Modules

Here is a list of all modules:

- MQTT Packet (MQP) Buffer structure
- Helper Macros for RX PUBLISH
- LIBRARY Generated Error Codes
- Information to establish a secure connection.
- Abstraction of Network Services on a platform
- Options for platform to configure network
- The Server Library API(s)
  - Helper Macros for RX CONNECT
- The Server Daemon API(s)

MQTT Packet (MQP) Buffer structure

Classes

struct	**mqtt_packet**

Detailed Description

The core construct to encapsulate, construct and process a message

---

Helper Macros for RX PUBLISH

Defines

#define	**MQP_PUB_TOP_BUF**(mqp) (MQP_VHEADER_BUF(mqp) + 2)
#define	**MQP_PUB_TOP_LEN**(mqp) (mqp->vh_len - 2 - (mqp->msg_id? 2 : 0))
#define	**MQP_PUB_PAY_BUF**(mqp) (mqp->pl_len? MQP_PAYLOAD_BUF(mqp) : NULL)
#define	**MQP_PUB_PAY_LEN**(mqp) (mqp->pl_len)

Define Documentation

#define MQP_PUB_PAY_BUF (mqp) (mqp->pl_len? MQP_PAYL(

Yields pointer to payload data

#define MQP_PUB_PAY_LEN (mqp) (mqp->pl_len)

Length or size of payload data

#define MQP_PUB_TOP_BUF (mqp) (MQP_VHEADER_BUF(mq

Yields pointer to topic content

#define MQP_PUB_TOP_LEN (mqp) (mqp->vh_len - 2 - (mqp->r

Length or size of topic content

LIBRARY Generated Error Codes

Defines

#define	**MQP_ERR_NETWORK** (-1)
#define	**MQP_ERR_TIMEOUT** (-2)
#define	**MQP_ERR_NET_OPS** (-3)
#define	**MQP_ERR_FNPARAM** (-4)
#define	**MQP_ERR_PKT_AVL** (-5)
#define	**MQP_ERR_PKT_LEN** (-6)
#define	**MQP_ERR_NOTCONN** (-7)
#define	**MQP_ERR_BADCALL** (-8)
#define	**MQP_ERR_CONTENT** (-9)
#define	**MQP_ERR_LIBQUIT** (-10)
#define	**MQP_ERR_NOT_DEF** (-32)

Detailed Description

Library provides these codes as return values in several routines

Define Documentation

#define MQP_ERR_BADCALL (-8)

Irrelevant call for LIB state

#define MQP_ERR_CONTENT (-9)

MSG / Data content has errors

#define MQP_ERR_FNPARAM (-4)

Invalid parameter(s) provided

#define MQP_ERR_LIBQUIT (-10)

Needs reboot library has quit

#define MQP_ERR_NET_OPS (-3)

Platform Net Ops un-available

#define MQP_ERR_NETWORK (-1)

Problem in network (sock err)

#define MQP_ERR_NOT_DEF (-32)

Value other than defined ones

#define MQP_ERR_NOTCONN (-7)

Lib isn't CONNECTED to server

#define MQP_ERR_PKT_AVL (-5)

No pkts are available in pool

#define MQP_ERR_PKT_LEN (-6)

Inadequate free buffer in pkt

#define MQP_ERR_TIMEOUT (-2)

Net transaction has timed out

Information to establish a secure connection.

Classes

struct	**secure_conn**

Detailed Description

This is implementation specific and is targeted for the network services.

Specifically, the MQTT implementation makes no assumption or use of this construct. The client library merely passes information from the app to the network service layer.

Abstraction of Network Services on a platform

Classes

struct	**device_net_services**

Detailed Description

Services to enable the MQTT Client-Server communication over network

These services are invoked by the MQTT Library.

Options for platform to configure network

Defines

#define	**DEV_NETCONN_OPT_TCP** 0x01
#define	**DEV_NETCONN_OPT_UDP** 0x02
#define	**DEV_NETCONN_OPT_IP6** 0x04
#define	**DEV_NETCONN_OPT_URL** 0x08
#define	**DEV_NETCONN_OPT_SEC** 0x10

Define Documentation

#define DEV_NETCONN_OPT_IP6 0x04

Assert for IPv6, otherwise it is IPv4

#define DEV_NETCONN_OPT_SEC 0x10

Assert to indicate a secure connection

#define DEV_NETCONN_OPT_TCP 0x01

Assert to indicate TCP net connection

#define DEV_NETCONN_OPT_UDP 0x02

Assert to create a local UDP port bind

#define DEV_NETCONN_OPT_URL 0x08

Assert if the network address is a URL

The Server Library API(s)

Classes | Modules | Defines | Functions

Classes

struct	**mqtt_server_msg_cbs**
struct	**mqtt_server_lib_cfg**

Modules

Helper Macros for RX CONNECT

Defines

```
#define MQP_SERVER_RX_LEN 1024
```

Functions

i32	**mqtt_vh_msg_send** (void *ctx_cl, u8 msg_type, enum **mqtt_qos** qos, bool has_vh, u16 vh_data)
i32	**mqtt_vh_msg_send_locked** (void *ctx_cl, u8 msg_type, enum **mqtt_qos** qos, bool has_vh, u16 vh_data)
i32	**mqtt_server_pub_dispatch** (void *ctx_cl, struct **mqtt_packet** *mqp, bool dup)
i32	**mqtt_server_pub_dispatch_locked** (void *ctx_cl, struct **mqtt_packet** *mqp, bool dup)
i32	**mqtt_server_run** (u32 wait_secs)
i32	**mqtt_server_register_net_svc** (const struct **device_net_services** *net)
i32	**mqtt_server_lib_init** (const struct **mqtt_server_lib_cfg** *cfg, const struct **mqtt_server_msg_cbs** *cbs)

Define Documentation

#define MQP_SERVER_RX_LEN 1024

Max size(B) of RX Buffer for MQTT Server

Function Documentation

i32 mqtt_server_lib_init (const struct mqtt_server_lib_cfg * cfg,
const struct mqtt_server_msg_cbs * cbs
)

Initialize the MQTT Server Packet library This routine initializes the packet and network constructs that are required to manage the multiple network connetions. The server packet LIB must be initialized prior to invoking of any other routine or service.

> **Note:**
> This routine must be invoked only once in an run of the system.

If there are more than one application (tasks) that utilize the services of the server packet library, then certain configuration must be set in the LIB

See also:
struct **mqtt_server_lib_cfg**

The application should also provision the platform network specific network services into the packet library

See also:
mqtt_server_register_net_svc.

Once, the server LIB has been intialized successfully, it must be put into action, to listen to requests for incoming connections, by invoking the API **mqtt_server_run**.

Parameters:

[in]	**cfg**	configuration information for the MQTT server packet library
[in]	**cbs**	callback routines that LIB will invoke into the application

Returns:
0 on success otherwise -1.

i32 mqtt_server_pub_dispatch (void * ctx_cl,
struct mqtt_packet * mqp,
bool dup
)

Dispatch application constructed PUBLISH message to the client. Prior to sending the message to the client, this routine shall update the fixed-header to account for the duplicate flag that has been indicated by the caller.

The caller must populate the payload information of topic and data before invoking this service. In addition, the application must prepare, for the packet, the fix header leaving aside the duplicate flag - this flag shall be included in the fix heder by the LIB.

This service facilitates the application to re-use, iteratively, a single PUBLISH packet for multiple remote clients that have subscribed to the same topic for which the data is being published. The LIB does not free-up the MQTT packet after sending the packet to the remote client and the application is responsible for managing the packet container / memory

Parameters:

[in]	**ctx_cl**	handle to the underlying network context in the LIB. This handle is provided to the application by the LIB in the CONNECT callback.
[in]	**mqp**	app created PUBLISH message alongwith the fixed header
[in]	**dup**	is this a duplicate message that is being sent to client?

Returns:
on success, the number of bytes transferred. Otherwise, error defined in **LIBRARY Generated Error Codes**

i32 mqtt_server_pub_dispatch_locked (void * ctx_c
struct mqtt_packet * mqp,
bool dup
)

mqtt_server_pub_dispatch() with mutual exclusion (in multi-task application). This routine ensures that the LIB sends the specified packet onto the network in a manner that excludes execution of any other control. This API has been enabled to support the scenarios, where the multi-tasked user application has chosen to use a mutex object other than the one provisioned in the packet LIB to streamline / serialize accesses to the services of the packet LIB.

Refer to **mqtt_server_pub_dispatch** for other details.

i32 mqtt_server_register_net_svc (const struct device_net_service

Abstraction for the device specific network services Network services for communication with the clients

Parameters:

`[in]` **net** refers to network services supported by the platform

Returns:

on success, 0, otherwise -1

Abstraction of Network Services on a platform

Note:

all entries in net must be supported by the platform.

i32 mqtt_server_run (u32 wait_secs)

Run the server packet LIB for the specified wait time. This routine

yields the control back to the application after the specified duration of wait time. Such an arrangement enable the application to make overall progress to meet it intended functionality.

The wait time implies the maximum intermediate duration between the reception of two successive messages from the server. If no message is received before the expiry of the wait time, the routine returns. However, the routine would continue to block, in case, messages are being received within the successive period of wait time.

Parameters:

[in]	**wait_secs**	maximum time to wait for a message from the server

Note:
if the value of MQP_ERR_LIBQUIT is returned, then system must be restarted.

```
i32 mqtt_vh_msg_send ( void *           ctx_cl,
                       u8               msg_type,
                       enum mqtt_qos    qos,
                       bool             has_vh,
                       u16              vh_data
                     )
```

Send a Variable Header only message to the client. Application should this routine to send PUBREL and PUBCOMP messages.

Parameters:

[in]	**ctx_cl**	handle to the underlying network context in the LIB. This handle is provided to the application by the LIB in the CONNECT callback.
[in]	**msg_type**	message type that has to be sent to the client

[in]	**qos**	QoS with which the message needs to send to server
[in]	**has_vh**	does this message has data in the variable header?
[in]	**vh_data**	data <MSB:LSB> to be included in the variable header

Returns:

on success, the number of bytes transferred. Otherwise, errors defined in **LIBRARY Generated Error Codes**

i32 mqtt_vh_msg_send_locked (void * ctx_cl,
u8 msg_type,
enum mqtt_qos qos,
bool has_vh,
u16 vh_data
)

mqtt_vh_msg_send() with mutual exclusion (in multi-task application). This routine ensures that the LIB sends the specified VH MSG onto the network in a manner that excludes execution of any other control. This API has been enabled to support the scenarios, where the multi-tasked user application has chosen to use a mutex object other than the one provisioned in the packet LIB to streamline / serialize accesses to the services of the packet LIB.

Refer to **mqtt_vh_msg_send** for details

Helper Macros for RX CONNECT

The Server Library API(s)

Defines

#define	**MQ_CONN_UTF8_BUF**(utf8) ((utf8)? (utf8)->buffer : NULL)
#define	**MQ_CONN_UTF8_LEN**(utf8) ((utf8)? (utf8)->length : 0)
#define	**MQC_UTF8_CLIENTID**(utf8_vec) (utf8_vec[0])
#define	**MQC_UTF8_WILL_TOP**(utf8_vec) (utf8_vec[1])
#define	**MQC_UTF8_WILL_MSG**(utf8_vec) (utf8_vec[2])
#define	**MQC_UTF8_USERNAME**(utf8_vec) (utf8_vec[3])
#define	**MQC_UTF8_PASSWORD**(utf8_vec) (utf8_vec[4])
#define	**MQC_CLIENTID_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(MC
#define	**MQC_CLIENTID_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(MC
#define	**MQC_WILL_TOP_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(M
#define	**MQC_WILL_TOP_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(M
#define	**MQC_WILL_MSG_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(M
#define	**MQC_WILL_MSG_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(M
#define	**MQC_USERNAME_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(
#define	**MQC_USERNAME_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(U
#define	**MQC_PASSWORD_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(
#define	**MQC_PASSWORD_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(

Define Documentation

#define MQ_CONN_UTF8_BUF (utf8) ((utf8)? (utf8)->buffer : NU

Yields pointer to the UTF8 content

#define MQ_CONN_UTF8_LEN (utf8) ((utf8)? (utf8)->length : 0)

Length or size of the UTF8 content

#define MQC_UTF8_CLIENTID (utf8_vec) (utf8_vec[0])

Returns Client ID

#define MQC_UTF8_PASSWORD (utf8_vec) (utf8_vec[4])

Returns Pass Word

#define MQC_UTF8_USERNAME (utf8_vec) (utf8_vec[3])

Returns User Name

#define MQC_UTF8_WILL_MSG (utf8_vec) (utf8_vec[2])

Returns Will Data

#define MQC_UTF8_WILL_TOP (utf8_vec) (utf8_vec[1])

Returns Will Topic

The Server Daemon API(s)

Classes

struct	**mqtt_server_app_cbs**
struct	**mqtt_server_app_cfg**

Functions

i32	**mqtt_server_topic_enroll** (const void *app_hnd, const struct **utf8_string** *topic, enum **mqtt_qos** qos)
i32	**mqtt_server_topic_disenroll** (const void *app_hnd, const struct **utf8_string** *topic)
i32	**mqtt_server_app_pub_send** (const struct **utf8_string** *topic, const u8 *data_buf, u32 data_len, enum **mqtt_qos** qos, bool retain)
void *	**mqtt_server_app_register** (const struct **mqtt_server_app_cbs** *cbs, const i8 *name)
i32	**mqtt_server_init** (const struct **mqtt_server_lib_cfg** *lib_cfg, const struct **mqtt_server_app_cfg** *app_cfg)

Function Documentation

i32 mqtt_server_app_pub_send (const struct utf8_string * topic,
const u8 * data_bu
u32 data_le
enum mqtt_qos qos,
bool retain
)

Send data to network for a topic This routine offers the binary data along-with associated properties for a specific topic to the server. The server, based on the subscriptions from the remote clients and the enrollments made by the local applications for this topic, will send the binary data and its qualifiers, adjusted for the maximum subscribed or enrolled QoS, to the remote clients and the local applications.

