unsigned short ∗packetid, unsigned char ∗buf, int buflen)
- DLLExport int MQTTSerialize_ack_size ()
- int MQTTPacket_len (int rem_len)
- DLLExport int MQTTPacket_equals (MQTTString ∗a, char ∗b)
- DLLExport int MQTTPacket_encode (unsigned char ∗buf, int length)
- int MQTTPacket_decode (int(∗getcharfn)(unsigned char ∗, int), int ∗value)
- int MQTTPacket_decodeBuf (unsigned char ∗buf, int ∗value)
- int readInt (unsigned char ∗∗pptr)
- char readChar (unsigned char ∗∗pptr)
- void writeChar (unsigned char ∗∗pptr, char c)
- void writeInt (unsigned char ∗∗pptr, int anInt)
- int readMQTTLenString (MQTTString ∗mqttstring, unsigned char ∗∗pptr, unsigned char ∗enddata)
- void writeCString (unsigned char ∗∗pptr, const char ∗string)
- void writeMQTTString (unsigned char ∗∗pptr, MQTTString mqttstring)
- DLLExport int MQTTPacket_read (unsigned char ∗buf, int buflen, int(∗getfn)(unsigned char ∗, int))
- int MQTTPacket_readnb (unsigned char ∗buf, int buflen, MQTTTransport ∗trp)
- const char ∗ MQTTPacket_msgTypesToString (enum msgTypes)

5.30.1 Macro Definition Documentation

5.30.1.1 DLLExport `#define DLLExport`

5.30.1.2 DLLImport `#define DLLImport`

5.30.1.3 MQTTString_initializer `#define MQTTString_initializer {NULL, {0, NULL}}`

5.30.2 Enumeration Type Documentation

5.30.2.1 errors `enum errors`

Enumerator

MQTTPACKET_BUFFER_TOO_SHORT	
MQTTPACKET_READ_ERROR	
MQTTPACKET_READ_COMPLETE	

5.30.2.2 msgTypes `enum msgTypes`

Enumerator

CONNECT	
CONNACK	
PUBLISH	
PUBACK	
PUBREC	
PUBREL	
PUBCOMP	
SUBSCRIBE	
SUBACK	
UNSUBSCRIBE	
UNSUBACK	
PINGREQ	
PINGRESP	
DISCONNECT	

5.30.3 Function Documentation

5.30.3.1 MQTTDeserialize_ack()

```
DLLExport int MQTTDeserialize_ack (
    unsigned char * packettype,
    unsigned char * dup,
    unsigned short * packetid,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into an ack

Parameters

packettype	returned integer - the MQTT packet type
dup	returned integer - the MQTT dup flag
packetid	returned integer - the MQTT packet identifier
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.30.3.2 MQTTPacket_decode()

```
int MQTTPacket_decode (
    int(*)(unsigned char *, int) getcharfn,
    int * value )
```

Decodes the message length according to the MQTT algorithm

Parameters

getcharfn	pointer to function to read the next character from the data source
value	the decoded length returned

Returns

the number of bytes read from the socket

5.30.3.3 MQTTPacket_decodeBuf()

```
int MQTTPacket_decodeBuf (
    unsigned char * buf,
    int * value )
```

5.30.3.4 MQTTPacket_encode()

```
DLLExport int MQTTPacket_encode (
    unsigned char * buf,
    int length )
```

Encodes the message length according to the MQTT algorithm

Parameters

buf	the buffer into which the encoded data is written
length	the length to be encoded

Returns

the number of bytes written to buffer

Here is the caller graph for this function:

5.30.3.5 MQTTPacket_equals()

```
DLLExport int MQTTPacket_equals (
    MQTTString * a,
    char * bptr )
```

Compares an MQTTString to a C string

Parameters

a	the MQTTString to compare
bptr	the C string to compare

Returns

boolean - equal or not

Here is the caller graph for this function:

5.30.3.6 MQTTPacket_len()

```
int MQTTPacket_len (
    int rem_len )
```

Here is the caller graph for this function:

5.30.3.7 MQTTPacket_msgTypesToString()

```
const char* MQTTPacket_msgTypesToString (
    enum msgTypes )
```

5.30.3.8 MQTTPacket_read()

```
DLLExport int MQTTPacket_read (
    unsigned char * buf,
    int buflen,
    int(*)(unsigned char *, int) getfn )
```

Helper function to read packet data from some source into a buffer

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
getfn	pointer to a function which will read any number of bytes from the needed source

Returns

integer MQTT packet type, or -1 on error

Note

the whole message must fit into the caller's buffer

5.30.3.9 MQTTPacket_readnb()

```
int MQTTPacket_readnb (
    unsigned char * buf,
    int buflen,
    MQTTTransport * trp )
```

Helper function to read packet data from some source into a buffer, non-blocking

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
trp	pointer to a transport structure holding what is needed to solve getting data from it

Returns

integer MQTT packet type, 0 for call again, or -1 on error

Note

the whole message must fit into the caller's buffer

5.30.3.10 MQTTSerialize_ack()

```
DLLExport int MQTTSerialize_ack (
    unsigned char * buf,
    int buflen,
    unsigned char packettype,
    unsigned char dup,
    unsigned short packetid )
```

Serializes the ack packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
type	the MQTT packet type
dup	the MQTT dup flag
packetid	the MQTT packet identifier

Returns

serialized length, or error if 0

5.30.3.11 MQTTSerialize_ack_size()

```
DLLExport int MQTTSerialize_ack_size ( )
```

5.30.3.12 MQTTstrlen()

```
int MQTTstrlen (
    MQTTString mqttstring )
```

Return the length of the MQTTstring - C string if there is one, otherwise the length delimited string

Parameters

mqttstring	the string to return the length of

Returns

the length of the string

5.30.3.13 readChar()

```
char readChar (
    unsigned char ** pptr )
```

Reads one character from the input buffer.

Parameters

pptr	pointer to the input buffer - incremented by the number of bytes used & returned

Returns

the character read

5.30.3.14 readInt()

```
int readInt (
    unsigned char ** pptr )
```

Calculates an integer from two bytes read from the input buffer

Parameters

pptr	pointer to the input buffer - incremented by the number of bytes used & returned

Returns

the integer value calculated

5.30.3.15 readMQTTLenString()

```
int readMQTTLenString (
    MQTTString * mqttstring,
    unsigned char ** pptr,
    unsigned char * enddata )
```

Parameters

mqttstring	the MQTTString structure into which the data is to be read
pptr	pointer to the output buffer - incremented by the number of bytes used & returned
enddata	pointer to the end of the data: do not read beyond

Returns

1 if successful, 0 if not

5.30.3.16 writeChar()

```
void writeChar (
    unsigned char ** pptr,
    char c )
```

Writes one character to an output buffer.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
c	the character to write

Here is the caller graph for this function:

5.30.3.17 writeCString()

```
void writeCString (
    unsigned char ** pptr,
    const char * string )
```

Writes a "UTF" string to an output buffer. Converts C string to length-delimited.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
string	the C string to write

Here is the caller graph for this function:

5.30.3.18 writeInt()

```
void writeInt (
    unsigned char ** pptr,
    int anInt )
```

Writes an integer as 2 bytes to an output buffer.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
anInt	the integer to write

Here is the caller graph for this function:

5.30.3.19 writeMQTTString()

```
void writeMQTTString (
    unsigned char ** pptr,
    MQTTString mqttstring )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.31 st-device-sdk-c/src/include/mqtt/iot_mqtt_publish.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define DLLImport
- #define DLLExport

Functions

- DLLExport int MQTTSerialize_publish (unsigned char ∗buf, int buflen, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, unsigned char ∗payload, int payloadlen)
- DLLExport int MQTTSerialize_publish_size (int qos, MQTTString topicName, int payloadlen)
- DLLExport int MQTTSerialize_publish_header (unsigned char ∗buf, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, int payloadlen)
- DLLExport int MQTTDeserialize_publish (unsigned char ∗dup, int ∗qos, unsigned char ∗retained, unsigned short ∗packetid, MQTTString ∗topicName, unsigned char ∗∗payload, int ∗payloadlen, unsigned char ∗buf, int len)
- DLLExport int MQTTSerialize_puback (unsigned char ∗buf, int buflen, unsigned short packetid)
- DLLExport int MQTTSerialize_pubrel (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid)
- DLLExport int MQTTSerialize_pubcomp (unsigned char ∗buf, int buflen, unsigned short packetid)

5.31.1 Macro Definition Documentation

5.31.1.1 DLLExport `#define DLLExport`

5.31.1.2 DLLImport `#define DLLImport`

5.31.2 Function Documentation

5.31.2.1 MQTTDeserialize_publish()

```
DLLExport int MQTTDeserialize_publish (
    unsigned char * dup,
    int * qos,
    unsigned char * retained,
    unsigned short * packetid,
    MQTTString * topicName,
    unsigned char ** payload,
    int * payloadlen,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into publish data

Parameters

dup	returned integer - the MQTT dup flag
qos	returned integer - the MQTT QoS value
retained	returned integer - the MQTT retained flag
packetid	returned integer - the MQTT packet identifier
topicName	returned MQTTString - the MQTT topic in the publish
payload	returned byte buffer - the MQTT publish payload
payloadlen	returned integer - the length of the MQTT payload
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success

5.31.2.2 MQTTSerialize_puback()

```
DLLExport int MQTTSerialize_puback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes a puback packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.31.2.3 MQTTSerialize_pubcomp()

```
DLLExport int MQTTSerialize_pubcomp (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes a pubrel packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.31.2.4 MQTTSerialize_publish()

```
DLLExport int MQTTSerialize_publish (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    int qos,
```

```
unsigned char retained,
unsigned short packetid,
MQTTString topicName,
unsigned char * payload,
int payloadlen )
```

Serializes the supplied publish data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
dup	integer - the MQTT dup flag
qos	integer - the MQTT QoS value
retained	integer - the MQTT retained flag
packetid	integer - the MQTT packet identifier
topicName	MQTTString - the MQTT topic in the publish
payload	byte buffer - the MQTT publish payload
payloadlen	integer - the length of the MQTT payload

Returns

the length of the serialized data. <= 0 indicates error

5.31.2.5 MQTTSerialize_publish_header()

```
DLLExport int MQTTSerialize_publish_header (
unsigned char * buf,
unsigned char dup,
int qos,
unsigned char retained,
unsigned short packetid,
MQTTString topicName,
int payloadlen )
```

Here is the call graph for this function:

5.31.2.6 MQTTSerialize_publish_size()

```
DLLExport int MQTTSerialize_publish_size (
    int qos,
    MQTTString topicName,
    int payloadlen )
```

Here is the call graph for this function:

5.31.2.7 MQTTSerialize_pubrel()

```
DLLExport int MQTTSerialize_pubrel (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid )
```

Serializes a pubrel packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.32 st-device-sdk-c/src/include/mqtt/iot_mqtt_stacktrace.h File Reference

```
#include <stdio.h>
```

Include dependency graph for iot_mqtt_stacktrace.h:

This graph shows which files directly or indirectly include this file:

Macros

- #define NOSTACKTRACE 1
- #define FUNC_ENTRY
- #define FUNC_ENTRY_NOLOG
- #define FUNC_ENTRY_MED
- #define FUNC_ENTRY_MAX
- #define FUNC_EXIT
- #define FUNC_EXIT_NOLOG
- #define FUNC_EXIT_MED
- #define FUNC_EXIT_MAX
- #define FUNC_EXIT_RC(x)
- #define FUNC_EXIT_MED_RC(x)
- #define FUNC_EXIT_MAX_RC(x)

5.32.1 Macro Definition Documentation

5.32.1.1 FUNC_ENTRY `#define FUNC_ENTRY`

5.32.1.2 FUNC_ENTRY_MAX `#define FUNC_ENTRY_MAX`

5.32.1.3 FUNC_ENTRY_MED `#define FUNC_ENTRY_MED`

5.32.1.4 FUNC_ENTRY_NOLOG `#define FUNC_ENTRY_NOLOG`

5.32.1.5 FUNC_EXIT `#define FUNC_EXIT`

5.32.1.6 FUNC_EXIT_MAX `#define FUNC_EXIT_MAX`

5.32.1.7 FUNC_EXIT_MAX_RC `#define FUNC_EXIT_MAX_RC(`
x)

5.32.1.8 FUNC_EXIT_MED `#define FUNC_EXIT_MED`

5.32.1.9 FUNC_EXIT_MED_RC `#define FUNC_EXIT_MED_RC(`
x)

5.32.1.10 FUNC_EXIT_NOLOG `#define FUNC_EXIT_NOLOG`

5.32.1.11 FUNC_EXIT_RC `#define FUNC_EXIT_RC(`
x)

5.32.1.12 NOSTACKTRACE `#define NOSTACKTRACE 1`

5.33 st-device-sdk-c/src/include/mqtt/iot_mqtt_subscribe.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define DLLImport
- #define DLLExport

Functions

- DLLExport int MQTTSerialize_subscribe (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ], int requestedQoSs[ ])
- int MQTTSerialize_subscribe_size (int count, MQTTString topicFilters[ ])
- DLLExport int MQTTDeserialize_subscribe (unsigned char ∗dup, unsigned short ∗packetid, int maxcount, int ∗count, MQTTString topicFilters[ ], int requestedQoSs[ ], unsigned char ∗buf, int len)
- DLLExport int MQTTSerialize_suback (unsigned char ∗buf, int buflen, unsigned short packetid, int count, int ∗grantedQoSs)
- DLLExport int MQTTDeserialize_suback (unsigned short ∗packetid, int maxcount, int ∗count, int granted↩ QoSs[ ], unsigned char ∗buf, int len)

5.33.1 Macro Definition Documentation

5.33.1.1 DLLExport `#define DLLExport`

5.33.1.2 DLLImport `#define DLLImport`

5.33.2 Function Documentation

5.33.2.1 MQTTDeserialize_suback()

```
DLLExport int MQTTDeserialize_suback (
    unsigned short * packetid,
    int maxcount,
    int * count,
    int grantedQoSs[],
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into suback data

Parameters

packetid	returned integer - the MQTT packet identifier
maxcount	- the maximum number of members allowed in the grantedQoSs array
count	returned integer - number of members in the grantedQoSs array
grantedQoSs	returned array of integers - the granted qualities of service
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.33.2.2 MQTTDeserialize_subscribe()

```
DLLExport int MQTTDeserialize_subscribe (
    unsigned char * dup,
    unsigned short * packetid,
    int maxcount,
    int * count,
    MQTTString topicFilters[],
    int requestedQoSs[],
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into subscribe data

Parameters

dup	integer returned - the MQTT dup flag
packetid	integer returned - the MQTT packet identifier
maxcount	- the maximum number of members allowed in the topicFilters and requestedQoSs arrays
count	- number of members in the topicFilters and requestedQoSs arrays
topicFilters	- array of topic filter names
requestedQoSs	- array of requested QoS
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

the length of the serialized data. <= 0 indicates error

5.33.2.3 MQTTSerialize_suback()

```
DLLExport int MQTTSerialize_suback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid,
```

```
int count,
int * grantedQoSs )
```

Serializes the supplied suback data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier
count	- number of members in the grantedQoSs array
grantedQoSs	- array of granted QoS

Returns

the length of the serialized data. <= 0 indicates error

5.33.2.4 MQTTSerialize_subscribe()

```
DLLExport int MQTTSerialize_subscribe (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[],
    int requestedQoSs[] )
```

Serializes the supplied subscribe data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied bufferr
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier
count	- number of members in the topicFilters and reqQos arrays
topicFilters	- array of topic filter names
requestedQoSs	- array of requested QoS

Returns

the length of the serialized data. <= 0 indicates error

Here is the caller graph for this function:

5.33.2.5 MQTTSerialize_subscribe_size()

```
int MQTTSerialize_subscribe_size (
    int count,
    MQTTString topicFilters[] )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.34 st-device-sdk-c/src/include/mqtt/iot_mqtt_unsubscribe.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define DLLImport
- #define DLLExport

Functions

- DLLExport int MQTTSerialize_unsubscribe (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ])
- DLLExport int MQTTSerialize_unsubscribe_size (int count, MQTTString topicFilters[ ])
- DLLExport int MQTTDeserialize_unsubscribe (unsigned char ∗dup, unsigned short ∗packetid, int max_count, int ∗count, MQTTString topicFilters[ ], unsigned char ∗buf, int len)
- DLLExport int MQTTSerialize_unsuback (unsigned char ∗buf, int buflen, unsigned short packetid)
- DLLExport int MQTTDeserialize_unsuback (unsigned short ∗packetid, unsigned char ∗buf, int len)

5.34.1 Macro Definition Documentation

5.34.1.1 DLLExport `#define DLLExport`

5.34.1.2 DLLImport `#define DLLImport`

5.34.2 Function Documentation

5.34.2.1 MQTTDeserialize_unsuback()

```
DLLExport int MQTTDeserialize_unsuback (
    unsigned short * packetid,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into unsuback data

Parameters

packetid	returned integer - the MQTT packet identifier
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.34.2.2 MQTTDeserialize_unsubscribe()

```
DLLExport int MQTTDeserialize_unsubscribe (
    unsigned char * dup,
    unsigned short * packetid,
    int maxcount,
    int * count,
    MQTTString topicFilters[],
    unsigned char * buf,
    int len )
```

Deserializes the supplied (wire) buffer into unsubscribe data

Parameters

dup	integer returned - the MQTT dup flag
packetid	integer returned - the MQTT packet identifier

Parameters

maxcount	- the maximum number of members allowed in the topicFilters and requestedQoSs arrays
count	- number of members in the topicFilters and requestedQoSs arrays
topicFilters	- array of topic filter names
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

the length of the serialized data. <= 0 indicates error

5.34.2.3 MQTTSerialize_unsuback()

```
DLLExport int MQTTSerialize_unsuback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes the supplied unsuback data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

the length of the serialized data. <= 0 indicates error

5.34.2.4 MQTTSerialize_unsubscribe()

```
DLLExport int MQTTSerialize_unsubscribe (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[] )
```

Serializes the supplied unsubscribe data into the supplied buffer, ready for sending

Parameters

buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier
count	- number of members in the topicFilters array
topicFilters	- array of topic filter names

Returns

the length of the serialized data. <= 0 indicates error

Here is the caller graph for this function:

5.34.2.5 MQTTSerialize_unsubscribe_size()

```
DLLExport int MQTTSerialize_unsubscribe_size (
    int count,
    MQTTString topicFilters[] )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.35 st-device-sdk-c/src/include/os/iot_os_net.h File Reference

`#include "iot_os_util.h"`

Include dependency graph for iot_os_net.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_net_socket

  Contains "network management structure" data.

Typedefs

- typedef struct iot_net_socket iot_net_socket

  Contains "network management structure" data.

Functions

- void iot_os_net_print_status (iot_net_socket ∗n)

  print network status
- int iot_os_net_select (iot_net_socket ∗n, unsigned int timeout_ms)

  check network socket status
- void iot_os_net_init (iot_net_socket ∗)

  Initialize the network structure.
- int iot_os_net_connet (iot_net_socket ∗n, char ∗addr, int port)

  connect with server
- void iot_os_net_disconnect (iot_net_socket ∗n)

  disconnect with server

5.35.1 Typedef Documentation

5.35.1.1 iot_net_socket `typedef struct iot_net_socket iot_net_socket`

Contains "network management structure" data.

5.35.2 Function Documentation

5.35.2.1 iot_os_net_connet()

```
int iot_os_net_connet (
    iot_net_socket * n,
    char * addr,
    int port )
```

connect with server

Parameters

n	- iot_net_socket struct
addr	- server address
port	- server port

Returns

connect status

5.35.2.2 iot_os_net_disconnect()

```
void iot_os_net_disconnect (
    iot_net_socket * n )
```

disconnect with server

Parameters

n	- iot_net_socket struct

Returns

void

5.35.2.3 iot_os_net_init()

```
void iot_os_net_init (
    iot_net_socket * )
```

Initialize the network structure.

Parameters

n	- iot_net_socket struct

Returns

void

5.35.2.4 iot_os_net_print_status()

```
void iot_os_net_print_status (
    iot_net_socket * n )
```

print network status

Parameters

n	- iot_net_socket struct

Returns

void

Here is the caller graph for this function:

5.35.2.5 iot_os_net_select()

```
int iot_os_net_select (
    iot_net_socket * n,
    unsigned int timeout_ms )
```

check network socket status

Parameters

n	- iot_net_socket struct
timeout_ms	- timeout in miliseconds

Returns

> 0 : there is some data to read == 0 : there is no data < 0 : there is error in network

Here is the caller graph for this function:

5.36 st-device-sdk-c/src/include/os/iot_os_util.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_os_mutex

  Contains a mutex data.

Macros

- #define IOT_DELAY(x) iot_os_delay(x)
- #define IOT_OS_MAX_DELAY iot_os_max_delay
- #define IOT_OS_TRUE iot_os_true
- #define IOT_OS_FALSE iot_os_false

Typedefs

- typedef void ∗ iot_os_thread
- typedef void iot_os_queue
- typedef void iot_os_eventgroup
- typedef void iot_os_sem
- typedef void ∗ iot_os_timer
- typedef struct iot_os_mutex iot_os_mutex

  Contains a mutex data.

Functions

- int iot_os_thread_create (void *thread_function, const char *name, int stack_size, void *data, int priority, iot_os_thread *thread_handle)

  create and run thread
- void iot_os_thread_delete (iot_os_thread thread_handle)

  delete thread
- void iot_os_thread_yield ()

  yield task
- iot_os_queue * iot_os_queue_create (int queue_length, int item_size)

  create queue
- int iot_os_queue_reset (iot_os_queue *queue_handle)

  reset queue
- void iot_os_queue_delete (iot_os_queue *queue_handle)

  delete queue
- int iot_os_queue_send (iot_os_queue *queue_handle, void *data, unsigned int wait_time_ms)

  send message to the back of queue.
- int iot_os_queue_receive (iot_os_queue *queue_handle, void *data, unsigned int wait_time_ms)

  receive message from the front of queue.
- iot_os_eventgroup * iot_os_eventgroup_create (void)

  create eventgroup
- void iot_os_eventgroup_delete (iot_os_eventgroup *eventgroup_handle)

  delete eventgroup
- unsigned int iot_os_eventgroup_wait_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits↩_to_wait_for, const int clear_on_exit, const int wait_for_all_bits, const unsigned int wait_time_ms)

  wait for bit of group of bits to become set
- unsigned int iot_os_eventgroup_set_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits_↩to_set)

  set bit/bits of eventgroup
- unsigned int iot_os_eventgroup_clear_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits↩_to_clear)

  clear bit/bits of eventgroup
- int iot_os_mutex_init (iot_os_mutex *mutex)

  create mutex
- int iot_os_mutex_lock (iot_os_mutex *mutex)

  mutex lock
- int iot_os_mutex_unlock (iot_os_mutex *mutex)

  mutex unlock
- void iot_os_delay (unsigned int delay_ms)

  delay a thread
- void iot_os_timer_init (iot_os_timer *timer)

  init timer
- char iot_os_timer_isexpired (iot_os_timer timer)

  check timer expired
- void iot_os_timer_count_ms (iot_os_timer timer, unsigned int timeout_ms)

  set timer count
- unsigned int iot_os_timer_left_ms (iot_os_timer timer)

  return remaining time in ms unit
- void iot_os_timer_destroy (iot_os_timer *timer)

  destroy timer

Variables

- const unsigned int iot_os_max_delay
- const unsigned int iot_os_true
- const unsigned int iot_os_false

5.36.1 Macro Definition Documentation

5.36.1.1 IOT_DELAY

```
#define IOT_DELAY(
    x ) iot_os_delay(x)
```

5.36.1.2 IOT_OS_FALSE

```
#define IOT_OS_FALSE iot_os_false
```

5.36.1.3 IOT_OS_MAX_DELAY

```
#define IOT_OS_MAX_DELAY iot_os_max_delay
```

5.36.1.4 IOT_OS_TRUE

```
#define IOT_OS_TRUE iot_os_true
```

5.36.2 Typedef Documentation

5.36.2.1 iot_os_eventgroup

```
typedef void iot_os_eventgroup
```

5.36.2.2 iot_os_mutex

```
typedef struct iot_os_mutex iot_os_mutex
```

Contains a mutex data.

5.36.2.3 iot_os_queue

```
typedef void iot_os_queue
```

5.36.2.4 iot_os_sem `typedef void iot_os_sem`

5.36.2.5 iot_os_thread `typedef void* iot_os_thread`

5.36.2.6 iot_os_timer `typedef void* iot_os_timer`

5.36.3 Function Documentation

5.36.3.1 iot_os_delay()

```
void iot_os_delay (
    unsigned int delay_ms )
```

delay a thread

This function will delay thread for given time

Parameters

in	*delay*	time to wait

Here is the caller graph for this function:

5.36.3.2 iot_os_eventgroup_clear_bits()

```
unsigned int iot_os_eventgroup_clear_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_clear )
```

clear bit/bits of eventgroup

This function will clear bit/bits of eventgroup

Parameters

in	*eventgroup_handle*	handle of eventgroup to cleat bit/bits
in	*bits_to_clear*	bitwise value of bit/bits to clear ex) use 0b101 will clear bit0 and bit2 ex) use 0b100 will clear only bit2

Returns

return is bits of event BEFORE clear bit.

5.36.3.3 iot_os_eventgroup_create()

```
iot_os_eventgroup* iot_os_eventgroup_create ( void )
```

create eventgroup

This function create eventgroup and return eventgroup handle

Returns

return is eventgroup handle of eventgroup created by this function If eventgroup was not created, NULL is returned.

Here is the caller graph for this function:

5.36.3.4 iot_os_eventgroup_delete()

```
void iot_os_eventgroup_delete ( iot_os_eventgroup * eventgroup_handle )
```

delete eventgroup

This function delete eventgroup

Parameters

in	*eventgroup_handle*	eventgroup handle of eventgroup to delete

5.36.3.5 iot_os_eventgroup_set_bits()

```
unsigned int iot_os_eventgroup_set_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_set )
```

set bit/bits of eventgroup

This function will set bit/bits of eventgroup

Parameters

in	*eventgroup_handle*	handle of eventgroup to set bit/bits
in	*bits_to_set*	bitwise value of bit/bits to set ex) use 0b101 will set bit0 and bit2 ex) use 0b100 will set only bit2

Returns

return is bits of event after set bit/bits. if you set 'clear_on_exit' to IOT_OS_TRUE, it can be 0 because of other task waiting for bits

Here is the caller graph for this function:

5.36.3.6 iot_os_eventgroup_wait_bits()

```
unsigned int iot_os_eventgroup_wait_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_wait_for,
    const int clear_on_exit,
    const int wait_for_all_bits,
    const unsigned int wait_time_ms )
```

wait for bit of group of bits to become set

This function will wait for event bit

Parameters

in	*eventgroup_handle*	handle of eventgroup waiting for
in	*bits_to_wait_for*	bitwise value of bit/bits. ex) use 0b101 to wait bit 0 and/or bit 2
in	*clear_on_exit*	if this value is IOT_OS_TRUE, bits in 'bits_to_wait_for' will be cleared
in	*wait_for_all_bits*	if this value is IOT_OS_TRUE, wait for ""ALL"" bits in 'bits_to_wait_for'
in	*wait_time_ms*	maximum time to wait until all/one of bits are set

Returns

return is bits of eventgroup. if you wait for bit0 and bit3, can check with was setted, return will be 0b101 = 5

5.36.3.7 iot_os_mutex_init()

```
int iot_os_mutex_init (
    iot_os_mutex * mutex )
```

create mutex

This function will create mutex

Parameters

out	*mutex*	handle of mutex created by this function

Returns

IOT_ERROR_NONE : success otherwise : fail

Here is the caller graph for this function:

5.36.3.8 iot_os_mutex_lock()

```
int iot_os_mutex_lock (
    iot_os_mutex * mutex )
```

mutex lock

This function will lock mutex before critical section

Parameters

in	*mutex*	handle of mutex

Returns

IOT_ERROR_NONE : success otherwise : fail

Here is the caller graph for this function:

5.36.3.9 iot_os_mutex_unlock()

```
int iot_os_mutex_unlock (
    iot_os_mutex * mutex )
```

mutex unlock

This function will unlock mutex after critical section

Parameters

in	*mutex*	handle of mutex

Returns

IOT_ERROR_NONE : success otherwise : fail

5.36.3.10 iot_os_queue_create()

```
iot_os_queue* iot_os_queue_create (
    int queue_length,
    int item_size )
```

create queue

This function create queue and return queue handle

Parameters

in	*queue_length*	maximum number of queue item
in	*item_size*	size of item

Returns

return is queue handle of queue created by this function If queue was not created, NULL is returned.

Here is the caller graph for this function:

5.36.3.11 iot_os_queue_delete()

```
void iot_os_queue_delete (
    iot_os_queue * queue_handle )
```

delete queue

This function delete queue

Parameters

in	*queue_handle*	handle of queue to be deleted.

5.36.3.12 iot_os_queue_receive()

```
int iot_os_queue_receive (
    iot_os_queue * queue_handle,
    void * data,
    unsigned int wait_time_ms )
```

receive message from the front of queue.

This function will receive item from the front of queue

Parameters

in	*queue_handle*	handle of queue to receive item
in	*data*	buffer for item received from queue
in	*wait_time_ms*	maximum time to wait until queue is not empty

Returns

IOT_OS_TRUE : success otherwise : fail

5.36.3.13 iot_os_queue_reset()

```
int iot_os_queue_reset (
    iot_os_queue * queue_handle )
```

reset queue

This function reset queue

Parameters

in	*queue_handle*	handle of queue to reset.

Returns

IOT_OS_TRUE : success otherwise : fail

5.36.3.14 iot_os_queue_send()

```
int iot_os_queue_send (
    iot_os_queue * queue_handle,
    void * data,
    unsigned int wait_time_ms )
```

send message to the back of queue.

This function will send item to the back of queue

Parameters

in	*queue_handle*	handle of queue to save item
in	*data*	item to be saved in queue
in	*wait_time_ms*	maximum time to wait until queue is available

Returns

IOT_OS_TRUE : success otherwise : fail

Here is the caller graph for this function:

5.36.3.15 iot_os_thread_create()

```
int iot_os_thread_create (
    void * thread_function,
    const char * name,
    int stack_size,
    void * data,
    int priority,
    iot_os_thread * thread_handle )
```

create and run thread

This function creates and runs thread

Parameters

in	*thread_function*	function pointer for thread loop
in	*name*	name of thread
in	*stack_size*	stack size of thread
in	*data*	data that will passed into the thread as the parameter
in	*priority*	priority of thread created by this function
out	*thread_handle*	thread handle of thread created by this function (optional, NULL is possible)

Returns

IOT_OS_TRUE : success otherwise : fail

Here is the caller graph for this function:

5.36.3.16 iot_os_thread_delete()

```
void iot_os_thread_delete (
    iot_os_thread thread_handle )
```

delete thread

This function delete thread

Parameters

in	*thread_handle*	thread handle of thread to delete(if NULL is passed, delete current thread)

5.36.3.17 iot_os_thread_yield() `void iot_os_thread_yield ( )`

yield task

This function yields task

5.36.3.18 iot_os_timer_count_ms() `void iot_os_timer_count_ms ( iot_os_timer timer, unsigned int timeout_ms )`

set timer count

This function will set timer count in ms unit.

Parameters

in	*timer*	timer handle
in	*count*	count to set in ms unit

Here is the caller graph for this function:

5.36.3.19 iot_os_timer_destroy() `void iot_os_timer_destroy ( iot_os_timer * timer )`

destroy timer

This function will destroy timer struct

Parameters

in	*timer*	pointer of timer to destroy

Here is the caller graph for this function:

5.36.3.20 iot_os_timer_init() `void iot_os_timer_init (iot_os_timer * timer )`

init timer

This function will init timer struct

Parameters

in	*timer*	pointer of timer to init

Here is the caller graph for this function:

5.36.3.21 iot_os_timer_isexpired() `char iot_os_timer_isexpired (iot_os_timer timer )`

check timer expired

This function will check if timer is expired

Parameters

in	*timer*	timer handle

Returns

1 : timer is expired 0 : timer is not expired

Here is the caller graph for this function:

5.36.3.22 iot_os_timer_left_ms()

```
unsigned int iot_os_timer_left_ms ( iot_os_timer timer )
```

return remaining time in ms unit

This function will return remaining time in ms unit

Parameters

in	*timer*	timer handle

Returns

0 : timer is expired non-zero : remaining time

Here is the caller graph for this function:

5.36.4 Variable Documentation

5.36.4.1 iot_os_false `const unsigned int iot_os_false`

5.36.4.2 iot_os_max_delay `const unsigned int iot_os_max_delay`

5.36.4.3 iot_os_true `const unsigned int iot_os_true`

5.37 st-device-sdk-c/src/include/st_dev.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include "st_dev_version.h"
```

Include dependency graph for st_dev.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct iot_pin_t

  Contains a pin values for pin type onboarding process.
- struct iot_cap_val_t

  Contains a various type of data which can be int, double, string and string array.
- struct iot_cap_cmd_data_t

  Contains data for "command" payload.
- union noti_data_raw_t

  Contains data for raw data of each notification.
- struct noti_data_raw_t::_rate_limit
- struct noti_data_raw_t::_quota
- struct iot_noti_data_t

  Contains data for notification data.

