Here is a list of all related documentation pages:

- MQTT Client Library - App Notes
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MQTT Client Library - App Notes**
Overview

This narrative outlines the usage / working aspects and operational modes that are available to the users to make use of the services of the MQTT Client Library to build applications that are just apt for the targeted use-case.
QoS Messages

In MQTT, a QoS0 message does not warrant the remote entity to acknowledge the receipt of the message and the protocol does not guarantee to the sender the delivery of the message to the recipient. In other words, the message from the sender may or may not get delivered to the recipient and therefore, the QoS0 messages get associated with the "at most once" delivery concept. However, with QoS1 (at least once, delivery) and QoS2 (exactly once), the messages need to be acknowledged (in a differing manner). This section enumerates the client LIB specific implementation aspects of QoS1 and QoS2 messages.

The non QoS0 messages in MQTT protocol need some additional processing. Specifically,

- Ack QoS1 and QoS2 packets (received from the server)
  The client LIB 'context' relieves the application from sending the ACK messages to the server by implementing the required support to send the corresponding ACK as and when required.

- Track QoS1 and QoS2 message sequences
  The client LIB 'context' manages the non-QoS0 packets to track and ascertain the sequence of the message ID(s) from the server. In addition, the 'context' resends the unacknowledged QoS1 / QoS2 packets to the MQTT server at the time of next MQTT connection. The packets are resent to the MQTT server, only if, the on-going session (connection) has not been configured for the 'clean session', the ACK(s) for these packets have not been received and the next iteration of the MQTT session (connection) also does not assert the 'clean session' parameter.

In case, the on-going session for a 'context' has been configured with for a 'clean session', the references to un-acknowledged packets are removed from the 'context' at the time of termination of the network connection to the server. In other case, where the on-going session in the 'context' has not been configured for a 'clean session', however, in the next iteration of the MQTT connection,
the 'clean session' parameter gets asserted, the client LIB 'context' will drop all the references to the un-acknowledged packets at the time of the establishment of the second MQTT connection.

When required to resend the unacknowledged packets to the server, the client LIB 'context', when configured to operate in the MQTT 3.1.1 mode, resends only the PUBLISH packets to the server. Whereas, in the MQTT 3.1 mode, the client LIB 'context', additionally, resends to the server, any unacknowledged SUBSCRIBE and UNSUBSCRIBE messages.
Context Overview

The MQTT Client LIB can support simultaneous MQTT connections to the servers. And the operational configuration and other related aspects of such a connection, in this software design, is referred as a 'context'. Therefore, the implied and intuitive sequence for the usage of MQTT Client library evolves to the following:

- Initialize LIB (singleton operation)
- Create one or more 'context(s)'
- Bidirectional transfer of MQTT packets
- Delete 'contexts'
- Exit LIB (singleton operation)

The Client LIB must be initialized only once in the system and the 'contexts' can be created and / or deleted by one or more applications (tasks). In a system, if more than one application is going utilize the services of the Client LIB, then the user must provision the platform specific support for Mutex.

Given, the richness of the features offered by the Client LIB, the services available to manage the contexts and transfer of MQTT packets need some elaboration. Specifically, the following aspects of the 'context' makes the Client LIB flexible and scalable across the deployments.

- 'Context' Management: Either in a Group or as an individual
- 'Context' Operation: Either callback (CB) or sync-wait (WT)

The afore mentioned parameters can be configured for a 'context' at the time of its creation but the restrictions outlined below must be adhered.

- There can be only one group of 'contexts' in the system. However, there can be as many as possible, the individual 'contexts' in the system along-with or without a single group-text.
- All the 'contexts' in the group will use the same mode of operation i.e. callback mode or otherwise. In other words, it is not possible to operate the constituent 'contexts' of the group in different modes. And the table below further enumerates the permissible
combination of 'contexts' in the system.

**Permissible 'Context' combination**

The concept of 'group of contexts' is aimed at enabling a single application to manage multiple 'contexts' or connections to the same or different servers. Conversely, a single 'individual context' can be managed by an independent application. The Client LIB, for managing the 'group contexts' allocates, additional resources - primarily, the Client LIB is required to incorporate the 'loopback' (UDP) port in the system and the user should carefully choose and configure, without creating any conflict, the 'loopback' port in the system.