Parameters:

[in]	**topic**	UTF8 topic Name for which data has been published
[in]	**data_buf**	the published binary data for the topic
[in]	**data_len**	the length of the binary data
[in]	**qos**	quality of service of the message / data
[in]	**retain**	should the server retain the data?

Returns:

on success, the length of data sent, otherwise -1 on error.

void* mqtt_server_app_register (const struct mqtt_server_app_cb
const i8 *
)

Register an application with the server. This routine makes known to the server about an application identified by its name and creates a context / reference for the application in the server.

An application intending to utlize the servicse of the MQTT server must be first registered with the server.

Parameters:

[in]	**cbs**	callback routines from the application to be invoked by the server
[in]	**name**	refers to the name of the application. The application must retain the memory used by the 'name' after the function call. The server does not copy the name into its internal buffers.

Returns:

a valid handle to context of application in the server, othewise NULL on error

i32 mqtt_server_init (const struct mqtt_server_lib_cfg * lib_cfg, const struct mqtt_server_app_cfg * app_cfg)

Initialize the MQTT Server (Task / Daemon). This routine initializes the server implementation and sets it up using the provided configuration. The server implementation must be initialized prior to invoking of any other routine or service.

This routine should be invoked as part of the platform initialization.

Note:

This routine must be invoked only once in an run of the system. This routine internally invokes the **mqtt_server_lib_init( )** and therefore, there is no need to invoke the **mqtt_server_lib_init( )** explicitly.

The server needs to be in a state to listen to incoming MQTT connection requests. Therefore, the platform sequence after provisioning the buffer using the API **mqtt_server_register_net_svc**, must invoke the API **mqtt_server_run**, in an infinite loop, to keep the server daemon active.

Parameters:

[in]	**lib_cfg**	configuration information for the MQTT server packet library.
[in]	**app_cfg**	configuration information for the server applications.

Returns:

0 on success, otherwise -1 on error

i32 mqtt_server_topic_disenroll (const void * app_hn
const struct utf8_string * topic
)

Cancel previous enrollment to receive data for a topic This routines terminates the previous registration, if any, made by the application to receive any publishised data for the specified topic. Once, the enrollment is removed, the application, there after, will not receive any data for this topic from the server.

Parameters:

[in]	**app_hnd**	handle to the application context in the server. This handle is provided by server **mqtt_server_app_register()**
[in]	**topic**	UTF8 based string for which the application needs to stop getting the published data

Returns:

0 on success, otherwise a negative value on error.

i32 mqtt_server_topic_enroll (const void * app_hnd,
const struct utf8_string * topic,
enum mqtt_qos qos
)

Enroll with server to receive data for a topic This routine registers with

the server, the specified topic for which the application should receive any published data from the network. Whenever, any of the remote clients that are connected to the server or applications, this or other, publishes data for the specified topic, the server will present the published information to this application.

As part of the enrollment, the application should provide the maxmimum QoS with which the server should provide the published data. If the topic data received by the server bears a QoS higher than the one specified by the application, the server shall lower it to the QoS level preferred by the application. Otherwise, the QoS of the data shall be presented 'as-is'. In other words, the application should expect a published data with a lower QoS.

Parameters:

[in]	**app_hnd**	handle to the application context in the server. This handle is provided by server **mqtt_server_app_register()**
[in]	**topic**	UTF8 based string for which the application needs to start getting the published data
[in]	**qos**	the meximum QoS the application intends to receive data for this particular topic

Returns:

0 on sucess, otherwise a negative value on error.

Class List

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

client_ctx	
device_net_services	
mqtt_ack_wlist	
mqtt_packet	
mqtt_server_app_cbs	
mqtt_server_app_cfg	
mqtt_server_lib_cfg	
mqtt_server_msg_cbs	
pub_qos2_cq	
secure_conn	
utf8_string	
utf8_strqos	

Public Attributes

client_ctx Struct Reference

List of all members.

Public Attributes

void *	**usr**
i32	**net**
i8	**remote_ip** [16]
u32	**ip_length**
u32	**timeout**
u16	**ka_secs**
u32	**flags**
struct **client_ctx** *	**next**

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

device_net_services Struct Reference

Abstraction of Network Services on a platform

List of all members.

Public Attributes

i32(*	**open**)(u32 nwconn_opts, const i8 *server_addr, u16 port_number, const struct **secure_conn** *nw_security)
i32(*	**send**)(i32 comm, const u8 *buf, u32 len, void *ctx)
i32(*	**recv**)(i32 comm, u8 *buf, u32 len, u32 wait_secs, bool *err_timeo, void *ctx)
i32(*	**send_dest**)(i32 comm, const u8 *buf, u32 len, u16 dest_port, const i8 *dest_ip, u32 ip_len)
i32(*	**recv_from**)(i32 comm, u8 *buf, u32 len, u16 *from_port, i8 *from_ip, u32 *ip_len)
i32(*	**close**)(i32 comm)
i32(*	**listen**)(u32 nwconn_opts, u16 port_number, const struct **secure_conn** *nw_security)
i32(*	**accept**)(i32 **listen**, i8 *client_ip, u32 *ip_length)
i32(*	**io_mon**)(i32 *recv_cvec, i32 *send_cvec, i32 *rsvd_cvec, u32 wait_secs)
u32(*	**time**)(void)

Member Data Documentation

i32(* device_net_services::accept)(i32 listen, i8 *client_ip, u32 *ip_l

Accept an incominng connection. This routine creates a new communication channel for the (remote) requesting client.

Parameters:

[in]	**listen**	handle to listen for the incoming connection requests from the remote clients
[out]	**client_ip**	IP address of the connected client. This value is valid only on successful return of the routine. The place holder must provide memory to store atleast 16 bytes.
[in,out]	**ip_length**	Length of IP address. It is provided by the caller to declare the length of the place holder and updated by routine to indicate the length of the connected client's IP address.

Returns:

on success, a valid handle to the new connection, otherwise NULL

i32(* device_net_services::close)(i32 comm)

Close communication connection

i32(* device_net_services::io_mon)(i32 *recv_cvec, i32 *send_cvec

Monitor activity on communication handles. The routine blocks for the specified period of time to monitor activity, if any, on each of the

communication handle that has been provided in one or more vector sets. At the expiry of the wait time, this function must identify the handles, on which, acitvities were observed.

A particular collection of communication handles are packed as an array or in a vector and is passed to the routine. A NULL handle in the vector indicates the termination of the vector and can effectively used to account for the size of the vector.

Similarly, at end the end of the wait period, the routine must provide a vector of the handles for which activity was observed.

Parameters:

`[in,out]`	**recv_hvec**	a vector of handles which must be monitored for receive activities.
`[in,out]`	**send_hvec**	a vector of handles which must be monitored for send activities.
`[in,out]`	**rsvd_hvec**	reserved for future use.
`[in]`	**wait_secs**	time to wait and monitor activity on communication handles provided in one or more sets. If set to 0, the routine returns immediately.

Returns:

on success, the total number of handles for which activity was observed. This number can be 0, if no activity was observed on any of the provided handle in the specified time. Otherwise, -1 on error.

i32(* device_net_services::listen)(u32 nwconn_opts, u16 port_num

Listen to incoming connection from clients. This routine prepares the system to listen on the specified port for the incoming network connections from the remote clients.

Parameters:

`[in]`	**nwconn_opts**	Implementation specific construct to

		enumerate server address and / or connection related details
[in]	**port_number**	Network port number, typically, 1883 or 8883
[in]	**nw_security**	Information to establish a secure connection with client. Set it to NULL, if not used. **Information to establish a secure connection.**

Returns:
a valid handle to listening contruct, otherwise NULL

i32(* device_net_services::open)(u32 nwconn_opts, const i8 *serv

Set up a communication channel with a server or set up a local port. This routine opens up either a "connection oriented" communication channel with the specified server or set up a local configuration for "connectionless" transactions.

Parameters:

[in]	**nwconn_opts**	Implementation specific construct to enumerate server address and / or connection related details
[in]	**server_addr**	URL or IP address (string) or other server reference. For setting up a local (UDP) port, set it to NULL.
[in]	**port_number**	Network port number, typically, 1883 or 8883 for remote severs. For setting up a local (UDP) port, use an intended port number.
[in]	**nw_security**	Information to establish a secure connection with server. Set it to NULL, if not used. **Information to establish a secure connection.**

Returns:
a valid handle to connection, otherwise NULL

i32(* device_net_services::recv)(i32 comm, u8 *buf, u32 len, u32 wa

Receive data from the "connection oriented" channel. The routine blocks till the time, there is either a data that has been received from the server or the time to await data from the server has expired.

Parameters:

[in]	**comm**	Handle to network connection as returned by **accept()**.
[out]	**buf**	place-holder to which data from network should be written into.
[in]	**len**	maximum length of 'buf'
[in]	**wait_secs**	maximum time to await data from network. If exceeded, the routine returns error with the err_timeo flag set as true.
[out]	**err_timeo**	if set, indicates that error is due to timeout.
[in]	**ctx**	reference to the MQTT connection context

Returns:

on success, number of bytes received, 0 on connection reset, otherwise -1 on error. In case, error (-1) is due to the time-out, then the implementation should set flag err_timeo as true.

i32(* device_net_services::recv_from)(i32 comm, u8 *buf, u32 len, u

Receive data on a local port sent by any network element. The routine blocks till the time, data has been received on the local port from any remote network element.

Parameters:

[in]	**comm**	handle to network connection as return by **open()**.
[in]	**buf**	place-holder to which data from network should be written into.

[in]	**len**	maximum lengh of 'buf'
[out]	**from_port**	place-holder for the port of the sender network entity
[out]	**from_ip**	place-holder to retrieve the IP address of the sender network entity. The memory space must be provisioned to store atleast 16 bytes.
[in,out]	**ip_len**	length of IP address. It is provided by the caller to declare the length of the place holder and updated by routine to indicate the length of the remote network entity's IP address.

Returns:

on success, number of bytes received, 0 on connection reset, otherwise -1 on errir.

i32(* device_net_services::send)(i32 comm, const u8 *buf, u32 len,

Send data onto the "connection oriented" network. The routine blocks till the time, the data has been copied into the network stack for dispatch on to the "connection oriented" network.

Parameters:

[in]	**comm**	handle to network connection as returned by **open()**.
[in]	**buf**	refers to the data that is intended to be sent
[in]	**len**	length of the data
[in]	**ctx**	reference to the MQTT connection context

Returns:

on success, the number of bytes sent, 0 on connection reset, otherwise -1

i32(* device_net_services::send_dest)(i32 comm, const u8 *buf, u3

Send data to particular port on the specified network element. The routine blocks till the time, the data has been copied into the network stack for dispatch to the "specified" network entity.

Parameters:

[in]	**comm**	handle to network connection as returned by **open()**.
[in]	**buf**	refers to data that is intended to be sent
[in]	**len**	length of the data
[in]	**dest_port**	network port to which data is to be sent.
[in]	**dest_ip**	IP address of the entity to which data is to be sent.
[in]	**ip_len**	length of the destination IP address.

Returns:

on success, the number of bytes sent, 0 on connection reset, otherwise -1.

u32(* device_net_services::time)(void)

Get Time (in seconds). Provides a monotonically incrementing value of a time service in unit of seconds. The implementation should ensure that associated timer hardware or the clock module remains active through the low power states of the system. Such an arrangement ensures that MQTT Library is able to track the Keep-Alive time across the cycles of low power states. It would be typical of battery operated systems to transition to low power states during the period of inactivity or otherwise to conserve battery.

In the absence of a sustained time reference across the low power states, if the system transitions away from the active state, the MQTT Library, then may not be able to effectively monitor the Keep Alive duration.

It is the responsbililty of the implementation to manage the roll- over problem of the hardware and ensure the integrity of the time value is maintained.

Returns:
time in seconds

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

mqtt_ack_wlist Struct Reference

List of all members.