Macros

- #define MAX_CAP_ARG (5)

Typedefs

- typedef enum iot_status iot_status_t

  Contains a enumeration values for types of iot_status.
- typedef enum iot_stat_lv iot_stat_lv_t

  Contains a enumeration values for types of iot_status level.
- typedef void ∗ IOT_CTX
- typedef void ∗ IOT_CAP_HANDLE
- typedef void ∗ IOT_EVENT
- typedef struct iot_pin_t iot_pin_t

  Contains a pin values for pin type onboarding process.
- typedef enum iot_cap_val_type iot_cap_val_type_t

  Contains a enumeration values for types of capability.
- typedef enum iot_noti_type iot_noti_type_t

  Contains a enumeration values for types of notification.
- typedef void(∗ st_status_cb) (iot_status_t iot_status, iot_stat_lv_t stat_lv, void ∗usr_data)
- typedef void(∗ st_cap_init_cb) (IOT_CAP_HANDLE ∗cap_handle, void ∗init_usr_data)
- typedef void(∗ st_cap_noti_cb) (iot_noti_data_t ∗noti_data, void ∗noti_usr_data)
- typedef void(∗ st_cap_cmd_cb) (IOT_CAP_HANDLE ∗cap_handle, iot_cap_cmd_data_t ∗cmd_data, void ∗usr_data)

Enumerations

- enum iot_status { IOT_STATUS_IDLE = (1 << 0), IOT_STATUS_PROVISIONING = (1 << 1), IOT_STATUS_NEED_INTERACT = (1 << 2), IOT_STATUS_CONNECTING = (1 << 3), IOT_STATUS_ALL }

  Contains a enumeration values for types of iot_status.
- enum iot_stat_lv { IOT_STAT_LV_STAY, IOT_STAT_LV_START, IOT_STAT_LV_DONE, IOT_STAT_LV_FAIL }

  Contains a enumeration values for types of iot_status level.
- enum iot_cap_val_type { IOT_CAP_VAL_TYPE_UNKNOWN = -1, IOT_CAP_VAL_TYPE_INTEGER, IOT_CAP_VAL_TYPE_NUMBER, IOT_CAP_VAL_TYPE_INT_OR_NUM, IOT_CAP_VAL_TYPE_STRING, IOT_CAP_VAL_TYPE_STR_ARRAY }

  Contains a enumeration values for types of capability.
- enum iot_noti_type { IOT_NOTI_TYPE_UNKNOWN = -1, IOT_NOTI_TYPE_DEV_DELETED, IOT_NOTI_TYPE_RATE_LIMIT, IOT_NOTI_TYPE_QUOTA_REACHED }

  Contains a enumeration values for types of notification.

Functions

- IOT_EVENT * st_cap_attr_create_int (char *attribute, int integer, char *unit)

  Create IOT_EVENT data with integer `value`.
- IOT_EVENT * st_cap_attr_create_number (char *attribute, double number, char *unit)

  Create IOT_EVENT data with real number(double) `value`.
- IOT_EVENT * st_cap_attr_create_string (char *attribute, char *string, char *unit)

  Create IOT_EVENT data with string `value`.
- IOT_EVENT * st_cap_attr_create_string_array (char *attribute, uint8_t str_num, char *string_array[ ], char *unit)

  Create IOT_EVENT data with string array `value`.
- void st_cap_attr_free (IOT_EVENT *event)

  Free IOT_EVENT data.
- int st_cap_attr_send (IOT_CAP_HANDLE *cap_handle, uint8_t evt_num, IOT_EVENT *event[ ])

  Request to publish deviceEvent.
- IOT_CAP_HANDLE * st_cap_handle_init (IOT_CTX *iot_ctx, const char *component, const char *capability, st_cap_init_cb init_cb, void *init_usr_data)

  Create and initialize a capability handle.
- int st_conn_set_noti_cb (IOT_CTX *iot_ctx, st_cap_noti_cb noti_cb, void *noti_usr_data)

  Register callback function for notification event.
- int st_cap_cmd_set_cb (IOT_CAP_HANDLE *cap_handle, const char *cmd_type, st_cap_cmd_cb cmd_cb, void *usr_data)

  Register callback function for command message.
- IOT_CTX * st_conn_init (unsigned char *onboarding_config, unsigned int onboarding_config_len, unsigned char *device_info, unsigned int device_info_len)

  st-iot-core initialize function
- int st_conn_start (IOT_CTX *iot_ctx, st_status_cb status_cb, iot_status_t maps, void *usr_data, iot_pin_t *pin_num)

  st-iot-core server connection function
- int st_conn_cleanup (IOT_CTX *iot_ctx, bool reboot)

  st-iot-core device clean-up function
- void st_conn_ownership_confirm (IOT_CTX *iot_ctx, bool confirm)

  easysetup user confirm report function

5.37.1 Macro Definition Documentation

5.37.1.1 MAX_CAP_ARG `#define MAX_CAP_ARG (5)`

5.37.2 Typedef Documentation

5.37.2.1 IOT_CAP_HANDLE `typedef void* IOT_CAP_HANDLE`

5.37.2.2 iot_cap_val_type_t `typedef enum iot_cap_val_type iot_cap_val_type_t`

Contains a enumeration values for types of capability.

5.37.2.3 IOT_CTX `typedef void* IOT_CTX`

5.37.2.4 IOT_EVENT `typedef void* IOT_EVENT`

5.37.2.5 iot_noti_type_t `typedef enum iot_noti_type iot_noti_type_t`

Contains a enumeration values for types of notification.

5.37.2.6 iot_pin_t `typedef struct iot_pin_t iot_pin_t`

Contains a pin values for pin type onboarding process.

5.37.2.7 iot_stat_lv_t `typedef enum iot_stat_lv iot_stat_lv_t`

Contains a enumeration values for types of iot_status level.

5.37.2.8 iot_status_t `typedef enum iot_status iot_status_t`

Contains a enumeration values for types of iot_status.

5.37.2.9 st_cap_cmd_cb `typedef void(* st_cap_cmd_cb) (IOT_CAP_HANDLE *cap_handle, iot_cap_cmd_data_t *cmd_data, void *usr_data)`

5.37.2.10 st_cap_init_cb `typedef void(* st_cap_init_cb) (IOT_CAP_HANDLE *cap_handle, void *init↩ _usr_data)`

5.37.2.11 st_cap_noti_cb `typedef void(* st_cap_noti_cb) (iot_noti_data_t *noti_data, void *noti_usr_data)`

5.37.2.12 st_status_cb `typedef void(* st_status_cb) (iot_status_t iot_status, iot_stat_lv_t stat_lv, void *usr_data)`

5.37.3 Enumeration Type Documentation

5.37.3.1 iot_cap_val_type `enum iot_cap_val_type`

Contains a enumeration values for types of capability.

Enumerator

IOT_CAP_VAL_TYPE_UNKNOWN	For undefined type.
IOT_CAP_VAL_TYPE_INTEGER	For integer.
IOT_CAP_VAL_TYPE_NUMBER	For float number.
IOT_CAP_VAL_TYPE_INT_OR_NUM	For integer or float number.
IOT_CAP_VAL_TYPE_STRING	For NULL-terminated string.
IOT_CAP_VAL_TYPE_STR_ARRAY	For array of NULL-terminated strings.

5.37.3.2 iot_noti_type `enum iot_noti_type`

Contains a enumeration values for types of notification.

Enumerator

IOT_NOTI_TYPE_UNKNOWN	For undefined type.
IOT_NOTI_TYPE_DEV_DELETED	For device deleted event.
IOT_NOTI_TYPE_RATE_LIMIT	For rate limit event.
IOT_NOTI_TYPE_QUOTA_REACHED	For data quota reached event.

5.37.3.3 iot_stat_lv `enum iot_stat_lv`

Contains a enumeration values for types of iot_status level.

Enumerator

IOT_STAT_LV_STAY	meanings for staying level with each status
IOT_STAT_LV_START	meanings for start level with each status
IOT_STAT_LV_DONE	meanings for done level with each status
IOT_STAT_LV_FAIL	meanings for fail level with each status

5.37.3.4 iot_status `enum iot_status`

Contains a enumeration values for types of iot_status.

Enumerator

IOT_STATUS_IDLE	For idle status, not connected.
IOT_STATUS_PROVISIONING	For provisioning status. do onboarding process.
IOT_STATUS_NEED_INTERACT	For user interation status. need to interact with user.
IOT_STATUS_CONNECTING	For server connecting status. do connecting server.
IOT_STATUS_ALL	

5.37.4 Function Documentation

5.37.4.1 st_cap_attr_create_int()

```
IOT_EVENT* st_cap_attr_create_int (
    char * attribute,
    int integer,
    char * unit )
```

Create IOT_EVENT data with integer `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*integer*	The integer to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.37.4.2 st_cap_attr_create_number()

```
IOT_EVENT* st_cap_attr_create_number (
    char * attribute,
    double number,
    char * unit )
```

Create IOT_EVENT data with real number(double) `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*number*	The double number to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.37.4.3 st_cap_attr_create_string()

```
IOT_EVENT* st_cap_attr_create_string (
    char * attribute,
    char * string,
    char * unit )
```

Create IOT_EVENT data with string `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*string*	The string to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.37.4.4 st_cap_attr_create_string_array()

```
IOT_EVENT* st_cap_attr_create_string_array (
    char * attribute,
    uint8_t str_num,
    char * string_array[],
    char * unit )
```

Create IOT_EVENT data with string array `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*str_num*	The number of strings in the string_array.
in	*string_array*	The pointer of string array to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.37.4.5 st_cap_attr_free()

```
void st_cap_attr_free (
    IOT_EVENT * event )
```

Free IOT_EVENT data.

This function frees IOT_EVENT data.

Parameters

in	*event*	The IOT_EVENT data to free.

5.37.4.6 st_cap_attr_send()

```
int st_cap_attr_send (
    IOT_CAP_HANDLE * cap_handle,
    uint8_t evt_num,
    IOT_EVENT * event[] )
```

Request to publish deviceEvent.

This function creates a deviceEvent with the list of IOT_EVENT data, and requests to publish it. When there is no error, this function returns sequence number, which is unique value to identify the deviceEvent message.

Parameters

in	*cap_handle*	The IOT_CAP_HANDLE to publish a deviceEvent.
in	*evt_num*	The number of IOT_EVENT data in the event.
in	*event*	The IOT_EVENT data list to create the deviceEvent.

Returns

return `sequence number`(which is positive integer) if successful, negative integer for error case.

5.37.4.7 st_cap_cmd_set_cb()

```
int st_cap_cmd_set_cb (
    IOT_CAP_HANDLE * cap_handle,
    const char * cmd_type,
    st_cap_cmd_cb cmd_cb,
    void * usr_data )
```

Register callback function for command message.

This function registers user callback for command message from ST server. If the `capability`(used to create handle) and `cmd_type` of command message are same with input arguments, the callback function will be called.

Parameters

in	*cap_handle*	The capability handle to register cb function.
in	*cmd_type*	The commands interested to process.
in	*cmd_cb*	The callback function invoked when command is received.
in	*usr_data*	User data for cmd_cb.

Returns

return `(0)` if it works successfully, non-zero for error case.

Warning

User callback must return because MQTT works after user callback has ended.

5.37.4.8 st_cap_handle_init()

```
IOT_CAP_HANDLE* st_cap_handle_init (
    IOT_CTX * iot_ctx,
    const char * component,
    const char * capability,
    st_cap_init_cb init_cb,
    void * init_usr_data )
```

Create and initialize a capability handle.

This function creates a capability handle, and initializes it with input args.

Parameters

in	*iot_ctx*	The iot context.
in	*component*	Component string. Default component name is "main".
in	*capability*	Capability string. This should be matched with "id" value of capability definition json format.
in	*init_cb*	The function which is called to initialize device state.
in	*init_usr_data*	User data for init_cb.

Returns

Pointer of created capability handle.

5.37.4.9 st_conn_cleanup()

```
int st_conn_cleanup (
    IOT_CTX * iot_ctx,
    bool reboot )
```

st-iot-core device clean-up function

This function cleans-up all DATA including provisioning & registered data

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*reboot*	boolean set true for auto-reboot of system, else false.

Returns

return `(0)` if it works successfully, non-zero for error case.

Here is the call graph for this function:

5.37.4.10 st_conn_init()

```
IOT_CTX* st_conn_init (
    unsigned char * onboarding_config,
    unsigned int onboarding_config_len,
    unsigned char * device_info,
    unsigned int device_info_len )
```

st-iot-core initialize function

This function initializes st-iot-core for target

Parameters

in	*onboarding_config*	starting pointer of onboarding_config.json contents
in	*onboarding_config_len*	size of onboarding_config.json contents
in	*device_info*	starting pointer of device_info.json contents
in	*device_info_len*	size of device_info.json contents

Returns

return IOT_CTX handle(a pointer) if it succeeds, or NULL if it fails

Here is the call graph for this function:

5.37.4.11 st_conn_ownership_confirm()

```
void st_conn_ownership_confirm (
    IOT_CTX * iot_ctx,
    bool confirm )
```

easysetup user confirm report function

This function reports the user confirmation to easysetup

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*confirm*	user confirmation result

Here is the call graph for this function:

5.37.4.12 st_conn_set_noti_cb()

```
int st_conn_set_noti_cb (
    IOT_CTX * iot_ctx,
    st_cap_noti_cb noti_cb,
    void * noti_usr_data )
```

Register callback function for notification event.

This function registers user callback function which will be called when notification event occurs(such as `rate limit`, `delete device`).

Parameters

in	*iot_ctx*	The iot context.
in	*noti_cb*	The callback function which will be called when notification event occurs.
in	*noti_usr_data*	User data for noti_cb.

Returns

return `(0)` if it works successfully, non-zero for error case.

Warning

User callback must return because MQTT works after user callback has ended

5.37.4.13 st_conn_start()

```
int st_conn_start (
    IOT_CTX * iot_ctx,
    st_status_cb status_cb,
    iot_status_t maps,
    void * usr_data,
    iot_pin_t * pin_num )
```

st-iot-core server connection function

This function tries to connect server

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*status_cb*	user callback function to receive status of st-iot-core
in	*maps*	status of st-iot-core interested to receive through status_cb
in	*usr_data*	user data(a pointer) to use in status_cb
in	*pin_num*	if PIN ownership validation type used, valid 8 digit pin should be set. otherwise set null.

Returns

return `(0)` if it works successfully, non-zero for error case.

5.38 st-device-sdk-c/src/include/st_dev_version.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define VER_MAJOR (1)
- #define VER_MINOR (0)
- #define VER_PATCH (0)
- #define STDK_VERSION(a, b, c) (((a) << 16) + ((b) << 8) + (c))
- #define STDK_VERSION_CODE (STDK_VERSION(VER_MAJOR,VER_MINOR,VER_PATCH))

5.38.1 Macro Definition Documentation

5.38.1.1 STDK_VERSION

```
#define STDK_VERSION(
    a,
    b,
    c ) (((a) << 16) + ((b) << 8) + (c))
```

5.38.1.2 STDK_VERSION_CODE

```
#define STDK_VERSION_CODE (STDK_VERSION(VER_MAJOR,VER_MINOR,VER_PATCH))
```

5.38.1.3 VER_MAJOR `#define VER_MAJOR (1)`

5.38.1.4 VER_MINOR `#define VER_MINOR (0)`

5.38.1.5 VER_PATCH `#define VER_PATCH (0)`

5.39 st-device-sdk-c/src/iot_api.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_debug.h"
#include "iot_easysetup.h"
#include "iot_crypto.h"
#include "iot_nv_data.h"
#include "iot_os_util.h"
#include "cJSON.h"
```

Include dependency graph for iot_api.c:

Functions

- iot_error_t iot_command_send (struct iot_context ∗ctx, enum iot_command_type new_cmd, const void ∗param, int param_size)

  send command to iot main task
- iot_error_t iot_easysetup_request (struct iot_context ∗ctx, enum iot_easysetup_step step, const void ∗payload)

send easysetup cgi payload manipulation request

- iot_error_t iot_state_update (struct iot_context ∗ctx, iot_state_t new_state, int opt)

  update iot state
- void iot_api_onboarding_config_mem_free (struct iot_devconf_prov_data ∗devconf)

  free onboarding config memory
- iot_error_t iot_api_onboarding_config_load (unsigned char ∗onboarding_config, unsigned int onboarding_↩ config_len, struct iot_devconf_prov_data ∗devconf)

  load "onboarding_config.json" from application source directory
- iot_error_t iot_get_time_in_sec (char ∗buf, size_t buf_len)

  get time data by sec
- iot_error_t iot_get_time_in_ms (char ∗buf, size_t buf_len)

  get time data in msec
- void iot_api_device_info_mem_free (struct iot_device_info ∗device_info)

  free device info memory
- iot_error_t iot_api_device_info_load (unsigned char ∗device_info, unsigned int device_info_len, struct iot_device_info ∗info)

  load "device_info.json" from application source directory
- void iot_api_prov_data_mem_free (struct iot_device_prov_data ∗prov)

  free prov data memory
- iot_error_t iot_api_read_device_identity (unsigned char ∗nv_prof, unsigned int nv_prof_len, const char ∗object, char ∗∗nv_data)

  Extract required data from "device_info.json" which is located in application source directory.
- iot_error_t iot_device_cleanup (struct iot_context ∗ctx)

  device cleanup

5.39.1 Function Documentation

5.39.1.1 iot_api_device_info_load()

```
iot_error_t iot_api_device_info_load (
    unsigned char * device_info,
    unsigned int device_info_len,
    struct iot_device_info * info )
```

load "device_info.json" from application source directory

"device_info.json" should be updated by application developer
This function parses downloaded "device_info.json" to be used for EasySetup
Only firmwareVersion will be parsed by this api. others are handled by another api

Parameters

in	*device_info*	start pointer of json data
in	*device_info_len*	json data length
out	*info*	"device_info.json" will be parsed and mapped to this internal structure

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.

Return values

IOT_ERROR_MEM_ALLOC	memory allocation failure.

example

```
{
    "deviceInfo": {
        "firmwareVersion": "FwVer0011A",
    ...
    }
}
```

Here is the caller graph for this function:

5.39.1.2 iot_api_device_info_mem_free()

```
void iot_api_device_info_mem_free (
    struct iot_device_info * info )
```

free device info memory

this function frees the loaded device's information

Parameters

in	*info*	loaded device's information

5.39.1.3 iot_api_onboarding_config_load()

```
iot_error_t iot_api_onboarding_config_load (
    unsigned char * onboarding_config,
    unsigned int onboarding_config_len,
    struct iot_devconf_prov_data * devconf )
```

load "onboarding_config.json" from application source directory

"onboarding_config.json" can be downloaded from SmartThings Developer Workspace
This function parses downloaded "onboarding_config.json" to be used for EasySetup

Parameters

in	*onboarding_config*	start pointer of json data
in	*onboarding_config_len*	json data length
out	*devconf*	"onboarding_config.json" will be parsed and mapped to this internal structure

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.
IOT_ERROR_MEM_ALLOC	memory allocation failure.
IOT_ERROR_CRYPTO_SHA256	sha256 error.
IOT_ERROR_CRYPTO_BASE64	base64 error.
IOT_ERROR_CRYPTO_BASE64_URLSAFE	base64 urlsafe error.

example

```
{
   "onboardingConfig": {
       "deviceOnboardingID": "NAME", // max. 13 character. this will be prefix of soft-ap ssid.
       "mnId": "MNID", // mnId for developer and/or manufacturer. "MNID" shouldn't be used.
       "setupId": "999", // 3-digit Device onboarding ID for this device.
       "vid": "VID", // VID(Vendor ID) for this profile.
       "deviceTypeId": "TYPE", // Device type which is selected from Developer Workspace.
       "ownershipValidationType": [ "JUSTWORKS", "BUTTON", "PIN", "QR" ],
           // "JUSTWORKS" for confirming without user interaction.
           // "BUTTON" for confirming by pressing builtin button.
           // "PIN" for confirming by matching 8-digit number PIN
           // "QR" for confirming by scanning a QR code by SmartThings app.
       "identityType": "ED25519 or CERTIFICATE" // ED25519 or X.509 CERTIFICATE
    }
}
```

Here is the caller graph for this function:

5.39.1.4 iot_api_onboarding_config_mem_free()

```
void iot_api_onboarding_config_mem_free (
    struct iot_devconf_prov_data * devconf )
```

free onboarding config memory

this function frees the loaded onboarding configuration

Parameters

in	*devconf*	loaded onboarding configuration

5.39.1.5 iot_api_prov_data_mem_free()

```
void iot_api_prov_data_mem_free (
```

```
struct iot_device_prov_data * prov )
```

free prov data memory

this function frees the loaded provisioning data

Parameters

in	*prov*	loaded provisioning data

Here is the caller graph for this function:

5.39.1.6 iot_api_read_device_identity()

```
iot_error_t iot_api_read_device_identity (
    unsigned char * device_nv_info,
    unsigned int device_nv_info_len,
    const char * object,
    char ** nv_data )
```

Extract required data from "device_info.json" which is located in application source directory.

"device_info.json" should be updated by application developer
This function parses downloaded "device_info.json" to be used for EasySetup

Parameters

in	*device_nv_info*	starting pointer of json data
in	*device_nv_info_len*	json data length
in	*object*	object name for searching json data.
out	*nv_data*	"device_info.json" will be parsed by "object" and mapped to this pointer

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_UNINITIALIZED	invalid json value.
IOT_ERROR_MEM_ALLOC	memory allocation failure.

example

```
{
 "nvProfile": {
    "privateKey": "privateKey", // Client (= Device) Private key
    "publicKey": "publicKey", // Client (= Device) Public key
    "serialNumber": "serialNumber" // Device Serial Number
 }
}
```

Here is the caller graph for this function:

5.39.1.7 iot_command_send()

```
iot_error_t iot_command_send (
    struct iot_context * ctx,
    enum iot_command_type cmd_type,
    const void * param,
    int param_size )
```

send command to iot main task

this function sends specific command to iot-task via queue

Parameters

in	*ctx*	iot-core context
in	*cmd_type*	actual specific command type
in	*param*	additional parameter data for each command
in	*param_size*	additional parameter size

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_MEM_ALLOC	memory allocation failed
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

Here is the caller graph for this function:

5.39.1.8 iot_device_cleanup()

```
iot_error_t iot_device_cleanup (
    struct iot_context * ctx )
```

device cleanup

this function triggers clean-up process. All registered data will be removed

Parameters

in	*ctx*	iot-core context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.39.1.9 iot_easysetup_request()

```
iot_error_t iot_easysetup_request (
    struct iot_context * ctx,
    enum iot_easysetup_step step,
    const void * payload )
```

send easysetup cgi payload manipulation request

easysetup cgi payload manipulation should be done at iot-task. This function sends payload to iot-task via queue

Parameters

in	*ctx*	iot-core context
in	*step*	indicates which uri(command) is dealing with
in	*payload*	payload data - mostly json data

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

5.39.1.10 iot_get_time_in_ms()

```
iot_error_t iot_get_time_in_ms (
    char * buf,
    size_t buf_len )
```

get time data in msec

this function tries to get time value in millisecond by string

Parameters

in	*buf*	buffer point to contain millisecond based string value
in	*buf_len*	size of allocated buffer for string

Return values

IOT_ERROR_NONE	success.

5.39.1.11 iot_get_time_in_sec()

```
iot_error_t iot_get_time_in_sec (
    char * buf,
    size_t buf_len )
```

get time data by sec

this function tries to get time value in second by string

Parameters

in	*buf*	buffer point to contain second based string value
in	*buf_len*	size of allocated buffer for string

Return values

IOT_ERROR_NONE	success.

5.39.1.12 iot_state_update()

```
iot_error_t iot_state_update (
    struct iot_context * ctx,
    iot_state_t new_state,
    int need_interact )
```

update iot state

this function tries to update iot-state using iot_command_send internally

Parameters

in	*ctx*	iot-core context
in	*new_state*	new iot-state to update
in	*need_interact*	additional parameter data for each command

Return values

IOT_ERROR_NONE	success.
IOT_ERROR_MEM_ALLOC	memory allocation failed
IOT_ERROR_BAD_REQ	queue send error

Here is the call graph for this function:

5.40 st-device-sdk-c/src/iot_capability.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <cJSON.h>
#include "iot_internal.h"
#include "iot_util.h"
#include "iot_debug.h"
#include "iot_capability.h"
#include "iot_os_util.h"
#include "iot_bsp_system.h"
```

Include dependency graph for iot_capability.c:

Macros

- #define MAX_SQNUM 0x7FFFFFFF

Functions

- IOT_EVENT * st_cap_attr_create_int (char *attribute, int integer, char *unit)

  Create IOT_EVENT data with integer `value`.
- IOT_EVENT * st_cap_attr_create_number (char *attribute, double number, char *unit)

  Create IOT_EVENT data with real number(double) `value`.
- IOT_EVENT * st_cap_attr_create_string (char *attribute, char *string, char *unit)

  Create IOT_EVENT data with string `value`.
- IOT_EVENT * st_cap_attr_create_string_array (char *attribute, uint8_t str_num, char *string_array[ ], char *unit)

  Create IOT_EVENT data with string array `value`.
- void st_cap_attr_free (IOT_EVENT *event)

  Free IOT_EVENT data.
- IOT_CAP_HANDLE * st_cap_handle_init (IOT_CTX *iot_ctx, const char *component, const char *capability, st_cap_init_cb init_cb, void *init_usr_data)

  Create and initialize a capability handle.
- int st_conn_set_noti_cb (IOT_CTX *iot_ctx, st_cap_noti_cb noti_cb, void *noti_usr_data)

  Register callback function for notification event.
- int st_cap_cmd_set_cb (IOT_CAP_HANDLE *cap_handle, const char *cmd_type, st_cap_cmd_cb cmd_cb, void *usr_data)

  Register callback function for command message.
- int st_cap_attr_send (IOT_CAP_HANDLE *cap_handle, uint8_t evt_num, IOT_EVENT *event[ ])

  Request to publish deviceEvent.
- void iot_noti_sub_cb (struct iot_context *ctx, char *payload)

  callback for mqtt noti msg
- void iot_cap_sub_cb (iot_cap_handle_list_t *cap_handle_list, char *payload)

  callback for mqtt command msg
- void iot_cap_call_init_cb (iot_cap_handle_list_t *cap_handle_list)

  call init callback

5.40.1 Macro Definition Documentation

5.40.1.1 MAX_SQNUM `#define MAX_SQNUM 0x7FFFFFFF`

5.40.2 Function Documentation

5.40.2.1 iot_cap_call_init_cb()

```
void iot_cap_call_init_cb (
    iot_cap_handle_list_t * cap_handle_list )
```

call init callback

this function is used to call all allocated capability callbacks when target is connected

Parameters

in	*cap_handle_list*	allocated capability handle list

5.40.2.2 iot_cap_sub_cb()

```
void iot_cap_sub_cb (
    iot_cap_handle_list_t * cap_handle_list,
    char * payload )
```

callback for mqtt command msg

this function is used to handle command message from server

Parameters

in	*cap_handle_list*	allocated capability handle list
in	*payload*	received raw message from server

Here is the caller graph for this function:

5.40.2.3 iot_noti_sub_cb()

```
void iot_noti_sub_cb (
    struct iot_context * ctx,
    char * payload )
```

callback for mqtt noti msg

this function is used to handle notification message from server

Parameters

in	*ctx*	iot-core context
in	*payload*	received raw message from server

Here is the caller graph for this function:

5.40.2.4 st_cap_attr_create_int()

```
IOT_EVENT* st_cap_attr_create_int (
    char * attribute,
    int integer,
    char * unit )
```

Create IOT_EVENT data with integer `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*integer*	The integer to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.40.2.5 st_cap_attr_create_number()

```
IOT_EVENT* st_cap_attr_create_number (
    char * attribute,
    double number,
    char * unit )
```

Create IOT_EVENT data with real number(double) `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*number*	The double number to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.40.2.6 st_cap_attr_create_string()

```
IOT_EVENT* st_cap_attr_create_string (
    char * attribute,
    char * string,
    char * unit )
```

Create IOT_EVENT data with string `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*string*	The string to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.40.2.7 st_cap_attr_create_string_array()

```
IOT_EVENT* st_cap_attr_create_string_array (
    char * attribute,
    uint8_t str_num,
    char * string_array[],
    char * unit )
```

Create IOT_EVENT data with string array `value`.

This function creates a new IOT_EVENT data with input parameters. Once it returns, user has full responsibility for deallocating event data by using st_cap_attr_free.

Parameters

in	*attribute*	The attribute string of IOT_EVENT data.
in	*str_num*	The number of strings in the string_array.
in	*string_array*	The pointer of string array to add to IOT_EVENT data.
in	*unit*	The unit string if needed. Otherwise NULL.

Returns

Pointer of `IOT_EVENT` which is used to publish device status.

Warning

Must call st_cap_attr_free to free IOT_EVENT data after using it.

See also

st_cap_attr_send

5.40.2.8 st_cap_attr_free()

```
void st_cap_attr_free (
    IOT_EVENT * event )
```

Free IOT_EVENT data.

This function frees IOT_EVENT data.

Parameters

in	*event*	The IOT_EVENT data to free.

5.40.2.9 st_cap_attr_send()

```
int st_cap_attr_send (
    IOT_CAP_HANDLE * cap_handle,
    uint8_t evt_num,
    IOT_EVENT * event[] )
```

Request to publish deviceEvent.

This function creates a deviceEvent with the list of IOT_EVENT data, and requests to publish it. When there is no error, this function returns sequence number, which is unique value to identify the deviceEvent message.

Parameters

in	*cap_handle*	The IOT_CAP_HANDLE to publish a deviceEvent.
in	*evt_num*	The number of IOT_EVENT data in the event.
in	*event*	The IOT_EVENT data list to create the deviceEvent.

Returns

return `sequence number`(which is positive integer) if successful, negative integer for error case.

5.40.2.10 st_cap_cmd_set_cb()

```
int st_cap_cmd_set_cb (
    IOT_CAP_HANDLE * cap_handle,
    const char * cmd_type,
    st_cap_cmd_cb cmd_cb,
    void * usr_data )
```

Register callback function for command message.

This function registers user callback for command message from ST server. If the `capability`(used to create handle) and `cmd_type` of command message are same with input arguments, the callback function will be called.

Parameters

in	*cap_handle*	The capability handle to register cb function.
in	*cmd_type*	The commands interested to process.
in	*cmd_cb*	The callback function invoked when command is received.
in	*usr_data*	User data for cmd_cb.

Returns

return `(0)` if it works successfully, non-zero for error case.

Warning

User callback must return because MQTT works after user callback has ended.

5.40.2.11 st_cap_handle_init()

```
IOT_CAP_HANDLE* st_cap_handle_init (
    IOT_CTX * iot_ctx,
    const char * component,
    const char * capability,
```

```
st_cap_init_cb init_cb,
void * init_usr_data )
```

Create and initialize a capability handle.

This function creates a capability handle, and initializes it with input args.

Parameters

in	*iot_ctx*	The iot context.
in	*component*	Component string. Default component name is "main".
in	*capability*	Capability string. This should be matched with "id" value of capability definition json format.
in	*init_cb*	The function which is called to initialize device state.
in	*init_usr_data*	User data for init_cb.

Returns

Pointer of created capability handle.

5.40.2.12 st_conn_set_noti_cb()

```
int st_conn_set_noti_cb (
    IOT_CTX * iot_ctx,
    st_cap_noti_cb noti_cb,
    void * noti_usr_data )
```

Register callback function for notification event.

This function registers user callback function which will be called when notification event occurs(such as `rate limit`, `delete device`).

Parameters

in	*iot_ctx*	The iot context.
in	*noti_cb*	The callback function which will be called when notification event occurs.
in	*noti_usr_data*	User data for noti_cb.

Returns

return `(0)` if it works successfully, non-zero for error case.

Warning

User callback must return because MQTT works after user callback has ended

5.41 st-device-sdk-c/src/iot_crypto.c File Reference

```
#include <string.h>
#include "iot_main.h"
#include "iot_debug.h"
```

```
#include "iot_crypto_internal.h"
```

Include dependency graph for iot_crypto.c:

Functions

- iot_error_t iot_crypto_pk_init (iot_crypto_pk_context_t ∗ctx, iot_crypto_pk_info_t ∗info)

  Initialize a context by passed private key data.
- void iot_crypto_pk_free (iot_crypto_pk_context_t ∗ctx)

  Cleanup the context.
- iot_error_t iot_crypto_pk_sign (iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size_t ∗slen)

  Generate a signature.
- iot_error_t iot_crypto_pk_verify (iot_crypto_pk_context_t ∗ctx, unsigned char ∗input, size_t ilen, unsigned char ∗sig, size_t slen)

  Verify the signature.

5.41.1 Function Documentation

5.41.1.1 iot_crypto_pk_free()

```
void iot_crypto_pk_free (
    iot_crypto_pk_context_t * ctx )
```

Cleanup the context.