Public Attributes

struct **mqtt_packet** *	**head**
struct **mqtt_packet** *	**tail**

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commo**

Public Attributes

mqtt_packet Struct Reference

MQTT Packet (MQP) Buffer structure

List of all members.

Public Attributes

u8	**msg_type**
u8	**fh_byte1**
u16	**msg_id**
u8	**n_refs**
u8	**pad** [3]
u8	**offset**
u8	**fh_len**
u16	**vh_len**
u32	**pl_len**
u32	**private**
u32	**maxlen**
u8 *	**buffer**
void(*	**free**)(struct **mqtt_packet** *mqp)
struct **mqtt_packet** *	**next**

Member Data Documentation

u8* mqtt_packet::buffer

The attached buffer

u8 mqtt_packet::fh_byte1

Fixed Header: Byte1

u8 mqtt_packet::fh_len

Fix Header Length

void(* mqtt_packet::free)(struct mqtt_packet *mqp)

Method to free this packet to a particular pool

u32 mqtt_packet::maxlen

Maximum buffer size

u16 mqtt_packet::msg_id

Msg transaction ID

u8 mqtt_packet::msg_type

MQTT Message Type

u8 mqtt_packet::n_refs

users of this msg

u8 mqtt_packet::offset

Start of data index

u32 mqtt_packet::pl_len

Pay Load Length

u16 mqtt_packet::vh_len

Var Header Length

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

mqtt_server_app_cbs Struct Reference

The Server Daemon API(s)

List of all members.

Public Attributes

u16(*	**connect**)(const struct **utf8_string** *client_id, const struct **utf8_string** *user_name, const struct **utf8_string** *pass_word, void **app_usr)
void(*	**publish**)(const struct **utf8_string** *topic, const u8 *payload, u32 pay_len, bool dup, u8 qos, bool retain)
void(*	**disconn**)(const void *app_usr, bool due2err)

Member Data Documentation

u16(* mqtt_server_app_cbs::connect)(const struct utf8_string *clie

Connection request from remote client - assess credentials. This routine presents, to the application, the information about the credentials of the remote client that is trying to establish a MQTT connection with the server. The application shall utilize these data in conjunction with its policy to either allow or deny connection to the server.

Should the application choose to allow the remote client to establish a MQTT connection, then it must provide in 'app-user' (place-holder), a handle that refers to the user of this connection. The server will provide this handle in follow-up routines to enable the application to refer to the associated connection in its implementation.

Parameters:

[in]	**client_id**	UTF8 based ID of the remote client - is set to NULL to indicate a zero-length client id.
[in]	**user_name**	UTF8 based user-name provided by the remote client. It is set to NULL if the client did not provide an user name
[in]	**pass_word**	UTF8 based pass-word provided by the remote client. It is set to NULL if the client did not provide a pass-word
[out]	**app_usr**	place-holder for application to provide a handle to the user of this specific connection / client.

Returns:

16bit value for the variable header in the CONNACK message. The MSB in the return value refers to the 8bit parameter of the acknowledge flags and must be set 0. The LSB in the return value refers to the 8bit 'return code' parameter in the CONNACK message and must be set accordingly.

void(* mqtt_server_app_cbs::disconn)(const void *app_usr, bool d

Notify disconnection to the remote client. This routine is invoked by the server to declare to the application to a particular client has been terminated and follow-up, if needed, can now commence. This routine aids the application by indicating if an error condition had caused the disconnection.

Parameters:

[in]	**app_usr**	handle to refer to the user of the connection in the application
[in]	**due2err**	has the connection been closed due to an error?

Returns:

none

void(* mqtt_server_app_cbs::publish)(const struct utf8_string *top

Indicate a PUBLISH message from the network. This routine presents, to the application, the topic and the related data along with other qualifiers published by one of the clients associated with this server. The application must enroll with the server the particular topic for which the data should be notified.

Parameters:

[in]	**topic**	UTF8 topic Name for which data has been published
[in]	**data_buf**	the published binary data for the topic
[in]	**data_len**	the length of the binary data
[in]	**dup**	is this a duplicate data from remote client?
[in]	**qos**	quality of service of the message / data
[in]	**retain**	should the server retain the data?

Returns:
none

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/**server_core.h**

mqtt_server_app_cfg Struct Reference

The Server Daemon API(s)

```
#include <server_core.h>
```

List of all members.

Public Attributes

void *	**place_holder**

Detailed Description

Configuration for the applications that utilize the MQTT Server Daemon. At the moment this configuration is not used and has been incorporated to support forward compatibility (future use)

Member Data Documentation

void* mqtt_server_app_cfg::place_holder

Dummy, not used as of now

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/**server_core.h**

Public Attributes

mqtt_server_lib_cfg Struct Reference

The Server Library API(s)

List of all members.

Public Attributes

u16	**listener_port**
u16	**loopback_port**
void *	**mutex**
void(*	**mutex_lockin**)(void ***mutex**)
void(*	**mutex_unlock**)(void ***mutex**)
i32(*	**debug_printf**)(const i8 *format,...)
bool	**aux_debug_en**

Member Data Documentation

bool mqtt_server_lib_cfg::aux_debug_en

Assert to indicate additional debug info

i32(* mqtt_server_lib_cfg::debug_printf)(const i8 *format,...)

Debug, mandatory

u16 mqtt_server_lib_cfg::listener_port

Port to listen to incoming network connections from the clients

u16 mqtt_server_lib_cfg::loopback_port

If the server application has more than one task and / or supports at-least one plug-in, then a non-zero value must be provided. Otherwise, this parameter must be set to zero. This parameter is used by the implementation to synchronize multiple tasks for the network connection.

void* mqtt_server_lib_cfg::mutex

For a multi-task enviroment, provide a handle to platform mutex

---

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/server_pkts.h

mqtt_server_msg_cbs Struct Reference

The Server Library API(s)

```
#include <server_pkts.h>
```

List of all members.

Public Attributes

u16(*	**connect_rx**)(void *ctx_cl, u8 conn_flags, struct **utf8_string** *const *utf8_vec, void **usr)
bool(*	**sub_msg_rx**)(void *usr_cl, const struct **utf8_strqos** *qos_topics, u32 n_topics, u16 msg_id, u8 *acks)
bool(*	**un_sub_msg**)(void *usr_cl, const struct **utf8_string** *topics, u32 n_topics, u16 msg_id)
bool(*	**pub_msg_rx**)(void *usr_cl, const struct **utf8_string** *topic, const u8 *data_buf, u32 data_len, u16 msg_id, bool dup, enum **mqtt_qos** qos, bool retain)
bool(*	**ack_notify**)(void *usr_cl, u8 msg_type, u16 msg_id)
void(*	**on_cl_net_close**)(void *usr_cl, bool due2err)
void(*	**on_connack_send**)(void *usr_cl, bool clean_session)

Detailed Description

Working Principle for implementing the call-back services:
Implementation of the call-back services should report in return value, only about errors found in the RX processing. Specifically, the status of TX as a follow-up to RX message (represented as a call-back service) need not be reported to the server packet library.

Error observed in TX (supported by appropriate API(s) / services in the service packet library) is recorded in the 'context' and shall be dealt in the next iteration of RX loop.

Member Data Documentation

bool(* mqtt_server_msg_cbs::ack_notify)(void *usr_cl, u8 msg_typ

Notify the acknowledgement that was received from the remote client. Following are the messages that are notified by the server LIB.

PUBACK, PUBREC, PUBREL, PUBCOMP

On return from this routine, if the application has found problem in processing the ACK message, then the LIB will simply terminate the associated network connection

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**msg_type**	refers to the type of ACK message
[in]	**msg_id**	the associated message ID provided by the client

Returns:

application should return false if the ACK was not expected by it or no reference was found for it. Otherwise true.

u16(* mqtt_server_msg_cbs::connect_rx)(void *ctx_cl, u8 conn_fla

Indicate the CONNECT Message from the client This routine provides, to the application, information about the connection request that a remote client has made. The application should utilize the credential and other data presented by the LIB to authenticate, analyze and finally, approve or deny the request.

Application at its discretion can also imply deployment specific policies to make decision about allowing or refusing the request.

The return value of this routine is a 16bit value that commensurates

with the 'variable header' of the CONNACK message. Specifically, the LSB of the 16bit return value corresponds to the 8bit 'return code' parameter in the CONNACK message and the MSB to the 'acknowledge flags'. The application must set a valid return value.

The LIB uses the return value to compose the CONNACK message to the remote client. If the LSB of return value is a 'non zero' value, then the LIB, after sending the CONNACK message to the remote client, will terminate the network connection.

Parameters:

[in]	**ctx_cl**	handle to the underlying network context in the LIB
[in]	**conn_flags**	options received in CONNECT message from server
[in]	**utf8_vec**	vector / array of pointers to UTF8 information. The order of UTF8 information is client-id, will topic, will-message, user-name and pass-word. A NULL in vector element indicates absence of that particular UTF8 information.
[out]	**usr**	place holder for application to provide connection specific handle. In subsequent calls from the implementation this handle will be passed as a parameter to enable application to refer to the particular active connection.

Returns:

16bit value for the variable header of the CONNACK message

void(* mqtt_server_msg_cbs::on_cl_net_close)(void *usr_cl, bool

Notify that network connection to client has been closed. This routine is invoked by the LIB to declare to the application that the network connection to a particular client has been terminated and follow-up, if

needed, can now commence. If configured, removal of the client session and / or dispatch of the WILL message, will be typical aspects, among others, to follow-up. The routine aids the application by indicating if an error condition had caused the closure.

This routine is invoked by the LIB irrespective of the source entity, server or client, that has caused the closure of the network.

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**due2err**	has the connection been closed due to an error?

void(* mqtt_server_msg_cbs::on_connack_send)(void *usr_cl, boo

Notify that CONNACK has been sent to the specified client. This routine is invoked by the LIB to enable the application to make progress and follow-up on the session information for the particular client. Specifically, this routine facilitates the application to either delete the session or re-send / sync-up the pending messages associated with the client. The follow-up action is depenedent upon the 'clean_session' option in the CONNECT message from the client.

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**clean_session**	was a clean session requested in CONNECT?

bool(* mqtt_server_msg_cbs::pub_msg_rx)(void *usr_cl, const str

Indicate the PUBLISH Message from the client. This routine provides, to the application, the binary data along-with its qualifiers and the topic to which a remote client has published data.

On return from this routine, if the application has found a problem in

processing of the contents of the PUBLISH message, the LIB will simply terminate the associated network connection. Otherwise, depending upon the QoS level of the PUBLISH message, the LIB shall dispatch the ACK (PUBACK or PUBREC) to the client, thereby, relieveing the application from this support.

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**topic**	UTF8 Topic Name for which data has been published
[in]	**data_buf**	the published binary data for the topic
[in]	**data_len**	the length of the binary data
[in]	**msg_id**	the associated message ID provided by the client
[in]	**dup**	has client indicated this as a duplicate message
[in]	**qos**	quality of service of the message
[in]	**retain**	should the server retain the data?

Returns:

The application should return false, if it encounters any problem in the processing of data, topic and related resources. Otherwise, true.

bool(* mqtt_server_msg_cbs::sub_msg_rx)(void *usr_cl, const str

Indicate the SUBSCRIBE Message from the client This routine provides, to the application, information about the topics that the remote client intends to subscribe to.

On return from this routine, if the application has found a problem in the processing of message, then the LIB will simply terminate the associated network connection.

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**qos_topics**	an array of topic along-with its qos
[in]	**n_topics**	the count / number of elements in the array
[in]	**msg_id**	the associated message ID provided by the client
[in]	**acks**	place holder array for the application to provide finalized qos for each of the subscribed topic. The order of ack is same as that of qos_topics

Returns:

The application should return false, if it encounters any problem in the processing of topic. Otherwise, true.

Note:

The memory space pointed by the 'buffer' field in the elements of 'qos_topics' array has an additional byte available beyond the size of memory indicated by the 'length' field. This extra byte can be used by the application to NUL terminate the buffer. **This quirk is applicable to this routine only.**

bool(* mqtt_server_msg_cbs::un_sub_msg)(void *usr_cl, const str

Indicate the UNSUBSCRIBE Message from the client This routine provides, to the application, information about the topics that the remote client intends to unsubscribe.

On return from this routine, if the application has found a problem in the processing of message, then the LIB will simply terminate the associated network connection.

Parameters:

[in]	**usr_cl**	handle to connection context in the application
[in]	**topics**	an array of topic in the message
[in]	**n_topics**	the count / number of elements in the array

[in] **msg_id** the associated message ID provided by the client

Returns:

The application should return false, if it encounters any problem in the processing of topic. Otherwise, true.

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/**server_pkts.h**

Public Attributes

pub_qos2_cq Struct Reference

List of all members.

Public Attributes

u16	**id_vec** [MAX_PUBREL_INFLT]
u8	**n_free**
u8	**rd_idx**
u8	**wr_idx**

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

secure_conn Struct Reference

Information to establish a secure connection.

List of all members.

Public Attributes

void *	**method**
void *	**cipher**
u32	**n_file**
char **	**files**

Member Data Documentation

void* secure_conn::cipher

Reference to information about cryptograph ciphers

char** secure_conn::files

Reference to array of file-names used for security

void* secure_conn::method

Reference to information about protocol or methods

u32 secure_conn::n_file

Count of secure connection related files, certs...

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

utf8_string Struct Reference

```
#include <mqtt_common.h>
```

List of all members.

Public Attributes

i8 *	**buffer**
u16	**length**

Detailed Description

Description of UTF8 information as used by MQTT Library.

Member Data Documentation

i8* utf8_string::buffer

Refers to UTF8 content

u16 utf8_string::length

Length of UTF8 content

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Public Attributes

utf8_strqos Struct Reference

```
#include <mqtt_common.h>
```

List of all members.

Public Attributes

i8 *	**buffer**
u16	**length**
enum **mqtt_qos**	**qosreq**

Detailed Description

Construct to create Topic to SUBSCRIBE

Member Data Documentation

i8* utf8_strqos::buffer

Refers to UTF8 content

u16 utf8_strqos::length

Length of UTF8 content

enum mqtt_qos utf8_strqos::qosreq

QoS Level for content

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/**mqtt_commc**

Class Index

C | D | M | P | S | U

C

client_ctx

D

device_net_services

M

mqtt_ack_wlist

mqtt_packet

mqtt_server_app_cbs

mqtt_server_app_cf

mqtt_server_lib_cfg

mqtt_server_msg_cb

P

C | D | M | P | S | U

Here is a list of all documented class members with links to the class documentation for each member:

- a -

- accept : **device_net_services**
- ack_notify : **mqtt_server_msg_cbs**
- aux_debug_en : **mqtt_server_lib_cfg**

- b -

- buffer : **mqtt_packet** , **utf8_strqos** , **utf8_string**

- c -

- cipher : **secure_conn**
- close : **device_net_services**
- connect : **mqtt_server_app_cbs**
- connect_rx : **mqtt_server_msg_cbs**

- d -

- debug_printf : **mqtt_server_lib_cfg**
- disconn : **mqtt_server_app_cbs**

- f -

- fh_byte1 : **mqtt_packet**
- fh_len : **mqtt_packet**

- files : **secure_conn**
- free : **mqtt_packet**

- i -

- io_mon : **device_net_services**

- l -

- length : **utf8_string** , **utf8_strqos**
- listen : **device_net_services**
- listener_port : **mqtt_server_lib_cfg**
- loopback_port : **mqtt_server_lib_cfg**

- m -

- maxlen : **mqtt_packet**
- method : **secure_conn**
- msg_id : **mqtt_packet**
- msg_type : **mqtt_packet**
- mutex : **mqtt_server_lib_cfg**

- n -

- n_file : **secure_conn**
- n_refs : **mqtt_packet**

- o -

- offset : **mqtt_packet**
- on_cl_net_close : **mqtt_server_msg_cbs**
- on_connack_send : **mqtt_server_msg_cbs**
- open : **device_net_services**

- p -

- pl_len : **mqtt_packet**
- place_holder : **mqtt_server_app_cfg**
- pub_msg_rx : **mqtt_server_msg_cbs**
- publish : **mqtt_server_app_cbs**

- q -

- qosreq : **utf8_strqos**

- r -

- recv : **device_net_services**
- recv_from : **device_net_services**

- s -

- send : **device_net_services**
- send_dest : **device_net_services**
- sub_msg_rx : **mqtt_server_msg_cbs**

- t -

- time : **device_net_services**

- u -

- un_sub_msg : **mqtt_server_msg_cbs**

- v -

- vh_len : **mqtt_packet**

File List

Here is a list of all documented files with brief descriptions:

MainPage.h [code]

D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_comm [code]

D:/my_data/GIT/network_apps/netapps/mqtt/server/server_core.h [code]

D:/my_data/GIT/network_apps/netapps/mqtt/server/server_pkts.h [code]

D:/my_data/GIT/network_apps/netapps/mqtt/con File Reference

```
#include <stdbool.h> #include <stdlib.h>
#include <stdio.h>
```

Go to the source code of this file.

Classes

struct	**mqtt_packet**
struct	**utf8_string**
struct	**mqtt_ack_wlist**
struct	**utf8_strqos**
struct	**secure_conn**
struct	**device_net_services**
struct	**pub_qos2_cq**
struct	**client_ctx**

Defines

#define	**MQTT_COMMON_VERSTR** "1.0.0"
#define	**MIN**(a, b) ((a > b)? b : a)
#define	**MQTT_CONNECT** 0x01
#define	**MQTT_CONNACK** 0x02
#define	**MQTT_PUBLISH** 0x03
#define	**MQTT_PUBACK** 0x04
#define	**MQTT_PUBREC** 0x05
#define	**MQTT_PUBREL** 0x06
#define	**MQTT_PUBCOMP** 0x07
#define	**MQTT_SUBSCRIBE** 0x08
#define	**MQTT_SUBACK** 0x09
#define	**MQTT_UNSUBSCRIBE** 0x0A
#define	**MQTT_UNSUBACK** 0x0B
#define	**MQTT_PINGREQ** 0x0C
#define	**MQTT_PINGRSP** 0x0D
#define	**MQTT_DISCONNECT** 0x0E
#define	**MAX_FH_LEN** 0x05
#define	**MAX_REMLEN_BYTES** (MAX_FH_LEN - 1)
#define	**MAKE_FH_BYTE1**(msg_type,flags) (u8)((msg_type << 4) | fl
#define	**MAKE_FH_FLAGS**(bool_dup, enum_qos, bool_retain) (u8)((3) | (enum_qos << 1) | bool_retain) & 0xF)
#define	**QOS_VALUE**(enum_qos) (u8)(enum_qos & 0x3)
#define	**QFL_VALUE** 0x80
#define	**DUP_FLAG_VAL**(bool_val) (u8)(bool_val << 3)
#define	**BOOL_RETAIN**(fh_byte1) ((fh_byte1 & 0x1)? true : false)
#define	**BOOL_DUP**(fh_byte1) ((fh_byte1 & 0x8)? true : false)
#define	**ENUM_QOS**(fh_byte1) (enum **mqtt_qos**)((fh_byte1 & 0x6) >>
#define	**MSG_TYPE**(fh_byte1) (u8)((fh_byte1 & 0xf0) >> 4)
#define	**MQP_FHEADER_BUF**(mqp) (mqp->buffer + mqp->offset)
#define	**MQP_VHEADER_BUF**(mqp) (MQP_FHEADER_BUF(mqp) + >fh_len)
#define	**MQP_PAYLOAD_BUF**(mqp) (MQP_VHEADER_BUF(mqp) + >vh_len)

#define	**MQP_CONTENT_LEN**(mqp) (mqp->fh_len + mqp->vh_len +
#define	**MQP_FREEBUF_LEN**(mqp)
#define	**MQP_FHEADER_VAL**(mqp) (mqp->fh_byte1)
#define	**MQP_FHEADER_MSG**(mqp) (MSG_TYPE(MQP_FHEADER
#define	**MQP_FHEADER_FLG**(mqp) (MSG_FLAGS(MQP_FHEADEF
#define	**DEFINE_MQP_VEC**(num_mqp, mqp_vec) static struct **mqtt_** mqp_vec[num_mqp];
#define	**DEFINE_MQP_BUF_VEC**(num_mqp, mqp_vec, buf_len, buf_
#define	**MQP_PUB_TOP_BUF**(mqp) (MQP_VHEADER_BUF(mqp) +
#define	**MQP_PUB_TOP_LEN**(mqp) (mqp->vh_len - 2 - (mqp->msg_
#define	**MQP_PUB_PAY_BUF**(mqp) (mqp->pl_len? MQP_PAYLOAD_ NULL)
#define	**MQP_PUB_PAY_LEN**(mqp) (mqp->pl_len)
#define	**MQP_ERR_NETWORK** (-1)
#define	**MQP_ERR_TIMEOUT** (-2)
#define	**MQP_ERR_NET_OPS** (-3)
#define	**MQP_ERR_FNPARAM** (-4)
#define	**MQP_ERR_PKT_AVL** (-5)
#define	**MQP_ERR_PKT_LEN** (-6)
#define	**MQP_ERR_NOTCONN** (-7)
#define	**MQP_ERR_BADCALL** (-8)
#define	**MQP_ERR_CONTENT** (-9)
#define	**MQP_ERR_LIBQUIT** (-10)
#define	**MQP_ERR_NOT_DEF** (-32)
#define	**DEV_NETCONN_OPT_TCP** 0x01
#define	**DEV_NETCONN_OPT_UDP** 0x02
#define	**DEV_NETCONN_OPT_IP6** 0x04
#define	**DEV_NETCONN_OPT_URL** 0x08
#define	**DEV_NETCONN_OPT_SEC** 0x10
#define	**MAX_PUBREL_INFLT** 8
#define	**KA_TIMEOUT_NONE** 0xffffffff

Typedefs

typedef int	**i32**
typedef unsigned int	**u32**
typedef unsigned char	**u8**
typedef char	**i8**
typedef unsigned short	**u16**
typedef short	**i16**

Enumerations

enum **mqtt_qos** { **MQTT_QOS0**, **MQTT_QOS1**, **MQTT_QOS2** }

Functions

void	**mqp_free** (struct **mqtt_packet** *mqp)
void	**mqp_reset** (struct **mqtt_packet** *mqp)
void	**mqp_init** (struct **mqtt_packet** *mqp, u8 offset)
i32	**mqp_buf_wr_utf8** (u8 *buf, const struct **utf8_string** *utf8)
i32	**mqp_buf_tail_wr_remlen** (u8 *buf, u32 remlen)
i32	**mqp_buf_rd_remlen** (u8 *buf, u32 *remlen)
i32	**mqp_pub_append_topic** (struct **mqtt_packet** *mqp, const struct **utf8_string** *topic, u16 msg_id)
i32	**mqp_pub_append_data** (struct **mqtt_packet** *mqp, const u8 *data_buf, u32 data_len)
bool	**mqp_proc_msg_id_ack_rx** (struct **mqtt_packet** *mqp_raw, bool has_payload)
bool	**mqp_proc_pub_rx** (struct **mqtt_packet** *mqp_raw)
bool	**mqp_ack_wlist_append** (struct **mqtt_ack_wlist** *list, struct **mqtt_packet** *elem)
struct **mqtt_packet** *	**mqp_ack_wlist_remove** (struct **mqtt_ack_wlist** *list, u16 msg_id)
void	**mqp_ack_wlist_purge** (struct **mqtt_ack_wlist** *list)
i32	**mqp_prep_fh** (struct **mqtt_packet** *mqp, u8 flags)
i32	**mqp_recv** (i32 net, const struct **device_net_services** *net_ops, struct **mqtt_packet** *mqp, u32 wait_secs, bool *timed_out, void *ctx)
void	**qos2_pub_cq_reset** (struct **pub_qos2_cq** *cq)
bool	**qos2_pub_cq_logup** (struct **pub_qos2_cq** *cq, u16 msg_id)
bool	**qos2_pub_cq_unlog** (struct **pub_qos2_cq** *cq, u16 msg_id)

bool	**qos2_pub_cq_check** (struct **pub_qos2_cq** *cq, u16 msg_id)
void	**cl_ctx_reset** (struct **client_ctx** *cl_ctx)
void	**cl_ctx_timeout_insert** (struct **client_ctx** **head, struct **client_ctx** *elem)
void	**cl_ctx_remove** (struct **client_ctx** **head, struct **client_ctx** *elem)
void	**cl_ctx_timeout_update** (struct **client_ctx** *cl_ctx, u32 now_secs)

Detailed Description

This file incorporates constructs that are common to both client and server implementation.

The applications are not expected to utlize the routines made available in this module module.

Note:
the routines in this module do not check for availability and correctness of the input parameters

Warning:
The module is expected to under-go changes whilst incorporating support for the server. Therefore, it is suggested that applications do not rely on the services provided in this module.

Define Documentation

#define DEFINE_MQP_BUF_VEC (num_mqp, mqp_vec, buf_len, buf_vec)

Value:

```
DEFINE_MQP_VEC(num_mqp, mqp_vec);
                \
        static u8 buf_vec[num_mqp][buf_len];
```

#define MAX_FH_LEN 0x05

MAX Length of Fixed Header

#define MAX_REMLEN_BYTES (MAX_FH_LEN - 1)

Max number of bytes in remaining length field

#define MQP_FREEBUF_LEN (mqp)

Value:

```
(mqp->maxlen - mqp->offset -       \
                                MQP_CONTENT_LEN(mq
p))
```

#define MQTT_COMMON_VERSTR "1.0.0"

Version of Common LIB

#define MQTT_CONNECT 0x01

MQTT Message Types

#define QFL_VALUE 0x80

QOS Failure value (SUBACK)

Enumeration Type Documentation

enum mqtt_qos

MQTT Quality of Service

Enumerator:

MQTT_QOS0 QoS Level 0

MQTT_QOS1 QoS Level 1

MQTT_QOS2 QoS Level 2

Function Documentation

i32 mqp_buf_rd_remlen (u8 * buf,
u32 * remlen
)

Read MQTT construct 'Remaining Length' from leading bytes of the buffer. The 'remaining length' is written in the format as outlined in the MQTT specification.