Parameters

in	*ctx*	a pointer to a buffer to cleanup

5.41.1.2 iot_crypto_pk_init()

```
iot_error_t iot_crypto_pk_init (
```

```
iot_crypto_pk_context_t * ctx,
iot_crypto_pk_info_t * info )
```

Initialize a context by passed private key data.

Parameters

in	*ctx*	a pointer to a buffer to handle crypto context
in	*info*	a pointer to a buffer containing private key data

Return values

IOT_ERROR_NONE	context is sucessfully initialized
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE	private key data has not supported algorithm

5.41.1.3 iot_crypto_pk_sign()

```
iot_error_t iot_crypto_pk_sign (
    iot_crypto_pk_context_t * ctx,
    unsigned char * input,
    size_t ilen,
    unsigned char * sig,
    size_t * slen )
```

Generate a signature.

Parameters

in	*ctx*	a pointer to a buffer containing crypto context
in	*input*	a pointer to a buffer to generate a signature
in	*ilen*	the size of buffer pointed by input in bytes
out	*sig*	a pointer to a buffer to store a signature
out	*slen*	the bytes written to sig

Return values

IOT_ERROR_NONE	the signature is sucessfully generated
IOT_ERROR_CRYPTO_PK_PARSEKEY	failed to parse device certificate
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN	a length of private key is wrong
IOT_ERROR_CRYPTO_PK_SIGN	failed to generate signature

5.41.1.4 iot_crypto_pk_verify()

```
iot_error_t iot_crypto_pk_verify (
    iot_crypto_pk_context_t * ctx,
```

```
unsigned char * input,
size_t ilen,
unsigned char * sig,
size_t slen )
```

Verify the signature.

Parameters

in	*ctx*	a pointer to a buffer containing crypto context
in	*input*	a pointer to a buffer to generate a signature
in	*ilen*	the size of buffer pointed by input in bytes
in	*sig*	a pointer to a buffer containing the signature
in	*slen*	the size of buffer pointed by sig in bytes

Return values

IOT_ERROR_NONE	the signature is sucessfully verified
IOT_ERROR_CRYPTO_PK_NULL_FUNC	verify function is not implemented
IOT_ERROR_CRYPTO_PK_PARSEKEY	failed to parse device certificate
IOT_ERROR_CRYPTO_PK_INVALID_CTX	ctx is null
IOT_ERROR_CRYPTO_PK_INVALID_ARG	info is null
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN	a length of private key is wrong
IOT_ERROR_CRYPTO_PK_VERIFY	the signature is mismatched

5.42 st-device-sdk-c/src/iot_jwt.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "iot_internal.h"
#include "iot_debug.h"
#include "iot_error.h"
#include "iot_crypto.h"
#include "iot_jwt.h"
#include "iot_util.h"
#include "cJSON.h"
```

Include dependency graph for iot_jwt.c:

Functions

- iot_error_t iot_jwt_create (char **token, const char *sn, iot_crypto_pk_info_t *pk_info)

  Create a JWT as proof of the device's identity.

5.42.1 Function Documentation

5.42.1.1 iot_jwt_create()

```
iot_error_t iot_jwt_create (
    char ** token,
    const char * sn,
    iot_crypto_pk_info_t * pk_info )
```

Create a JWT as proof of the device's identity.

This function makes a JWT string to connect to ST Cloud. Pass the JWT as password. Supported key types are RS256 and ED25519.

Parameters

out	*token*	a pointer of buffer to store a formatted and signed string
in	*sn*	device serial number as user name
in	*pk_info*	private key data

Return values

IOT_ERROR_NONE	JWT is sucessfully generated
IOT_ERROR_JWT_MALLOC	no more available heap memory
IOT_ERROR_JWT_CJSON	failed to make json

5.43 st-device-sdk-c/src/iot_main.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_debug.h"
#include "iot_bsp_wifi.h"
#include "iot_nv_data.h"
#include "iot_easysetup.h"
#include "iot_capability.h"
#include "iot_os_util.h"
#include "iot_util.h"
#include "iot_bsp_system.h"
```

Include dependency graph for iot_main.c:

Macros

- #define NEXT_STATE_TIMEOUT_MS (100000)
- #define EASYSETUP_TIMEOUT_MS (300000) /∗ 5 min ∗/

Functions

- IOT_CTX ∗ st_conn_init (unsigned char ∗onboarding_config, unsigned int onboarding_config_len, unsigned char ∗device_info, unsigned int device_info_len)

  st-iot-core initialize function
- int st_conn_start (IOT_CTX ∗iot_ctx, st_status_cb status_cb, iot_status_t maps, void ∗usr_data, iot_pin_t ∗pin_num)

  st-iot-core server connection function
- int st_conn_cleanup (IOT_CTX ∗iot_ctx, bool reboot)

  st-iot-core device clean-up function

5.43.1 Macro Definition Documentation

5.43.1.1 EASYSETUP_TIMEOUT_MS

```
#define EASYSETUP_TIMEOUT_MS (300000) /* 5 min */
```

5.43.1.2 NEXT_STATE_TIMEOUT_MS

```
#define NEXT_STATE_TIMEOUT_MS (100000)
```

5.43.2 Function Documentation

5.43.2.1 st_conn_cleanup()

```
int st_conn_cleanup (
    IOT_CTX * iot_ctx,
    bool reboot )
```

st-iot-core device clean-up function

This function cleans-up all DATA including provisioning & registered data

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*reboot*	boolean set true for auto-reboot of system, else false.

Returns

return `(0)` if it works successfully, non-zero for error case.

Here is the call graph for this function:

5.43.2.2 st_conn_init()

```
IOT_CTX* st_conn_init (
    unsigned char * onboarding_config,
    unsigned int onboarding_config_len,
    unsigned char * device_info,
    unsigned int device_info_len )
```

st-iot-core initialize function

This function initializes st-iot-core for target

Parameters

in	*onboarding_config*	starting pointer of onboarding_config.json contents
in	*onboarding_config_len*	size of onboarding_config.json contents
in	*device_info*	starting pointer of device_info.json contents
in	*device_info_len*	size of device_info.json contents

Returns

return IOT_CTX handle(a pointer) if it succeeds, or NULL if it fails

Here is the call graph for this function:

5.43.2.3 st_conn_start()

```
int st_conn_start (
    IOT_CTX * iot_ctx,
    st_status_cb status_cb,
    iot_status_t maps,
    void * usr_data,
    iot_pin_t * pin_num )
```

st-iot-core server connection function

This function tries to connect server

Parameters

in	*iot_ctx*	iot_context handle generated by iot_main_init()
in	*status_cb*	user callback function to receive status of st-iot-core
in	*maps*	status of st-iot-core interested to receive through status_cb
in	*usr_data*	user data(a pointer) to use in status_cb
in	*pin_num*	if PIN ownership validation type used, valid 8 digit pin should be set. otherwise set null.

Returns

return `(0)` if it works successfully, non-zero for error case.

5.44 st-device-sdk-c/src/iot_mqtt.c File Reference

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <cJSON.h>
#include "iot_main.h"
#include "iot_internal.h"
#include "iot_uuid.h"
#include "iot_util.h"
#include "iot_debug.h"
#include "iot_nv_data.h"
#include "iot_bsp_system.h"
```

Include dependency graph for iot_mqtt.c:

Functions

- iot_error_t iot_mqtt_connect (struct iot_mqtt_ctx ∗target_cli, char ∗username, char ∗sign_data)

  mqtt connect
- iot_error_t iot_mqtt_publish (struct iot_context ∗ctx, void ∗payload)

  pyblish mqtt message
- void _iot_mqtt_noti_sub_callback (MessageData ∗md, void ∗userData)
- void _iot_mqtt_cmd_sub_callback (MessageData ∗md, void ∗userData)
- iot_error_t iot_mqtt_subscribe (struct iot_mqtt_ctx ∗mqtt_ctx)

  subscribe mqtt topic
- iot_error_t iot_mqtt_unsubscribe (struct iot_mqtt_ctx ∗mqtt_ctx)

  unsubscribe mqtt topic
- void iot_mqtt_disconnect (struct iot_mqtt_ctx ∗target_cli)

  mqtt disconnect
- iot_error_t iot_mqtt_registration (struct iot_mqtt_ctx ∗mqtt_ctx)

  register device to ST server

5.44.1 Function Documentation

5.44.1.1 _iot_mqtt_cmd_sub_callback()

```
void _iot_mqtt_cmd_sub_callback (
    MessageData * md,
    void * userData )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.44.1.2 _iot_mqtt_noti_sub_callback()

```
void _iot_mqtt_noti_sub_callback (
    MessageData * md,
    void * userData )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.44.1.3 iot_mqtt_connect()

```
iot_error_t iot_mqtt_connect (
    struct iot_mqtt_ctx * target_cli,
    char * username,
    char * sign_data )
```

mqtt connect

this function connects ST server by mqtt

Parameters

in	*target_cli*	mqtt handling context
in	*username*	username to connect ST server based on mqtt protocol
in	*sign_data*	specific password that was jwt token-based to connect ST server

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.44.1.4 iot_mqtt_disconnect()

```
void iot_mqtt_disconnect (
    struct iot_mqtt_ctx * target_cli )
```

mqtt disconnect

this function is used to disconnect from server

Parameters

in	*target_cli*	mqtt handling context

Here is the call graph for this function:

5.44.1.5 iot_mqtt_publish()

```
iot_error_t iot_mqtt_publish (
    struct iot_context * ctx,
    void * payload )
```

pyblish mqtt message

this function is used to publish command & notification message

Parameters

in	*ctx*	iot-core context
in	*payload*	raw message sending to server

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.44.1.6 iot_mqtt_registration()

```
iot_error_t iot_mqtt_registration (
    struct iot_mqtt_ctx * mqtt_ctx )
```

register device to ST server

this function is used to handle the registration process

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.44.1.7 iot_mqtt_subscribe()

```
iot_error_t iot_mqtt_subscribe (
    struct iot_mqtt_ctx * mqtt_ctx )
```

subscribe mqtt topic

this function is used to subscribe command & notification topics

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.44.1.8 iot_mqtt_unsubscribe()

```
iot_error_t iot_mqtt_unsubscribe (
    struct iot_mqtt_ctx * mqtt_ctx )
```

unsubscribe mqtt topic

this function is used to unsubscribe command & notification topics

Parameters

in	*mqtt_ctx*	mqtt handling context

Return values

IOT_ERROR_NONE	success.

Here is the call graph for this function:

5.45 st-device-sdk-c/src/iot_nv_data.c File Reference

```
#include <string.h>
#include <stdlib.h>
#include "iot_nv_data.h"
#include "iot_bsp_fs.h"
#include "iot_bsp_nv_data.h"
#include "iot_debug.h"
#include "iot_util.h"
#include "certs/root_ca.h"
#include "iot_internal.h"
```

Include dependency graph for iot_nv_data.c:

Macros

- #define IOT_NVD_MAX_DATA_LEN (2048)
- #define IOT_NVD_MAX_ID_LEN (36)
- #define IOT_NVD_MAX_PW_LEN (64)
- #define IOT_NVD_MAX_BSSID_LEN (6)
- #define IOT_NVD_MAX_UID_LEN (128)

Functions

- iot_error_t _iot_nv_read_data (const char *path, char *data, unsigned int size)
- iot_error_t _iot_nv_write_data (const char *path, const char *data, unsigned int size)
- iot_error_t iot_nv_init (unsigned char *device_info, unsigned int device_info_len)

  Initialize a nv file-system.
- iot_error_t iot_nv_deinit ()

  Deinitialize a nv file-system.
- iot_error_t iot_nv_get_prov_data (struct iot_device_prov_data *prov_data)

  Get provisioning data from the nv file-system.
- iot_error_t iot_nv_set_prov_data (struct iot_device_prov_data *prov_data)

  Set provisioning data to the nv file-system.
- iot_error_t iot_nv_erase_prov_data ()

  Erase wifi/cloud provisioning data.
- iot_error_t iot_nv_get_wifi_prov_data (struct iot_wifi_prov_data *wifi_prov)

  Get wifi provisioning data from the nv file-system.
- iot_error_t iot_nv_set_wifi_prov_data (struct iot_wifi_prov_data *wifi_prov)

  Set wifi provisioning data to the nv file-system.
- iot_error_t iot_nv_get_cloud_prov_data (struct iot_cloud_prov_data *cloud_prov)

  Get cloud provisioning data from the nv file-system.
- iot_error_t iot_nv_set_cloud_prov_data (struct iot_cloud_prov_data *cloud_prov)

  Set cloud provisioning data to the nv file-system.
- iot_error_t iot_nv_get_private_key (char **key, unsigned int *len)

  Get a private key from the nv file-system.
- iot_error_t iot_nv_get_public_key (char **key, unsigned int *len)

  Get a public key from the nv file-system.
- iot_error_t iot_nv_get_root_certificate (char **cert, unsigned int *len)

  Get a root cert from the nv file-system.
- iot_error_t iot_nv_get_client_certificate (char **cert, unsigned int *len)

  Get a client cert from the nv file-system.
- iot_error_t iot_nv_get_device_id (char **device_id, unsigned int *len)

  Get a device id from the nv file-system.
- iot_error_t iot_nv_set_device_id (const char *device_id)

  Set a device id to the nv file-system.
- iot_error_t iot_nv_get_serial_number (char **sn, unsigned int *len)

  Get a serial number from the nv file-system.
- iot_error_t iot_nv_erase (iot_nvd_t nv_type)

  Erase a nv data.

5.45.1 Macro Definition Documentation

5.45.1.1 IOT_NVD_MAX_BSSID_LEN `#define IOT_NVD_MAX_BSSID_LEN (6)`

5.45.1.2 IOT_NVD_MAX_DATA_LEN `#define IOT_NVD_MAX_DATA_LEN (2048)`

5.45.1.3 IOT_NVD_MAX_ID_LEN `#define IOT_NVD_MAX_ID_LEN (36)`

5.45.1.4 IOT_NVD_MAX_PW_LEN `#define IOT_NVD_MAX_PW_LEN (64)`

5.45.1.5 IOT_NVD_MAX_UID_LEN `#define IOT_NVD_MAX_UID_LEN (128)`

5.45.2 Function Documentation

5.45.2.1 _iot_nv_read_data()

```
iot_error_t _iot_nv_read_data (
    const char * path,
    char * data,
    unsigned int size )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.2 _iot_nv_write_data()

```
iot_error_t _iot_nv_write_data (
    const char * path,
    const char * data,
    unsigned int size )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.3 iot_nv_deinit()

```
iot_error_t iot_nv_deinit ( )
```

Deinitialize a nv file-system.

Return values

IOT_ERROR_NONE	NV file-system deinit successful.
IOT_ERROR_DEINIT_FAIL	NV file-system deinit failed.

See also

iot_bsp_fs_deinit

Here is the call graph for this function:

5.45.2.4 iot_nv_erase()

```
iot_error_t iot_nv_erase (
    iot_nvd_t nv_type )
```

Erase a nv data.

This function erase the nv data completely in File-system.

Parameters

in	*nv_type*	The type of nv data to erase.

Return values

IOT_ERROR_NONE	NV data erase successful.
IOT_ERROR_INVALID_ARGS	Invalid nv type.
IOT_ERROR_NV_DATA_ERROR	NV data erase failed.
IOT_ERROR_NV_DATA_NOT_EXIST	NV data does not exist.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.5 iot_nv_erase_prov_data() `iot_error_t iot_nv_erase_prov_data ( )`

Erase wifi/cloud provisioning data.

This function erases the wifi/cloud provisioning data in File-system.

Return values

IOT_ERROR_NONE	NV data erase successful.
IOT_ERROR_NV_DATA_ERROR	NV data erase failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.6 iot_nv_get_client_certificate()

```
iot_error_t iot_nv_get_client_certificate (
    char ** cert,
    unsigned int * len )
```

Get a client cert from the nv file-system.

Parameters

out	*cert*	A pointer to data array to store the client cert from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.45.2.7 iot_nv_get_cloud_prov_data()

```
iot_error_t iot_nv_get_cloud_prov_data (
    struct iot_cloud_prov_data * cloud_prov )
```

Get cloud provisioning data from the nv file-system.

Parameters

out	*cloud_prov*	A pointer to data structure to store the cloud provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.8 iot_nv_get_device_id()

```
iot_error_t iot_nv_get_device_id (
    char ** device_id,
    unsigned int * len )
```

Get a device id from the nv file-system.

Parameters

out	*device_id*	A pointer to data array to store the device id from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.45.2.9 iot_nv_get_private_key()

```
iot_error_t iot_nv_get_private_key (
    char ** key,
    unsigned int * len )
```

Get a private key from the nv file-system.

Parameters

out	*key*	A pointer to data array to store the private key from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.45.2.10 iot_nv_get_prov_data()

```
iot_error_t iot_nv_get_prov_data (
    struct iot_device_prov_data * prov_data )
```

Get provisioning data from the nv file-system.

Parameters

out	*prov_data*	A pointer to data structure to store the provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

See also

iot_nv_get_wifi_prov_data

iot_nv_get_cloud_prov_data

Here is the call graph for this function:

5.45.2.11 iot_nv_get_public_key()

```
iot_error_t iot_nv_get_public_key (
    char ** key,
    unsigned int * len )
```

Get a public key from the nv file-system.

Parameters

out	*key*	A pointer to data array to store the public key from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

5.45.2.12 iot_nv_get_root_certificate()

```
iot_error_t iot_nv_get_root_certificate (
    char ** cert,
    unsigned int * len )
```

Get a root cert from the nv file-system.

Parameters

out	*cert*	A pointer to data array to store the root cert from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

5.45.2.13 iot_nv_get_serial_number()

```
iot_error_t iot_nv_get_serial_number (
    char ** sn,
    unsigned int * len )
```

Get a serial number from the nv file-system.

Parameters

out	*sn*	A pointer to data array to store the serial number from the nv file-system.
out	*len*	The length of the nv data.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.14 iot_nv_get_wifi_prov_data()

```
iot_error_t iot_nv_get_wifi_prov_data (
    struct iot_wifi_prov_data * wifi_prov )
```

Get wifi provisioning data from the nv file-system.

Parameters

out	*wifi_prov*	A pointer to data structure to store the wifi provisioning data from the nv file-system.

Return values

IOT_ERROR_NONE	Get nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Get nv data failed.

Warning

The caller is always responsible to free the allocated pointer after using the data.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.15 iot_nv_init()

```
iot_error_t iot_nv_init (
    unsigned char * device_info,
    unsigned int device_info_len )
```

Initialize a nv file-system.

This function initializes the nv file-system of the device. You must call this function before using nv management function.

Return values

IOT_ERROR_NONE	NV file-system init successful.
IOT_ERROR_INIT_F↩ AIL	NV file-system init failed.

See also

iot_bsp_fs_init

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.16 iot_nv_set_cloud_prov_data()

```
iot_error_t iot_nv_set_cloud_prov_data (
    struct iot_cloud_prov_data * cloud_prov )
```

Set cloud provisioning data to the nv file-system.

Parameters

in	*cloud_prov*	A pointer to cloud provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.45.2.17 iot_nv_set_device_id()

```
iot_error_t iot_nv_set_device_id (
    const char * device_id )
```

Set a device id to the nv file-system.

Parameters

in	*device↩ _id*	The string of device id from MQTT Server.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

5.45.2.18 iot_nv_set_prov_data()

```
iot_error_t iot_nv_set_prov_data (
    struct iot_device_prov_data * prov_data )
```

Set provisioning data to the nv file-system.

Parameters

in	*prov_data*	A pointer to provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

See also

iot_nv_set_wifi_prov_data
iot_nv_set_cloud_prov_data

Here is the call graph for this function:

5.45.2.19 iot_nv_set_wifi_prov_data()

```
iot_error_t iot_nv_set_wifi_prov_data (
    struct iot_wifi_prov_data * wifi_prov )
```

Set wifi provisioning data to the nv file-system.

Parameters

in	*wifi_prov*	A pointer to wifi provisioning data structure.

Return values

IOT_ERROR_NONE	Set nv data successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.
IOT_ERROR_NV_DATA_ERROR	Set nv data failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.46 st-device-sdk-c/src/iot_util.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "iot_main.h"
#include "iot_util.h"
#include "iot_debug.h"
#include "iot_bsp_random.h"
```

Include dependency graph for iot_util.c:

Functions

- iot_error_t iot_util_convert_str_uuid (const char ∗str, struct iot_uuid ∗uuid)

  uuid type string to iot_uuid struct converting function for st-iot-core
- iot_error_t iot_util_convert_uuid_str (struct iot_uuid ∗uuid, char ∗str, int max_sz)

  iot_uuid struct based value to uuid type string converting function for st-iot-core
- iot_error_t iot_util_get_random_uuid (struct iot_uuid ∗uuid)

  To get random uuid based on iot_uuid struct.
- iot_error_t iot_util_convert_str_mac (char ∗str, struct iot_mac ∗mac)

  To convert WIFI mac string into iot_mac struct value.
- iot_error_t iot_util_convert_mac_str (struct iot_mac ∗mac, char ∗str, int max_sz)

  To convert iot_mac value intto WIFI mac string.
- uint16_t iot_util_convert_channel_freq (uint8_t channel)

  To convert Wi-Fi channel into frequency value.
- iot_error_t iot_util_url_parse (char ∗url, url_parse_t ∗output)

  parse url with protocol, domain, port number parts for st-iot-core

5.46.1 Function Documentation

5.46.1.1 iot_util_convert_channel_freq() `uint16_t iot_util_convert_channel_freq ( uint8_t channel )`

To convert Wi-Fi channel into frequency value.

This function tries to convert from the channel to frequency

Parameters

in	*Wi-Fi*	channel

Returns

Wi-Fi frequency

Here is the caller graph for this function:

5.46.1.2 iot_util_convert_mac_str()

```
iot_error_t iot_util_convert_mac_str (
    struct iot_mac * mac,
    char * str,
    int max_sz )
```

To convert iot_mac value intto WIFI mac string.

This function tries to convert from the iot_mac struct value to string

Parameters

in	*mac*	converting wanted iot_mac struct value pointer
in	*str*	allocated memory pointer for converted WIFI mac string
in	*max_sz*	max size of allocated memory pointer

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_INVALID_ARGS	invalid arguments

5.46.1.3 iot_util_convert_str_mac()

```
iot_error_t iot_util_convert_str_mac (
    char * str,
    struct iot_mac * mac )
```

To convert WIFI mac string into iot_mac struct value.

This function tries to convert from the string to iot_mac struct value

Parameters

in	*str*	WIFI mac string pointer such as 'xx:xx:xx:xx:xx:xx'
in	*mac*	allocated iot_mac struct pointer to get iot_mac value from str

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_INVALID_ARGS	invalid arguments

5.46.1.4 iot_util_convert_str_uuid()

```
iot_error_t iot_util_convert_str_uuid (
    const char * str,
    struct iot_uuid * uuid )
```

uuid type string to iot_uuid struct converting function for st-iot-core

This function tries to convert from uuid type string to iot_uuid struct

Parameters

in	*str*	uuid type string pointer such as 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
in	*uuid*	allocated iot_uuid struct pointer to get iot_uuid value from str

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the caller graph for this function:

5.46.1.5 iot_util_convert_uuid_str()

```
iot_error_t iot_util_convert_uuid_str (
    struct iot_uuid * uuid,
    char * str,
    int max_sz )
```

iot_uuid struct based value to uuid type string converting function for st-iot-core

This function tries to convert from iot_uuid struct based value to uuid type string

Parameters

in	*uuid*	converting wanted iot_uuid struct value pointer
in	*str*	allocated memory pointer for converted uuid type string
in	*max_sz*	max size of allocated memory pointer

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the caller graph for this function:

5.46.1.6 iot_util_get_random_uuid()

```
iot_error_t iot_util_get_random_uuid (
    struct iot_uuid * uuid )
```

To get random uuid based on iot_uuid struct.

This function tries to make random iot_uuid values

Parameters

in	*uuid*	allocated iot_uuid struct pointer to get random iot_uuid value

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

Here is the call graph for this function:

5.46.1.7 iot_util_url_parse()

```
iot_error_t iot_util_url_parse (
    char * url,
    url_parse_t * output )
```

parse url with protocol, domain, port number parts for st-iot-core

This function parse give url protocol, domain, port number parts

Parameters

in	*url*	null-terminated url string like "https://example.sample.com:1234"
out	*output*	parsed output with url_parse_t type

Returns

return IOT_ERROR_NONE on success, or iot_error_t errors if it fails

5.47 st-device-sdk-c/src/iot_uuid.c File Reference

```
#include <string.h>
#include <sys/time.h>
#include "iot_main.h"
#include "iot_crypto.h"
#include "iot_uuid.h"
#include "iot_bsp_random.h"
#include "iot_debug.h"
```

Include dependency graph for iot_uuid.c:

Functions

- iot_error_t iot_random_uuid_from_mac (struct iot_uuid ∗uuid)

  Change mac to random uuid This function changes mac to random uuid.
- iot_error_t iot_uuid_from_mac (struct iot_uuid ∗uuid)

  Change mac to uuid This function changes mac to the uuid.

5.47.1 Function Documentation

5.47.1.1 iot_random_uuid_from_mac()

```
iot_error_t iot_random_uuid_from_mac (
    struct iot_uuid * uuid )
```

Change mac to random uuid This function changes mac to random uuid.

Parameters

out	*uuid*	created random uuid

Returns

IOT_ERROR_NONE on success

Here is the call graph for this function:

Here is the caller graph for this function:

5.47.1.2 iot_uuid_from_mac()

```
iot_error_t iot_uuid_from_mac (
    struct iot_uuid * uuid )
```

Change mac to uuid This function changes mac to the uuid.

Parameters

out	*uuid*	created uuid

Returns

IOT_ERROR_NONE on success

Here is the call graph for this function:

5.48 st-device-sdk-c/src/mqtt/client/iot_mqtt_client.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "iot_debug.h"
#include "iot_mqtt_client.h"
```

Include dependency graph for iot_mqtt_client.c:

Functions

- bool MQTTClientInit (MQTTClient ∗c, iot_net_socket ∗network, unsigned int command_timeout_ms, unsigned char ∗sendbuf, size_t sendbuf_size, unsigned char ∗readbuf, size_t readbuf_size)
- int deliverMessage (MQTTClient ∗c, MQTTString ∗topicName, MQTTMessage ∗message)
- int keepalive (MQTTClient ∗c)
- void MQTTCleanSession (MQTTClient ∗c)
- void MQTTCloseSession (MQTTClient ∗c)

- int cycle (MQTTClient ∗c, iot_os_timer timer)
- int MQTTYield (MQTTClient ∗c, int timeout_ms)
- int waitfor (MQTTClient ∗c, int packet_type, iot_os_timer timer)
- int MQTTConnectWithResults (MQTTClient ∗c, MQTTPacket_connectData ∗options, MQTTConnackData ∗data)
- int MQTTConnect (MQTTClient ∗c, MQTTPacket_connectData ∗options)
- int MQTTSetMessageHandler (MQTTClient ∗c, const char ∗topicFilter, messageHandler messageHandler, void ∗userData)
- int MQTTSubscribeWithResults (MQTTClient ∗c, const char ∗topicFilter, enum QoS qos, messageHandler messageHandler, MQTTSubackData ∗data, void ∗userData)
- int MQTTSubscribe (MQTTClient ∗c, const char ∗topicFilter, enum QoS qos, messageHandler messageHandler, void ∗userData)
- int MQTTUnsubscribe (MQTTClient ∗c, const char ∗topicFilter)
- int MQTTPublish (MQTTClient ∗c, const char ∗topicName, MQTTMessage ∗message)
- int MQTTDisconnect (MQTTClient ∗c)

5.48.1 Function Documentation

5.48.1.1 cycle()

```
int cycle (
    MQTTClient * c,
    iot_os_timer timer )
```

Here is the caller graph for this function:

5.48.1.2 deliverMessage()

```
int deliverMessage (
    MQTTClient * c,
    MQTTString * topicName,
    MQTTMessage * message )
```

Here is the call graph for this function:

5.48.1.3 keepalive()

```
int keepalive (
    MQTTClient * c )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.4 MQTTCleanSession()

```
void MQTTCleanSession (
    MQTTClient * c )
```

Here is the caller graph for this function:

5.48.1.5 MQTTClientInit()

```
bool MQTTClientInit (
    MQTTClient * client,
    iot_net_socket * network,
    unsigned int command_timeout_ms,
    unsigned char * sendbuf,
    size_t sendbuf_size,
    unsigned char * readbuf,
    size_t readbuf_size )
```

Create an MQTT client object

Parameters

client	
network	
command_timeout_ms	

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.6 MQTTCloseSession()

```
void MQTTCloseSession (
    MQTTClient * c )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.7 MQTTConnect()

```
int MQTTConnect (
    MQTTClient * client,
    MQTTPacket_connectData * options )
```

MQTT Connect - send an MQTT connect packet down the network and wait for a Connack The nework object must be connected to the network endpoint before calling this

Parameters

options	- connect options

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.8 MQTTConnectWithResults()

```
int MQTTConnectWithResults (
    MQTTClient * client,
    MQTTPacket_connectData * options,
    MQTTConnackData * data )
```

MQTT Connect - send an MQTT connect packet down the network and wait for a Connack The nework object must be connected to the network endpoint before calling this

Parameters

options	- connect options

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.9 MQTTDisconnect()

```
int MQTTDisconnect (
    MQTTClient * client )
```

MQTT Disconnect - send an MQTT disconnect packet and close the connection

Parameters

client	- the client object to use

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.10 MQTTPublish()

```
int MQTTPublish (
    MQTTClient * client,
    const char * ,
    MQTTMessage *  )
```

MQTT Publish - send an MQTT publish packet and wait for all acks to complete for all QoSs

Parameters

client	- the client object to use
topic	- the topic to publish to
message	- the message to send

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.11 MQTTSetMessageHandler()

```
int MQTTSetMessageHandler (
    MQTTClient * c,
    const char * topicFilter,
    messageHandler messageHandler,
    void * userData )
```

MQTT SetMessageHandler - set or remove a per topic message handler

Parameters

client	- the client object to use
topicFilter	- the topic filter set the message handler for
messageHandler	- pointer to the message handler function or NULL to remove

Returns

success code

5.48.1.12 MQTTSubscribe()

```
int MQTTSubscribe (
    MQTTClient * client,
    const char * topicFilter,
    enum QoS,
    messageHandler ,
    void * userData )
```

MQTT Subscribe - send an MQTT subscribe packet and wait for suback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to subscribe to
message	- the message to send

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.13 MQTTSubscribeWithResults()

```
int MQTTSubscribeWithResults (
    MQTTClient * client,
    const char * topicFilter,
    enum QoS,
    messageHandler ,
    MQTTSubackData * data,
    void * userData )
```

MQTT Subscribe - send an MQTT subscribe packet and wait for suback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to subscribe to
message	- the message to send
data	- suback granted QoS returned

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.14 MQTTUnsubscribe()

```
int MQTTUnsubscribe (
    MQTTClient * client,
    const char * topicFilter )
```

MQTT Subscribe - send an MQTT unsubscribe packet and wait for unsuback before returning.

Parameters

client	- the client object to use
topicFilter	- the topic filter to unsubscribe from

Returns

success code

Here is the call graph for this function:

Here is the caller graph for this function:

5.48.1.15 MQTTYield()

```
int MQTTYield (
    MQTTClient * client,
    int time )
```

MQTT Yield - MQTT background

Parameters

client	- the client object to use
time	- the time, in milliseconds, to yield for

Returns

success code

Here is the call graph for this function:

5.48.1.16 waitfor()

```
int waitfor (
    MQTTClient * c,
    int packet_type,
    iot_os_timer timer )
```

Here is the call graph for this function:

5.49 st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_client.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_connect_client.c:

Functions

- int MQTTSerialize_connectLength (MQTTPacket_connectData ∗options)
- int MQTTSerialize_connect (unsigned char ∗buf, int buflen, MQTTPacket_connectData ∗options)
- int MQTTSerialize_connect_size (MQTTPacket_connectData ∗options)
- int MQTTDeserialize_connack (unsigned char ∗sessionPresent, unsigned char ∗connack_rc, unsigned char ∗buf, int buflen)
- int MQTTSerialize_zero (unsigned char ∗buf, int buflen, unsigned char packettype)
- int MQTTSerialize_disconnect (unsigned char ∗buf, int buflen)
- int MQTTSerialize_disconnect_size ()
- int MQTTSerialize_pingreq (unsigned char ∗buf, int buflen)
- int MQTTSerialize_pingreq_size ()

5.49.1 Function Documentation

5.49.1.1 MQTTDeserialize_connack()

```
int MQTTDeserialize_connack (
    unsigned char * sessionPresent,
    unsigned char * connack_rc,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into connack data - return code

Parameters

sessionPresent	the session present flag returned (only for MQTT 3.1.1)
connack_rc	returned integer value of the connack return code
buf	the raw buffer data, of the correct length determined by the remaining length field
len	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.49.1.2 MQTTSerialize_connect()

```
int MQTTSerialize_connect (
    unsigned char * buf,
    int buflen,
    MQTTPacket_connectData * options )
```

Serializes the connect options into the buffer.