Parameters:

[in]	**buf**	refers to memory to head-read 'Remaining Length' from
[in]	**remlen**	place-holder for The 'Remaining Length' value

Returns:

in success, number of header bytes read, otherwise -1 on error

i32 mqp_buf_tail_wr_remlen (u8 * buf,
u32 remlen
)

Write the MQTT construct 'Remaining Length' into trailing end of buffer. The 'remaining length' is written in the format as outlined in the MQTT specification.

The implementation assumes availability of at-least 4 bytes in the buffer. Depending on the value of 'Remaining Length' appropriate trailing bytes in the buffer would be used.

Parameters:

[in]	**buf**	refers to memory to tail-write 'Remaining Length' into
[in]	**remlen**	The 'Remaining Length' value

Returns:
in success, number of trailing bytes used, otherwise -1 on error

i32 mqp_buf_wr_utf8 (u8 * buf,
const struct utf8_string * utf8
)

Write UTF8 information into the buffer. The UTF8 information includes content and its length.

Warning:
The routine does not check for correctness of the paramters.

Parameters:
`[in]` **buf** refers to memory to write UTF8 information into
`[in]` **utf8** contains UTF8 information to be written

Returns:
on success, number of bytes written, otherwise -1 on error.

void mqp_free (struct mqtt_packet * mqp)

Free a MQTT Packet Buffer Puts back the packet buffer in to the appropriate pool.

Parameters:
`[in]` **mqp** packet buffer to be freed

Returns:
none

void mqp_init (struct mqtt_packet * mqp,
u8 offset
)

Initializes attributes of the MQTT Packet Holder. This routine sets number of users of the MQTT Packet Holder to 1. However, it leaves, if already provisioned, the reference to buffer and its size un-altered.

Parameters:

`[in]` **mqp** packet buffer to be initialized

`[in]` **offset** index in buffer to indicate start of the contents

Returns:

none

i32 mqp_prep_fh (struct mqtt_packet * mqp,
u8 flags
)

Prepare the Fixed-Header of the MQTT Packet (before being sent to network) Based on the contents of the mqtt packet and the combination of DUP, QoS and Retain flags as outlined the MQTT specification, the routine updates, among others, significant internal fields such as 'remaining length' and 'fixed header length' in the packet construct and embeds the fixed header, so created, in the packet buffer.

This service must be utilized on a packet that has been already populated with all the payload data, topics and other contents. The fixed header must be the final step in the compostion of MQTT packet prior to its dispatch to the server.

Returns size, in bytes, of the fixed-header, otherwise -1 on error.

bool mqp_proc_msg_id_ack_rx (struct mqtt_packet * mqp_raw,
bool has_payloa
)

Construct a packet for Message ID enabled ACK received from network Process the raw ACK message information to update the

packet holder.

Warning:
This routine does not check for correctness of the input parameters.

Parameters:

[in]	**mqp_raw**	holds a raw buffer from the network
[in]	**has_payload**	asserted, if ACK message should have a payload

Returns:
on success, true, otherwise false

bool mqp_proc_pub_rx (struct mqtt_packet * mqp_raw)

Construct a packet for PUBLISH message received from the network Process the raw PUB message information to update the packet holder.

Warning:
This routine does not check for correctness of the input parameters.

Parameters:

[in]	**mqp_raw**	holds a raw buffer from the network

Returns:
on success, true, other wise false

i32 mqp_pub_append_data (struct mqtt_packet * mqp,
const u8 * data_buf,
u32 data_len
)

Include payload data for publishing The payload data is associated

with a topic.

Warning:
This routine does not check for correctness of the input parameters.

Parameters:

[in]	**mqp**	packet buffer in which payload data must be included.
[in]	**data_buf**	data to be included in the packet buffer
[in]	**data_len**	length of the data to be included in the packet.

Returns:
on success, number of bytes appended, otherwise -1 on error.

Note:
A 'topic' must be appended prior to inclusion of pulished data.

---

i32 mqp_pub_append_topic (struct mqtt_packet * mqp,
const struct utf8_string * topic,
u16 msg_id
)

Include variable header Topic as part of PUB Message construction. Inclusion of a Topic also encompasses incorporation of the message ID.

The topic refers to the subject for which data will be published by the client or the server. The topic entity must be appended into the packet buffer prior to the inclusion of the payload (data).

Warning:
This routine does not check for correctness of the input parameters.

Parameters:

[in] **mqp** packet buffer in which topic must be included.
[in] **topic** UTF8 information
[in] **msg_id** Message or Packet transaction ID

Returns:
on success, number of bytes appended, otherwise -1 on error.

Note:
A 'topic' must be appended prior to inclusion of pulished data.

void mqp_reset (struct mqtt_packet * mqp)

Resets the attributes of MQTT Packet Holder to its init state Not all fields are reset - entities such as offset, n_refs in addition to buffer information are not updated.

Parameters:
[in] **mqp** packet buffer to be reset

Returns:
none

See also:
mqp_init

D:/my_data/GIT/network_apps/netapps/mqtt/ser File Reference

```
#include "server_pkts.h"
```

Go to the source code of this file.

Classes

struct	**mqtt_server_app_cbs**
struct	**mqtt_server_app_cfg**

Functions

i32	**mqtt_server_topic_enroll** (const void *app_hnd, const struct **utf8_string** *topic, enum **mqtt_qos** qos)
i32	**mqtt_server_topic_disenroll** (const void *app_hnd, const struct **utf8_string** *topic)
i32	**mqtt_server_app_pub_send** (const struct **utf8_string** *topic, const u8 *data_buf, u32 data_len, enum **mqtt_qos** qos, bool retain)
void *	**mqtt_server_app_register** (const struct **mqtt_server_app_cbs** *cbs, const i8 *name)
i32	**mqtt_server_init** (const struct **mqtt_server_lib_cfg** *lib_cfg, const struct **mqtt_server_app_cfg** *app_cfg)

Detailed Description

The MQTT server daemon, a task, provisions the high level abstractions for the smart applicaltions. This is an intelligent layer that utilizes the services of the MQTT Server Packet Library and is responsible for the functions of the topic management, the client management and support of multiple server applications.

The light-weight server enables the services of the MQTT protocol for an application to either extend and / or control the existing functions to suit the deployment specific scenario. These applications in practice are the plug-ins / add-ons to the core server functionalities. Specifically, these applications, among others capabilities, can be used for analyzing and approving the credentials of the remote clients, acting as a bridge between a MQTT external client and the server, and a snooper to learn about all the data transactions to / from the server.

The server is targeted to conform to MQTT 3.1.1 specification.

The services of the server are multi-task safe. Platform specific atomicity constructs are used, through abstractions, by the server to maintain data coherency and synchronization.

The server offers the following compile line configurable parameters (-D opt)

- **CFG_SR_MAX_MQP_TX_LEN:** the constant buffer length allocated for a TX.

- **CFG_SR_MAX_SUBTOP_LEN:** the maximum buffer size to hold a sub-topic. For e.g., in the topic /x/y/z, the phrase /x, /y and z are sub-topics.

- **CFG_SR_MAX_TOPIC_NODE:** the maximum number of topic nodes in server. For e.g., in the topic /x/y/z, there are three nodes (/x, /y and z).

- **CFG_SR_MAX_CL_ID_SIZE:** the maximum length of the client-id

string.

- **CFG_SR_MAX_NUM_CLIENT:** the maximum number of clients to be managed. Note this is different from the maximum number of 'contexts'. A large number of clients can be managed using fewer number of 'contexts' (connections).

> **Note:**
> Any future extensions & development must follow the following guidelines. A new API or an extension to the existing API a) must be rudimentary b) must not imply a rule or policy (including a state machine) b) must ensure simple design and implementation

---

D:/my_data/GIT/network_apps/netapps/mqtt/ser File Reference

```
#include "mqtt_common.h"
```

Go to the source code of this file.

Classes

struct	**mqtt_server_msg_cbs**
struct	**mqtt_server_lib_cfg**

Defines

#define	**MQ_CONN_UTF8_BUF**(utf8) ((utf8)? (utf8)->buffer : NULL)
#define	**MQ_CONN_UTF8_LEN**(utf8) ((utf8)? (utf8)->length : 0)
#define	**MQC_UTF8_CLIENTID**(utf8_vec) (utf8_vec[0])
#define	**MQC_UTF8_WILL_TOP**(utf8_vec) (utf8_vec[1])
#define	**MQC_UTF8_WILL_MSG**(utf8_vec) (utf8_vec[2])
#define	**MQC_UTF8_USERNAME**(utf8_vec) (utf8_vec[3])
#define	**MQC_UTF8_PASSWORD**(utf8_vec) (utf8_vec[4])
#define	**MQC_CLIENTID_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(MC
#define	**MQC_CLIENTID_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(MC
#define	**MQC_WILL_TOP_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(M
#define	**MQC_WILL_TOP_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(M
#define	**MQC_WILL_MSG_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(M
#define	**MQC_WILL_MSG_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(M
#define	**MQC_USERNAME_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(
#define	**MQC_USERNAME_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(U
#define	**MQC_PASSWORD_BUF**(utf8_vec) MQ_CONN_UTF8_BUF(
#define	**MQC_PASSWORD_LEN**(utf8_vec) MQ_CONN_UTF8_LEN(
#define	**MQP_SERVER_RX_LEN** 1024

Functions

i32	**mqtt_vh_msg_send** (void *ctx_cl, u8 msg_type, enum **mqtt_qos** qos, bool has_vh, u16 vh_data)
i32	**mqtt_vh_msg_send_locked** (void *ctx_cl, u8 msg_type, enum **mqtt_qos** qos, bool has_vh, u16 vh_data)
i32	**mqtt_server_pub_dispatch** (void *ctx_cl, struct **mqtt_packet** *mqp, bool dup)
i32	**mqtt_server_pub_dispatch_locked** (void *ctx_cl, struct **mqtt_packet** *mqp, bool dup)
i32	**mqtt_server_run** (u32 wait_secs)
i32	**mqtt_server_register_net_svc** (const struct **device_net_services** *net)
i32	**mqtt_server_lib_init** (const struct **mqtt_server_lib_cfg** *cfg, const struct **mqtt_server_msg_cbs** *cbs)

Detailed Description

The C library provisions the interface / API(s) for the MQTT Server Packet LIB.

This is a light-weight library that enables the services of the MQTT protcol for user's server application(s) to exchange the MQTT packets with one or more remote clients. The Server Packet LIB is a simple and easy-to-use implementation to support both un-packing of the messages received from the remote clients and formation of packets to be sent to the remote clients.

The library is targeted to conform to MQTT 3.1.1 specification.

The Server Packet LIB is a highly portable software and implies a very limited set of dependencies on a platform. Importantly, these limited dependencies are common features used in the embedded and the networking world, and can be easily adapted to the target platforms. The services of the library are multi-task safe. Platform specific atomicity constructs are used, through abstractions, by the library to maintain data coherency and synchronization. In addition, the library can be configured to support several in-flight messages.

The Server Packet LIB can support multiple and simultaneous MQTT connections from clients. However, the responsibility of managing the clients and topics, authentication and approval for connections and supporting any other needs that specific to the deployment remains with the user application.

The number of the network connections that the Server Packet LIB can support is configurable through the compile line option / flag **-DCFG_SR_MQTT_CTXS** . In addition, the maximum length of the RX buffer used by the server is also configurable through the compile line option / flag **-DCFG_SR_MAX_MQP_RX_LEN** .