Parameters

buf	the buffer into which the packet will be serialized
len	the length in bytes of the supplied buffer
options	the options to be used to build the connect packet

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.49.1.3 MQTTSerialize_connect_size() `int MQTTSerialize_connect_size ( MQTTPacket_connectData * options )`

Here is the call graph for this function:

Here is the caller graph for this function:

5.49.1.4 MQTTSerialize_connectLength() `int MQTTSerialize_connectLength ( MQTTPacket_connectData * options )`

Determines the length of the MQTT connect packet that would be produced using the supplied connect options.

Parameters

options	the options to be used to build the connect packet

Returns

the length of buffer needed to contain the serialized version of the packet

Here is the caller graph for this function:

5.49.1.5 MQTTSerialize_disconnect()

```
int MQTTSerialize_disconnect (
    unsigned char * buf,
    int buflen )
```

Serializes a disconnect packet into the supplied buffer, ready for writing to a socket

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer, to avoid overruns

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.49.1.6 MQTTSerialize_disconnect_size() `int MQTTSerialize_disconnect_size ( )`

Here is the caller graph for this function:

5.49.1.7 MQTTSerialize_pingreq()

```
int MQTTSerialize_pingreq (
    unsigned char * buf,
    int buflen )
```

Serializes a disconnect packet into the supplied buffer, ready for writing to a socket

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer, to avoid overruns

Returns

serialized length, or error if 0

Here is the caller graph for this function:

5.49.1.8 MQTTSerialize_pingreq_size() `int MQTTSerialize_pingreq_size ( )`

Here is the caller graph for this function:

5.49.1.9 MQTTSerialize_zero()

```
int MQTTSerialize_zero (
    unsigned char * buf,
    int buflen,
    unsigned char packettype )
```

Serializes a 0-length packet into the supplied buffer, ready for writing to a socket

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer, to avoid overruns
packettype	the message type

Returns

serialized length, or error if 0

5.50 st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_server.c File Reference

```
#include "iot_mqtt_stacktrace.h"
#include "iot_mqtt_packet.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_connect_server.c:

Macros

- #define min(a, b) ((a < b) ? a : b)

Functions

- int MQTTPacket_checkVersion (MQTTString *protocol, int version)
- int MQTTDeserialize_connect (MQTTPacket_connectData *data, unsigned char *buf, int len)
- int MQTTSerialize_connack (unsigned char *buf, int buflen, unsigned char connack_rc, unsigned char sessionPresent)

5.50.1 Macro Definition Documentation

5.50.1.1 min

```
#define min(
    a,
    b ) ((a < b) ? a : b)
```

5.50.2 Function Documentation

5.50.2.1 MQTTDeserialize_connect()

```
int MQTTDeserialize_connect (
    MQTTPacket_connectData * data,
    unsigned char * buf,
    int len )
```

Deserializes the supplied (wire) buffer into connect data structure

Parameters

data	the connect data structure to be filled out
buf	the raw buffer data, of the correct length determined by the remaining length field
len	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.50.2.2 MQTTPacket_checkVersion()

```
int MQTTPacket_checkVersion (
    MQTTString * protocol,
    int version )
```

Validates MQTT protocol name and version combinations

Parameters

protocol	the MQTT protocol name as an MQTTString
version	the MQTT protocol version number, as in the connect packet

Returns

correct MQTT combination? 1 is true, 0 is false

5.50.2.3 MQTTSerialize_connack()

```
int MQTTSerialize_connack (
    unsigned char * buf,
    int buflen,
    unsigned char connack_rc,
    unsigned char sessionPresent )
```

Serializes the connack packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
connack_rc	the integer connack return code to be used
sessionPresent	the MQTT 3.1.1 sessionPresent flag

Returns

serialized length, or error if 0

5.51 st-device-sdk-c/src/mqtt/packet/iot_mqtt_deserialize_publish.c File Reference

```
#include "iot_mqtt_stacktrace.h"
#include "iot_mqtt_packet.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_deserialize_publish.c:

Macros

- #define min(a, b) ((a < b) ? 1 : 0)

Functions

- int MQTTDeserialize_publish (unsigned char *dup, int *qos, unsigned char *retained, unsigned short *packetid, MQTTString *topicName, unsigned char **payload, int *payloadlen, unsigned char *buf, int buflen)
- int MQTTDeserialize_ack (unsigned char *packettype, unsigned char *dup, unsigned short *packetid, unsigned char *buf, int buflen)

5.51.1 Macro Definition Documentation

5.51.1.1 min

```
#define min(
    a,
    b ) ((a < b) ? 1 : 0)
```

5.51.2 Function Documentation

5.51.2.1 MQTTDeserialize_ack()

```
int MQTTDeserialize_ack (
    unsigned char * packettype,
    unsigned char * dup,
    unsigned short * packetid,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into an ack

Parameters

packettype	returned integer - the MQTT packet type
dup	returned integer - the MQTT dup flag
packetid	returned integer - the MQTT packet identifier
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.51.2.2 MQTTDeserialize_publish()

```
int MQTTDeserialize_publish (
    unsigned char * dup,
    int * qos,
    unsigned char * retained,
    unsigned short * packetid,
    MQTTString * topicName,
    unsigned char ** payload,
    int * payloadlen,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into publish data

Parameters

dup	returned integer - the MQTT dup flag
qos	returned integer - the MQTT QoS value
retained	returned integer - the MQTT retained flag
packetid	returned integer - the MQTT packet identifier
topicName	returned MQTTString - the MQTT topic in the publish
payload	returned byte buffer - the MQTT publish payload
payloadlen	returned integer - the length of the MQTT payload
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success

5.52 st-device-sdk-c/src/mqtt/packet/iot_mqtt_format.c File Reference

```
#include "iot_mqtt_stacktrace.h"
#include "iot_mqtt_packet.h"
```

```
#include <string.h>
```

Include dependency graph for iot_mqtt_format.c:

Functions

- const char ∗ MQTTPacket_getName (unsigned short packetid)
- int MQTTStringFormat_connect (char ∗strbuf, int strbuflen, MQTTPacket_connectData ∗data)
- int MQTTStringFormat_connack (char ∗strbuf, int strbuflen, unsigned char connack_rc, unsigned char sessionPresent)
- int MQTTStringFormat_publish (char ∗strbuf, int strbuflen, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, unsigned char ∗payload, int payloadlen)
- int MQTTStringFormat_ack (char ∗strbuf, int strbuflen, unsigned char packettype, unsigned char dup, unsigned short packetid)
- int MQTTStringFormat_subscribe (char ∗strbuf, int strbuflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ], int requestedQoSs[ ])
- int MQTTStringFormat_suback (char ∗strbuf, int strbuflen, unsigned short packetid, int count, int ∗granted↩ QoSs)
- int MQTTStringFormat_unsubscribe (char ∗strbuf, int strbuflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ])

Variables

- const char ∗ MQTTPacket_names [ ]

5.52.1 Function Documentation

5.52.1.1 MQTTPacket_getName()

```
const char* MQTTPacket_getName (
    unsigned short packetid )
```

5.52.1.2 MQTTStringFormat_ack()

```
int MQTTStringFormat_ack (
    char * strbuf,
    int strbuflen,
    unsigned char packettype,
    unsigned char dup,
    unsigned short packetid )
```

5.52.1.3 MQTTStringFormat_connack()

```
int MQTTStringFormat_connack (
    char * strbuf,
    int strbuflen,
    unsigned char connack_rc,
    unsigned char sessionPresent )
```

5.52.1.4 MQTTStringFormat_connect()

```
int MQTTStringFormat_connect (
    char * strbuf,
    int strbuflen,
    MQTTPacket_connectData * data )
```

5.52.1.5 MQTTStringFormat_publish()

```
int MQTTStringFormat_publish (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    int qos,
    unsigned char retained,
    unsigned short packetid,
    MQTTString topicName,
    unsigned char * payload,
    int payloadlen )
```

5.52.1.6 MQTTStringFormat_suback()

```
int MQTTStringFormat_suback (
    char * strbuf,
    int strbuflen,
    unsigned short packetid,
    int count,
    int * grantedQoSs )
```

5.52.1.7 MQTTStringFormat_subscribe()

```
int MQTTStringFormat_subscribe (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[],
    int requestedQoSs[] )
```

5.52.1.8 MQTTStringFormat_unsubscribe()

```
int MQTTStringFormat_unsubscribe (
    char * strbuf,
    int strbuflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[] )
```

5.52.2 Variable Documentation

5.52.2.1 MQTTPacket_names

```
const char* MQTTPacket_names[]
```

Initial value:

```
=
{
    "RESERVED", "CONNECT", "CONNACK", "PUBLISH", "PUBACK", "PUBREC", "PUBREL",
    "PUBCOMP", "SUBSCRIBE", "SUBACK", "UNSUBSCRIBE", "UNSUBACK",
    "PINGREQ", "PINGRESP", "DISCONNECT"
}
```

5.53 st-device-sdk-c/src/mqtt/packet/iot_mqtt_packet.c File Reference

```
#include "iot_mqtt_stacktrace.h"
#include "iot_mqtt_packet.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_packet.c:

Macros

- #define MAX_NO_OF_REMAINING_LENGTH_BYTES 4

Functions

- int MQTTPacket_encode (unsigned char ∗buf, int length)
- int MQTTPacket_decode (int(∗getcharfn)(unsigned char ∗, int), int ∗value)
- int MQTTPacket_len (int rem_len)
- int bufchar (unsigned char ∗c, int count)
- int MQTTPacket_decodeBuf (unsigned char ∗buf, int ∗value)
- int readInt (unsigned char ∗∗pptr)
- char readChar (unsigned char ∗∗pptr)
- void writeChar (unsigned char ∗∗pptr, char c)
- void writeInt (unsigned char ∗∗pptr, int anInt)
- void writeCString (unsigned char ∗∗pptr, const char ∗string)
- int getLenStringLen (char ∗ptr)
- void writeMQTTString (unsigned char ∗∗pptr, MQTTString mqttstring)
- int readMQTTLenString (MQTTString ∗mqttstring, unsigned char ∗∗pptr, unsigned char ∗enddata)
- int MQTTstrlen (MQTTString mqttstring)
- int MQTTPacket_equals (MQTTString ∗a, char ∗bptr)
- int MQTTPacket_read (unsigned char ∗buf, int buflen, int(∗getfn)(unsigned char ∗, int))
- int MQTTPacket_readnb (unsigned char ∗buf, int buflen, MQTTTransport ∗trp)
- const char ∗ MQTTPacket_msgTypesToString (enum msgTypes msgType)

5.53.1 Macro Definition Documentation

5.53.1.1 MAX_NO_OF_REMAINING_LENGTH_BYTES `#define MAX_NO_OF_REMAINING_LENGTH_BYTES 4`

5.53.2 Function Documentation

5.53.2.1 bufchar()

```
int bufchar (
    unsigned char * c,
    int count )
```

5.53.2.2 getLenStringLen()

```
int getLenStringLen (
    char * ptr )
```

5.53.2.3 MQTTPacket_decode()

```
int MQTTPacket_decode (
    int(*)(unsigned char *, int) getcharfn,
    int * value )
```

Decodes the message length according to the MQTT algorithm

Parameters

getcharfn	pointer to function to read the next character from the data source
value	the decoded length returned

Returns

the number of bytes read from the socket

5.53.2.4 MQTTPacket_decodeBuf()

```
int MQTTPacket_decodeBuf (
    unsigned char * buf,
    int * value )
```

5.53.2.5 MQTTPacket_encode()

```
int MQTTPacket_encode (
    unsigned char * buf,
    int length )
```

Encodes the message length according to the MQTT algorithm

Parameters

buf	the buffer into which the encoded data is written
length	the length to be encoded

Returns

the number of bytes written to buffer

Here is the caller graph for this function:

5.53.2.6 MQTTPacket_equals()

```
int MQTTPacket_equals (
    MQTTString * a,
    char * bptr )
```

Compares an MQTTString to a C string

Parameters

a	the MQTTString to compare
bptr	the C string to compare

Returns

boolean - equal or not

Here is the caller graph for this function:

5.53.2.7 MQTTPacket_len()

```
int MQTTPacket_len (
    int rem_len )
```

Here is the caller graph for this function:

5.53.2.8 MQTTPacket_msgTypesToString()

```
const char* MQTTPacket_msgTypesToString (
    enum msgTypes msgType )
```

5.53.2.9 MQTTPacket_read()

```
int MQTTPacket_read (
    unsigned char * buf,
    int buflen,
    int(*)(unsigned char *, int) getfn )
```

Helper function to read packet data from some source into a buffer

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
getfn	pointer to a function which will read any number of bytes from the needed source

Returns

integer MQTT packet type, or -1 on error

Note

the whole message must fit into the caller's buffer

5.53.2.10 MQTTPacket_readnb()

```
int MQTTPacket_readnb (
    unsigned char * buf,
    int buflen,
    MQTTTransport * trp )
```

Helper function to read packet data from some source into a buffer, non-blocking

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
trp	pointer to a transport structure holding what is needed to solve getting data from it

Returns

integer MQTT packet type, 0 for call again, or -1 on error

Note

the whole message must fit into the caller's buffer

5.53.2.11 MQTTstrlen()

```
int MQTTstrlen (
    MQTTString mqttstring )
```

Return the length of the MQTTstring - C string if there is one, otherwise the length delimited string

Parameters

mqttstring	the string to return the length of

Returns

the length of the string

5.53.2.12 readChar()

```
char readChar (
    unsigned char ** pptr )
```

Reads one character from the input buffer.

Parameters

pptr	pointer to the input buffer - incremented by the number of bytes used & returned

Returns

the character read

5.53.2.13 readInt()

```
int readInt (
    unsigned char ** pptr )
```

Calculates an integer from two bytes read from the input buffer

Parameters

pptr	pointer to the input buffer - incremented by the number of bytes used & returned

Returns

the integer value calculated

5.53.2.14 readMQTTLenString()

```
int readMQTTLenString (
    MQTTString * mqttstring,
    unsigned char ** pptr,
    unsigned char * enddata )
```

Parameters

mqttstring	the MQTTString structure into which the data is to be read
pptr	pointer to the output buffer - incremented by the number of bytes used & returned
enddata	pointer to the end of the data: do not read beyond

Returns

1 if successful, 0 if not

5.53.2.15 writeChar()

```
void writeChar (
    unsigned char ** pptr,
    char c )
```

Writes one character to an output buffer.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
c	the character to write

Here is the caller graph for this function:

5.53.2.16 writeCString()

```
void writeCString (
    unsigned char ** pptr,
    const char * string )
```

Writes a "UTF" string to an output buffer. Converts C string to length-delimited.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
string	the C string to write

Here is the caller graph for this function:

5.53.2.17 writeInt()

```
void writeInt (
    unsigned char ** pptr,
    int anInt )
```

Writes an integer as 2 bytes to an output buffer.

Parameters

pptr	pointer to the output buffer - incremented by the number of bytes used & returned
anInt	the integer to write

Here is the caller graph for this function:

5.53.2.18 writeMQTTString()

```
void writeMQTTString (
    unsigned char ** pptr,
    MQTTString mqttstring )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.54 st-device-sdk-c/src/mqtt/packet/iot_mqtt_serialize_publish.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_serialize_publish.c:

Functions

- int MQTTSerialize_publishLength (int qos, MQTTString topicName, int payloadlen)

- int MQTTSerialize_publish (unsigned char ∗buf, int buflen, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, unsigned char ∗payload, int payloadlen)
- int MQTTSerialize_publish_size (int qos, MQTTString topicName, int payloadlen)
- int MQTTSerialize_publish_header (unsigned char ∗buf, unsigned char dup, int qos, unsigned char retained, unsigned short packetid, MQTTString topicName, int payloadlen)
- int MQTTSerialize_ack (unsigned char ∗buf, int buflen, unsigned char packettype, unsigned char dup, unsigned short packetid)
- int MQTTSerialize_ack_size ()
- int MQTTSerialize_puback (unsigned char ∗buf, int buflen, unsigned short packetid)
- int MQTTSerialize_pubrel (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid)
- int MQTTSerialize_pubcomp (unsigned char ∗buf, int buflen, unsigned short packetid)

5.54.1 Function Documentation

5.54.1.1 MQTTSerialize_ack()

```
int MQTTSerialize_ack (
    unsigned char * buf,
    int buflen,
    unsigned char packettype,
    unsigned char dup,
    unsigned short packetid )
```

Serializes the ack packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
type	the MQTT packet type
dup	the MQTT dup flag
packetid	the MQTT packet identifier

Returns

serialized length, or error if 0

5.54.1.2 MQTTSerialize_ack_size()

```
int MQTTSerialize_ack_size ( )
```

5.54.1.3 MQTTSerialize_puback()

```
int MQTTSerialize_puback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes a puback packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.54.1.4 MQTTSerialize_pubcomp()

```
int MQTTSerialize_pubcomp (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes a pubrel packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.54.1.5 MQTTSerialize_publish()

```
int MQTTSerialize_publish (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    int qos,
    unsigned char retained,
    unsigned short packetid,
    MQTTString topicName,
    unsigned char * payload,
    int payloadlen )
```

Serializes the supplied publish data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
dup	integer - the MQTT dup flag
qos	integer - the MQTT QoS value

Parameters

retained	integer - the MQTT retained flag
packetid	integer - the MQTT packet identifier
topicName	MQTTString - the MQTT topic in the publish
payload	byte buffer - the MQTT publish payload
payloadlen	integer - the length of the MQTT payload

Returns

the length of the serialized data. <= 0 indicates error

5.54.1.6 MQTTSerialize_publish_header()

```
int MQTTSerialize_publish_header (
    unsigned char * buf,
    unsigned char dup,
    int qos,
    unsigned char retained,
    unsigned short packetid,
    MQTTString topicName,
    int payloadlen )
```

Here is the call graph for this function:

5.54.1.7 MQTTSerialize_publish_size()

```
int MQTTSerialize_publish_size (
    int qos,
    MQTTString topicName,
    int payloadlen )
```

Here is the call graph for this function:

5.54.1.8 MQTTSerialize_publishLength()

```
int MQTTSerialize_publishLength (
    int qos,
    MQTTString topicName,
    int payloadlen )
```

Determines the length of the MQTT publish packet that would be produced using the supplied parameters

Parameters

qos	the MQTT QoS of the publish (packetid is omitted for QoS 0)
topicName	the topic name to be used in the publish
payloadlen	the length of the payload to be sent

Returns

the length of buffer needed to contain the serialized version of the packet

Here is the caller graph for this function:

5.54.1.9 MQTTSerialize_pubrel()

```
int MQTTSerialize_pubrel (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid )
```

Serializes a pubrel packet into the supplied buffer.

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier

Returns

serialized length, or error if 0

5.55 st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_client.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_subscribe_client.c:

Functions

- int MQTTSerialize_subscribeLength (int count, MQTTString topicFilters[ ])
- int MQTTSerialize_subscribe (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ], int requestedQoSs[ ])
- int MQTTSerialize_subscribe_size (int count, MQTTString topicFilters[ ])
- int MQTTDeserialize_suback (unsigned short ∗packetid, int maxcount, int ∗count, int grantedQoSs[ ], unsigned char ∗buf, int buflen)

5.55.1 Function Documentation

5.55.1.1 MQTTDeserialize_suback()

```
int MQTTDeserialize_suback (
    unsigned short * packetid,
    int maxcount,
    int * count,
    int grantedQoSs[],
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into suback data

Parameters

packetid	returned integer - the MQTT packet identifier
maxcount	- the maximum number of members allowed in the grantedQoSs array
count	returned integer - number of members in the grantedQoSs array
grantedQoSs	returned array of integers - the granted qualities of service
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.55.1.2 MQTTSerialize_subscribe()

```
int MQTTSerialize_subscribe (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[],
    int requestedQoSs[] )
```

Serializes the supplied subscribe data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied bufferr
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier
count	- number of members in the topicFilters and reqQos arrays
topicFilters	- array of topic filter names
requestedQoSs	- array of requested QoS

Returns

the length of the serialized data. <= 0 indicates error

Here is the caller graph for this function:

5.55.1.3 MQTTSerialize_subscribe_size()

```
int MQTTSerialize_subscribe_size (
    int count,
    MQTTString topicFilters[] )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.55.1.4 MQTTSerialize_subscribeLength()

```
int MQTTSerialize_subscribeLength (
    int count,
    MQTTString topicFilters[] )
```

Determines the length of the MQTT subscribe packet that would be produced using the supplied parameters

Parameters

count	the number of topic filter strings in topicFilters
topicFilters	the array of topic filter strings to be used in the publish

Returns

the length of buffer needed to contain the serialized version of the packet

Here is the caller graph for this function:

5.56 st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_server.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_subscribe_server.c:

Functions

- int MQTTDeserialize_subscribe (unsigned char ∗dup, unsigned short ∗packetid, int maxcount, int ∗count, MQTTString topicFilters[ ], int requestedQoSs[ ], unsigned char ∗buf, int buflen)
- int MQTTSerialize_suback (unsigned char ∗buf, int buflen, unsigned short packetid, int count, int ∗granted↩ QoSs)

5.56.1 Function Documentation

5.56.1.1 MQTTDeserialize_subscribe()

```
int MQTTDeserialize_subscribe (
    unsigned char * dup,
    unsigned short * packetid,
    int maxcount,
    int * count,
    MQTTString topicFilters[],
    int requestedQoSs[],
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into subscribe data

Parameters

dup	integer returned - the MQTT dup flag
packetid	integer returned - the MQTT packet identifier
maxcount	- the maximum number of members allowed in the topicFilters and requestedQoSs arrays
count	- number of members in the topicFilters and requestedQoSs arrays
topicFilters	- array of topic filter names
requestedQoSs	- array of requested QoS
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

the length of the serialized data. <= 0 indicates error

5.56.1.2 MQTTSerialize_suback()

```
int MQTTSerialize_suback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid,
    int count,
    int * grantedQoSs )
```

Serializes the supplied suback data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier
count	- number of members in the grantedQoSs array
grantedQoSs	- array of granted QoS

Returns

the length of the serialized data. <= 0 indicates error

5.57 st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_client.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_unsubscribe_client.c:

Functions

- int MQTTSerialize_unsubscribeLength (int count, MQTTString topicFilters[ ])
- int MQTTSerialize_unsubscribe (unsigned char ∗buf, int buflen, unsigned char dup, unsigned short packetid, int count, MQTTString topicFilters[ ])
- int MQTTSerialize_unsubscribe_size (int count, MQTTString topicFilters[ ])
- int MQTTDeserialize_unsuback (unsigned short ∗packetid, unsigned char ∗buf, int buflen)

5.57.1 Function Documentation

5.57.1.1 MQTTDeserialize_unsuback()

```
int MQTTDeserialize_unsuback (
    unsigned short * packetid,
    unsigned char * buf,
    int buflen )
```

Deserializes the supplied (wire) buffer into unsuback data

Parameters

packetid	returned integer - the MQTT packet identifier
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

error code. 1 is success, 0 is failure

5.57.1.2 MQTTSerialize_unsubscribe()

```
int MQTTSerialize_unsubscribe (
    unsigned char * buf,
    int buflen,
    unsigned char dup,
    unsigned short packetid,
    int count,
    MQTTString topicFilters[] )
```

Serializes the supplied unsubscribe data into the supplied buffer, ready for sending

Parameters

buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer
dup	integer - the MQTT dup flag
packetid	integer - the MQTT packet identifier
count	- number of members in the topicFilters array
topicFilters	- array of topic filter names

Returns

the length of the serialized data. <= 0 indicates error

Here is the caller graph for this function:

5.57.1.3 MQTTSerialize_unsubscribe_size()

```
int MQTTSerialize_unsubscribe_size (
    int count,
    MQTTString topicFilters[] )
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.57.1.4 MQTTSerialize_unsubscribeLength()

```
int MQTTSerialize_unsubscribeLength (
    int count,
    MQTTString topicFilters[] )
```

Determines the length of the MQTT unsubscribe packet that would be produced using the supplied parameters

Parameters

count	the number of topic filter strings in topicFilters
topicFilters	the array of topic filter strings to be used in the publish

Returns

the length of buffer needed to contain the serialized version of the packet

Here is the caller graph for this function:

5.58 st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_server.c File Reference

```
#include "iot_mqtt_packet.h"
#include "iot_mqtt_stacktrace.h"
#include <string.h>
```

Include dependency graph for iot_mqtt_unsubscribe_server.c:

Functions

- int MQTTDeserialize_unsubscribe (unsigned char ∗dup, unsigned short ∗packetid, int maxcount, int ∗count, MQTTString topicFilters[ ], unsigned char ∗buf, int len)
- int MQTTSerialize_unsuback (unsigned char ∗buf, int buflen, unsigned short packetid)

5.58.1 Function Documentation

5.58.1.1 MQTTDeserialize_unsubscribe()

```
int MQTTDeserialize_unsubscribe (
    unsigned char * dup,
    unsigned short * packetid,
    int maxcount,
    int * count,
    MQTTString topicFilters[],
    unsigned char * buf,
    int len )
```

Deserializes the supplied (wire) buffer into unsubscribe data

Parameters

dup	integer returned - the MQTT dup flag
packetid	integer returned - the MQTT packet identifier
maxcount	- the maximum number of members allowed in the topicFilters and requestedQoSs arrays
count	- number of members in the topicFilters and requestedQoSs arrays
topicFilters	- array of topic filter names
buf	the raw buffer data, of the correct length determined by the remaining length field
buflen	the length in bytes of the data in the supplied buffer

Returns

the length of the serialized data. <= 0 indicates error

5.58.1.2 MQTTSerialize_unsuback()

```
int MQTTSerialize_unsuback (
    unsigned char * buf,
    int buflen,
    unsigned short packetid )
```

Serializes the supplied unsuback data into the supplied buffer, ready for sending

Parameters

buf	the buffer into which the packet will be serialized
buflen	the length in bytes of the supplied buffer
packetid	integer - the MQTT packet identifier

Returns

the length of the serialized data. <= 0 indicates error

5.59 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_debug_esp8266.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <esp_log.h>
#include <esp_system.h>
#include <esp_heap_caps.h>
#include "iot_bsp_debug.h"
```

Include dependency graph for iot_bsp_debug_esp8266.c:

Functions

- void iot_bsp_debug (iot_debug_level_t level, const char *tag, const char *fmt,...)

  Write message into the log.
- void iot_bsp_debug_check_heap (const char *tag, const char *func, const int line, const char *fmt,...)

  Check memory(heap) status.

Variables

- heap_region_t g_heap_region [HEAP_REGIONS_MAX]

5.59.1 Function Documentation

5.59.1.1 iot_bsp_debug()

```
void iot_bsp_debug (
    iot_debug_level_t level,
    const char * tag,
    const char * fmt,
    ... )
```

Write message into the log.

This function is not intended to be used directly. Instead, use one of IOT_ERROR, IOT_WARN, IOT_INFO, IOT_↩ DEBUG macros.

Parameters

in	*level*	log level • IOT_DEBUG_LEVEL_NONE • IOT_DEBUG_LEVEL_ERROR • IOT_DEBUG_LEVEL_WARN • IOT_DEBUG_LEVEL_INFO • IOT_DEBUG_LEVEL_DEBUG
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.59.1.2 iot_bsp_debug_check_heap()

```
void iot_bsp_debug_check_heap (
    const char * tag,
    const char * func,
    const int line,
    const char * fmt,
    ... )
```

Check memory(heap) status.

This function check memory(heap) status

Parameters

in	*tag*	tag
in	*func*	caller function
in	*line*	line number
in	*fmt*	user friendly string

5.59.2 Variable Documentation

5.59.2.1 g_heap_region `heap_region_t g_heap_region[HEAP_REGIONS_MAX]`

5.60 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_fs_esp8266.c File Reference

```
#include <stdlib.h>
#include <string.h>
#include <esp_err.h>
#include <nvs_flash.h>
#include "iot_bsp_fs.h"
#include "iot_bsp_nv_data.h"
#include "iot_crypto.h"
#include "iot_debug.h"
```

Include dependency graph for iot_bsp_fs_esp8266.c:

Macros

- #define STDK_NV_DATA_PARTITION "stnv"
- #define STDK_NV_DATA_NAMESPACE "stdk"

Functions

- iot_error_t iot_bsp_fs_init ()

  Initialize a file system.
- iot_error_t iot_bsp_fs_deinit ()

  Deinitialize a file system.

- iot_error_t iot_bsp_fs_open (const char *filename, iot_bsp_fs_open_mode_t mode, iot_bsp_fs_handle_t *handle)

  Open a file.
- iot_error_t iot_bsp_fs_read (iot_bsp_fs_handle_t handle, char *buffer, unsigned int length)

  Read a file.
- iot_error_t iot_bsp_fs_write (iot_bsp_fs_handle_t handle, const char *data, unsigned int length)

  Write a file.
- iot_error_t iot_bsp_fs_close (iot_bsp_fs_handle_t handle)

  Close a file.
- iot_error_t iot_bsp_fs_remove (const char *filename)

  Remove a file.

5.60.1 Macro Definition Documentation

5.60.1.1 STDK_NV_DATA_NAMESPACE `#define STDK_NV_DATA_NAMESPACE "stdk"`

5.60.1.2 STDK_NV_DATA_PARTITION `#define STDK_NV_DATA_PARTITION "stnv"`

5.60.2 Function Documentation

5.60.2.1 iot_bsp_fs_close() `iot_error_t iot_bsp_fs_close ( iot_bsp_fs_handle_t handle )`

Close a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().

Return values

IOT_ERROR_NONE	File close successful.

Here is the caller graph for this function:

5.60.2.2 iot_bsp_fs_deinit() `iot_error_t iot_bsp_fs_deinit ( )`

Deinitialize a file system.

Return values

IOT_ERROR_NONE	File-system deinit successful.
IOT_ERROR_DEINIT_FAIL	File-system deinit failed.

Here is the caller graph for this function:

5.60.2.3 iot_bsp_fs_init() `iot_error_t iot_bsp_fs_init ( )`

Initialize a file system.

This function initializes the file-system of the device. You must call this function before using file-management function (open, read, write, etc.)

Return values

IOT_ERROR_NONE	File-system init successful.
IOT_ERROR_INIT_F↩ AIL	File-system init failed.

Here is the caller graph for this function:

5.60.2.4 iot_bsp_fs_open()

```
iot_error_t iot_bsp_fs_open (
    const char * filename,
    iot_bsp_fs_open_mode_t mode,
    iot_bsp_fs_handle_t * handle )
```

Open a file.

Parameters

in	*filename*	File name.
in	*mode*	File open mode. FS_READONLY or FS_READWRITE.
out	*handle*	A pointer to the file handle.

Return values

IOT_ERROR_NONE	File open successful.
IOT_ERROR_FS_OPEN_F↩ AIL	File open failed.

Here is the caller graph for this function:

5.60.2.5 iot_bsp_fs_read()

```
iot_error_t iot_bsp_fs_read (
    iot_bsp_fs_handle_t handle,
    char * buffer,
    unsigned int length )
```

Read a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().
out	*buffer*	A pointer to buffer array to store the read data from the file.
in	*length*	The size of buffer.

Return values

IOT_ERROR_NONE	File read successful.
IOT_ERROR_FS_READ_F↩AIL	File read failed.
IOT_ERROR_FS_NO_FILE	No file.

Here is the caller graph for this function:

5.60.2.6 iot_bsp_fs_remove()

```
iot_error_t iot_bsp_fs_remove (
    const char * filename )
```

Remove a file.

Parameters

in	*filename*	File name.

Return values

IOT_ERROR_NONE	File remove successful.
IOT_ERROR_INVALID_ARGS	Invalid filename
IOT_ERROR_FS_REMOVE_FAIL	File remove failed.

Here is the caller graph for this function:

5.60.2.7 iot_bsp_fs_write()

```
iot_error_t iot_bsp_fs_write (
    iot_bsp_fs_handle_t handle,
    const char * data,
    unsigned int length )
```

Write a file.

Parameters

in	*handle*	This is iot_bsp_fs_handle_t handle from iot_bsp_fs_open().
in	*data*	A pointer to data array to write to the file.
in	*length*	The size of data.

Return values

IOT_ERROR_NONE	File write successful.
IOT_ERROR_FS_WRITE_FAIL	File write failed.

Here is the call graph for this function:

Here is the caller graph for this function:

5.61 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_nv_data_esp8266.c File Reference

```
#include "iot_bsp_nv_data.h"
#include "iot_debug.h"
```

Include dependency graph for iot_bsp_nv_data_esp8266.c:

Functions

- const char * iot_bsp_nv_get_data_path (iot_nvd_t nv_type)

  Get a nv data path.

5.61.1 Function Documentation

5.61.1.1 iot_bsp_nv_get_data_path()

```
const char* iot_bsp_nv_get_data_path (
    iot_nvd_t nv_type )
```

Get a nv data path.

This function will return the nv data path. The detail of data path depends on the file-system type. ex : espressif nvs-system - nvs key ("nvItemKey") linux file-system - file path ("/stnv/nvItemKey")

Parameters

in	*nv_type*	The type of nv data. declaration is in "iot_nv_data.h"

Returns

nv data path.

See also

iot_nv_data.h

Here is the caller graph for this function:

5.62 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_random_esp8266.c File Reference

```
#include "iot_bsp_random.h"
#include <esp_system.h>
```

Include dependency graph for iot_bsp_random_esp8266.c:

Functions

- unsigned int iot_bsp_random ()

  Generate random number.

5.62.1 Function Documentation

5.62.1.1 iot_bsp_random() `unsigned int iot_bsp_random ( )`

Generate random number.

This function generates and returns a random number.

Returns

unsigned int random number

Here is the caller graph for this function:

5.63 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_system_esp8266.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include "esp_system.h"
#include "esp8266/eagle_soc.h"
#include "esp8266/efuse_register.h"
#include "iot_bsp_system.h"
#include "iot_debug.h"
```

Include dependency graph for iot_bsp_system_esp8266.c:

Functions

- void iot_bsp_system_reboot ()

  Restart system.
- void iot_bsp_system_poweroff ()

  Shutdown system.
- iot_error_t iot_bsp_system_get_time_in_sec (char ∗buf, unsigned int buf_len)

  Get system time in second.
- iot_error_t iot_bsp_system_set_time_in_sec (const char ∗time_in_sec)

  Set system time in second.
- iot_error_t iot_bsp_system_get_uniqueid (unsigned char ∗∗uid, size_t ∗olen)

  Get device unique value.

5.63.1 Function Documentation

5.63.1.1 iot_bsp_system_get_time_in_sec()

```
iot_error_t iot_bsp_system_get_time_in_sec (
    char * buf,
    unsigned int buf_len )
```

Get system time in second.

Parameters

out	*buf*	A pointer to data array to store the system time in second.
in	*buf_len*	The length of buffer.

Return values

IOT_ERROR_NONE	Set time successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.

5.63.1.2 iot_bsp_system_get_uniqueid()

```
iot_error_t iot_bsp_system_get_uniqueid (
    unsigned char ** uid,
    size_t * olen )
```

Get device unique value.

The source of unique value

Parameters

out	*uid*	a pointer of pointer to a unique id buffer
out	*olen*	the bytes written to unique id buffer

Returns

iot_error_t

Return values

IOT_ERROR_NONE	success
IOT_ERROR_MEM_ALLOC	alloc failed for unique id buffer
IOT_ERROR_NOT_IMPLEMENTED	no way to make unique id

5.63.1.3 iot_bsp_system_poweroff()

```
void iot_bsp_system_poweroff ( )
```

Shutdown system.

This function shuts down system.

5.63.1.4 iot_bsp_system_reboot()

```
void iot_bsp_system_reboot ( )
```

Restart system.

This function restarts system.

5.63.1.5 iot_bsp_system_set_time_in_sec()

```
iot_error_t iot_bsp_system_set_time_in_sec (
    const char * time_in_sec )
```

Set system time in second.

Parameters

in	*time_in_sec*	A time value in second from struct timeval's tv_sec (ex : 1546300800)

Return values

IOT_ERROR_NONE	Set time successful.
IOT_ERROR_INVALID_ARGS	Invalid argument.

5.64 st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_wifi_esp8266.c File Reference

```
#include <string.h>
#include <time.h>
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "rom/ets_sys.h"
#include "esp_wifi.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "iot_debug.h"
#include "iot_bsp_wifi.h"
#include "iot_os_util.h"
#include "iot_util.h"
#include "lwip/apps/sntp.h"
```

Include dependency graph for iot_bsp_wifi_esp8266.c:

Functions

- iot_error_t iot_bsp_wifi_init ()

  Initialize Wi-Fi function.
- iot_error_t iot_bsp_wifi_set_mode (iot_wifi_conf ∗conf)

  Set the Wi-Fi mode.
- uint16_t iot_bsp_wifi_get_scan_result (iot_wifi_scan_result_t ∗scan_result)

  Get the AP scan result.
- iot_error_t iot_bsp_wifi_get_mac (struct iot_mac ∗wifi_mac)

  Get the Wi-Fi MAC.
- iot_wifi_freq_t iot_bsp_wifi_get_freq (void)

  Get the Wi-Fi Frequency band.

Variables

- const int WIFI_STA_START_BIT = BIT0
- const int WIFI_STA_CONNECT_BIT = BIT1
- const int WIFI_STA_DISCONNECT_BIT = BIT2
- const int WIFI_AP_START_BIT = BIT3
- const int WIFI_AP_STOP_BIT = BIT4
- const int WIFI_EVENT_BIT_ALL = BIT0|BIT1|BIT2|BIT3|BIT4

5.64.1 Function Documentation

5.64.1.1 iot_bsp_wifi_get_freq()

```
iot_wifi_freq_t iot_bsp_wifi_get_freq (
    void )
```

Get the Wi-Fi Frequency band.

This function get the Wi-Fi Frequency band

Returns

IOT_WIFI_FREQ_2_4G_ONLY : 2.4GHz only supported IOT_WIFI_FREQ_5G_ONLY : 5GHz only supported IOT_WIFI_FREQ_2_4G_5G_BOTH : 2.4GHz and 5GHz both supported

5.64.1.2 iot_bsp_wifi_get_mac()

```
iot_error_t iot_bsp_wifi_get_mac (
    struct iot_mac * wifi_mac )
```

Get the Wi-Fi MAC.

This function get the Wi-Fi MAC

Parameters

out	*iot_mac*	array to save Wi-Fi MAC

Returns

IOT_ERROR_NONE : Success IOT_ERROR_READ_FAIL

Here is the caller graph for this function:

5.64.1.3 iot_bsp_wifi_get_scan_result()

```
uint16_t iot_bsp_wifi_get_scan_result (
    iot_wifi_scan_result_t * scan_result )
```

Get the AP scan result.

This function get the scan result

Parameters

out	*iot_wifi_scan_result_t*	array to save AP list

Returns

number of APs found

Here is the call graph for this function:

5.64.1.4 iot_bsp_wifi_init()

```
iot_error_t iot_bsp_wifi_init ( )
```

Initialize Wi-Fi function.

This function initializes Wi-Fi

Returns

IOT_ERROR_NONE : succeed

Here is the caller graph for this function:

5.64.1.5 iot_bsp_wifi_set_mode()

```
iot_error_t iot_bsp_wifi_set_mode (
    iot_wifi_conf * conf )
```

Set the Wi-Fi mode.

This function set the wifi operating mode as scan, station and softap

Parameters

in	*mode*	Wi-Fi operation mode

Returns

IOT_ERROR_NONE : succeed

Here is the caller graph for this function:

5.64.2 Variable Documentation

5.64.2.1 WIFI_AP_START_BIT

```
const int WIFI_AP_START_BIT = BIT3
```

5.64.2.2 WIFI_AP_STOP_BIT `const int WIFI_AP_STOP_BIT = BIT4`

5.64.2.3 WIFI_EVENT_BIT_ALL `const int WIFI_EVENT_BIT_ALL = BIT0|BIT1|BIT2|BIT3|BIT4`

5.64.2.4 WIFI_STA_CONNECT_BIT `const int WIFI_STA_CONNECT_BIT = BIT1`

5.64.2.5 WIFI_STA_DISCONNECT_BIT `const int WIFI_STA_DISCONNECT_BIT = BIT2`

5.64.2.6 WIFI_STA_START_BIT `const int WIFI_STA_START_BIT = BIT0`

5.65 st-device-sdk-c/src/port/os/freertos/iot_os_net_freertos.c File Reference

```
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <errno.h>
#include <unistd.h>
#include "iot_os_net.h"
#include "iot_debug.h"
```

Include dependency graph for iot_os_net_freertos.c:

Functions

- void iot_os_net_print_status (iot_net_socket *n)

  print network status
- int iot_os_net_select (iot_net_socket *n, unsigned int timeout_ms)

  check network socket status
- void iot_os_net_disconnect (iot_net_socket *n)

  disconnect with server
- int iot_os_net_connet (iot_net_socket *n, char *addr, int port)

  connect with server
- void iot_os_net_init (iot_net_socket *n)

  Initialize the network structure.

5.65.1 Function Documentation

5.65.1.1 iot_os_net_connet()

```
int iot_os_net_connet (
    iot_net_socket * n,
    char * addr,
    int port )
```

connect with server

Parameters

n	- iot_net_socket struct
addr	- server address
port	- server port

Returns

connect status

5.65.1.2 iot_os_net_disconnect()

```
void iot_os_net_disconnect (
    iot_net_socket * n )
```

disconnect with server

Parameters

n	- iot_net_socket struct

Returns

void

5.65.1.3 iot_os_net_init()

```
void iot_os_net_init (
    iot_net_socket *  )
```

Initialize the network structure.

Parameters

n	- iot_net_socket struct

Returns

void

5.65.1.4 iot_os_net_print_status()

```
void iot_os_net_print_status (
    iot_net_socket * n )
```

print network status

Parameters

n	- iot_net_socket struct

Returns

void

Here is the caller graph for this function:

5.65.1.5 iot_os_net_select()

```
int iot_os_net_select (
    iot_net_socket * n,
    unsigned int timeout_ms )
```

check network socket status

Parameters

n	- iot_net_socket struct
timeout_ms	- timeout in miliseconds

Returns

> 0 : there is some data to read == 0 : there is no data < 0 : there is error in network

Here is the caller graph for this function:

5.66 st-device-sdk-c/src/port/os/freertos/iot_os_util_freertos.c File Reference

```
#include <string.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "event_groups.h"
#include "semphr.h"
#include "iot_error.h"
#include "iot_os_util.h"
```

Include dependency graph for iot_os_util_freertos.c:

Typedefs

- typedef struct Freertos_Timer Freertos_Timer

Functions

- int iot_os_thread_create (void *thread_function, const char *name, int stack_size, void *data, int priority, iot_os_thread *thread_handle)

  create and run thread
- void iot_os_thread_delete (iot_os_thread thread_handle)

  delete thread
- void iot_os_thread_yield ()

  yield task
- iot_os_queue * iot_os_queue_create (int queue_length, int item_size)

  create queue

- int iot_os_queue_reset (iot_os_queue *queue_handle)

  reset queue
- void iot_os_queue_delete (iot_os_queue *queue_handle)

  delete queue
- int iot_os_queue_send (iot_os_queue *queue_handle, void *data, unsigned int wait_time_ms)

  send message to the back of queue.
- int iot_os_queue_receive (iot_os_queue *queue_handle, void *data, unsigned int wait_time_ms)

  receive message from the front of queue.
- iot_os_eventgroup * iot_os_eventgroup_create (void)

  create eventgroup
- void iot_os_eventgroup_delete (iot_os_eventgroup *eventgroup_handle)

  delete eventgroup
- unsigned int iot_os_eventgroup_wait_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits↩_to_wait_for, const int clear_on_exit, const int wait_for_all_bits, const unsigned int wait_time_ms)

  wait for bit of group of bits to become set
- unsigned int iot_os_eventgroup_set_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits_↩to_set)

  set bit/bits of eventgroup
- unsigned int iot_os_eventgroup_clear_bits (iot_os_eventgroup *eventgroup_handle, const unsigned int bits↩_to_clear)

  clear bit/bits of eventgroup
- int iot_os_mutex_init (iot_os_mutex *mutex)

  create mutex
- int iot_os_mutex_lock (iot_os_mutex *mutex)

  mutex lock
- int iot_os_mutex_unlock (iot_os_mutex *mutex)

  mutex unlock
- void iot_os_delay (unsigned int delay_ms)

  delay a thread
- void iot_os_timer_count_ms (iot_os_timer timer, unsigned int timeout_ms)

  set timer count
- unsigned int iot_os_timer_left_ms (iot_os_timer timer)

  return remaining time in ms unit
- char iot_os_timer_isexpired (iot_os_timer timer)

  check timer expired
- void iot_os_timer_init (iot_os_timer *timer)

  init timer
- void iot_os_timer_destroy (iot_os_timer *timer)

  destroy timer

Variables

- const unsigned int iot_os_max_delay = portMAX_DELAY
- const unsigned int iot_os_true = pdTRUE
- const unsigned int iot_os_false = pdFALSE

5.66.1 Typedef Documentation

5.66.1.1 Freertos_Timer `typedef struct Freertos_Timer Freertos_Timer`

5.66.2 Function Documentation

5.66.2.1 iot_os_delay()

```
void iot_os_delay (
    unsigned int delay_ms )
```

delay a thread

This function will delay thread for given time

Parameters

in	*delay*	time to wait

Here is the caller graph for this function:

5.66.2.2 iot_os_eventgroup_clear_bits()

```
unsigned int iot_os_eventgroup_clear_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_clear )
```

clear bit/bits of eventgroup

This function will clear bit/bits of eventgroup

Parameters

in	*eventgroup_handle*	handle of eventgroup to cleat bit/bits
in	*bits_to_clear*	bitwise value of bit/bits to clear ex) use 0b101 will clear bit0 and bit2 ex) use 0b100 will clear only bit2

Returns

return is bits of event BEFORE clear bit.

5.66.2.3 iot_os_eventgroup_create()

```
iot_os_eventgroup* iot_os_eventgroup_create (
    void )
```

create eventgroup

This function create eventgroup and return eventgroup handle

Returns

return is eventgroup handle of eventgroup created by this function If eventgroup was not created, NULL is returned.

Here is the caller graph for this function:

5.66.2.4 iot_os_eventgroup_delete()

```
void iot_os_eventgroup_delete (
    iot_os_eventgroup * eventgroup_handle )
```

delete eventgroup

This function delete eventgroup

Parameters

in	*eventgroup_handle*	eventgroup handle of eventgroup to delete

5.66.2.5 iot_os_eventgroup_set_bits()

```
unsigned int iot_os_eventgroup_set_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_set )
```

set bit/bits of eventgroup

This function will set bit/bits of eventgroup

Parameters

in	*eventgroup_handle*	handle of eventgroup to set bit/bits
in	*bits_to_set*	bitwise value of bit/bits to set ex) use 0b101 will set bit0 and bit2 ex) use 0b100 will set only bit2

Returns

return is bits of event after set bit/bits. if you set 'clear_on_exit' to IOT_OS_TRUE, it can be 0 because of other task waiting for bits

Here is the caller graph for this function:

5.66.2.6 iot_os_eventgroup_wait_bits()

```
unsigned int iot_os_eventgroup_wait_bits (
    iot_os_eventgroup * eventgroup_handle,
    const unsigned int bits_to_wait_for,
    const int clear_on_exit,
    const int wait_for_all_bits,
    const unsigned int wait_time_ms )
```

wait for bit of group of bits to become set

This function will wait for event bit

Parameters

in	*eventgroup_handle*	handle of eventgroup waiting for
in	*bits_to_wait_for*	bitwise value of bit/bits. ex) use 0b101 to wait bit 0 and/or bit 2
in	*clear_on_exit*	if this value is IOT_OS_TRUE, bits in 'bits_to_wait_for' will be cleared
in	*wait_for_all_bits*	if this value is IOT_OS_TRUE, wait for ""ALL"" bits in 'bits_to_wait_for'
in	*wait_time_ms*	maximum time to wait until all/one of bits are set

Returns

return is bits of eventgroup. if you wait for bit0 and bit3, can check with was setted, return will be 0b101 = 5

5.66.2.7 iot_os_mutex_init()

```
int iot_os_mutex_init (
    iot_os_mutex * mutex )
```

create mutex

This function will create mutex

Parameters

out	*mutex*	handle of mutex created by this function

Returns

IOT_ERROR_NONE : success otherwise : fail

Here is the caller graph for this function:

5.66.2.8 iot_os_mutex_lock() `int iot_os_mutex_lock ( iot_os_mutex * mutex )`

mutex lock

This function will lock mutex before critical section

Parameters

in	*mutex*	handle of mutex

Returns

IOT_ERROR_NONE : success otherwise : fail

Here is the caller graph for this function:

5.66.2.9 iot_os_mutex_unlock()

```
int iot_os_mutex_unlock (
    iot_os_mutex * mutex )
```

mutex unlock

This function will unlock mutex after critical section

Parameters

in	*mutex*	handle of mutex

Returns

IOT_ERROR_NONE : success otherwise : fail

5.66.2.10 iot_os_queue_create()

```
iot_os_queue* iot_os_queue_create (
    int queue_length,
    int item_size )
```

create queue

This function create queue and return queue handle

Parameters

in	*queue_length*	maximum number of queue item
in	*item_size*	size of item

Returns

return is queue handle of queue created by this function If queue was not created, NULL is returned.

Here is the caller graph for this function:

5.66.2.11 iot_os_queue_delete()

```
void iot_os_queue_delete (
    iot_os_queue * queue_handle )
```

delete queue

This function delete queue

Parameters

in	*queue_handle*	handle of queue to be deleted.

5.66.2.12 iot_os_queue_receive()

```
int iot_os_queue_receive (
    iot_os_queue * queue_handle,
    void * data,
    unsigned int wait_time_ms )
```

receive message from the front of queue.

This function will receive item from the front of queue

Parameters

in	*queue_handle*	handle of queue to receive item
in	*data*	buffer for item received from queue
in	*wait_time_ms*	maximum time to wait until queue is not empty

Returns

IOT_OS_TRUE : success otherwise : fail

5.66.2.13 iot_os_queue_reset()

```
int iot_os_queue_reset (
    iot_os_queue * queue_handle )
```

reset queue

This function reset queue

Parameters

in	*queue_handle*	handle of queue to reset.

Returns

IOT_OS_TRUE : success otherwise : fail

5.66.2.14 iot_os_queue_send()

```
int iot_os_queue_send (
    iot_os_queue * queue_handle,
    void * data,
    unsigned int wait_time_ms )
```

send message to the back of queue.

This function will send item to the back of queue

Parameters

in	*queue_handle*	handle of queue to save item
in	*data*	item to be saved in queue
in	*wait_time_ms*	maximum time to wait until queue is available

Returns

IOT_OS_TRUE : success otherwise : fail

Here is the caller graph for this function:

5.66.2.15 iot_os_thread_create()

```
int iot_os_thread_create (
    void * thread_function,
    const char * name,
    int stack_size,
    void * data,
    int priority,
    iot_os_thread * thread_handle )
```

create and run thread

This function creates and runs thread

Parameters

in	*thread_function*	function pointer for thread loop
in	*name*	name of thread
in	*stack_size*	stack size of thread
in	*data*	data that will passed into the thread as the parameter
in	*priority*	priority of thread created by this function
out	*thread_handle*	thread handle of thread created by this function (optional, NULL is possible)

Returns

IOT_OS_TRUE : success otherwise : fail

Here is the caller graph for this function:

5.66.2.16 iot_os_thread_delete()

```
void iot_os_thread_delete (
    iot_os_thread thread_handle )
```

delete thread

This function delete thread

Parameters

in	*thread_handle*	thread handle of thread to delete(if NULL is passed, delete current thread)

5.66.2.17 iot_os_thread_yield()

```
void iot_os_thread_yield ( )
```

yield task

This function yields task

5.66.2.18 iot_os_timer_count_ms()

```
void iot_os_timer_count_ms (
    iot_os_timer timer,
    unsigned int timeout_ms )
```

set timer count

This function will set timer count in ms unit.

Parameters

in	*timer*	timer handle
in	*count*	count to set in ms unit

Here is the caller graph for this function:

5.66.2.19 iot_os_timer_destroy()

```
void iot_os_timer_destroy (
    iot_os_timer * timer )
```

destroy timer

This function will destroy timer struct

Parameters

in	*timer*	pointer of timer to destroy

Here is the caller graph for this function:

5.66.2.20 iot_os_timer_init()

```
void iot_os_timer_init (
    iot_os_timer * timer )
```

init timer

This function will init timer struct

Parameters

in	*timer*	pointer of timer to init

Here is the caller graph for this function:

5.66.2.21 iot_os_timer_isexpired()

```
char iot_os_timer_isexpired (
    iot_os_timer timer )
```

check timer expired

This function will check if timer is expired

Parameters

in	*timer*	timer handle

Returns

1 : timer is expired 0 : timer is not expired

Here is the caller graph for this function:

5.66.2.22 iot_os_timer_left_ms()

```
unsigned int iot_os_timer_left_ms ( iot_os_timer timer )
```

return remaining time in ms unit

This function will return remaining time in ms unit

Parameters

in	*timer*	timer handle

Returns

0 : timer is expired non-zero : remaining time

Here is the caller graph for this function:

5.66.3 Variable Documentation

5.66.3.1 iot_os_false

```
const unsigned int iot_os_false = pdFALSE
```

5.66.3.2 iot_os_max_delay

```
const unsigned int iot_os_max_delay = portMAX_DELAY
```

5.66.3.3 iot_os_true

```
const unsigned int iot_os_true = pdTRUE
```

Index

_IOT_NOTI_TYPE_DEV_DELETED
iot_main.h, 187
_IOT_NOTI_TYPE_JWT_EXPIRED
iot_main.h, 187
_IOT_NOTI_TYPE_QUOTA_REACHED
iot_main.h, 187
_IOT_NOTI_TYPE_RATE_LIMIT
iot_main.h, 187
_IOT_NOTI_TYPE_UNKNOWN
iot_main.h, 187
_iot_easysetup_gen_get_payload
iot_easysetup_http.c, 98
_iot_easysetup_gen_post_payload
iot_easysetup_http.c, 99
_iot_mqtt_cmd_sub_callback
iot_mqtt.c, 319
_iot_mqtt_noti_sub_callback
iot_mqtt.c, 319
_iot_noti_type
iot_main.h, 187
_iot_nv_read_data
iot_nv_data.c, 326
_iot_nv_write_data
iot_nv_data.c, 326

addr
iot_mac, 48
all
MQTTConnackFlags, 70
MQTTConnectFlags, 71
args_str
iot_cap_cmd_data_t, 12
ARRAY_SIZE
iot_easysetup_http.c, 98
authmode
iot_wifi_scan_result_t, 63

bits
MQTTConnackFlags, 70
MQTTConnectFlags, 72
MQTTHeader, 73
broker_port
iot_cloud_prov_data, 24
broker_url
iot_cloud_prov_data, 25
bssid
iot_wifi_conf, 60
iot_wifi_prov_data, 61
iot_wifi_scan_result_t, 63
buf
iot_mqtt_ctx, 50
MQTTClient, 67
buf_size
MQTTClient, 67
bufchar
iot_mqtt_packet.c, 376
byte
MQTTHeader, 73

cap_handle_list
iot_context, 28
capability
iot_cap_handle, 19
cleansession
MQTTClient, 67
MQTTConnectFlags, 72
MQTTPacket_connectData, 78
cli
iot_mqtt_ctx, 50
client_ctx
iot_context, 29
clientID
MQTTPacket_connectData, 78
cloud
iot_device_prov_data, 46
cmd_cb
iot_cap_cmd_set, 14
cmd_count
iot_context, 29
cmd_data
iot_cap_cmd_data_t, 12
cmd_err
iot_context, 29
cmd_filter
iot_mqtt_ctx, 50
cmd_list
iot_cap_handle, 19
cmd_queue
iot_context, 29
cmd_status
iot_context, 29
cmd_type
iot_cap_cmd_set, 14
iot_command, 26
COLOR_CYAN
iot_debug.h, 150
COLOR_END
iot_debug.h, 150
command
iot_cap_cmd_set_list, 16
command_timeout_ms
MQTTClient, 67
component
iot_cap_handle, 19
CONFIG_STDK_MQTT_DYNAMIC_BUFFER
iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_PING_RETRY
iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_PUBLISH_RETRY
iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_RECV_BUFFER

iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_RECV_CYCLE
iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_SEND_BUFFER
iot_mqtt_client.h, 214
CONFIG_STDK_MQTT_SEND_CYCLE
iot_mqtt_client.h, 214
CONNACK
iot_mqtt_packet.h, 238
connack_return_codes
iot_mqtt_connect.h, 228
CONNECT
iot_mqtt_packet.h, 238
count
noti_data_raw_t::_rate_limit, 9
cstring
MQTTString, 81
ctx
iot_cap_handle, 19
curr_otm_feature
iot_context, 29
curr_state
iot_context, 29
curve
iot_crypto_ed25519_keypair, 36
cycle
iot_mqtt_client.c, 349

data
MQTTLenString, 74
DefaultClient
iot_mqtt_client.h, 214
defaultMessageHandler
MQTTClient, 67
defaultUserData
MQTTClient, 67
deliverMessage
iot_mqtt_client.c, 349
devconf
iot_context, 29
device_info
iot_context, 30
device_onboarding_id
iot_devconf_prov_data, 43
device_type
iot_devconf_prov_data, 43
deviceId
iot_registered_data, 56
DISCONNECT
iot_mqtt_packet.h, 238
DLLExport
iot_mqtt_client.h, 214
iot_mqtt_connect.h, 227
iot_mqtt_packet.h, 237
iot_mqtt_publish.h, 247
iot_mqtt_subscribe.h, 253
iot_mqtt_unsubscribe.h, 258
DLLImport
iot_mqtt_client.h, 214
iot_mqtt_connect.h, 227
iot_mqtt_packet.h, 238
iot_mqtt_publish.h, 247
iot_mqtt_subscribe.h, 253
iot_mqtt_unsubscribe.h, 258
domain
url_parse_t, 85
dup
MQTTHeader, 73
MQTTMessage, 75

easysetup_req_queue
iot_context, 30
easysetup_resp_queue
iot_context, 30
EASYSETUP_TIMEOUT_MS
iot_main.c, 315
ENTER
iot_debug.h, 151
err
iot_easysetup_payload, 47
errors
iot_mqtt_packet.h, 238
ES_CONFIRM_MAX_DELAY
iot_easysetup_d2d.c, 105
es_crypto_cipher_info
iot_context, 30
es_httpd_handle
iot_context, 30
evt_type
iot_cap_evt_data_t, 17
evt_unit
iot_cap_evt_data_t, 17
evt_value
iot_cap_evt_data_t, 17

fd
iot_bsp_fs_handle_t, 11
filename
iot_bsp_fs_handle_t, 11
firmware_version
iot_device_info, 44
fn
iot_crypto_pk_context, 39
fp
MQTTClient::MessageHandlers, 65
Freertos_Timer
iot_os_util_freertos.c, 418
freq
iot_wifi_scan_result_t, 63
fs_close_custom
iot_easysetup_http.c, 100
fs_open_custom
iot_easysetup_http.c, 100
FS_READONLY
iot_bsp_fs.h, 113
FS_READWRITE
iot_bsp_fs.h, 113
fs_state_free

iot_easysetup_http.c, 100
fs_state_init
iot_easysetup_http.c, 100
FUNC_ENTRY
iot_mqtt_stacktrace.h, 251
FUNC_ENTRY_MAX
iot_mqtt_stacktrace.h, 252
FUNC_ENTRY_MED
iot_mqtt_stacktrace.h, 252
FUNC_ENTRY_NOLOG
iot_mqtt_stacktrace.h, 252
FUNC_EXIT
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_MAX
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_MAX_RC
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_MED
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_MED_RC
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_NOLOG
iot_mqtt_stacktrace.h, 252
FUNC_EXIT_RC
iot_mqtt_stacktrace.h, 252

g_heap_region
iot_bsp_debug_esp8266.c, 399
get_cgi_cmds
iot_easysetup_http.c, 102
getfn
MQTTTransport, 83
getLenStringLen
iot_mqtt_packet.c, 376
grantedQoS
MQTTSubackData, 82

handle
iot_cap_handle_list, 20
HASH_SIZE
iot_easysetup_d2d.c, 105
hash_token
iot_crypto_ecdh_params, 35
hash_token_len
iot_crypto_ecdh_params, 35
hashed_sn
iot_devconf_prov_data, 43
HIT
iot_debug.h, 151
httpd_cgi_handler
iot_easysetup_http.c, 100
httpd_post_begin
iot_easysetup_http.c, 101
httpd_post_finished
iot_easysetup_http.c, 101
httpd_post_receive_data
iot_easysetup_http.c, 101

id
iot_uuid, 59
MQTTMessage, 75
info
iot_crypto_pk_context, 39
init_cb
iot_cap_handle, 19
init_usr_data
iot_cap_handle, 19
int
MQTTConnectFlags, 72
integer
iot_cap_val_t, 23
iot_api.c
iot_api_device_info_load, 294
iot_api_device_info_mem_free, 295
iot_api_onboarding_config_load, 295
iot_api_onboarding_config_mem_free, 296
iot_api_prov_data_mem_free, 296
iot_api_read_device_identity, 297
iot_command_send, 298
iot_device_cleanup, 299
iot_easysetup_request, 300
iot_get_time_in_ms, 300
iot_get_time_in_sec, 301
iot_state_update, 301
iot_api_device_info_load
iot_api.c, 294
iot_internal.h, 167
iot_api_device_info_mem_free
iot_api.c, 295
iot_internal.h, 168
iot_api_onboarding_config_load
iot_api.c, 295
iot_internal.h, 168
iot_api_onboarding_config_mem_free
iot_api.c, 296
iot_internal.h, 169
iot_api_prov_data_mem_free
iot_api.c, 296
iot_internal.h, 169
iot_api_read_device_identity
iot_api.c, 297
iot_internal.h, 170
iot_bsp_debug
iot_bsp_debug.h, 111
iot_bsp_debug_esp8266.c, 398
iot_debug.h, 153
iot_bsp_debug.h
iot_bsp_debug, 111
iot_bsp_debug_check_heap, 111
iot_bsp_debug_check_heap
iot_bsp_debug.h, 111
iot_bsp_debug_esp8266.c, 398
iot_debug.h, 153
iot_bsp_debug_esp8266.c
g_heap_region, 399
iot_bsp_debug, 398
iot_bsp_debug_check_heap, 398

iot_bsp_fs.h
FS_READONLY, 113
FS_READWRITE, 113
iot_bsp_fs_close, 113
iot_bsp_fs_deinit, 114
iot_bsp_fs_init, 114
iot_bsp_fs_open, 115
iot_bsp_fs_open_from_stnv, 116
iot_bsp_fs_open_mode_t, 113
iot_bsp_fs_read, 116
iot_bsp_fs_remove, 117
iot_bsp_fs_write, 118
iot_bsp_fs_close
iot_bsp_fs.h, 113
iot_bsp_fs_esp8266.c, 400
iot_bsp_fs_deinit
iot_bsp_fs.h, 114
iot_bsp_fs_esp8266.c, 401
iot_bsp_fs_esp8266.c
iot_bsp_fs_close, 400
iot_bsp_fs_deinit, 401
iot_bsp_fs_init, 401
iot_bsp_fs_open, 402
iot_bsp_fs_read, 403
iot_bsp_fs_remove, 404
iot_bsp_fs_write, 404
STDK_NV_DATA_NAMESPACE, 400
STDK_NV_DATA_PARTITION, 400
iot_bsp_fs_handle_t, 10
fd, 11
filename, 11
iot_bsp_fs_init
iot_bsp_fs.h, 114
iot_bsp_fs_esp8266.c, 401
iot_bsp_fs_open
iot_bsp_fs.h, 115
iot_bsp_fs_esp8266.c, 402
iot_bsp_fs_open_from_stnv
iot_bsp_fs.h, 116
iot_bsp_fs_open_mode_t
iot_bsp_fs.h, 113
iot_bsp_fs_read
iot_bsp_fs.h, 116
iot_bsp_fs_esp8266.c, 403
iot_bsp_fs_remove
iot_bsp_fs.h, 117
iot_bsp_fs_esp8266.c, 404
iot_bsp_fs_write
iot_bsp_fs.h, 118
iot_bsp_fs_esp8266.c, 404
iot_bsp_nv_data.h
iot_bsp_nv_get_data_path, 120
iot_bsp_nv_data_esp8266.c
iot_bsp_nv_get_data_path, 406
iot_bsp_nv_get_data_path
iot_bsp_nv_data.h, 120
iot_bsp_nv_data_esp8266.c, 406
iot_bsp_random
iot_bsp_random.h, 121
iot_bsp_random_esp8266.c, 407
iot_bsp_random.h
iot_bsp_random, 121
iot_bsp_random_esp8266.c
iot_bsp_random, 407
iot_bsp_system.h
iot_bsp_system_get_time_in_sec, 123
iot_bsp_system_get_uniqueid, 123
iot_bsp_system_poweroff, 124
iot_bsp_system_reboot, 124
iot_bsp_system_set_time_in_sec, 124
IOT_POWEROFF, 123
IOT_REBOOT, 123
iot_bsp_system_esp8266.c
iot_bsp_system_get_time_in_sec, 408
iot_bsp_system_get_uniqueid, 409
iot_bsp_system_poweroff, 409
iot_bsp_system_reboot, 409
iot_bsp_system_set_time_in_sec, 409
iot_bsp_system_get_time_in_sec
iot_bsp_system.h, 123
iot_bsp_system_esp8266.c, 408
iot_bsp_system_get_uniqueid
iot_bsp_system.h, 123
iot_bsp_system_esp8266.c, 409
iot_bsp_system_poweroff
iot_bsp_system.h, 124
iot_bsp_system_esp8266.c, 409
iot_bsp_system_reboot
iot_bsp_system.h, 124
iot_bsp_system_esp8266.c, 409
iot_bsp_system_set_time_in_sec
iot_bsp_system.h, 124
iot_bsp_system_esp8266.c, 409
iot_bsp_wifi.h
iot_bsp_wifi_get_freq, 127
iot_bsp_wifi_get_mac, 127
iot_bsp_wifi_get_scan_result, 128
iot_bsp_wifi_init, 129
iot_bsp_wifi_set_mode, 129
IOT_SOFT_AP_CHANNEL, 126
IOT_WIFI_AUTH_MAX, 127
iot_wifi_auth_mode_t, 126
IOT_WIFI_AUTH_OPEN, 127
IOT_WIFI_AUTH_WEP, 127
IOT_WIFI_AUTH_WPA2_ENTERPRISE, 127
IOT_WIFI_AUTH_WPA2_PSK, 127
IOT_WIFI_AUTH_WPA_PSK, 127
IOT_WIFI_AUTH_WPA_WPA2_PSK, 127
IOT_WIFI_CMD_TIMEOUT, 126
IOT_WIFI_FREQ_2_4G_5G_BOTH, 127
IOT_WIFI_FREQ_2_4G_ONLY, 127
IOT_WIFI_FREQ_5G_ONLY, 127
iot_wifi_freq_t, 127
IOT_WIFI_MAX_BSSID_LEN, 126
IOT_WIFI_MAX_PASS_LEN, 126
IOT_WIFI_MAX_SCAN_RESULT, 126