> **Note:**
>
> Any future extensions & development must follow the following guidelines. A new API or an extension to the existing API a) must be rudimentary b) must not imply a rule or policy (including a state

machine) b) must ensure simple design and implementation

Here is a list of all documented file members with links to the documentation:

- d -

- DEV_NETCONN_OPT_IP6 : **mqtt_common.h**
- DEV_NETCONN_OPT_SEC : **mqtt_common.h**
- DEV_NETCONN_OPT_TCP : **mqtt_common.h**
- DEV_NETCONN_OPT_UDP : **mqtt_common.h**
- DEV_NETCONN_OPT_URL : **mqtt_common.h**

- m -

- MAX_FH_LEN : **mqtt_common.h**
- MAX_REMLEN_BYTES : **mqtt_common.h**
- MQ_CONN_UTF8_BUF : **server_pkts.h**
- MQ_CONN_UTF8_LEN : **server_pkts.h**
- MQC_UTF8_CLIENTID : **server_pkts.h**
- MQC_UTF8_PASSWORD : **server_pkts.h**
- MQC_UTF8_USERNAME : **server_pkts.h**
- MQC_UTF8_WILL_MSG : **server_pkts.h**
- MQC_UTF8_WILL_TOP : **server_pkts.h**
- mqp_buf_rd_remlen() : **mqtt_common.h**
- mqp_buf_tail_wr_remlen() : **mqtt_common.h**
- mqp_buf_wr_utf8() : **mqtt_common.h**
- MQP_ERR_BADCALL : **mqtt_common.h**
- MQP_ERR_CONTENT : **mqtt_common.h**
- MQP_ERR_FNPARAM : **mqtt_common.h**

- MQP_ERR_LIBQUIT : **mqtt_common.h**
- MQP_ERR_NET_OPS : **mqtt_common.h**
- MQP_ERR_NETWORK : **mqtt_common.h**
- MQP_ERR_NOT_DEF : **mqtt_common.h**
- MQP_ERR_NOTCONN : **mqtt_common.h**
- MQP_ERR_PKT_AVL : **mqtt_common.h**
- MQP_ERR_PKT_LEN : **mqtt_common.h**
- MQP_ERR_TIMEOUT : **mqtt_common.h**
- mqp_free() : **mqtt_common.h**
- mqp_init() : **mqtt_common.h**
- mqp_prep_fh() : **mqtt_common.h**
- mqp_proc_msg_id_ack_rx() : **mqtt_common.h**
- mqp_proc_pub_rx() : **mqtt_common.h**
- mqp_pub_append_data() : **mqtt_common.h**
- mqp_pub_append_topic() : **mqtt_common.h**
- MQP_PUB_PAY_BUF : **mqtt_common.h**
- MQP_PUB_PAY_LEN : **mqtt_common.h**
- MQP_PUB_TOP_BUF : **mqtt_common.h**
- MQP_PUB_TOP_LEN : **mqtt_common.h**
- mqp_reset() : **mqtt_common.h**
- MQP_SERVER_RX_LEN : **server_pkts.h**
- MQTT_COMMON_VERSTR : **mqtt_common.h**
- MQTT_CONNECT : **mqtt_common.h**
- mqtt_qos : **mqtt_common.h**
- MQTT_QOS0 : **mqtt_common.h**
- MQTT_QOS1 : **mqtt_common.h**
- MQTT_QOS2 : **mqtt_common.h**
- mqtt_server_app_pub_send() : **server_core.h**
- mqtt_server_app_register() : **server_core.h**
- mqtt_server_init() : **server_core.h**
- mqtt_server_lib_init() : **server_pkts.h**
- mqtt_server_pub_dispatch() : **server_pkts.h**
- mqtt_server_pub_dispatch_locked() : **server_pkts.h**
- mqtt_server_register_net_svc() : **server_pkts.h**
- mqtt_server_run() : **server_pkts.h**
- mqtt_server_topic_disenroll() : **server_core.h**
- mqtt_server_topic_enroll() : **server_core.h**
- mqtt_vh_msg_send() : **server_pkts.h**
- mqtt_vh_msg_send_locked() : **server_pkts.h**

- q -

- QFL_VALUE : **mqtt_common.h**

---

client_ctx Member List

This is the complete list of members for **client_ctx**, including all inherited members.

flags (defined in **client_ctx**)	**client_ctx**
ip_length (defined in **client_ctx**)	**client_ctx**
ka_secs (defined in **client_ctx**)	**client_ctx**
net (defined in **client_ctx**)	**client_ctx**
next (defined in **client_ctx**)	**client_ctx**
remote_ip (defined in **client_ctx**)	**client_ctx**
timeout (defined in **client_ctx**)	**client_ctx**
usr (defined in **client_ctx**)	**client_ctx**

D:/my_data/GIT/network_apps/netapps/mqtt/con

Go to the documentation of this file.

```
00001 /*****************************************
*********************************
00002 *
00003 *   Copyright (C) 2014 Texas Instruments Inc
orporated
00004 *
00005 *   All rights reserved. Property of Texas I
nstruments Incorporated.
00006 *   Restricted rights to use, duplicate or d
isclose this code are
00007 *   granted through contract.
00008 *
00009 *   The program may not be used without the
written permission of
00010 *   Texas Instruments Incorporated or agains
t the terms and conditions
00011 *   stipulated in the agreement under which
this program has been supplied,
00012 *   and under no circumstances can it be use
d with non-TI connectivity device.
00013 *
00014 *****************************************
*********************************/
00015
00016 /*
```

```
00017   mqtt_common.h
00018
00019   This module outlines the interfaces that a
re common to both client amd
00020   server components. The applications are no
t expected to utilize the
00021   services outlined in this module.
00022 */
00023
00024 #ifndef __MQTT_COMMON_H__
00025 #define __MQTT_COMMON_H__
00026
00042 #include <stdbool.h>
00043 #include <stdlib.h>
00044 #include <stdio.h>
00045
00046 #define MQTT_COMMON_VERSTR "1.0.0"
00048 typedef int             i32;
00049 typedef unsigned int   u32;
00050 typedef unsigned char   u8;
00051 typedef char            i8;
00052 typedef unsigned short u16;
00053 typedef short          i16;
00054
00055 #define MIN(a, b) ((a > b)? b : a)
00056
00058 #define MQTT_CONNECT      0x01
00059 #define MQTT_CONNACK      0x02
00060 #define MQTT_PUBLISH      0x03
00061 #define MQTT_PUBACK       0x04
00062 #define MQTT_PUBREC       0x05
00063 #define MQTT_PUBREL       0x06
00064 #define MQTT_PUBCOMP      0x07
00065 #define MQTT_SUBSCRIBE    0x08
00066 #define MQTT_SUBACK       0x09
00067 #define MQTT_UNSUBSCRIBE  0x0A
00068 #define MQTT_UNSUBACK     0x0B
```

```
00069 #define MQTT_PINGREQ      0x0C
00070 #define MQTT_PINGRSP      0x0D
00071 #define MQTT_DISCONNECT   0x0E
00072
00073 #define MAX_FH_LEN        0x05
00076 #define MAX_REMLEN_BYTES  (MAX_FH_LEN - 1)
00077
00078 #define MAKE_FH_BYTE1(msg_type,  flags) (u8)
((msg_type << 4) | flags)
00079
00080 #define MAKE_FH_FLAGS(bool_dup, enum_qos, bo
ol_retain)                    \
00081         (u8)(((bool_dup << 3) | (enum_qos <<
 1) | bool_retain) & 0xF)
00082
00083 #define QOS_VALUE(enum_qos) (u8)(enum_qos &
0x3)
00084 #define QFL_VALUE           0x80
00086 #define DUP_FLAG_VAL(bool_val) (u8)(bool_val
 << 3)
00087
00088 #define BOOL_RETAIN(fh_byte1)  ((fh_byte1 &
0x1)? true : false)
00089 #define BOOL_DUP(fh_byte1)     ((fh_byte1 &
0x8)? true : false)
00090 #define ENUM_QOS(fh_byte1)     (enum mqtt_qo
s)((fh_byte1 & 0x6) >> 1)
00091
00092 #define MSG_TYPE(fh_byte1)  (u8)((fh_byte1 &
 0xf0) >> 4)
00093
00094 static inline u32 buf_wr_nbytes(u8 *dst, con
st u8 *src, u32 n)
00095 {
00096         u32 c = n;
00097         while(c--)
00098                 *dst++ = *src++;
```

```
00099
00100         return n;
00101 }
00102
00103 static inline u32 buf_set(u8 *dst, u8 val, u
32 n)
00104 {
00105         u32 c = n;
00106         while(c--)
00107                 *dst++ = val;
00108
00109         return n;
00110 }
00111
00113 static inline u32 buf_wr_nbo_2B(u8 *buf, u16
 val)
00114 {
00115         buf[0] = (u8)((val >> 8) & 0xFF); /*
 MSB */
00116         buf[1] = (u8)((val)      & 0xFF); /*
 LSB */
00117         return 2;
00118 }
00119
00121 static inline u32 buf_rd_nbo_2B(const u8 *bu
f, u16 *val)
00122 {
00123         *val = (u16)((buf[0] << 8) | (buf[1]
));
00124         return 2;
00125 }
00126
00132 struct mqtt_packet {
00133
00134         u8                      msg_type;
00135         u8                      fh_byte1;
00137         u16                     msg_id;
```

```
00139         u8                      n_refs;
00140         u8                      pad[3];
00141
00142         u8                      offset;
00143         u8                      fh_len;
00144         u16                     vh_len;
00145         u32                     pl_len;
00147         u32                     private;
00148
00149         u32                     maxlen;
00150         u8                     *buffer;
00153         void                   (*free)(struct
mqtt_packet *mqp);
00154
00155         struct mqtt_packet     *next;
00156 };
00157
00160 #define MQP_FHEADER_BUF(mqp)  (mqp->buffer +
 mqp->offset)
00161 #define MQP_VHEADER_BUF(mqp)  (MQP_FHEADER_B
UF(mqp) + mqp->fh_len)
00162 #define MQP_PAYLOAD_BUF(mqp)  (MQP_VHEADER_B
UF(mqp) + mqp->vh_len)
00163
00164 #define MQP_CONTENT_LEN(mqp)  (mqp->fh_len +
 mqp->vh_len + mqp->pl_len)
00165 #define MQP_FREEBUF_LEN(mqp)  (mqp->maxlen -
 mqp->offset -      \
00166                                MQP_CONTENT_L
EN(mqp))
00167
00168 #define MQP_FHEADER_VAL(mqp)  (mqp->fh_byte1)

00169 #define MQP_FHEADER_MSG(mqp)  (MSG_TYPE(MQP_
FHEADER_VAL(mqp)))
00170 #define MQP_FHEADER_FLG(mqp)  (MSG_FLAGS(MQP
_FHEADER_VAL(mqp)))
```

```
00171
00172 #define DEFINE_MQP_VEC(num_mqp, mqp_vec)
            \
00173         static struct mqtt_packet mqp_vec[nu
m_mqp];
00174
00175 #define DEFINE_MQP_BUF_VEC(num_mqp, mqp_vec,
 buf_len, buf_vec)        \
00176         DEFINE_MQP_VEC(num_mqp, mqp_vec);
                          \
00177         static u8 buf_vec[num_mqp][buf_len];
00178
00179 /*-----------------------------------------
---------------------------
00180  * Heleper MACROS for PUBLISH-RX Message Pro
cessing
00181  *-----------------------------------------
---------------------------
00182  */
00183
00189 #define MQP_PUB_TOP_BUF(mqp) (MQP_VHEADER_BU
F(mqp) + 2)
00190
00192 #define MQP_PUB_TOP_LEN(mqp) (mqp->vh_len -
2 - (mqp->msg_id? 2 : 0))
00193
00195 #define MQP_PUB_PAY_BUF(mqp) (mqp->pl_len? M
QP_PAYLOAD_BUF(mqp) : NULL)
00196
00198 #define MQP_PUB_PAY_LEN(mqp) (mqp->pl_len)
00199
00205 #define WILL_RETAIN_VAL 0x20
00206 #define WILL_CONFIG_VAL 0x04
00207 #define CLEAN_START_VAL 0x02
00208 #define USER_NAME_OPVAL 0x80
00209 #define PASS_WORD_OPVAL 0x40
00210
```

```
00219 #define MQP_ERR_NETWORK   (-1)
00220 #define MQP_ERR_TIMEOUT   (-2)
00221 #define MQP_ERR_NET_OPS   (-3)
00222 #define MQP_ERR_FNPARAM   (-4)
00223 #define MQP_ERR_PKT_AVL   (-5)
00224 #define MQP_ERR_PKT_LEN   (-6)
00225 #define MQP_ERR_NOTCONN   (-7)
00226 #define MQP_ERR_BADCALL   (-8)
00227 #define MQP_ERR_CONTENT   (-9)
00228 #define MQP_ERR_LIBQUIT  (-10)
00231 #define MQP_ERR_NOT_DEF  (-32)
00235 /*-----------------------------------------
---------------------------
00236  * Common Operations
00237  *-----------------------------------------
---------------------------
00238  */
00239
00246 void mqp_free(struct mqtt_packet *mqp);
00247
00257 void mqp_reset(struct mqtt_packet *mqp);
00258
00268 void mqp_init(struct mqtt_packet *mqp, u8 of
fset);
00269
00271 static
00272 inline void mqp_buffer_attach(struct mqtt_pa
cket *mqp, u8 *buffer, u32 length,
00273                               u8 offset)
00274 {
00275         mqp_init(mqp, offset);
00276
00277         mqp->buffer = buffer;
00278         mqp->maxlen = length;
00279         mqp->free   = NULL;
00280
00281         return;
```

```
00282 }
00283
00285 struct utf8_string {
00286
00287         i8   *buffer;
00288         u16   length;
00289 };
00290
00300 i32 mqp_buf_wr_utf8(u8 *buf, const struct ut
f8_string *utf8);
00301
00314 i32 mqp_buf_tail_wr_remlen(u8 *buf, u32 reml
en);
00315
00324 i32 mqp_buf_rd_remlen(u8 *buf, u32 *remlen);
00325
00343 i32
00344 mqp_pub_append_topic(struct mqtt_packet *mqp
, const struct utf8_string *topic,
00345                         u16 msg_id);
00346
00360 i32 mqp_pub_append_data(struct mqtt_packet *
mqp, const u8 *data_buf,
00361                           u32 data_len);
00362
00373 bool mqp_proc_msg_id_ack_rx(struct mqtt_pack
et *mqp_raw, bool has_payload);
00374
00384 bool mqp_proc_pub_rx(struct mqtt_packet *mqp
_raw);
00385
00386 /*
00387    Wait-List of MQTT Messages for which ackn
oledge is pending from remote node.
00388 */
00389 struct mqtt_ack_wlist {
00390
```

```
00391         struct mqtt_packet *head;  /* Points
 to head of single linked-list. */
00392         struct mqtt_packet *tail;  /* Points
 to tail of single linked-list. */
00393 };
00394
00395 static inline bool mqp_ack_wlist_is_empty(st
ruct mqtt_ack_wlist *list)
00396 {
00397         return (NULL == list->head) ? true :
false;
00398 }
00399
00400 /*
00401    Add specified element into trailing end o
f list.
00402
00403    Returns, on success, true, otherwise fals
e.
00404 */
00405 bool mqp_ack_wlist_append(struct mqtt_ack_wl
ist *list,
00406                           struct mqtt_packet
    *elem);
00407
00408 /*
00409    Removes element that has specified msg_id
 from list.
00410
00411    Returns, on success, pointer to removed e
lement, otherwise NULL.
00412 */
00413 struct mqtt_packet *mqp_ack_wlist_remove(str
uct mqtt_ack_wlist *list,
00414                                         u1
6 msg_id);
00415 /*
```

```
00416    Removes and frees all elements in list.
00417 */
00418 void mqp_ack_wlist_purge(struct mqtt_ack_wli
st *list);
00419
00420 static inline bool is_wlist_empty(const stru
ct mqtt_ack_wlist *list)
00421 {
00422         return list->head? false : true;
00423 }
00424
00439 i32 mqp_prep_fh(struct mqtt_packet *mqp, u8
flags);
00440
00442 enum mqtt_qos {
00443
00444         MQTT_QOS0,
00445         MQTT_QOS1,
00446         MQTT_QOS2
00447 };
00448
00450 struct utf8_strqos {
00451
00452         i8              *buffer;
00453         u16              length;
00454         enum mqtt_qos    qosreq;
00455 };
00456
00457
00467 struct secure_conn {
00468
00469         void *method;
00470         void *cipher;
00471         u32   n_file;
00472         char **files;
00473 };
00474
```

```
00484 struct device_net_services {
00485
00489 #define DEV_NETCONN_OPT_TCP  0x01
00490 #define DEV_NETCONN_OPT_UDP  0x02
00491 #define DEV_NETCONN_OPT_IP6  0x04
00492 #define DEV_NETCONN_OPT_URL  0x08
00493 #define DEV_NETCONN_OPT_SEC  0x10
00512         i32 (*open)(u32 nwconn_opts, const i
8 *server_addr, u16 port_number,
00513                     const struct secure_conn
 *nw_security);
00514
00526         i32   (*send)(i32 comm, const u8 *bu
f, u32 len, void *ctx);
00527
00548         i32   (*recv)(i32 comm, u8 *buf, u32
 len, u32 wait_secs,
00549                       bool *err_timeo, void
*ctx);
00550
00565         i32   (*send_dest)(i32 comm, const u
8 *buf, u32 len, u16 dest_port,
00566                            const i8 *dest_ip
, u32 ip_len);
00567
00588         i32   (*recv_from)(i32 comm, u8 *buf
, u32 len, u16 *from_port,
00589                            i8 *from_ip, u32
*ip_len);
00590
00592         i32   (*close)(i32 comm);
00593
00605         i32 (*listen)(u32 nwconn_opts, u16 p
ort_number,
00606                       const struct secure_co
nn *nw_security);
00607
```

```
00623         i32 (*accept)(i32 listen, i8 *client
_ip, u32 *ip_length);
00624
00653         i32   (*io_mon)(i32 *recv_cvec, i32
*send_cvec,
00654                         i32 *rsvd_cvec,  u32
 wait_secs);
00655
00677         u32   (*time)(void);
00678 };
00679  /* device_net_services */
00681
00682 /* Receive data from the specified network a
nd read into the 'mqp' */
00683 i32 mqp_recv(i32  net,     const struct devi
ce_net_services *net_ops,
00684              struct mqtt_packet *mqp, u32 wa
it_secs, bool *timed_out,
00685              void *ctx);
00686
00687 /*-----------------------------------------
-----------------------------------
00688  * Data structure for managing the QoS2 PUB
RX packets and follow-ups
00689  *-----------------------------------------
-----------------------------------*/
00690
00691 #define MAX_PUBREL_INFLT 8 /* Must be kept a
s a value of 2^n */
00692
00693 struct pub_qos2_cq { /* Circular Queue CQ to
 track QOS2 PUB RX messages */
00694
00695         u16 id_vec[MAX_PUBREL_INFLT];  /* Ve
ctor to store RX Message-IDs */
00696         u8  n_free;                    /* Nu
m of free elements in vector */
```

```
00697         u8  rd_idx;                     /* In
dex to Read  next Message-ID */
00698         u8  wr_idx;                     /* In
dex to Write next Message-ID */
00699 };
00700
00701 /* Reset the specified Circular Queue (CQ) */

00702 void qos2_pub_cq_reset(struct pub_qos2_cq *c
q);
00703
00704 /* Append the message-id into the CQ tail. R
eturn true on success, else false */
00705 bool qos2_pub_cq_logup(struct pub_qos2_cq *c
q, u16 msg_id);
00706
00707 /* Remove the message-id from the CQ head. R
eturn true on success, else false */
00708 bool qos2_pub_cq_unlog(struct pub_qos2_cq *c
q, u16 msg_id);
00709
00710 /* Is the message-id available in the CQ ? R
eturn true on success, else false */
00711 bool qos2_pub_cq_check(struct pub_qos2_cq *c
q, u16 msg_id);
00712
00713 /* Get the count of message-ID(s) availalbe
in the CQ */
00714 static inline i32 qos2_pub_cq_count(struct p
ub_qos2_cq *cq)
00715 {
00716         return MAX_PUBREL_INFLT - cq->n_free
;
00717 }
00718
00719 struct client_ctx {
00720
```

```
00721         void         *usr;  /* Client Usr */
00722         i32           net;  /* Socket HND */
00723
00724         i8            remote_ip[16];
00725         u32           ip_length;
00726
00727         u32           timeout;
00728         u16           ka_secs;
00729
00730         u32           flags;
00731
00732         struct client_ctx *next;
00733 };
00734
00735 void cl_ctx_reset(struct client_ctx *cl_ctx)
;
00736 void cl_ctx_timeout_insert(struct client_ctx
 **head,
00737                            struct client_ctx
 *elem);
00738
00739 void cl_ctx_remove(struct client_ctx **head,
00740                    struct client_ctx *elem);
00741
00742 #define KA_TIMEOUT_NONE 0xffffffff  /* Diffe
rent than KA SECS = 0 */
00743 void cl_ctx_timeout_update(struct client_ctx
 *cl_ctx, u32 now_secs);
00744
00745 #endif
```

device_net_services Member List

This is the complete list of members for **device_net_services**, including all inherited members.

accept	**device_net_services**
close	**device_net_services**
io_mon	**device_net_services**
listen	**device_net_services**
open	**device_net_services**
recv	**device_net_services**
recv_from	**device_net_services**
send	**device_net_services**
send_dest	**device_net_services**
time	**device_net_services**

mqtt_ack_wlist Member List

This is the complete list of members for **mqtt_ack_wlist**, including all inherited members.

head (defined in **mqtt_ack_wlist**)	**mqtt_ack_wlist**
tail (defined in **mqtt_ack_wlist**)	**mqtt_ack_wlist**

mqtt_packet Member List

This is the complete list of members for **mqtt_packet**, including all inherited members.

buffer	**mqtt_packet**
fh_byte1	**mqtt_packet**
fh_len	**mqtt_packet**
free	**mqtt_packet**
maxlen	**mqtt_packet**
msg_id	**mqtt_packet**
msg_type	**mqtt_packet**
n_refs	**mqtt_packet**
next (defined in **mqtt_packet**)	**mqtt_packet**
offset	**mqtt_packet**
pad (defined in **mqtt_packet**)	**mqtt_packet**
pl_len	**mqtt_packet**
private (defined in **mqtt_packet**)	**mqtt_packet**
vh_len	**mqtt_packet**

mqtt_server_app_cbs Member List

This is the complete list of members for **mqtt_server_app_cbs**, including all inherited members.

connect	**mqtt_server_app_cbs**
disconn	**mqtt_server_app_cbs**
publish	**mqtt_server_app_cbs**

D:/my_data/GIT/network_apps/netapps/mqtt/ser

Go to the documentation of this file.

```
00001 /*****************************************
*********************************
00002 *
00003 *   Copyright (C) 2014 Texas Instruments Inc
orporated
00004 *
00005 *   All rights reserved. Property of Texas I
nstruments Incorporated.
00006 *   Restricted rights to use, duplicate or d
isclose this code are
00007 *   granted through contract.
00008 *
00009 *   The program may not be used without the
written permission of
00010 *   Texas Instruments Incorporated or agains
t the terms and conditions
00011 *   stipulated in the agreement under which
this program has been supplied,
00012 *   and under no circumstances can it be use
d with non-TI connectivity device.
00013 *
00014 *****************************************
*********************************/
00015
00016
```

```
00017 #ifndef __SERVER_CORE_H__
00018 #define __SERVER_CORE_H__
00019
00020 #include "server_pkts.h"
00021
00072 struct mqtt_server_app_cbs {
00073
00102         u16  (*connect)(const struct utf8_st
ring *client_id,
00103                         const struct utf8_st
ring *user_name,
00104                         const struct utf8_st
ring *pass_word,
00105                         void **app_usr);
00106
00122         void (*publish)(const struct utf8_st
ring *topic,
00123                         const u8 *payload, u
32 pay_len,
00124                         bool dup, u8 qos, bo
ol retain);
00125
00138         void (*disconn)(const void *app_usr,
bool due2err);
00139 };
00140
00164 i32 mqtt_server_topic_enroll(const void *app
_hnd,
00165                              const struct ut
f8_string *topic, enum mqtt_qos qos);
00166
00180 i32 mqtt_server_topic_disenroll(const void *
app_hnd,
00181                                 const struct
utf8_string *topic);
00182
00198 i32 mqtt_server_app_pub_send(const  struct u
```

```
tf8_string *topic,
00199                                   const u8 *data_
buf, u32 data_len,
00200                                   enum mqtt_qos q
os, bool retain);
00201
00219 void *mqtt_server_app_register(const struct
mqtt_server_app_cbs *cbs,
00220                                     const i8 *nam
e);
00221
00222
00227 struct mqtt_server_app_cfg {
00228
00229         void        *place_holder;
00230 };
00231
00254 i32 mqtt_server_init(const struct mqtt_serve
r_lib_cfg *lib_cfg,
00255                          const struct mqtt_serve
r_app_cfg *app_cfg);
00256  /* End of server_daemon */
00258
00259 #endif
00260
```

mqtt_server_app_cfg Member List

This is the complete list of members for **mqtt_server_app_cfg**, including all inherited members.

place_holder **mqtt_server_app_cfg**

mqtt_server_lib_cfg Member List

This is the complete list of members for **mqtt_server_lib_cfg**, including all inherited members.

aux_debug_en	**mqtt_server_lib_cfg**
debug_printf	**mqtt_server_lib_cfg**
listener_port	**mqtt_server_lib_cfg**
loopback_port	**mqtt_server_lib_cfg**
mutex	**mqtt_server_lib_cfg**
mutex_lockin (defined in **mqtt_server_lib_cfg**)	**mqtt_server_lib_cfg**
mutex_unlock (defined in **mqtt_server_lib_cfg**)	**mqtt_server_lib_cfg**