IOT_WIFI_MAX_SSID_LEN, 126
IOT_WIFI_MODE_OFF, 127
IOT_WIFI_MODE_P2P, 127
IOT_WIFI_MODE_SCAN, 127
IOT_WIFI_MODE_SOFTAP, 127
IOT_WIFI_MODE_STATION, 127
iot_wifi_mode_t, 127
IOT_WIFI_MODE_UNDEFINED, 127
iot_bsp_wifi_esp8266.c
iot_bsp_wifi_get_freq, 411
iot_bsp_wifi_get_mac, 411
iot_bsp_wifi_get_scan_result, 412
iot_bsp_wifi_init, 412
iot_bsp_wifi_set_mode, 413
WIFI_AP_START_BIT, 413
WIFI_AP_STOP_BIT, 413
WIFI_EVENT_BIT_ALL, 414
WIFI_STA_CONNECT_BIT, 414
WIFI_STA_DISCONNECT_BIT, 414
WIFI_STA_START_BIT, 414
iot_bsp_wifi_get_freq
iot_bsp_wifi.h, 127
iot_bsp_wifi_esp8266.c, 411
iot_bsp_wifi_get_mac
iot_bsp_wifi.h, 127
iot_bsp_wifi_esp8266.c, 411
iot_bsp_wifi_get_scan_result
iot_bsp_wifi.h, 128
iot_bsp_wifi_esp8266.c, 412
iot_bsp_wifi_init
iot_bsp_wifi.h, 129
iot_bsp_wifi_esp8266.c, 412
iot_bsp_wifi_set_mode
iot_bsp_wifi.h, 129
iot_bsp_wifi_esp8266.c, 413
IOT_BUF_RX_SIZE
iot_main.h, 186
IOT_BUF_TX_SIZE
iot_main.h, 186
iot_cap_call_init_cb
iot_capability.c, 303
iot_internal.h, 171
iot_cap_cmd_data_t, 11
args_str, 12
cmd_data, 12
num_args, 12
iot_cap_cmd_set, 13
cmd_cb, 14
cmd_type, 14
usr_data, 14
iot_cap_cmd_set_list, 14
command, 16
next, 16
iot_cap_cmd_set_list_t
iot_capability.h, 131
iot_cap_cmd_set_t
iot_capability.h, 131
iot_cap_evt_data_t, 16
evt_type, 17
evt_unit, 17
evt_value, 17
iot_capability.h, 131
IOT_CAP_HANDLE
st_dev.h, 282
iot_cap_handle, 18
capability, 19
cmd_list, 19
component, 19
ctx, 19
init_cb, 19
init_usr_data, 19
iot_cap_handle_list, 20
handle, 20
next, 20
iot_cap_handle_list_t
iot_main.h, 187
iot_cap_sub_cb
iot_capability.c, 304
iot_internal.h, 171
iot_cap_unit_t, 21
string, 21
type, 21
iot_cap_unit_type
iot_capability.h, 131
IOT_CAP_UNIT_TYPE_STRING
iot_capability.h, 132
IOT_CAP_UNIT_TYPE_UNUSED
iot_capability.h, 132
iot_cap_val_t, 22
integer, 23
number, 23
str_num, 23
string, 23
strings, 23
type, 23
iot_cap_val_type
st_dev.h, 283
IOT_CAP_VAL_TYPE_INT_OR_NUM
st_dev.h, 283
IOT_CAP_VAL_TYPE_INTEGER
st_dev.h, 283
IOT_CAP_VAL_TYPE_NUMBER
st_dev.h, 283
IOT_CAP_VAL_TYPE_STR_ARRAY
st_dev.h, 283
IOT_CAP_VAL_TYPE_STRING
st_dev.h, 283
iot_cap_val_type_t
st_dev.h, 282
IOT_CAP_VAL_TYPE_UNKNOWN
st_dev.h, 283
iot_capability.c
iot_cap_call_init_cb, 303
iot_cap_sub_cb, 304
iot_noti_sub_cb, 304
MAX_SQNUM, 303

st_cap_attr_create_int, 305
st_cap_attr_create_number, 305
st_cap_attr_create_string, 306
st_cap_attr_create_string_array, 306
st_cap_attr_free, 307
st_cap_attr_send, 307
st_cap_cmd_set_cb, 308
st_cap_handle_init, 308
st_conn_set_noti_cb, 310
iot_capability.h
iot_cap_cmd_set_list_t, 131
iot_cap_cmd_set_t, 131
iot_cap_evt_data_t, 131
iot_cap_unit_type, 131
IOT_CAP_UNIT_TYPE_STRING, 132
IOT_CAP_UNIT_TYPE_UNUSED, 132
iot_cloud_prov_data, 24
broker_port, 24
broker_url, 25
label, 25
location_id, 25
room_id, 25
IOT_CMD_STATE_HANDLE
iot_main.h, 188
iot_command, 25
cmd_type, 26
param, 26
IOT_COMMAND_CHECK_CLOUD_STATE
iot_main.h, 188
IOT_COMMAND_CHECK_PROV_STATUS
iot_main.h, 188
IOT_COMMAND_CLOUD_CONNECTING
iot_main.h, 188
IOT_COMMAND_CLOUD_REGISTERED
iot_main.h, 188
IOT_COMMAND_CLOUD_REGISTERING
iot_main.h, 188
IOT_COMMAND_NETWORK_MODE
iot_main.h, 188
IOT_COMMAND_NOTIFICATION_RECEIVED
iot_main.h, 188
IOT_COMMAND_READY_TO_CTL
iot_main.h, 188
IOT_COMMAND_SELF_CLEANUP
iot_main.h, 188
iot_command_send
iot_api.c, 298
iot_internal.h, 172
iot_command_type
iot_main.h, 187
IOT_COMMAND_TYPE_MAX
iot_main.h, 188
iot_connect_type
iot_main.h, 188
IOT_CONNECT_TYPE_COMMUNICATION
iot_main.h, 188
IOT_CONNECT_TYPE_REGISTRATION
iot_main.h, 188
iot_context, 26
cap_handle_list, 28
client_ctx, 29
cmd_count, 29
cmd_err, 29
cmd_queue, 29
cmd_status, 29
curr_otm_feature, 29
curr_state, 29
devconf, 29
device_info, 30
easysetup_req_queue, 30
easysetup_resp_queue, 30
es_crypto_cipher_info, 30
es_httpd_handle, 30
iot_events, 30
iot_reg_data, 30
lookup_id, 30
noti_cb, 31
noti_usr_data, 31
pin, 31
prov_data, 31
pub_queue, 31
reged_cli, 31
reported_stat, 31
req_state, 31
scan_num, 32
scan_result, 32
state_timer, 32
status_cb, 32
status_maps, 32
status_usr_data, 32
usr_events, 32
iot_crypto.c
iot_crypto_pk_free, 311
iot_crypto_pk_init, 311
iot_crypto_pk_sign, 312
iot_crypto_pk_verify, 312
iot_crypto.h
IOT_CRYPTO_ALIGN_B64_LEN, 135
iot_crypto_base64_decode, 139
iot_crypto_base64_decode_urlsafe, 140
iot_crypto_base64_encode, 140
iot_crypto_base64_encode_urlsafe, 141
IOT_CRYPTO_CAL_B64_LEN, 135
iot_crypto_cipher_aes, 141
IOT_CRYPTO_CIPHER_AES256, 139
IOT_CRYPTO_CIPHER_DECRYPT, 139
IOT_CRYPTO_CIPHER_ENCRYPT, 139
iot_crypto_cipher_get_align_size, 142
iot_crypto_cipher_info_t, 138
iot_crypto_cipher_mode_t, 138
iot_crypto_cipher_type_t, 139
IOT_CRYPTO_CIPHER_UNKNOWN, 139
iot_crypto_ecdh_gen_master_secret, 143
iot_crypto_ecdh_params_t, 138
iot_crypto_ed25519_convert_keypair, 143
iot_crypto_ed25519_convert_pubkey, 144

iot_crypto_ed25519_convert_seckey, 144
iot_crypto_ed25519_free_keypair, 144
iot_crypto_ed25519_init_keypair, 145
iot_crypto_ed25519_keypair_t, 138
IOT_CRYPTO_ED25519_LEN, 135
IOT_CRYPTO_IV_LEN, 135
iot_crypto_pk_context_t, 138
IOT_CRYPTO_PK_ED25519, 139
iot_crypto_pk_free, 145
iot_crypto_pk_funcs_t, 138
iot_crypto_pk_info_t, 138
iot_crypto_pk_init, 145
IOT_CRYPTO_PK_RSA, 139
iot_crypto_pk_sign, 146
IOT_CRYPTO_PK_TYPE_ED25519, 135
IOT_CRYPTO_PK_TYPE_RSA, 135
iot_crypto_pk_type_t, 139
IOT_CRYPTO_PK_UNKNOWN, 139
iot_crypto_pk_verify, 146
IOT_CRYPTO_SECRET_LEN, 135
iot_crypto_sha256, 147
IOT_CRYPTO_SHA256_LEN, 135
IOT_CRYPTO_SIGNATURE_LEN, 136
iot_crypto_ss_decrypt, 148
iot_crypto_ss_encrypt, 148
IOT_ERROR_CRYPTO_BASE64, 136
IOT_ERROR_CRYPTO_BASE64_URLSAFE, 136
IOT_ERROR_CRYPTO_CIPHER, 136
IOT_ERROR_CRYPTO_CIPHER_ALIGN, 136
IOT_ERROR_CRYPTO_CIPHER_IVLEN, 136
IOT_ERROR_CRYPTO_CIPHER_KEYLEN, 136
IOT_ERROR_CRYPTO_CIPHER_OUTSIZE, 136
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_MODE, 136
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE, 136
IOT_ERROR_CRYPTO_ED_KEY_CONVERT, 137
IOT_ERROR_CRYPTO_PK_ECDH, 137
IOT_ERROR_CRYPTO_PK_INVALID_ARG, 137
IOT_ERROR_CRYPTO_PK_INVALID_CTX, 137
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN, 137
IOT_ERROR_CRYPTO_PK_NULL_FUNC, 137
IOT_ERROR_CRYPTO_PK_PARSEKEY, 137
IOT_ERROR_CRYPTO_PK_SIGN, 137
IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE, 137
IOT_ERROR_CRYPTO_PK_VERIFY, 137
IOT_ERROR_CRYPTO_SHA256, 138
IOT_ERROR_CRYPTO_SS_KDF, 138
IOT_CRYPTO_ALIGN_B64_LEN
iot_crypto.h, 135
iot_crypto_base64_decode
iot_crypto.h, 139
iot_crypto_mbedtls.c, 90
iot_crypto_base64_decode_urlsafe
iot_crypto.h, 140
iot_crypto_mbedtls.c, 90
iot_crypto_base64_encode
iot_crypto.h, 140
iot_crypto_mbedtls.c, 91
iot_crypto_base64_encode_urlsafe
iot_crypto.h, 141
iot_crypto_mbedtls.c, 91
IOT_CRYPTO_CAL_B64_LEN
iot_crypto.h, 135
iot_crypto_cipher_aes
iot_crypto.h, 141
iot_crypto_mbedtls.c, 92
IOT_CRYPTO_CIPHER_AES256
iot_crypto.h, 139
IOT_CRYPTO_CIPHER_DECRYPT
iot_crypto.h, 139
IOT_CRYPTO_CIPHER_ENCRYPT
iot_crypto.h, 139
iot_crypto_cipher_get_align_size
iot_crypto.h, 142
iot_crypto_mbedtls.c, 93
iot_crypto_cipher_info, 33
iv, 33
iv_len, 34
key, 34
key_len, 34
mode, 34
type, 34
iot_crypto_cipher_info_t
iot_crypto.h, 138
iot_crypto_cipher_mode_t
iot_crypto.h, 138
iot_crypto_cipher_type_t
iot_crypto.h, 139
IOT_CRYPTO_CIPHER_UNKNOWN
iot_crypto.h, 139
iot_crypto_ecdh_gen_master_secret
iot_crypto.h, 143
iot_crypto_mbedtls.c, 94
iot_crypto_ecdh_params, 34
hash_token, 35
hash_token_len, 35
s_pubkey, 35
t_seckey, 35
iot_crypto_ecdh_params_t
iot_crypto.h, 138
iot_crypto_ed25519.c
iot_crypto_ed25519_convert_keypair, 87
iot_crypto_ed25519_convert_pubkey, 87
iot_crypto_ed25519_convert_seckey, 88
iot_crypto_ed25519_free_keypair, 88
iot_crypto_ed25519_init_keypair, 88
iot_crypto_ed25519_convert_keypair
iot_crypto.h, 143
iot_crypto_ed25519.c, 87
iot_crypto_ed25519_convert_pubkey
iot_crypto.h, 144
iot_crypto_ed25519.c, 87
iot_crypto_ed25519_convert_seckey

iot_crypto.h, 144
iot_crypto_ed25519.c, 88
iot_crypto_ed25519_free_keypair
iot_crypto.h, 144
iot_crypto_ed25519.c, 88
iot_crypto_ed25519_init_keypair
iot_crypto.h, 145
iot_crypto_ed25519.c, 88
iot_crypto_ed25519_keypair, 36
curve, 36
sign, 36
iot_crypto_ed25519_keypair_t
iot_crypto.h, 138
IOT_CRYPTO_ED25519_LEN
iot_crypto.h, 135
IOT_CRYPTO_IV_LEN
iot_crypto.h, 135
iot_crypto_keypair, 37
pubkey, 37
seckey, 37
iot_crypto_mbedtls.c
iot_crypto_base64_decode, 90
iot_crypto_base64_decode_urlsafe, 90
iot_crypto_base64_encode, 91
iot_crypto_base64_encode_urlsafe, 91
iot_crypto_cipher_aes, 92
iot_crypto_cipher_get_align_size, 93
iot_crypto_ecdh_gen_master_secret, 94
iot_crypto_sha256, 94
iot_crypto_pk_context, 38
fn, 39
info, 39
iot_crypto_pk_context_t
iot_crypto.h, 138
IOT_CRYPTO_PK_ED25519
iot_crypto.h, 139
iot_crypto_pk_free
iot_crypto.c, 311
iot_crypto.h, 145
iot_crypto_pk_funcs, 39
name, 40
sign, 40
verify, 40
iot_crypto_pk_funcs_t
iot_crypto.h, 138
iot_crypto_pk_info, 40
pubkey, 41
pubkey_len, 41
seckey, 41
seckey_len, 41
type, 41
iot_crypto_pk_info_t
iot_crypto.h, 138
iot_crypto_pk_init
iot_crypto.c, 311
iot_crypto.h, 145
IOT_CRYPTO_PK_RSA
iot_crypto.h, 139
iot_crypto_pk_sign
iot_crypto.c, 312
iot_crypto.h, 146
IOT_CRYPTO_PK_TYPE_ED25519
iot_crypto.h, 135
IOT_CRYPTO_PK_TYPE_RSA
iot_crypto.h, 135
iot_crypto_pk_type_t
iot_crypto.h, 139
IOT_CRYPTO_PK_UNKNOWN
iot_crypto.h, 139
iot_crypto_pk_verify
iot_crypto.c, 312
iot_crypto.h, 146
IOT_CRYPTO_SECRET_LEN
iot_crypto.h, 135
iot_crypto_sha256
iot_crypto.h, 147
iot_crypto_mbedtls.c, 94
IOT_CRYPTO_SHA256_LEN
iot_crypto.h, 135
IOT_CRYPTO_SIGNATURE_LEN
iot_crypto.h, 136
iot_crypto_ss.c
iot_crypto_ss_decrypt, 95
iot_crypto_ss_encrypt, 96
iot_crypto_ss_decrypt
iot_crypto.h, 148
iot_crypto_ss.c, 95
iot_crypto_ss_encrypt
iot_crypto.h, 148
iot_crypto_ss.c, 96
IOT_CTX
st_dev.h, 282
iot_ctx
iot_mqtt_ctx, 50
IOT_DEBUG
iot_debug.h, 151
iot_debug.h
COLOR_CYAN, 150
COLOR_END, 150
ENTER, 151
HIT, 151
iot_bsp_debug, 153
iot_bsp_debug_check_heap, 153
IOT_DEBUG, 151
IOT_DEBUG_CHECK, 151
IOT_DEBUG_LEVEL_DEBUG, 153
IOT_DEBUG_LEVEL_ERROR, 153
IOT_DEBUG_LEVEL_INFO, 153
IOT_DEBUG_LEVEL_MAX, 153
IOT_DEBUG_LEVEL_NONE, 153
iot_debug_level_t, 152
IOT_DEBUG_LEVEL_WARN, 153
IOT_DEBUG_PREFIX, 151
IOT_ERROR, 151
IOT_ERROR_CHECK, 151
IOT_INFO, 152

IOT_MEM_CHECK, 152
IOT_WARN, 152
IOT_WARN_CHECK, 152
LEAVE, 152
IOT_DEBUG_CHECK
iot_debug.h, 151
IOT_DEBUG_LEVEL_DEBUG
iot_debug.h, 153
IOT_DEBUG_LEVEL_ERROR
iot_debug.h, 153
IOT_DEBUG_LEVEL_INFO
iot_debug.h, 153
IOT_DEBUG_LEVEL_MAX
iot_debug.h, 153
IOT_DEBUG_LEVEL_NONE
iot_debug.h, 153
iot_debug_level_t
iot_debug.h, 152
IOT_DEBUG_LEVEL_WARN
iot_debug.h, 153
IOT_DEBUG_PREFIX
iot_debug.h, 151
IOT_DEFAULT_TIMEOUT
iot_internal.h, 166
IOT_DELAY
iot_os_util.h, 267
iot_devconf_prov_data, 42
device_onboarding_id, 43
device_type, 43
hashed_sn, 43
mnid, 43
ownership_validation_type, 43
pk_type, 43
setupid, 43
vid, 43
iot_device_cleanup
iot_api.c, 299
iot_internal.h, 173
iot_device_info, 44
firmware_version, 44
iot_device_prov_data, 45
cloud, 46
wifi, 46
iot_easysetup.h
iot_easysetup_create_ssid, 160
iot_easysetup_deinit, 160
iot_easysetup_init, 161
iot_easysetup_request_handler, 161
IOT_ERROR_EASYSETUP_400_BASE, 156
IOT_ERROR_EASYSETUP_500_BASE, 156
IOT_ERROR_EASYSETUP_CERTIFICATE_NOT_FOUND, 156
IOT_ERROR_EASYSETUP_CONFIRM_DENIED, 156
IOT_ERROR_EASYSETUP_CONFIRM_TIMEOUT, 156
IOT_ERROR_EASYSETUP_INTERNAL_CRITICAL_ERROR, 156
IOT_ERROR_EASYSETUP_INTERNAL_SERVER_ERROR, 156
IOT_ERROR_EASYSETUP_INTERNAL_WARNING, 156
IOT_ERROR_EASYSETUP_INVALID_BROKER_URL, 156
IOT_ERROR_EASYSETUP_INVALID_PAYLOAD, 156
IOT_ERROR_EASYSETUP_INVALID_PIN, 157
IOT_ERROR_EASYSETUP_INVALID_QR, 157
IOT_ERROR_EASYSETUP_INVALID_REQUEST, 157
IOT_ERROR_EASYSETUP_INVALID_SEQUENCE, 157
IOT_ERROR_EASYSETUP_MAX, 157
IOT_ERROR_EASYSETUP_NOT_SUPPORTED, 157
IOT_ERROR_EASYSETUP_PIN_NOT_FOUND, 157
IOT_ERROR_EASYSETUP_RAND_DECODE_ERROR, 157
IOT_ERROR_EASYSETUP_RPK_NOT_FOUND, 157
IOT_ERROR_EASYSETUP_SERIAL_NOT_FOUND, 157
IOT_ERROR_EASYSETUP_SHARED_KEY_CREATION_FAIL, 158
IOT_ERROR_EASYSETUP_SPUB_DECODE_ERROR, 158
IOT_ERROR_EASYSETUP_WIFI_SCANLIST_NOT_FOUND, 158
IOT_ES_URI_GET_DEVICEINFO, 158
IOT_ES_URI_GET_LOGS_DUMP, 158
IOT_ES_URI_GET_LOGS_SYSTEMINFO, 158
IOT_ES_URI_GET_POST_RESPONSE, 158
IOT_ES_URI_GET_WIFISCANINFO, 158
IOT_ES_URI_POST_CONFIRM, 158
IOT_ES_URI_POST_CONFIRMINFO, 158
IOT_ES_URI_POST_KEYINFO, 159
IOT_ES_URI_POST_LOGS, 159
IOT_ES_URI_POST_SETUPCOMPLETE, 159
IOT_ES_URI_POST_WIFIPROVISIONINGINFO, 159
IOT_OVF_TYPE_BUTTON, 159
IOT_OVF_TYPE_JUSTWORKS, 159
IOT_OVF_TYPE_PIN, 159
IOT_OVF_TYPE_QR, 159
OVF_BIT_BUTTON, 159
OVF_BIT_JUSTWORKS, 159
OVF_BIT_MAX_FEATURE, 159
OVF_BIT_PIN, 159
OVF_BIT_QR, 159
ownership_validation_feature, 159
iot_easysetup_create_ssid
iot_easysetup.h, 160
iot_easysetup_d2d.c, 106
iot_easysetup_crypto.c
iot_es_crypto_free_pk, 103

iot_es_crypto_init_pk, 103
iot_es_crypto_load_pk, 104
iot_easysetup_d2d.c
ES_CONFIRM_MAX_DELAY, 105
HASH_SIZE, 105
iot_easysetup_create_ssid, 106
iot_easysetup_request_handler, 107
JSON_H, 106
MAC_ADDR_BUFFER_SIZE, 106
PIN_SIZE, 106
st_conn_ownership_confirm, 107
URL_BUFFER_SIZE, 106
WIFIINFO_BUFFER_SIZE, 106
iot_easysetup_deinit
iot_easysetup.h, 160
iot_easysetup_http.c, 101
iot_easysetup_http.c
_iot_easysetup_gen_get_payload, 98
_iot_easysetup_gen_post_payload, 99
ARRAY_SIZE, 98
fs_close_custom, 100
fs_open_custom, 100
fs_state_free, 100
fs_state_init, 100
get_cgi_cmds, 102
httpd_cgi_handler, 100
httpd_post_begin, 101
httpd_post_finished, 101
httpd_post_receive_data, 101
iot_easysetup_deinit, 101
iot_easysetup_init, 102
LWIP_HTTPD_POST_MAX_PAYLOAD_LEN, 98
MAX_PAYLOAD_LENGTH, 98
post_cgi_cmds, 102
iot_easysetup_init
iot_easysetup.h, 161
iot_easysetup_http.c, 102
iot_easysetup_payload, 46
err, 47
payload, 47
step, 47
iot_easysetup_request
iot_api.c, 300
iot_internal.h, 173
iot_easysetup_request_handler
iot_easysetup.h, 161
iot_easysetup_d2d.c, 107
iot_easysetup_st_mqtt.c
iot_es_connect, 109
iot_es_disconnect, 109
iot_easysetup_step
iot_main.h, 188
IOT_EASYSETUP_STEP_CONFIRM
iot_main.h, 188
IOT_EASYSETUP_STEP_CONFIRMINFO
iot_main.h, 188
IOT_EASYSETUP_STEP_DEVICEINFO
iot_main.h, 188
IOT_EASYSETUP_STEP_KEYINFO
iot_main.h, 188
IOT_EASYSETUP_STEP_LOG_CREATE_DUMP
iot_main.h, 188
IOT_EASYSETUP_STEP_LOG_GET_DUMP
iot_main.h, 188
IOT_EASYSETUP_STEP_LOG_SYSTEMINFO
iot_main.h, 188
IOT_EASYSETUP_STEP_SETUPCOMPLETE
iot_main.h, 188
IOT_EASYSETUP_STEP_WIFIPROVIONINGINFO
iot_main.h, 188
IOT_EASYSETUP_STEP_WIFISCANINFO
iot_main.h, 188
IOT_ERROR
iot_debug.h, 151
iot_error.h
IOT_ERROR_BAD_REQ, 163
IOT_ERROR_CONNECT_FAIL, 163
IOT_ERROR_CRYPTO_BASE, 163
IOT_ERROR_DEINIT_FAIL, 163
IOT_ERROR_EASYSETUP_BASE, 163
IOT_ERROR_FS_CLOSE_FAIL, 163
IOT_ERROR_FS_DECRYPT_FAIL, 163
IOT_ERROR_FS_ENCRYPT_FAIL, 163
IOT_ERROR_FS_ENCRYPT_INIT, 163
IOT_ERROR_FS_NO_FILE, 163
IOT_ERROR_FS_OPEN_FAIL, 163
IOT_ERROR_FS_READ_FAIL, 163
IOT_ERROR_FS_REMOVE_FAIL, 163
IOT_ERROR_FS_WRITE_FAIL, 163
IOT_ERROR_INIT_FAIL, 163
IOT_ERROR_INVALID_ARGS, 163
IOT_ERROR_JWT_BASE, 163
IOT_ERROR_MEM_ALLOC, 163
IOT_ERROR_MQTT_CONNECT_FAIL, 163
IOT_ERROR_MQTT_NETCONN_FAIL, 163
IOT_ERROR_MQTT_PUBLISH_FAIL, 163
IOT_ERROR_MQTT_REJECT_CONNECT, 163
IOT_ERROR_MQTT_SERVER_UNAVAIL, 163
IOT_ERROR_NONE, 163
IOT_ERROR_NOT_IMPLEMENTED, 163
IOT_ERROR_NV_DATA_ERROR, 163
IOT_ERROR_NV_DATA_NOT_EXIST, 163
IOT_ERROR_PROV_FAIL, 163
IOT_ERROR_READ_FAIL, 163
IOT_ERROR_REG_UPDATED, 163
iot_error_t, 162, 163
IOT_ERROR_TIMEOUT, 163
IOT_ERROR_UNINITIALIZED, 163
IOT_ERROR_UUID_FAIL, 163
IOT_ERROR_WRITE_FAIL, 163
IOT_ERROR_BAD_REQ
iot_error.h, 163
IOT_ERROR_CHECK
iot_debug.h, 151
IOT_ERROR_CONNECT_FAIL
iot_error.h, 163

IOT_ERROR_CRYPTO_BASE
iot_error.h, 163
IOT_ERROR_CRYPTO_BASE64
iot_crypto.h, 136
IOT_ERROR_CRYPTO_BASE64_URLSAFE
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_ALIGN
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_IVLEN
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_KEYLEN
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_OUTSIZE
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_MODE
iot_crypto.h, 136
IOT_ERROR_CRYPTO_CIPHER_UNKNOWN_TYPE
iot_crypto.h, 136
IOT_ERROR_CRYPTO_ED_KEY_CONVERT
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_ECDH
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_INVALID_ARG
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_INVALID_CTX
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_INVALID_KEYLEN
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_NULL_FUNC
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_PARSEKEY
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_SIGN
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_UNKNOWN_KEYTYPE
iot_crypto.h, 137
IOT_ERROR_CRYPTO_PK_VERIFY
iot_crypto.h, 137
IOT_ERROR_CRYPTO_SHA256
iot_crypto.h, 138
IOT_ERROR_CRYPTO_SS_KDF
iot_crypto.h, 138
IOT_ERROR_DEINIT_FAIL
iot_error.h, 163
IOT_ERROR_EASYSETUP_400_BASE
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_500_BASE
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_BASE
iot_error.h, 163
IOT_ERROR_EASYSETUP_CERTIFICATE_NOT_FOUND
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_CONFIRM_DENIED
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_CONFIRM_TIMEOUT
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INTERNAL_CRITICAL_ERROR
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INTERNAL_SERVER_ERROR
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INTERNAL_WARNING
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INVALID_BROKER_URL
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INVALID_PAYLOAD
iot_easysetup.h, 156
IOT_ERROR_EASYSETUP_INVALID_PIN
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_INVALID_QR
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_INVALID_REQUEST
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_INVALID_SEQUENCE
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_MAX
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_NOT_SUPPORTED
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_PIN_NOT_FOUND
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_RAND_DECODE_ERROR
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_RPK_NOT_FOUND
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_SERIAL_NOT_FOUND
iot_easysetup.h, 157
IOT_ERROR_EASYSETUP_SHARED_KEY_CREATION_FAIL
iot_easysetup.h, 158
IOT_ERROR_EASYSETUP_SPUB_DECODE_ERROR
iot_easysetup.h, 158
IOT_ERROR_EASYSETUP_WIFI_SCANLIST_NOT_FOUND
iot_easysetup.h, 158
IOT_ERROR_FS_CLOSE_FAIL
iot_error.h, 163
IOT_ERROR_FS_DECRYPT_FAIL
iot_error.h, 163
IOT_ERROR_FS_ENCRYPT_FAIL
iot_error.h, 163
IOT_ERROR_FS_ENCRYPT_INIT
iot_error.h, 163
IOT_ERROR_FS_NO_FILE
iot_error.h, 163
IOT_ERROR_FS_OPEN_FAIL
iot_error.h, 163
IOT_ERROR_FS_READ_FAIL
iot_error.h, 163
IOT_ERROR_FS_REMOVE_FAIL
iot_error.h, 163
IOT_ERROR_FS_WRITE_FAIL
iot_error.h, 163
IOT_ERROR_INIT_FAIL
iot_error.h, 163
IOT_ERROR_INVALID_ARGS
iot_error.h, 163

IOT_ERROR_JWT_BASE
iot_error.h, 163
IOT_ERROR_JWT_CJSON
iot_jwt.h, 183
IOT_ERROR_JWT_INVALID_ARG
iot_jwt.h, 183
IOT_ERROR_JWT_MALLOC
iot_jwt.h, 183
IOT_ERROR_MEM_ALLOC
iot_error.h, 163
IOT_ERROR_MQTT_CONNECT_FAIL
iot_error.h, 163
IOT_ERROR_MQTT_NETCONN_FAIL
iot_error.h, 163
IOT_ERROR_MQTT_PUBLISH_FAIL
iot_error.h, 163
IOT_ERROR_MQTT_REJECT_CONNECT
iot_error.h, 163
IOT_ERROR_MQTT_SERVER_UNAVAIL
iot_error.h, 163
IOT_ERROR_NONE
iot_error.h, 163
IOT_ERROR_NOT_IMPLEMENTED
iot_error.h, 163
IOT_ERROR_NV_DATA_ERROR
iot_error.h, 163
IOT_ERROR_NV_DATA_NOT_EXIST
iot_error.h, 163
IOT_ERROR_PROV_FAIL
iot_error.h, 163
IOT_ERROR_READ_FAIL
iot_error.h, 163
IOT_ERROR_REG_UPDATED
iot_error.h, 163
iot_error_t
iot_error.h, 162, 163
IOT_ERROR_TIMEOUT
iot_error.h, 163
IOT_ERROR_UNINITIALIZED
iot_error.h, 163
IOT_ERROR_UUID_FAIL
iot_error.h, 163
IOT_ERROR_WRITE_FAIL
iot_error.h, 163
iot_es_connect
iot_easysetup_st_mqtt.c, 109
iot_internal.h, 174
iot_es_crypto_free_pk
iot_easysetup_crypto.c, 103
iot_internal.h, 175
iot_es_crypto_init_pk
iot_easysetup_crypto.c, 103
iot_internal.h, 175
iot_es_crypto_load_pk
iot_easysetup_crypto.c, 104
iot_internal.h, 175
iot_es_disconnect
iot_easysetup_st_mqtt.c, 109
iot_internal.h, 176
IOT_ES_URI_GET_DEVICEINFO
iot_easysetup.h, 158
IOT_ES_URI_GET_LOGS_DUMP
iot_easysetup.h, 158
IOT_ES_URI_GET_LOGS_SYSTEMINFO
iot_easysetup.h, 158
IOT_ES_URI_GET_POST_RESPONSE
iot_easysetup.h, 158
IOT_ES_URI_GET_WIFISCANINFO
iot_easysetup.h, 158
IOT_ES_URI_POST_CONFIRM
iot_easysetup.h, 158
IOT_ES_URI_POST_CONFIRMINFO
iot_easysetup.h, 158
IOT_ES_URI_POST_KEYINFO
iot_easysetup.h, 159
IOT_ES_URI_POST_LOGS
iot_easysetup.h, 159
IOT_ES_URI_POST_SETUPCOMPLETE
iot_easysetup.h, 159
IOT_ES_URI_POST_WIFIPROVISIONINGINFO
iot_easysetup.h, 159
IOT_EVENT
st_dev.h, 282
IOT_EVENT_BIT_ALL
iot_main.h, 186
IOT_EVENT_BIT_CAPABILITY
iot_main.h, 186
IOT_EVENT_BIT_COMMAND
iot_main.h, 186
IOT_EVENT_BIT_EASYSETUP_CONFIRM
iot_main.h, 186
IOT_EVENT_BIT_EASYSETUP_REQ
iot_main.h, 186
IOT_EVENT_BIT_EASYSETUP_RESP
iot_main.h, 186
iot_events
iot_context, 30
iot_get_time_in_ms
iot_api.c, 300
iot_internal.h, 176
iot_get_time_in_sec
iot_api.c, 301
iot_internal.h, 177
IOT_INFO
iot_debug.h, 152
iot_internal.h
iot_api_device_info_load, 167
iot_api_device_info_mem_free, 168
iot_api_onboarding_config_load, 168
iot_api_onboarding_config_mem_free, 169
iot_api_prov_data_mem_free, 169
iot_api_read_device_identity, 170
iot_cap_call_init_cb, 171
iot_cap_sub_cb, 171
iot_command_send, 172
IOT_DEFAULT_TIMEOUT, 166

iot_device_cleanup, 173
iot_easysetup_request, 173
iot_es_connect, 174
iot_es_crypto_free_pk, 175
iot_es_crypto_init_pk, 175
iot_es_crypto_load_pk, 175
iot_es_disconnect, 176
iot_get_time_in_ms, 176
iot_get_time_in_sec, 177
iot_mqtt_connect, 177
iot_mqtt_disconnect, 178
IOT_MQTT_KEEPALIVE_INTERVAL, 166
iot_mqtt_publish, 178
iot_mqtt_registration, 179
iot_mqtt_subscribe, 180
iot_mqtt_unsubscribe, 180
iot_noti_sub_cb, 181
IOT_PAYLOAD_SIZE, 166
IOT_PUB_QUEUE_LENGTH, 166
IOT_PUB_TOPIC_EVENT, 166
IOT_PUB_TOPIC_REGISTRATION, 166
IOT_QUEUE_LENGTH, 166
iot_state_update, 181
IOT_SUB_TOPIC_COMMAND, 166
IOT_SUB_TOPIC_NOTIFICATION, 166
IOT_SUB_TOPIC_REGISTRATION, 167
IOT_TASK_NAME, 167
IOT_TASK_PRIORITY, 167
IOT_TASK_STACK_SIZE, 167
IOT_TOPIC_SIZE, 167
iot_jwt.c
iot_jwt_create, 314
iot_jwt.h
IOT_ERROR_JWT_CJSON, 183
IOT_ERROR_JWT_INVALID_ARG, 183
IOT_ERROR_JWT_MALLOC, 183
iot_jwt_create, 183
iot_jwt_create
iot_jwt.c, 314
iot_jwt.h, 183
iot_mac, 47
addr, 48
iot_main.c
EASYSETUP_TIMEOUT_MS, 315
NEXT_STATE_TIMEOUT_MS, 315
st_conn_cleanup, 315
st_conn_init, 316
st_conn_start, 317
iot_main.h
_IOT_NOTI_TYPE_DEV_DELETED, 187
_IOT_NOTI_TYPE_JWT_EXPIRED, 187
_IOT_NOTI_TYPE_QUOTA_REACHED, 187
_IOT_NOTI_TYPE_RATE_LIMIT, 187
_IOT_NOTI_TYPE_UNKNOWN, 187
_iot_noti_type, 187
IOT_BUF_RX_SIZE, 186
IOT_BUF_TX_SIZE, 186
iot_cap_handle_list_t, 187
IOT_CMD_STATE_HANDLE, 188
IOT_COMMAND_CHECK_CLOUD_STATE, 188
IOT_COMMAND_CHECK_PROV_STATUS, 188
IOT_COMMAND_CLOUD_CONNECTING, 188
IOT_COMMAND_CLOUD_REGISTERED, 188
IOT_COMMAND_CLOUD_REGISTERING, 188
IOT_COMMAND_NETWORK_MODE, 188
IOT_COMMAND_NOTIFICATION_RECEIVED, 188
IOT_COMMAND_READY_TO_CTL, 188
IOT_COMMAND_SELF_CLEANUP, 188
iot_command_type, 187
IOT_COMMAND_TYPE_MAX, 188
iot_connect_type, 188
IOT_CONNECT_TYPE_COMMUNICATION, 188
IOT_CONNECT_TYPE_REGISTRATION, 188
iot_easysetup_step, 188
IOT_EASYSETUP_STEP_CONFIRM, 188
IOT_EASYSETUP_STEP_CONFIRMINFO, 188
IOT_EASYSETUP_STEP_DEVICEINFO, 188
IOT_EASYSETUP_STEP_KEYINFO, 188
IOT_EASYSETUP_STEP_LOG_CREATE_DUMP, 188
IOT_EASYSETUP_STEP_LOG_GET_DUMP, 188
IOT_EASYSETUP_STEP_LOG_SYSTEMINFO, 188
IOT_EASYSETUP_STEP_SETUPCOMPLETE, 188
IOT_EASYSETUP_STEP_WIFIPROVIONINGINFO, 188
IOT_EASYSETUP_STEP_WIFISCANINFO, 188
IOT_EVENT_BIT_ALL, 186
IOT_EVENT_BIT_CAPABILITY, 186
IOT_EVENT_BIT_COMMAND, 186
IOT_EVENT_BIT_EASYSETUP_CONFIRM, 186
IOT_EVENT_BIT_EASYSETUP_REQ, 186
IOT_EVENT_BIT_EASYSETUP_RESP, 186
IOT_REG_UUID_STR_LEN, 187
IOT_STATE_CHANGE_FAILED, 189
IOT_STATE_CLOUD_CONNECTED, 189
IOT_STATE_CLOUD_CONNECTING, 189
IOT_STATE_CLOUD_DISCONNECTED, 189
IOT_STATE_CLOUD_REGISTERING, 189
IOT_STATE_INITIALIZED, 189
iot_state_opt, 188
IOT_STATE_OPT_NEED_INTERACT, 188
IOT_STATE_OPT_NONE, 188
IOT_STATE_PROV_CONFIRMING, 189
IOT_STATE_PROV_DONE, 189
IOT_STATE_PROV_ENTER, 189
iot_state_t, 187
iot_state_type, 189
IOT_STATE_UNKNOWN, 189
IOT_WIFI_PROV_PASSWORD_LEN, 187
IOT_WIFI_PROV_SSID_LEN, 187
IOT_MEM_CHECK
iot_debug.h, 152
iot_mqtt.c

_iot_mqtt_cmd_sub_callback, 319
_iot_mqtt_noti_sub_callback, 319
iot_mqtt_connect, 320
iot_mqtt_disconnect, 321
iot_mqtt_publish, 321
iot_mqtt_registration, 322
iot_mqtt_subscribe, 322
iot_mqtt_unsubscribe, 323
iot_mqtt_client.c
cycle, 349
deliverMessage, 349
keepalive, 350
MQTTCleanSession, 351
MQTTClientInit, 351
MQTTCloseSession, 352
MQTTConnect, 352
MQTTConnectWithResults, 353
MQTTDisconnect, 354
MQTTPublish, 355
MQTTSetMessageHandler, 356
MQTTSubscribe, 357
MQTTSubscribeWithResults, 357
MQTTUnsubscribe, 358
MQTTYield, 360
waitfor, 361
iot_mqtt_client.h
CONFIG_STDK_MQTT_DYNAMIC_BUFFER, 214
CONFIG_STDK_MQTT_PING_RETRY, 214
CONFIG_STDK_MQTT_PUBLISH_RETRY, 214
CONFIG_STDK_MQTT_RECV_BUFFER, 214
CONFIG_STDK_MQTT_RECV_CYCLE, 214
CONFIG_STDK_MQTT_SEND_BUFFER, 214
CONFIG_STDK_MQTT_SEND_CYCLE, 214
DefaultClient, 214
DLLExport, 214
DLLImport, 214
IOT_MQTT_PRIORITY, 214
IOT_MQTT_STACK_SIZE, 215
MAX_MESSAGE_HANDLERS, 215
MAX_PACKET_ID, 215
MessageData, 215
messageHandler, 215
MQTT_BUFFER_OVERFLOW, 216
MQTT_DISCONNECTED, 216
MQTT_FAILURE, 216
MQTT_SUCCESS, 216
MQTTClient, 215
MQTTClientInit, 216
MQTTConnackData, 215
MQTTConnect, 217
MQTTConnectWithResults, 218
MQTTDisconnect, 219
MQTTMessage, 215
MQTTPublish, 220
MQTTSetMessageHandler, 221
MQTTSubackData, 215
MQTTSubscribe, 222
MQTTSubscribeWithResults, 222
MQTTUnsubscribe, 223
MQTTYield, 225
QoS, 215
QOS0, 216
QOS1, 216
QOS2, 216
returnCode, 216
SUBFAIL, 216
iot_mqtt_connect
iot_internal.h, 177
iot_mqtt.c, 320
iot_mqtt_connect.h
connack_return_codes, 228
DLLExport, 227
DLLImport, 227
MQTT_BAD_USERNAME_OR_PASSWORD, 228
MQTT_CLIENTID_REJECTED, 228
MQTT_CONNECTION_ACCEPTED, 228
MQTT_NOT_AUTHORIZED, 228
MQTT_SERVER_UNAVAILABLE, 228
MQTT_UNNACCEPTABLE_PROTOCOL, 228
MQTTDeserialize_connack, 228
MQTTDeserialize_connect, 229
MQTTPacket_connectData_initializer, 227
MQTTPacket_willOptions_initializer, 228
MQTTSerialize_connack, 229
MQTTSerialize_connect, 229
MQTTSerialize_connect_size, 231
MQTTSerialize_disconnect, 231
MQTTSerialize_disconnect_size, 232
MQTTSerialize_pingreq, 232
MQTTSerialize_pingreq_size, 233
iot_mqtt_connect_client.c
MQTTDeserialize_connack, 362
MQTTSerialize_connect, 363
MQTTSerialize_connect_size, 363
MQTTSerialize_connectLength, 364
MQTTSerialize_disconnect, 364
MQTTSerialize_disconnect_size, 366
MQTTSerialize_pingreq, 366
MQTTSerialize_pingreq_size, 367
MQTTSerialize_zero, 367
iot_mqtt_connect_server.c
min, 368
MQTTDeserialize_connect, 368
MQTTPacket_checkVersion, 370
MQTTSerialize_connack, 370
iot_mqtt_ctx, 48
buf, 50
cli, 50
cmd_filter, 50
iot_ctx, 50
mqtt_connected, 50
net, 50
noti_filter, 51
readbuf, 51
iot_mqtt_deserialize_publish.c
min, 371

MQTTDeserialize_ack, 371
MQTTDeserialize_publish, 372
iot_mqtt_disconnect
iot_internal.h, 178
iot_mqtt.c, 321
iot_mqtt_format.c
MQTTPacket_getName, 373
MQTTPacket_names, 375
MQTTStringFormat_ack, 373
MQTTStringFormat_connack, 374
MQTTStringFormat_connect, 374
MQTTStringFormat_publish, 374
MQTTStringFormat_suback, 374
MQTTStringFormat_subscribe, 374
MQTTStringFormat_unsubscribe, 374
iot_mqtt_format.h
MQTTFormat_toClientString, 234
MQTTFormat_toServerString, 234
MQTTPacket_getName, 234
MQTTStringFormat_ack, 235
MQTTStringFormat_connack, 235
MQTTStringFormat_connect, 235
MQTTStringFormat_publish, 235
MQTTStringFormat_suback, 235
MQTTStringFormat_subscribe, 235
MQTTStringFormat_unsubscribe, 236
IOT_MQTT_KEEPALIVE_INTERVAL
iot_internal.h, 166
iot_mqtt_packet.c
bufchar, 376
getLenStringLen, 376
MAX_NO_OF_REMAINING_LENGTH_BYTES, 376
MQTTPacket_decode, 376
MQTTPacket_decodeBuf, 377
MQTTPacket_encode, 377
MQTTPacket_equals, 377
MQTTPacket_len, 378
MQTTPacket_msgTypesToString, 378
MQTTPacket_read, 378
MQTTPacket_readnb, 379
MQTTstrlen, 379
readChar, 380
readInt, 380
readMQTTLenString, 380
writeChar, 381
writeCString, 381
writeInt, 382
writeMQTTString, 382
iot_mqtt_packet.h
CONNACK, 238
CONNECT, 238
DISCONNECT, 238
DLLExport, 237
DLLImport, 238
errors, 238
MQTTDeserialize_ack, 238
MQTTPACKET_BUFFER_TOO_SHORT, 238
MQTTPacket_decode, 239
MQTTPacket_decodeBuf, 239
MQTTPacket_encode, 239
MQTTPacket_equals, 240
MQTTPacket_len, 240
MQTTPacket_msgTypesToString, 241
MQTTPacket_read, 241
MQTTPACKET_READ_COMPLETE, 238
MQTTPACKET_READ_ERROR, 238
MQTTPacket_readnb, 241
MQTTSerialize_ack, 242
MQTTSerialize_ack_size, 242
MQTTString_initializer, 238
MQTTstrlen, 242
msgTypes, 238
PINGREQ, 238
PINGRESP, 238
PUBACK, 238
PUBCOMP, 238
PUBLISH, 238
PUBREC, 238
PUBREL, 238
readChar, 243
readInt, 243
readMQTTLenString, 243
SUBACK, 238
SUBSCRIBE, 238
UNSUBACK, 238
UNSUBSCRIBE, 238
writeChar, 244
writeCString, 244
writeInt, 245
writeMQTTString, 245
IOT_MQTT_PRIORITY
iot_mqtt_client.h, 214
iot_mqtt_publish
iot_internal.h, 178
iot_mqtt.c, 321
iot_mqtt_publish.h
DLLExport, 247
DLLImport, 247
MQTTDeserialize_publish, 247
MQTTSerialize_puback, 248
MQTTSerialize_pubcomp, 248
MQTTSerialize_publish, 248
MQTTSerialize_publish_header, 249
MQTTSerialize_publish_size, 249
MQTTSerialize_pubrel, 250
iot_mqtt_registration
iot_internal.h, 179
iot_mqtt.c, 322
iot_mqtt_serialize_publish.c
MQTTSerialize_ack, 384
MQTTSerialize_ack_size, 384
MQTTSerialize_puback, 384
MQTTSerialize_pubcomp, 385
MQTTSerialize_publish, 385
MQTTSerialize_publish_header, 386

MQTTSerialize_publish_size, 386
MQTTSerialize_publishLength, 387
MQTTSerialize_pubrel, 387
IOT_MQTT_STACK_SIZE
iot_mqtt_client.h, 215
iot_mqtt_stacktrace.h
FUNC_ENTRY, 251
FUNC_ENTRY_MAX, 252
FUNC_ENTRY_MED, 252
FUNC_ENTRY_NOLOG, 252
FUNC_EXIT, 252
FUNC_EXIT_MAX, 252
FUNC_EXIT_MAX_RC, 252
FUNC_EXIT_MED, 252
FUNC_EXIT_MED_RC, 252
FUNC_EXIT_NOLOG, 252
FUNC_EXIT_RC, 252
NOSTACKTRACE, 252
iot_mqtt_subscribe
iot_internal.h, 180
iot_mqtt.c, 322
iot_mqtt_subscribe.h
DLLExport, 253
DLLImport, 253
MQTTDeserialize_suback, 253
MQTTDeserialize_subscribe, 254
MQTTSerialize_suback, 254
MQTTSerialize_subscribe, 256
MQTTSerialize_subscribe_size, 256
iot_mqtt_subscribe_client.c
MQTTDeserialize_suback, 388
MQTTSerialize_subscribe, 389
MQTTSerialize_subscribe_size, 390
MQTTSerialize_subscribeLength, 390
iot_mqtt_subscribe_server.c
MQTTDeserialize_subscribe, 391
MQTTSerialize_suback, 392
iot_mqtt_unsubscribe
iot_internal.h, 180
iot_mqtt.c, 323
iot_mqtt_unsubscribe.h
DLLExport, 258
DLLImport, 258
MQTTDeserialize_unsuback, 258
MQTTDeserialize_unsubscribe, 258
MQTTSerialize_unsuback, 259
MQTTSerialize_unsubscribe, 259
MQTTSerialize_unsubscribe_size, 260
iot_mqtt_unsubscribe_client.c
MQTTDeserialize_unsuback, 393
MQTTSerialize_unsubscribe, 393
MQTTSerialize_unsubscribe_size, 394
MQTTSerialize_unsubscribeLength, 395
iot_mqtt_unsubscribe_server.c
MQTTDeserialize_unsubscribe, 396
MQTTSerialize_unsuback, 397
iot_net_socket, 51
iot_os_net.h, 262
read, 52
read_count, 52
socket, 52
write, 52
iot_noti_data_t, 53
raw, 54
type, 54
iot_noti_sub_cb
iot_capability.c, 304
iot_internal.h, 181
iot_noti_type
st_dev.h, 283
IOT_NOTI_TYPE_DEV_DELETED
st_dev.h, 284
IOT_NOTI_TYPE_QUOTA_REACHED
st_dev.h, 284
IOT_NOTI_TYPE_RATE_LIMIT
st_dev.h, 284
iot_noti_type_t
st_dev.h, 282
IOT_NOTI_TYPE_UNKNOWN
st_dev.h, 284
iot_nv_data.c
_iot_nv_read_data, 326
_iot_nv_write_data, 326
iot_nv_deinit, 327
iot_nv_erase, 328
iot_nv_erase_prov_data, 329
iot_nv_get_client_certificate, 329
iot_nv_get_cloud_prov_data, 330
iot_nv_get_device_id, 331
iot_nv_get_private_key, 332
iot_nv_get_prov_data, 333
iot_nv_get_public_key, 333
iot_nv_get_root_certificate, 334
iot_nv_get_serial_number, 334
iot_nv_get_wifi_prov_data, 335
iot_nv_init, 336
iot_nv_set_cloud_prov_data, 337
iot_nv_set_device_id, 338
iot_nv_set_prov_data, 339
iot_nv_set_wifi_prov_data, 339
IOT_NVD_MAX_BSSID_LEN, 325
IOT_NVD_MAX_DATA_LEN, 325
IOT_NVD_MAX_ID_LEN, 325
IOT_NVD_MAX_PW_LEN, 326
IOT_NVD_MAX_UID_LEN, 326
iot_nv_data.h
iot_nv_deinit, 191
iot_nv_erase, 192
iot_nv_erase_prov_data, 192
iot_nv_get_client_certificate, 193
iot_nv_get_cloud_prov_data, 194
iot_nv_get_device_id, 195
iot_nv_get_private_key, 196
iot_nv_get_prov_data, 197
iot_nv_get_public_key, 197
iot_nv_get_root_certificate, 198

iot_nv_get_serial_number, 198
iot_nv_get_wifi_prov_data, 199
iot_nv_init, 200
iot_nv_set_cloud_prov_data, 201
iot_nv_set_device_id, 202
iot_nv_set_prov_data, 203
iot_nv_set_wifi_prov_data, 203
IOT_NVD_AP_AUTH_TYPE, 191
IOT_NVD_AP_BSSID, 191
IOT_NVD_AP_PASS, 191
IOT_NVD_AP_SSID, 191
IOT_NVD_CA_CERT, 191
IOT_NVD_CLOUD_PROV_STATUS, 191
IOT_NVD_DEVICE_ID, 191
IOT_NVD_LABEL, 191
IOT_NVD_LOCATION_ID, 191
IOT_NVD_MAX, 191
IOT_NVD_PRIVATE_KEY, 191
IOT_NVD_PUBLIC_KEY, 191
IOT_NVD_ROOM_ID, 191
IOT_NVD_SERIAL_NUM, 191
IOT_NVD_SERVER_PORT, 191
IOT_NVD_SERVER_URL, 191
IOT_NVD_SUB_CERT, 191
iot_nvd_t, 190
IOT_NVD_WIFI_PROV_STATUS, 191
iot_nv_deinit
iot_nv_data.c, 327
iot_nv_data.h, 191
iot_nv_erase
iot_nv_data.c, 328
iot_nv_data.h, 192
iot_nv_erase_prov_data
iot_nv_data.c, 329
iot_nv_data.h, 192
iot_nv_get_client_certificate
iot_nv_data.c, 329
iot_nv_data.h, 193
iot_nv_get_cloud_prov_data
iot_nv_data.c, 330
iot_nv_data.h, 194
iot_nv_get_device_id
iot_nv_data.c, 331
iot_nv_data.h, 195
iot_nv_get_private_key
iot_nv_data.c, 332
iot_nv_data.h, 196
iot_nv_get_prov_data
iot_nv_data.c, 333
iot_nv_data.h, 197
iot_nv_get_public_key
iot_nv_data.c, 333
iot_nv_data.h, 197
iot_nv_get_root_certificate
iot_nv_data.c, 334
iot_nv_data.h, 198
iot_nv_get_serial_number
iot_nv_data.c, 334
iot_nv_data.h, 198
iot_nv_get_wifi_prov_data
iot_nv_data.c, 335
iot_nv_data.h, 199
iot_nv_init
iot_nv_data.c, 336
iot_nv_data.h, 200
iot_nv_set_cloud_prov_data
iot_nv_data.c, 337
iot_nv_data.h, 201
iot_nv_set_device_id
iot_nv_data.c, 338
iot_nv_data.h, 202
iot_nv_set_prov_data
iot_nv_data.c, 339
iot_nv_data.h, 203
iot_nv_set_wifi_prov_data
iot_nv_data.c, 339
iot_nv_data.h, 203
IOT_NVD_AP_AUTH_TYPE
iot_nv_data.h, 191
IOT_NVD_AP_BSSID
iot_nv_data.h, 191
IOT_NVD_AP_PASS
iot_nv_data.h, 191
IOT_NVD_AP_SSID
iot_nv_data.h, 191
IOT_NVD_CA_CERT
iot_nv_data.h, 191
IOT_NVD_CLOUD_PROV_STATUS
iot_nv_data.h, 191
IOT_NVD_DEVICE_ID
iot_nv_data.h, 191
IOT_NVD_LABEL
iot_nv_data.h, 191
IOT_NVD_LOCATION_ID
iot_nv_data.h, 191
IOT_NVD_MAX
iot_nv_data.h, 191
IOT_NVD_MAX_BSSID_LEN
iot_nv_data.c, 325
IOT_NVD_MAX_DATA_LEN
iot_nv_data.c, 325
IOT_NVD_MAX_ID_LEN
iot_nv_data.c, 325
IOT_NVD_MAX_PW_LEN
iot_nv_data.c, 326
IOT_NVD_MAX_UID_LEN
iot_nv_data.c, 326
IOT_NVD_PRIVATE_KEY
iot_nv_data.h, 191
IOT_NVD_PUBLIC_KEY
iot_nv_data.h, 191
IOT_NVD_ROOM_ID
iot_nv_data.h, 191
IOT_NVD_SERIAL_NUM
iot_nv_data.h, 191
IOT_NVD_SERVER_PORT

iot_nv_data.h, 191
IOT_NVD_SERVER_URL
iot_nv_data.h, 191
IOT_NVD_SUB_CERT
iot_nv_data.h, 191
iot_nvd_t
iot_nv_data.h, 190
IOT_NVD_WIFI_PROV_STATUS
iot_nv_data.h, 191
iot_os_delay
iot_os_util.h, 268
iot_os_util_freertos.c, 419
iot_os_eventgroup
iot_os_util.h, 267
iot_os_eventgroup_clear_bits
iot_os_util.h, 268
iot_os_util_freertos.c, 419
iot_os_eventgroup_create
iot_os_util.h, 269
iot_os_util_freertos.c, 419
iot_os_eventgroup_delete
iot_os_util.h, 269
iot_os_util_freertos.c, 420
iot_os_eventgroup_set_bits
iot_os_util.h, 269
iot_os_util_freertos.c, 420
iot_os_eventgroup_wait_bits
iot_os_util.h, 270
iot_os_util_freertos.c, 421
IOT_OS_FALSE
iot_os_util.h, 267
iot_os_false
iot_os_util.h, 279
iot_os_util_freertos.c, 430
IOT_OS_MAX_DELAY
iot_os_util.h, 267
iot_os_max_delay
iot_os_util.h, 279
iot_os_util_freertos.c, 431
iot_os_mutex, 54
iot_os_util.h, 267
sem, 55
iot_os_mutex_init
iot_os_util.h, 271
iot_os_util_freertos.c, 421
iot_os_mutex_lock
iot_os_util.h, 271
iot_os_util_freertos.c, 423
iot_os_mutex_unlock
iot_os_util.h, 272
iot_os_util_freertos.c, 424
iot_os_net.h
iot_net_socket, 262
iot_os_net_connet, 262
iot_os_net_disconnect, 262
iot_os_net_init, 262
iot_os_net_print_status, 264
iot_os_net_select, 264
iot_os_net_connet
iot_os_net.h, 262
iot_os_net_freertos.c, 415
iot_os_net_disconnect
iot_os_net.h, 262
iot_os_net_freertos.c, 415
iot_os_net_freertos.c
iot_os_net_connet, 415
iot_os_net_disconnect, 415
iot_os_net_init, 415
iot_os_net_print_status, 416
iot_os_net_select, 416
iot_os_net_init
iot_os_net.h, 262
iot_os_net_freertos.c, 415
iot_os_net_print_status
iot_os_net.h, 264
iot_os_net_freertos.c, 416
iot_os_net_select
iot_os_net.h, 264
iot_os_net_freertos.c, 416
iot_os_queue
iot_os_util.h, 267
iot_os_queue_create
iot_os_util.h, 272
iot_os_util_freertos.c, 424
iot_os_queue_delete
iot_os_util.h, 273
iot_os_util_freertos.c, 424
iot_os_queue_receive
iot_os_util.h, 273
iot_os_util_freertos.c, 425
iot_os_queue_reset
iot_os_util.h, 273
iot_os_util_freertos.c, 425
iot_os_queue_send
iot_os_util.h, 274
iot_os_util_freertos.c, 425
iot_os_sem
iot_os_util.h, 267
iot_os_thread
iot_os_util.h, 268
iot_os_thread_create
iot_os_util.h, 274
iot_os_util_freertos.c, 426
iot_os_thread_delete
iot_os_util.h, 275
iot_os_util_freertos.c, 427
iot_os_thread_yield
iot_os_util.h, 275
iot_os_util_freertos.c, 427
iot_os_timer
iot_os_util.h, 268
iot_os_timer_count_ms
iot_os_util.h, 276
iot_os_util_freertos.c, 427
iot_os_timer_destroy
iot_os_util.h, 276

iot_os_util_freertos.c, 428
iot_os_timer_init
iot_os_util.h, 277
iot_os_util_freertos.c, 428
iot_os_timer_isexpired
iot_os_util.h, 277
iot_os_util_freertos.c, 429
iot_os_timer_left_ms
iot_os_util.h, 278
iot_os_util_freertos.c, 430
IOT_OS_TRUE
iot_os_util.h, 267
iot_os_true
iot_os_util.h, 279
iot_os_util_freertos.c, 431
iot_os_util.h
IOT_DELAY, 267
iot_os_delay, 268
iot_os_eventgroup, 267
iot_os_eventgroup_clear_bits, 268
iot_os_eventgroup_create, 269
iot_os_eventgroup_delete, 269
iot_os_eventgroup_set_bits, 269
iot_os_eventgroup_wait_bits, 270
IOT_OS_FALSE, 267
iot_os_false, 279
IOT_OS_MAX_DELAY, 267
iot_os_max_delay, 279
iot_os_mutex, 267
iot_os_mutex_init, 271
iot_os_mutex_lock, 271
iot_os_mutex_unlock, 272
iot_os_queue, 267
iot_os_queue_create, 272
iot_os_queue_delete, 273
iot_os_queue_receive, 273
iot_os_queue_reset, 273
iot_os_queue_send, 274
iot_os_sem, 267
iot_os_thread, 268
iot_os_thread_create, 274
iot_os_thread_delete, 275
iot_os_thread_yield, 275
iot_os_timer, 268
iot_os_timer_count_ms, 276
iot_os_timer_destroy, 276
iot_os_timer_init, 277
iot_os_timer_isexpired, 277
iot_os_timer_left_ms, 278
IOT_OS_TRUE, 267
iot_os_true, 279
iot_os_util_freertos.c
Freertos_Timer, 418
iot_os_delay, 419
iot_os_eventgroup_clear_bits, 419
iot_os_eventgroup_create, 419
iot_os_eventgroup_delete, 420
iot_os_eventgroup_set_bits, 420
iot_os_eventgroup_wait_bits, 421
iot_os_false, 430
iot_os_max_delay, 431
iot_os_mutex_init, 421
iot_os_mutex_lock, 423
iot_os_mutex_unlock, 424
iot_os_queue_create, 424
iot_os_queue_delete, 424
iot_os_queue_receive, 425
iot_os_queue_reset, 425
iot_os_queue_send, 425
iot_os_thread_create, 426
iot_os_thread_delete, 427
iot_os_thread_yield, 427
iot_os_timer_count_ms, 427
iot_os_timer_destroy, 428
iot_os_timer_init, 428
iot_os_timer_isexpired, 429
iot_os_timer_left_ms, 430
iot_os_true, 431
IOT_OVF_TYPE_BUTTON
iot_easysetup.h, 159
IOT_OVF_TYPE_JUSTWORKS
iot_easysetup.h, 159
IOT_OVF_TYPE_PIN
iot_easysetup.h, 159
IOT_OVF_TYPE_QR
iot_easysetup.h, 159
IOT_PAYLOAD_SIZE
iot_internal.h, 166
iot_pin_t, 55
pin, 55
st_dev.h, 282
IOT_POWEROFF
iot_bsp_system.h, 123
IOT_PUB_QUEUE_LENGTH
iot_internal.h, 166
IOT_PUB_TOPIC_EVENT
iot_internal.h, 166
IOT_PUB_TOPIC_REGISTRATION
iot_internal.h, 166
IOT_QUEUE_LENGTH
iot_internal.h, 166
iot_random_uuid_from_mac
iot_uuid.c, 346
iot_uuid.h, 210
IOT_REBOOT
iot_bsp_system.h, 123
iot_reg_data
iot_context, 30
IOT_REG_UUID_STR_LEN
iot_main.h, 187
iot_registered_data, 56
deviceId, 56
new_reged, 57
updated, 57
IOT_SOFT_AP_CHANNEL
iot_bsp_wifi.h, 126

iot_stat_lv
st_dev.h, 284
IOT_STAT_LV_DONE
st_dev.h, 284
IOT_STAT_LV_FAIL
st_dev.h, 284
IOT_STAT_LV_START
st_dev.h, 284
IOT_STAT_LV_STAY
st_dev.h, 284
iot_stat_lv_t
st_dev.h, 282
iot_state
iot_state_data, 58
IOT_STATE_CHANGE_FAILED
iot_main.h, 189
IOT_STATE_CLOUD_CONNECTED
iot_main.h, 189
IOT_STATE_CLOUD_CONNECTING
iot_main.h, 189
IOT_STATE_CLOUD_DISCONNECTED
iot_main.h, 189
IOT_STATE_CLOUD_REGISTERING
iot_main.h, 189
iot_state_data, 57
iot_state, 58
opt, 58
IOT_STATE_INITIALIZED
iot_main.h, 189
iot_state_opt
iot_main.h, 188
IOT_STATE_OPT_NEED_INTERACT
iot_main.h, 188
IOT_STATE_OPT_NONE
iot_main.h, 188
IOT_STATE_PROV_CONFIRMING
iot_main.h, 189
IOT_STATE_PROV_DONE
iot_main.h, 189
IOT_STATE_PROV_ENTER
iot_main.h, 189
iot_state_t
iot_main.h, 187
iot_state_type
iot_main.h, 189
IOT_STATE_UNKNOWN
iot_main.h, 189
iot_state_update
iot_api.c, 301
iot_internal.h, 181
iot_status
st_dev.h, 284
IOT_STATUS_ALL
st_dev.h, 284
IOT_STATUS_CONNECTING
st_dev.h, 284
IOT_STATUS_IDLE
st_dev.h, 284
IOT_STATUS_NEED_INTERACT
st_dev.h, 284
IOT_STATUS_PROVISIONING
st_dev.h, 284
iot_status_t
st_dev.h, 282
IOT_SUB_TOPIC_COMMAND
iot_internal.h, 166
IOT_SUB_TOPIC_NOTIFICATION
iot_internal.h, 166
IOT_SUB_TOPIC_REGISTRATION
iot_internal.h, 167
IOT_TASK_NAME
iot_internal.h, 167
IOT_TASK_PRIORITY
iot_internal.h, 167
IOT_TASK_STACK_SIZE
iot_internal.h, 167
IOT_TOPIC_SIZE
iot_internal.h, 167
iot_util.c
iot_util_convert_channel_freq, 341
iot_util_convert_mac_str, 342
iot_util_convert_str_mac, 342
iot_util_convert_str_uuid, 343
iot_util_convert_uuid_str, 343
iot_util_get_random_uuid, 344
iot_util_url_parse, 345
iot_util.h
iot_util_convert_channel_freq, 206
iot_util_convert_mac_str, 206
iot_util_convert_str_mac, 207
iot_util_convert_str_uuid, 207
iot_util_convert_uuid_str, 208
iot_util_get_random_uuid, 208
iot_util_url_parse, 209
iot_util_convert_channel_freq
iot_util.c, 341
iot_util.h, 206
iot_util_convert_mac_str
iot_util.c, 342
iot_util.h, 206
iot_util_convert_str_mac
iot_util.c, 342
iot_util.h, 207
iot_util_convert_str_uuid
iot_util.c, 343
iot_util.h, 207
iot_util_convert_uuid_str
iot_util.c, 343
iot_util.h, 208
iot_util_get_random_uuid
iot_util.c, 344
iot_util.h, 208
iot_util_url_parse
iot_util.c, 345
iot_util.h, 209
iot_uuid, 58

id, 59
iot_uuid.c
iot_random_uuid_from_mac, 346
iot_uuid_from_mac, 347
iot_uuid.h
iot_random_uuid_from_mac, 210
iot_uuid_from_mac, 211
iot_uuid_from_mac
iot_uuid.c, 347
iot_uuid.h, 211
IOT_WARN
iot_debug.h, 152
IOT_WARN_CHECK
iot_debug.h, 152
IOT_WIFI_AUTH_MAX
iot_bsp_wifi.h, 127
iot_wifi_auth_mode_t
iot_bsp_wifi.h, 126
IOT_WIFI_AUTH_OPEN
iot_bsp_wifi.h, 127
IOT_WIFI_AUTH_WEP
iot_bsp_wifi.h, 127
IOT_WIFI_AUTH_WPA2_ENTERPRISE
iot_bsp_wifi.h, 127
IOT_WIFI_AUTH_WPA2_PSK
iot_bsp_wifi.h, 127
IOT_WIFI_AUTH_WPA_PSK
iot_bsp_wifi.h, 127
IOT_WIFI_AUTH_WPA_WPA2_PSK
iot_bsp_wifi.h, 127
IOT_WIFI_CMD_TIMEOUT
iot_bsp_wifi.h, 126
iot_wifi_conf, 59
bssid, 60
mode, 60
pass, 60
ssid, 60
IOT_WIFI_FREQ_2_4G_5G_BOTH
iot_bsp_wifi.h, 127
IOT_WIFI_FREQ_2_4G_ONLY
iot_bsp_wifi.h, 127
IOT_WIFI_FREQ_5G_ONLY
iot_bsp_wifi.h, 127
iot_wifi_freq_t
iot_bsp_wifi.h, 127
IOT_WIFI_MAX_BSSID_LEN
iot_bsp_wifi.h, 126
IOT_WIFI_MAX_PASS_LEN
iot_bsp_wifi.h, 126
IOT_WIFI_MAX_SCAN_RESULT
iot_bsp_wifi.h, 126
IOT_WIFI_MAX_SSID_LEN
iot_bsp_wifi.h, 126
IOT_WIFI_MODE_OFF
iot_bsp_wifi.h, 127
IOT_WIFI_MODE_P2P
iot_bsp_wifi.h, 127
IOT_WIFI_MODE_SCAN
iot_bsp_wifi.h, 127
IOT_WIFI_MODE_SOFTAP
iot_bsp_wifi.h, 127
IOT_WIFI_MODE_STATION
iot_bsp_wifi.h, 127
iot_wifi_mode_t
iot_bsp_wifi.h, 127
IOT_WIFI_MODE_UNDEFINED
iot_bsp_wifi.h, 127
iot_wifi_prov_data, 61
bssid, 61
password, 62
security_type, 62
ssid, 62
IOT_WIFI_PROV_PASSWORD_LEN
iot_main.h, 187
IOT_WIFI_PROV_SSID_LEN
iot_main.h, 187
iot_wifi_scan_result_t, 62
authmode, 63
bssid, 63
freq, 63
rssi, 63
ssid, 63
ipstack
MQTTClient, 67
isconnected
MQTTClient, 68
iv
iot_crypto_cipher_info, 33
iv_len
iot_crypto_cipher_info, 34