D:/my_data/GIT/network_apps/netapps/mqtt/ser

Go to the documentation of this file.

```
00001 /*****************************************
*********************************
00002 *
00003 *   Copyright (C) 2014 Texas Instruments Inc
orporated
00004 *
00005 *   All rights reserved. Property of Texas I
nstruments Incorporated.
00006 *   Restricted rights to use, duplicate or d
isclose this code are
00007 *   granted through contract.
00008 *
00009 *   The program may not be used without the
written permission of
00010 *   Texas Instruments Incorporated or agains
t the terms and conditions
00011 *   stipulated in the agreement under which
this program has been supplied,
00012 *   and under no circumstances can it be use
d with non-TI connectivity device.
00013 *
00014 *****************************************
*********************************/
00015
00016 #ifndef __SERVER_PKTS_H__
```

```
00017 #define __SERVER_PKTS_H__
00018
00019 #include "mqtt_common.h"
00020
00021 /*-----------------------------------------
-----------------------------------
00022  * Note: Do not create additional dependency
 of this file on any header other
00023  * than mqtt_common.h. Specifically, server_
pkts.[hc] in conjunction with the
00024  * mqtt_common.[hc] files must be facilitate
d to create a stand-alone library.
00025  *-----------------------------------------
-----------------------------------
00026  */
00027
00076 #define MQ_CONN_UTF8_BUF(utf8)      ((utf8)?
 (utf8)->buffer : NULL)
00077
00079 #define MQ_CONN_UTF8_LEN(utf8)      ((utf8)?
 (utf8)->length : 0)
00080
00081 #define MQC_UTF8_CLIENTID(utf8_vec) (utf8_ve
c[0])
00082 #define MQC_UTF8_WILL_TOP(utf8_vec) (utf8_ve
c[1])
00083 #define MQC_UTF8_WILL_MSG(utf8_vec) (utf8_ve
c[2])
00084 #define MQC_UTF8_USERNAME(utf8_vec) (utf8_ve
c[3])
00085 #define MQC_UTF8_PASSWORD(utf8_vec) (utf8_ve
c[4])
00087 #define MQC_CLIENTID_BUF(utf8_vec)  MQ_CONN_
UTF8_BUF(MQC_UTF8_CLIENTID(utf8_vec))
00088 #define MQC_CLIENTID_LEN(utf8_vec)  MQ_CONN_
UTF8_LEN(MQC_UTF8_CLIENTID(utf8_vec))
00089
```

```
00090 #define MQC_WILL_TOP_BUF(utf8_vec)  MQ_CONN_
UTF8_BUF(MQC_UTF8_WILL_TOP(utf8_vec))
00091 #define MQC_WILL_TOP_LEN(utf8_vec)  MQ_CONN_
UTF8_LEN(MQC_UTF8_WILL_TOP(utf8_vec))
00092
00093 #define MQC_WILL_MSG_BUF(utf8_vec)  MQ_CONN_
UTF8_BUF(MQC_UTF8_WILL_MSG(utf8_vec))
00094 #define MQC_WILL_MSG_LEN(utf8_vec)  MQ_CONN_
UTF8_LEN(MQC_UTF8_WILL_MSG(utf8_vec))
00095
00096 #define MQC_USERNAME_BUF(utf8_vec)  MQ_CONN_
UTF8_BUF(MQC_UTF8_USERNAME(utf8_vec))
00097 #define MQC_USERNAME_LEN(utf8_vec)  MQ_CONN_
UTF8_LEN(MQC_UTF8_USERNAME(utf8_vec))
00098
00099 #define MQC_PASSWORD_BUF(utf8_vec)  MQ_CONN_
UTF8_BUF(MQC_UTF8_PASSWORD(utf8_vec))
00100 #define MQC_PASSWORD_LEN(utf8_vec)  MQ_CONN_
UTF8_LEN(MQC_UTF8_PASSWORD(utf8_vec))
00101
00104 #ifndef CFG_SR_MAX_MQP_RX_LEN
00105 #define MQP_SERVER_RX_LEN  1024
00106 #else
00107 #define MQP_SERVER_RX_LEN  CFG_SR_MAX_MQP_RX
_LEN
00108 #endif
00109
00122 i32 mqtt_vh_msg_send(void *ctx_cl, u8 msg_ty
pe, enum mqtt_qos qos,
00123                      bool has_vh, u16 vh_dat
a);
00124
00134 i32 mqtt_vh_msg_send_locked(void *ctx_cl, u8
 msg_type, enum mqtt_qos qos,
00135                                bool has_vh, u16
 vh_data);
00136
```

```
00160 i32 mqtt_server_pub_dispatch(void *ctx_cl, s
truct mqtt_packet *mqp, bool dup);
00161
00171 i32 mqtt_server_pub_dispatch_locked(void *ct
x_cl, struct mqtt_packet *mqp,
00172                                     bool dup
);
00173
00190 i32 mqtt_server_run(u32 wait_secs);
00191
00201 i32 mqtt_server_register_net_svc(const struct
 device_net_services *net);
00202
00203
00214 struct mqtt_server_msg_cbs {
00215
00249         u16  (*connect_rx)(void *ctx_cl, u8
conn_flags,
00250                            struct utf8_string
 * const *utf8_vec, void **usr);
00251
00277         bool (*sub_msg_rx)(void *usr_cl, con
st struct utf8_strqos *qos_topics,
00278                            u32 n_topics, u16
msg_id, u8 *acks);
00279
00296         bool (*un_sub_msg)(void *usr_cl, con
st struct utf8_string *topics,
00297                            u32 n_topics, u16
 msg_id);
00298
00324         bool (*pub_msg_rx)(void *usr_cl, con
st struct utf8_string *topic,
00325                            const u8 *data_bu
f, u32 data_len, u16 msg_id,
00326                            bool dup, enum mq
tt_qos qos, bool retain);
```

```
00327
00343         bool (*ack_notify)(void *usr_cl, u8
msg_type, u16 msg_id);
00344
00359         void (*on_cl_net_close)(void *usr_cl
, bool due2err);
00360
00371         void (*on_connack_send)(void *usr_cl
, bool clean_session);
00372 };
00373
00374 struct mqtt_server_lib_cfg {
00375
00377         u16   listener_port;
00378
00385         u16   loopback_port;
00386
00388         void  *mutex;
00389         void (*mutex_lockin)(void *mutex);
00390         void (*mutex_unlock)(void *mutex);
00391
00392         i32  (*debug_printf)(const i8 *forma
t, ...);
00393         bool   aux_debug_en;
00395 };
00396
00420 i32 mqtt_server_lib_init(const struct mqtt_s
erver_lib_cfg *cfg,
00421                          const struct mqtt_s
erver_msg_cbs *cbs);
00422  /* End of server_pktlib */
00424
00425 #endif
```

mqtt_server_msg_cbs Member List

This is the complete list of members for **mqtt_server_msg_cbs**, including all inherited members.

ack_notify	**mqtt_server_msg_cbs**
connect_rx	**mqtt_server_msg_cbs**
on_cl_net_close	**mqtt_server_msg_cbs**
on_connack_send	**mqtt_server_msg_cbs**
pub_msg_rx	**mqtt_server_msg_cbs**
sub_msg_rx	**mqtt_server_msg_cbs**
un_sub_msg	**mqtt_server_msg_cbs**

pub_qos2_cq Member List

This is the complete list of members for **pub_qos2_cq**, including all inherited members.

id_vec (defined in **pub_qos2_cq**)	**pub_qos2_cq**
n_free (defined in **pub_qos2_cq**)	**pub_qos2_cq**
rd_idx (defined in **pub_qos2_cq**)	**pub_qos2_cq**
wr_idx (defined in **pub_qos2_cq**)	**pub_qos2_cq**

secure_conn Member List

This is the complete list of members for **secure_conn**, including all inherited members.

cipher	**secure_conn**
files	**secure_conn**
method	**secure_conn**
n_file	**secure_conn**

utf8_string Member List

This is the complete list of members for **utf8_string**, including all inherited members.

buffer	**utf8_string**
length	**utf8_string**

utf8_strqos Member List

This is the complete list of members for **utf8_strqos**, including all inherited members.

buffer	**utf8_strqos**
length	**utf8_strqos**
qosreq	**utf8_strqos**

- a -

- accept : **device_net_services**
- ack_notify : **mqtt_server_msg_cbs**
- aux_debug_en : **mqtt_server_lib_cfg**

- b -

- buffer : **mqtt_packet** , **utf8_strqos** , **utf8_string**

- c -

- cipher : **secure_conn**
- close : **device_net_services**
- connect : **mqtt_server_app_cbs**
- connect_rx : **mqtt_server_msg_cbs**

- d -

- debug_printf : **mqtt_server_lib_cfg**
- disconn : **mqtt_server_app_cbs**

- f -

- fh_byte1 : **mqtt_packet**
- fh_len : **mqtt_packet**
- files : **secure_conn**

- free : **mqtt_packet**

- i -

- io_mon : **device_net_services**

- l -

- length : **utf8_string** , **utf8_strqos**
- listen : **device_net_services**
- listener_port : **mqtt_server_lib_cfg**
- loopback_port : **mqtt_server_lib_cfg**

- m -

- maxlen : **mqtt_packet**
- method : **secure_conn**
- msg_id : **mqtt_packet**
- msg_type : **mqtt_packet**
- mutex : **mqtt_server_lib_cfg**

- n -

- n_file : **secure_conn**
- n_refs : **mqtt_packet**

- o -

- offset : **mqtt_packet**
- on_cl_net_close : **mqtt_server_msg_cbs**
- on_connack_send : **mqtt_server_msg_cbs**
- open : **device_net_services**

- p -

- pl_len : **mqtt_packet**
- place_holder : **mqtt_server_app_cfg**
- pub_msg_rx : **mqtt_server_msg_cbs**
- publish : **mqtt_server_app_cbs**

- q -

- qosreq : **utf8_strqos**

- r -

- recv : **device_net_services**
- recv_from : **device_net_services**

- s -

- send : **device_net_services**
- send_dest : **device_net_services**
- sub_msg_rx : **mqtt_server_msg_cbs**

- t -

- time : **device_net_services**

- u -

- un_sub_msg : **mqtt_server_msg_cbs**

- v -

- vh_len : **mqtt_packet**

MainPage.h

```
00001
```

- mqp_buf_rd_remlen() : **mqtt_common.h**
- mqp_buf_tail_wr_remlen() : **mqtt_common.h**
- mqp_buf_wr_utf8() : **mqtt_common.h**
- mqp_free() : **mqtt_common.h**
- mqp_init() : **mqtt_common.h**
- mqp_prep_fh() : **mqtt_common.h**
- mqp_proc_msg_id_ack_rx() : **mqtt_common.h**
- mqp_proc_pub_rx() : **mqtt_common.h**
- mqp_pub_append_data() : **mqtt_common.h**
- mqp_pub_append_topic() : **mqtt_common.h**
- mqp_reset() : **mqtt_common.h**
- mqtt_server_app_pub_send() : **server_core.h**
- mqtt_server_app_register() : **server_core.h**
- mqtt_server_init() : **server_core.h**
- mqtt_server_lib_init() : **server_pkts.h**
- mqtt_server_pub_dispatch() : **server_pkts.h**
- mqtt_server_pub_dispatch_locked() : **server_pkts.h**
- mqtt_server_register_net_svc() : **server_pkts.h**
- mqtt_server_run() : **server_pkts.h**
- mqtt_server_topic_disenroll() : **server_core.h**
- mqtt_server_topic_enroll() : **server_core.h**
- mqtt_vh_msg_send() : **server_pkts.h**
- mqtt_vh_msg_send_locked() : **server_pkts.h**

- mqtt_qos : **mqtt_common.h**

- MQTT_QOS0 : **mqtt_common.h**
- MQTT_QOS1 : **mqtt_common.h**
- MQTT_QOS2 : **mqtt_common.h**

- d -

- DEV_NETCONN_OPT_IP6 : **mqtt_common.h**
- DEV_NETCONN_OPT_SEC : **mqtt_common.h**
- DEV_NETCONN_OPT_TCP : **mqtt_common.h**
- DEV_NETCONN_OPT_UDP : **mqtt_common.h**
- DEV_NETCONN_OPT_URL : **mqtt_common.h**

- m -

- MAX_FH_LEN : **mqtt_common.h**
- MAX_REMLEN_BYTES : **mqtt_common.h**
- MQ_CONN_UTF8_BUF : **server_pkts.h**
- MQ_CONN_UTF8_LEN : **server_pkts.h**
- MQC_UTF8_CLIENTID : **server_pkts.h**
- MQC_UTF8_PASSWORD : **server_pkts.h**
- MQC_UTF8_USERNAME : **server_pkts.h**
- MQC_UTF8_WILL_MSG : **server_pkts.h**
- MQC_UTF8_WILL_TOP : **server_pkts.h**
- MQP_ERR_BADCALL : **mqtt_common.h**
- MQP_ERR_CONTENT : **mqtt_common.h**
- MQP_ERR_FNPARAM : **mqtt_common.h**
- MQP_ERR_LIBQUIT : **mqtt_common.h**
- MQP_ERR_NET_OPS : **mqtt_common.h**
- MQP_ERR_NETWORK : **mqtt_common.h**
- MQP_ERR_NOT_DEF : **mqtt_common.h**

- MQP_ERR_NOTCONN : **mqtt_common.h**
- MQP_ERR_PKT_AVL : **mqtt_common.h**
- MQP_ERR_PKT_LEN : **mqtt_common.h**
- MQP_ERR_TIMEOUT : **mqtt_common.h**
- MQP_PUB_PAY_BUF : **mqtt_common.h**
- MQP_PUB_PAY_LEN : **mqtt_common.h**
- MQP_PUB_TOP_BUF : **mqtt_common.h**
- MQP_PUB_TOP_LEN : **mqtt_common.h**
- MQP_SERVER_RX_LEN : **server_pkts.h**
- MQTT_COMMON_VERSTR : **mqtt_common.h**
- MQTT_CONNECT : **mqtt_common.h**

- q -

- QFL_VALUE : **mqtt_common.h**