JSON_H
iot_easysetup_d2d.c, 106

keepalive
iot_mqtt_client.c, 350
keepAliveInterval
MQTTClient, 68
MQTTPacket_connectData, 78
key
iot_crypto_cipher_info, 34
key_len
iot_crypto_cipher_info, 34

label
iot_cloud_prov_data, 25
last_received
MQTTClient, 68
last_sent
MQTTClient, 68
LEAVE
iot_debug.h, 152
len
MQTTLenString, 75
MQTTTransport, 83
lenstring
MQTTString, 81

limit
noti_data_raw_t::_quota, 8
location_id
iot_cloud_prov_data, 25
lookup_id
iot_context, 30
LWIP_HTTPD_POST_MAX_PAYLOAD_LEN
iot_easysetup_http.c, 98

MAC_ADDR_BUFFER_SIZE
iot_easysetup_d2d.c, 106
MAX_CAP_ARG
st_dev.h, 282
MAX_MESSAGE_HANDLERS
iot_mqtt_client.h, 215
MAX_NO_OF_REMAINING_LENGTH_BYTES
iot_mqtt_packet.c, 376
MAX_PACKET_ID
iot_mqtt_client.h, 215
MAX_PAYLOAD_LENGTH
iot_easysetup_http.c, 98
MAX_SQNUM
iot_capability.c, 303
message
MessageData, 65
MQTTPacket_willOptions, 80
MessageData, 64
iot_mqtt_client.h, 215
message, 65
topicName, 65
messageHandler
iot_mqtt_client.h, 215
messageHandlers
MQTTClient, 68
min
iot_mqtt_connect_server.c, 368
iot_mqtt_deserialize_publish.c, 371
mnid
iot_devconf_prov_data, 43
mode
iot_crypto_cipher_info, 34
iot_wifi_conf, 60
MQTT_BAD_USERNAME_OR_PASSWORD
iot_mqtt_connect.h, 228
MQTT_BUFFER_OVERFLOW
iot_mqtt_client.h, 216
MQTT_CLIENTID_REJECTED
iot_mqtt_connect.h, 228
mqtt_connected
iot_mqtt_ctx, 50
MQTT_CONNECTION_ACCEPTED
iot_mqtt_connect.h, 228
MQTT_DISCONNECTED
iot_mqtt_client.h, 216
MQTT_FAILURE
iot_mqtt_client.h, 216
MQTT_NOT_AUTHORIZED
iot_mqtt_connect.h, 228
MQTT_SERVER_UNAVAILABLE
iot_mqtt_connect.h, 228
MQTT_SUCCESS
iot_mqtt_client.h, 216
MQTT_UNNACCEPTABLE_PROTOCOL
iot_mqtt_connect.h, 228
MQTTCleanSession
iot_mqtt_client.c, 351
MQTTClient, 66
buf, 67
buf_size, 67
cleansession, 67
command_timeout_ms, 67
defaultMessageHandler, 67
defaultUserData, 67
iot_mqtt_client.h, 215
ipstack, 67
isconnected, 68
keepAliveInterval, 68
last_received, 68
last_sent, 68
messageHandlers, 68
next_packetid, 68
ping_outstanding, 68
ping_retry_count, 68
ping_wait, 68
readbuf, 68
readbuf_size, 68
MQTTClient::MessageHandlers, 65
fp, 65
topicFilter, 65
userData, 66
MQTTClientInit
iot_mqtt_client.c, 351
iot_mqtt_client.h, 216
MQTTCloseSession
iot_mqtt_client.c, 352
MQTTConnackData, 69
iot_mqtt_client.h, 215
rc, 69
sessionPresent, 69
MQTTConnackFlags, 70
all, 70
bits, 70
reserved, 70
sessionpresent, 70
MQTTConnect
iot_mqtt_client.c, 352
iot_mqtt_client.h, 217
MQTTConnectFlags, 71
all, 71
bits, 72
cleansession, 72
int, 72
password, 72
username, 72
will, 72
willQoS, 72
willRetain, 72

MQTTConnectWithResults
iot_mqtt_client.c, 353
iot_mqtt_client.h, 218
MQTTDeserialize_ack
iot_mqtt_deserialize_publish.c, 371
iot_mqtt_packet.h, 238
MQTTDeserialize_connack
iot_mqtt_connect.h, 228
iot_mqtt_connect_client.c, 362
MQTTDeserialize_connect
iot_mqtt_connect.h, 229
iot_mqtt_connect_server.c, 368
MQTTDeserialize_publish
iot_mqtt_deserialize_publish.c, 372
iot_mqtt_publish.h, 247
MQTTDeserialize_suback
iot_mqtt_subscribe.h, 253
iot_mqtt_subscribe_client.c, 388
MQTTDeserialize_subscribe
iot_mqtt_subscribe.h, 254
iot_mqtt_subscribe_server.c, 391
MQTTDeserialize_unsuback
iot_mqtt_unsubscribe.h, 258
iot_mqtt_unsubscribe_client.c, 393
MQTTDeserialize_unsubscribe
iot_mqtt_unsubscribe.h, 258
iot_mqtt_unsubscribe_server.c, 396
MQTTDisconnect
iot_mqtt_client.c, 354
iot_mqtt_client.h, 219
MQTTFormat_toClientString
iot_mqtt_format.h, 234
MQTTFormat_toServerString
iot_mqtt_format.h, 234
MQTTHeader, 73
bits, 73
byte, 73
dup, 73
qos, 74
retain, 74
type, 74
MQTTLenString, 74
data, 74
len, 75
MQTTMessage, 75
dup, 75
id, 75
iot_mqtt_client.h, 215
payload, 76
payloadlen, 76
qos, 76
retained, 76
MQTTPACKET_BUFFER_TOO_SHORT
iot_mqtt_packet.h, 238
MQTTPacket_checkVersion
iot_mqtt_connect_server.c, 370
MQTTPacket_connectData, 76
cleansession, 78
clientID, 78
keepAliveInterval, 78
MQTTVersion, 78
password, 78
struct_id, 78
struct_version, 78
username, 78
will, 78
willFlag, 78
MQTTPacket_connectData_initializer
iot_mqtt_connect.h, 227
MQTTPacket_decode
iot_mqtt_packet.c, 376
iot_mqtt_packet.h, 239
MQTTPacket_decodeBuf
iot_mqtt_packet.c, 377
iot_mqtt_packet.h, 239
MQTTPacket_encode
iot_mqtt_packet.c, 377
iot_mqtt_packet.h, 239
MQTTPacket_equals
iot_mqtt_packet.c, 377
iot_mqtt_packet.h, 240
MQTTPacket_getName
iot_mqtt_format.c, 373
iot_mqtt_format.h, 234
MQTTPacket_len
iot_mqtt_packet.c, 378
iot_mqtt_packet.h, 240
MQTTPacket_msgTypesToString
iot_mqtt_packet.c, 378
iot_mqtt_packet.h, 241
MQTTPacket_names
iot_mqtt_format.c, 375
MQTTPacket_read
iot_mqtt_packet.c, 378
iot_mqtt_packet.h, 241
MQTTPACKET_READ_COMPLETE
iot_mqtt_packet.h, 238
MQTTPACKET_READ_ERROR
iot_mqtt_packet.h, 238
MQTTPacket_readnb
iot_mqtt_packet.c, 379
iot_mqtt_packet.h, 241
MQTTPacket_willOptions, 79
message, 80
qos, 80
retained, 80
struct_id, 80
struct_version, 80
topicName, 80
MQTTPacket_willOptions_initializer
iot_mqtt_connect.h, 228
MQTTPublish
iot_mqtt_client.c, 355
iot_mqtt_client.h, 220
MQTTSerialize_ack
iot_mqtt_packet.h, 242

iot_mqtt_serialize_publish.c, 384
MQTTSerialize_ack_size
iot_mqtt_packet.h, 242
iot_mqtt_serialize_publish.c, 384
MQTTSerialize_connack
iot_mqtt_connect.h, 229
iot_mqtt_connect_server.c, 370
MQTTSerialize_connect
iot_mqtt_connect.h, 229
iot_mqtt_connect_client.c, 363
MQTTSerialize_connect_size
iot_mqtt_connect.h, 231
iot_mqtt_connect_client.c, 363
MQTTSerialize_connectLength
iot_mqtt_connect_client.c, 364
MQTTSerialize_disconnect
iot_mqtt_connect.h, 231
iot_mqtt_connect_client.c, 364
MQTTSerialize_disconnect_size
iot_mqtt_connect.h, 232
iot_mqtt_connect_client.c, 366
MQTTSerialize_pingreq
iot_mqtt_connect.h, 232
iot_mqtt_connect_client.c, 366
MQTTSerialize_pingreq_size
iot_mqtt_connect.h, 233
iot_mqtt_connect_client.c, 367
MQTTSerialize_puback
iot_mqtt_publish.h, 248
iot_mqtt_serialize_publish.c, 384
MQTTSerialize_pubcomp
iot_mqtt_publish.h, 248
iot_mqtt_serialize_publish.c, 385
MQTTSerialize_publish
iot_mqtt_publish.h, 248
iot_mqtt_serialize_publish.c, 385
MQTTSerialize_publish_header
iot_mqtt_publish.h, 249
iot_mqtt_serialize_publish.c, 386
MQTTSerialize_publish_size
iot_mqtt_publish.h, 249
iot_mqtt_serialize_publish.c, 386
MQTTSerialize_publishLength
iot_mqtt_serialize_publish.c, 387
MQTTSerialize_pubrel
iot_mqtt_publish.h, 250
iot_mqtt_serialize_publish.c, 387
MQTTSerialize_suback
iot_mqtt_subscribe.h, 254
iot_mqtt_subscribe_server.c, 392
MQTTSerialize_subscribe
iot_mqtt_subscribe.h, 256
iot_mqtt_subscribe_client.c, 389
MQTTSerialize_subscribe_size
iot_mqtt_subscribe.h, 256
iot_mqtt_subscribe_client.c, 390
MQTTSerialize_subscribeLength
iot_mqtt_subscribe_client.c, 390
MQTTSerialize_unsuback
iot_mqtt_unsubscribe.h, 259
iot_mqtt_unsubscribe_server.c, 397
MQTTSerialize_unsubscribe
iot_mqtt_unsubscribe.h, 259
iot_mqtt_unsubscribe_client.c, 393
MQTTSerialize_unsubscribe_size
iot_mqtt_unsubscribe.h, 260
iot_mqtt_unsubscribe_client.c, 394
MQTTSerialize_unsubscribeLength
iot_mqtt_unsubscribe_client.c, 395
MQTTSerialize_zero
iot_mqtt_connect_client.c, 367
MQTTSetMessageHandler
iot_mqtt_client.c, 356
iot_mqtt_client.h, 221
MQTTString, 81
cstring, 81
lenstring, 81
MQTTString_initializer
iot_mqtt_packet.h, 238
MQTTStringFormat_ack
iot_mqtt_format.c, 373
iot_mqtt_format.h, 235
MQTTStringFormat_connack
iot_mqtt_format.c, 374
iot_mqtt_format.h, 235
MQTTStringFormat_connect
iot_mqtt_format.c, 374
iot_mqtt_format.h, 235
MQTTStringFormat_publish
iot_mqtt_format.c, 374
iot_mqtt_format.h, 235
MQTTStringFormat_suback
iot_mqtt_format.c, 374
iot_mqtt_format.h, 235
MQTTStringFormat_subscribe
iot_mqtt_format.c, 374
iot_mqtt_format.h, 235
MQTTStringFormat_unsubscribe
iot_mqtt_format.c, 374
iot_mqtt_format.h, 236
MQTTstrlen
iot_mqtt_packet.c, 379
iot_mqtt_packet.h, 242
MQTTSubackData, 82
grantedQoS, 82
iot_mqtt_client.h, 215
MQTTSubscribe
iot_mqtt_client.c, 357
iot_mqtt_client.h, 222
MQTTSubscribeWithResults
iot_mqtt_client.c, 357
iot_mqtt_client.h, 222
MQTTTransport, 82
getfn, 83
len, 83
multiplier, 83

rem_len, 83
sck, 83
state, 83
MQTTUnsubscribe
iot_mqtt_client.c, 358
iot_mqtt_client.h, 223
MQTTVersion
MQTTPacket_connectData, 78
MQTTYield
iot_mqtt_client.c, 360
iot_mqtt_client.h, 225
msgTypes
iot_mqtt_packet.h, 238
multiplier
MQTTTransport, 83

name
iot_crypto_pk_funcs, 40
net
iot_mqtt_ctx, 50
new_reged
iot_registered_data, 57
next
iot_cap_cmd_set_list, 16
iot_cap_handle_list, 20
next_packetid
MQTTClient, 68
NEXT_STATE_TIMEOUT_MS
iot_main.c, 315
NOSTACKTRACE
iot_mqtt_stacktrace.h, 252
noti_cb
iot_context, 31
noti_data_raw_t, 84
quota, 84
rate_limit, 85
noti_data_raw_t::_quota, 8
limit, 8
used, 9
noti_data_raw_t::_rate_limit, 9
count, 9
remainingTime, 10
sequenceNumber, 10
threshold, 10
noti_filter
iot_mqtt_ctx, 51
noti_usr_data
iot_context, 31
num_args
iot_cap_cmd_data_t, 12
number
iot_cap_val_t, 23

opt
iot_state_data, 58
OVF_BIT_BUTTON
iot_easysetup.h, 159
OVF_BIT_JUSTWORKS
iot_easysetup.h, 159
OVF_BIT_MAX_FEATURE
iot_easysetup.h, 159
OVF_BIT_PIN
iot_easysetup.h, 159
OVF_BIT_QR
iot_easysetup.h, 159
ownership_validation_feature
iot_easysetup.h, 159
ownership_validation_type
iot_devconf_prov_data, 43

param
iot_command, 26
pass
iot_wifi_conf, 60
password
iot_wifi_prov_data, 62
MQTTConnectFlags, 72
MQTTPacket_connectData, 78
payload
iot_easysetup_payload, 47
MQTTMessage, 76
payloadlen
MQTTMessage, 76
pin
iot_context, 31
iot_pin_t, 55
PIN_SIZE
iot_easysetup_d2d.c, 106
ping_outstanding
MQTTClient, 68
ping_retry_count
MQTTClient, 68
ping_wait
MQTTClient, 68
PINGREQ
iot_mqtt_packet.h, 238
PINGRESP
iot_mqtt_packet.h, 238
pk_type
iot_devconf_prov_data, 43
port
url_parse_t, 86
post_cgi_cmds
iot_easysetup_http.c, 102
protocol
url_parse_t, 86
prov_data
iot_context, 31
pub_queue
iot_context, 31
PUBACK
iot_mqtt_packet.h, 238
PUBCOMP
iot_mqtt_packet.h, 238
pubkey
iot_crypto_keypair, 37
iot_crypto_pk_info, 41
pubkey_len

iot_crypto_pk_info, 41
PUBLISH
iot_mqtt_packet.h, 238
PUBREC
iot_mqtt_packet.h, 238
PUBREL
iot_mqtt_packet.h, 238

QoS
iot_mqtt_client.h, 215
qos
MQTTHeader, 74
MQTTMessage, 76
MQTTPacket_willOptions, 80
QOS0
iot_mqtt_client.h, 216
QOS1
iot_mqtt_client.h, 216
QOS2
iot_mqtt_client.h, 216
quota
noti_data_raw_t, 84

rate_limit
noti_data_raw_t, 85
raw
iot_noti_data_t, 54
rc
MQTTConnackData, 69
read
iot_net_socket, 52
read_count
iot_net_socket, 52
readbuf
iot_mqtt_ctx, 51
MQTTClient, 68
readbuf_size
MQTTClient, 68
readChar
iot_mqtt_packet.c, 380
iot_mqtt_packet.h, 243
readInt
iot_mqtt_packet.c, 380
iot_mqtt_packet.h, 243
readMQTTLenString
iot_mqtt_packet.c, 380
iot_mqtt_packet.h, 243
reged_cli
iot_context, 31
rem_len
MQTTTransport, 83
remainingTime
noti_data_raw_t::_rate_limit, 10
reported_stat
iot_context, 31
req_state
iot_context, 31
reserved
MQTTConnackFlags, 70
retain
MQTTHeader, 74
retained
MQTTMessage, 76
MQTTPacket_willOptions, 80
returnCode
iot_mqtt_client.h, 216
room_id
iot_cloud_prov_data, 25
rssi
iot_wifi_scan_result_t, 63

s_pubkey
iot_crypto_ecdh_params, 35
scan_num
iot_context, 32
scan_result
iot_context, 32
sck
MQTTTransport, 83
seckey
iot_crypto_keypair, 37
iot_crypto_pk_info, 41
seckey_len
iot_crypto_pk_info, 41
security_type
iot_wifi_prov_data, 62
sem
iot_os_mutex, 55
sequenceNumber
noti_data_raw_t::_rate_limit, 10
sessionPresent
MQTTConnackData, 69
sessionpresent
MQTTConnackFlags, 70
setupid
iot_devconf_prov_data, 43
sign
iot_crypto_ed25519_keypair, 36
iot_crypto_pk_funcs, 40
socket
iot_net_socket, 52
ssid
iot_wifi_conf, 60
iot_wifi_prov_data, 62
iot_wifi_scan_result_t, 63
st-device-sdk-c/doc/mainpage.dox, 86
st-device-sdk-c/src/crypto/iot_crypto_ed25519.c, 86
st-device-sdk-c/src/crypto/mbedtls/iot_crypto_mbedtls.c, 89
st-device-sdk-c/src/crypto/ss/iot_crypto_ss.c, 95
st-device-sdk-c/src/easysetup/http/iot_easysetup_http.c, 97
st-device-sdk-c/src/easysetup/iot_easysetup_crypto.c, 103
st-device-sdk-c/src/easysetup/iot_easysetup_d2d.c, 104
st-device-sdk-c/src/easysetup/iot_easysetup_st_mqtt.c, 108

st-device-sdk-c/src/include/bsp/iot_bsp_debug.h, 110
st-device-sdk-c/src/include/bsp/iot_bsp_fs.h, 112
st-device-sdk-c/src/include/bsp/iot_bsp_nv_data.h, 119
st-device-sdk-c/src/include/bsp/iot_bsp_random.h, 121
st-device-sdk-c/src/include/bsp/iot_bsp_system.h, 122
st-device-sdk-c/src/include/bsp/iot_bsp_wifi.h, 124
st-device-sdk-c/src/include/iot_capability.h, 130
st-device-sdk-c/src/include/iot_crypto.h, 132
st-device-sdk-c/src/include/iot_crypto_internal.h, 149
st-device-sdk-c/src/include/iot_debug.h, 150
st-device-sdk-c/src/include/iot_easysetup.h, 154
st-device-sdk-c/src/include/iot_error.h, 162
st-device-sdk-c/src/include/iot_internal.h, 164
st-device-sdk-c/src/include/iot_jwt.h, 182
st-device-sdk-c/src/include/iot_main.h, 184
st-device-sdk-c/src/include/iot_nv_data.h, 189
st-device-sdk-c/src/include/iot_util.h, 204
st-device-sdk-c/src/include/iot_uuid.h, 210
st-device-sdk-c/src/include/mqtt/iot_mqtt_client.h, 212
st-device-sdk-c/src/include/mqtt/iot_mqtt_connect.h, 226
st-device-sdk-c/src/include/mqtt/iot_mqtt_format.h, 233
st-device-sdk-c/src/include/mqtt/iot_mqtt_packet.h, 236
st-device-sdk-c/src/include/mqtt/iot_mqtt_publish.h, 246
st-device-sdk-c/src/include/mqtt/iot_mqtt_stacktrace.h, 250
st-device-sdk-c/src/include/mqtt/iot_mqtt_subscribe.h, 253
st-device-sdk-c/src/include/mqtt/iot_mqtt_unsubscribe.h, 257
st-device-sdk-c/src/include/os/iot_os_net.h, 261
st-device-sdk-c/src/include/os/iot_os_util.h, 265
st-device-sdk-c/src/include/st_dev.h, 279
st-device-sdk-c/src/include/st_dev_version.h, 292
st-device-sdk-c/src/iot_api.c, 293
st-device-sdk-c/src/iot_capability.c, 302
st-device-sdk-c/src/iot_crypto.c, 310
st-device-sdk-c/src/iot_jwt.c, 313
st-device-sdk-c/src/iot_main.c, 314
st-device-sdk-c/src/iot_mqtt.c, 318
st-device-sdk-c/src/iot_nv_data.c, 324
st-device-sdk-c/src/iot_util.c, 340
st-device-sdk-c/src/iot_uuid.c, 345
st-device-sdk-c/src/mqtt/client/iot_mqtt_client.c, 348
st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_client.c, 362
st-device-sdk-c/src/mqtt/packet/iot_mqtt_connect_server.c, 368
st-device-sdk-c/src/mqtt/packet/iot_mqtt_deserialize_publish.c, 371
st-device-sdk-c/src/mqtt/packet/iot_mqtt_format.c, 372
st-device-sdk-c/src/mqtt/packet/iot_mqtt_packet.c, 375
st-device-sdk-c/src/mqtt/packet/iot_mqtt_serialize_publish.c, 383
st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_client.c, 388
st-device-sdk-c/src/mqtt/packet/iot_mqtt_subscribe_server.c, 391
st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_client.c, 393
st-device-sdk-c/src/mqtt/packet/iot_mqtt_unsubscribe_server.c, 395
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_debug_esp8266.c, 397
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_fs_esp8266.c, 399
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_nv_data_esp8266.c, 405
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_random_esp8266.c, 407
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_system_esp8266.c, 408
st-device-sdk-c/src/port/bsp/esp8266/iot_bsp_wifi_esp8266.c, 410
st-device-sdk-c/src/port/os/freertos/iot_os_net_freertos.c, 414
st-device-sdk-c/src/port/os/freertos/iot_os_util_freertos.c, 417
st_cap_attr_create_int
  iot_capability.c, 305
  st_dev.h, 284
st_cap_attr_create_number
  iot_capability.c, 305
  st_dev.h, 285
st_cap_attr_create_string
  iot_capability.c, 306
  st_dev.h, 285
st_cap_attr_create_string_array
  iot_capability.c, 306
  st_dev.h, 286
st_cap_attr_free
  iot_capability.c, 307
  st_dev.h, 287
st_cap_attr_send
  iot_capability.c, 307
  st_dev.h, 287
st_cap_cmd_cb
  st_dev.h, 283
st_cap_cmd_set_cb
  iot_capability.c, 308
  st_dev.h, 287
st_cap_handle_init
  iot_capability.c, 308
  st_dev.h, 288
st_cap_init_cb
  st_dev.h, 283
st_cap_noti_cb
  st_dev.h, 283
st_conn_cleanup
  iot_main.c, 315
  st_dev.h, 288
st_conn_init
  iot_main.c, 316
  st_dev.h, 289
st_conn_ownership_confirm
  iot_easysetup_d2d.c, 107

st_dev.h, 290
st_conn_set_noti_cb
iot_capability.c, 310
st_dev.h, 291
st_conn_start
iot_main.c, 317
st_dev.h, 291
st_dev.h
IOT_CAP_HANDLE, 282
iot_cap_val_type, 283
IOT_CAP_VAL_TYPE_INT_OR_NUM, 283
IOT_CAP_VAL_TYPE_INTEGER, 283
IOT_CAP_VAL_TYPE_NUMBER, 283
IOT_CAP_VAL_TYPE_STR_ARRAY, 283
IOT_CAP_VAL_TYPE_STRING, 283
iot_cap_val_type_t, 282
IOT_CAP_VAL_TYPE_UNKNOWN, 283
IOT_CTX, 282
IOT_EVENT, 282
iot_noti_type, 283
IOT_NOTI_TYPE_DEV_DELETED, 284
IOT_NOTI_TYPE_QUOTA_REACHED, 284
IOT_NOTI_TYPE_RATE_LIMIT, 284
iot_noti_type_t, 282
IOT_NOTI_TYPE_UNKNOWN, 284
iot_pin_t, 282
iot_stat_lv, 284
IOT_STAT_LV_DONE, 284
IOT_STAT_LV_FAIL, 284
IOT_STAT_LV_START, 284
IOT_STAT_LV_STAY, 284
iot_stat_lv_t, 282
iot_status, 284
IOT_STATUS_ALL, 284
IOT_STATUS_CONNECTING, 284
IOT_STATUS_IDLE, 284
IOT_STATUS_NEED_INTERACT, 284
IOT_STATUS_PROVISIONING, 284
iot_status_t, 282
MAX_CAP_ARG, 282
st_cap_attr_create_int, 284
st_cap_attr_create_number, 285
st_cap_attr_create_string, 285
st_cap_attr_create_string_array, 286
st_cap_attr_free, 287
st_cap_attr_send, 287
st_cap_cmd_cb, 283
st_cap_cmd_set_cb, 287
st_cap_handle_init, 288
st_cap_init_cb, 283
st_cap_noti_cb, 283
st_conn_cleanup, 288
st_conn_init, 289
st_conn_ownership_confirm, 290
st_conn_set_noti_cb, 291
st_conn_start, 291
st_status_cb, 283
st_dev_version.h
STDK_VERSION, 292
STDK_VERSION_CODE, 292
VER_MAJOR, 292
VER_MINOR, 293
VER_PATCH, 293
st_status_cb
st_dev.h, 283
state
MQTTTransport, 83
state_timer
iot_context, 32
status_cb
iot_context, 32
status_maps
iot_context, 32
status_usr_data
iot_context, 32
STDK_NV_DATA_NAMESPACE
iot_bsp_fs_esp8266.c, 400
STDK_NV_DATA_PARTITION
iot_bsp_fs_esp8266.c, 400
STDK_VERSION
st_dev_version.h, 292
STDK_VERSION_CODE
st_dev_version.h, 292
step
iot_easysetup_payload, 47
str_num
iot_cap_val_t, 23
string
iot_cap_unit_t, 21
iot_cap_val_t, 23
strings
iot_cap_val_t, 23
struct_id
MQTTPacket_connectData, 78
MQTTPacket_willOptions, 80
struct_version
MQTTPacket_connectData, 78
MQTTPacket_willOptions, 80
SUBACK
iot_mqtt_packet.h, 238
SUBFAIL
iot_mqtt_client.h, 216
SUBSCRIBE
iot_mqtt_packet.h, 238

t_seckey
iot_crypto_ecdh_params, 35
threshold
noti_data_raw_t::_rate_limit, 10
topicFilter
MQTTClient::MessageHandlers, 65
topicName
MessageData, 65
MQTTPacket_willOptions, 80
type
iot_cap_unit_t, 21
iot_cap_val_t, 23

iot_crypto_cipher_info, 34
iot_crypto_pk_info, 41
iot_noti_data_t, 54
MQTTHeader, 74

UNSUBACK
iot_mqtt_packet.h, 238
UNSUBSCRIBE
iot_mqtt_packet.h, 238
updated
iot_registered_data, 57
URL_BUFFER_SIZE
iot_easysetup_d2d.c, 106
url_parse_t, 85
domain, 85
port, 86
protocol, 86
used
noti_data_raw_t::_quota, 9
userData
MQTTClient::MessageHandlers, 66
username
MQTTConnectFlags, 72
MQTTPacket_connectData, 78
usr_data
iot_cap_cmd_set, 14
usr_events
iot_context, 32

VER_MAJOR
st_dev_version.h, 292
VER_MINOR
st_dev_version.h, 293
VER_PATCH
st_dev_version.h, 293
verify
iot_crypto_pk_funcs, 40
vid
iot_devconf_prov_data, 43

waitfor
iot_mqtt_client.c, 361
wifi
iot_device_prov_data, 46
WIFI_AP_START_BIT
iot_bsp_wifi_esp8266.c, 413
WIFI_AP_STOP_BIT
iot_bsp_wifi_esp8266.c, 413
WIFI_EVENT_BIT_ALL
iot_bsp_wifi_esp8266.c, 414
WIFI_STA_CONNECT_BIT
iot_bsp_wifi_esp8266.c, 414
WIFI_STA_DISCONNECT_BIT
iot_bsp_wifi_esp8266.c, 414
WIFI_STA_START_BIT
iot_bsp_wifi_esp8266.c, 414
WIFIINFO_BUFFER_SIZE
iot_easysetup_d2d.c, 106
will
MQTTConnectFlags, 72
MQTTPacket_connectData, 78
willFlag
MQTTPacket_connectData, 78
willQoS
MQTTConnectFlags, 72
willRetain
MQTTConnectFlags, 72
write
iot_net_socket, 52
writeChar
iot_mqtt_packet.c, 381
iot_mqtt_packet.h, 244
writeCString
iot_mqtt_packet.c, 381
iot_mqtt_packet.h, 244
writeInt
iot_mqtt_packet.c, 382
iot_mqtt_packet.h, 245
writeMQTTString
iot_mqtt_packet.c, 382
iot_mqtt_packet.h, 245