It is anticipated that an application which utilizes the services of the Client LIB can be a multi-task composition and the Client LIB will be required to maintain integrity of the operation and resources. In such a scenario, the application must indicate to the Client LIB that a dedicated RX task is being used to handle the packets from the network for a given 'context'. Such a configuration for the 'context' will enable the Client LIB to enforce steps to maintain integrity of the network resources.
Receive Operation

While the services for transmission of data to the network remains same across the various modes of 'Context', the receive operation has several flavors and are enumerated in the table below. Depending on the needs of the use-case, the application can choose the appropriate options.

RX 'Context' operation - options
Buffer Pool

The client 'context' allocates buffer internally to transact the network packets. Specifically, the 'context' needs to allocate a buffer for sending a message to the network. Similarly, the 'context', whilst operating in the call-back mode, must provision a buffer to receive a message from the network and hand it to the application. In the MQTT client LIB, the allocation of a buffer is made from a pre-configured pool and the user of the client LIB, as part of the initialization sequence, must provision adequate resources in the pool.

In MQTT, the QoS1 & QoS2 messages / packets need to be acknowledged by the remote entity. For transmission, the number of in-flight messages relates to the maximum number of packets that can remain un-acknowledged at any given point of time. For example, the value of single in-flight message would imply that there can be only one un-acknowledged message in the client 'context' and the subsequent packet can be scheduled for transmission, only if, the previous non-QoS0 packet has been acknowledged. After sending the packet to the network and if required, getting the ACK from the peer, the client LIB returns the packet to the buffer pool.

On the receiving side, if required, the application can take over the ownership of a packet buffer delivered by the client LIB 'context' for further processing. Such an arrangement promotes 'zero copy' philosophy in a low power system. The mechanism of 'zero-copy' is suitable for scenarios, in which, the application is required to store or queue the packet for a differed or later processing. After the completion of the processing, the application has to 'free' the packet that it had taken from the client LIB 'context'. When freed, the packet is returned to the buffer pool of the client LIB.

The number of in-flight TX messages in the network and the number of RX packets including the ones that can be handed over to the application are dependent on the resources that have been made available in the buffer pool. In other words, the user of the client LIB must provision an adequate number of packet-buffers to commensurate with the intended run time configuration of transmit and receive buffers.
across all the 'contexts'. The routine / interface `mqtt_client_buffers_register` can be used to set-up the buffer pool.

Formula 1: **'Context' buffer pool = 'Context' in-flight TX buffers + 'Context' RX buffers (callback mode only)**

Formula 2: **Client 'LIB' buffer pool = Sum of all 'Contexts' buffer pool**

**Note:**
- The RX buffers include the buffers that can be potentially handed over to the application. It is strongly recommended that the application ensures that the client 'context' always has ownership of one RX buffer at any time to support the incoming message from the network.

Formula 3: **'Context' buffer pool (with clean session as false) = 'Context' buffer pool (with clean session as true) + 1 (for CONNECT message)**

The additional buffer for the 'clean session' = false configuration is required to support the dispatch of the CONNECT message to the network to establish the MQTT connection with the broker. The need of the additional buffer arises for the cases, in which, all the previous non-QOS0 message(s) (PUB / SUB / UNSUB) has / have not been acknowledged and the user application is holding all the RX buffers.

For example, for a minimalistic configuration of one in-flight message and one RX buffer with no handover of the packet to the user application, the size of the buffer pool for the configuration having 'clean session' as true will be 2. Where as, the size of buffer pool for the configuration 'clean session' as false will be \((2 + 1) = 3\).

**Note:**
- The client LIB does not support segregation of the buffers for sending (in-flight) and receiving. This responsibility is left to the application that uses the client 'context'.
Summary:

- The buffer pool must be provisioned with adequate resources to handle both the TX and RX operations.
- The size of the buffer pool must account for all the 'contexts' that will be used or has been configured.
- The number of buffers for a 'context' is established by adding the count of intended 'in-flight' messages to the network and when operating the 'context' in the callback mode, the number of overall RX packets including the ones that can be handed over to the application.
- The 'context' with 'clean session' = false needs an additional buffer to ensure that it is able to send the CONNECT message.
- The 'context' that is operating in the sync-wait mode does not need to include the RX packets in the configuration of the buffer pool.
- All the buffers in the pool has same size, therefore, the pool must be created for buffer that has adequate length.
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 Client Library API(s)
- Options for application to config CTX
- Options for App to configure network
<table>
<thead>
<tr>
<th>Files</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT Packet (MQP) Buffer structure</td>
<td></td>
</tr>
</tbody>
</table>
Classes

struct mqtt_packet
Detailed Description

The core construct to encapsulate, construct and process a message
**Defines**

<table>
<thead>
<tr>
<th>#define</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MQP_PUB_TOP_BUF</strong></td>
<td>(MQP_VHEADER_BUF(mqp) + 2)</td>
</tr>
<tr>
<td><strong>MQP_PUB_TOP_LEN</strong></td>
<td>(mqp-&gt;vh_len - 2 - (mqp-&gt;msg_id? 2 : 0))</td>
</tr>
<tr>
<td><strong>MQP_PUB_PAY_BUF</strong></td>
<td>(mqp-&gt;pl_len? MQP_PAYLOAD_BUF(mqp) : NULL)</td>
</tr>
<tr>
<td><strong>MQP_PUB_PAY_LEN</strong></td>
<td>(mqp-&gt;pl_len)</td>
</tr>
</tbody>
</table>
Define Documentation

```c
#define MQP_PUB_PAY_BUF (mqp) (mqp->pl_len? MQP_PAYLOAD_BUF(mqp) : NULL)
```

Yields pointer to payload data

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

Length or size of payload data

```c
#define MQP_PUB_TOP_BUF (mqp) (MQP_VHEADER_BUF(mqp) + 2)
```

Yields pointer to topic content

```c
#define MQP_PUB_TOP_LEN (mqp) (mqp->vh_len - 2 - (mqp->msg_id? 2 : 0))
```

Length or size of topic content
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBRARY Generated Error Codes</td>
<td></td>
<td>Defines</td>
<td></td>
</tr>
</tbody>
</table>
## Defines

```c
#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

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define MQP_ERR_BADCALL (-8)</code></td>
<td>Irrelevant call for LIB state</td>
</tr>
<tr>
<td><code>#define MQP_ERR_CONTENT (-9)</code></td>
<td>MSG / Data content has errors</td>
</tr>
<tr>
<td><code>#define MQP_ERR_FNPARAM (-4)</code></td>
<td>Invalid parameter(s) provided</td>
</tr>
<tr>
<td><code>#define MQP_ERR_LIBQUIT (-10)</code></td>
<td>Needs reboot library has quit</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NET_OPS (-3)</code></td>
<td>Platform Net Ops un-available</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NETWORK (-1)</code></td>
<td>Problem in network (sock err)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NOT_DEF (-32)</code></td>
<td>Value other than defined ones</td>
</tr>
</tbody>
</table>
#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
<table>
<thead>
<tr>
<th>Files</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Information to establish a secure connection.</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>secure_conn</th>
</tr>
</thead>
</table>
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.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abstraction of Network Services on a platform</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Classes

```c
struct device_net_services
```
Detailed Description

Services to enable the MQTT Client-Server communication over network

These services are invoked by the MQTT Library.
<table>
<thead>
<tr>
<th>Files</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options for platform to configure network</td>
<td></td>
</tr>
</tbody>
</table>
#Defines

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DEV_NETCONN_OPT_TCP</code></td>
<td>0x01</td>
</tr>
<tr>
<td><code>DEV_NETCONN_OPT_UDP</code></td>
<td>0x02</td>
</tr>
<tr>
<td><code>DEV_NETCONN_OPT_IP6</code></td>
<td>0x04</td>
</tr>
<tr>
<td><code>DEV_NETCONN_OPT_URL</code></td>
<td>0x08</td>
</tr>
<tr>
<td><code>DEV_NETCONN_OPT_SEC</code></td>
<td>0x10</td>
</tr>
</tbody>
</table>
## 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
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Related Pages</th>
<th>Modules</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### The Client Library API(s)

Classes | Defines | Functions
# Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>mqtt_client_ctx_cbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>struct</td>
<td>mqtt_client_lib_cfg</td>
</tr>
</tbody>
</table>
#define MQTT_CLIENT_VERSTR "1.0.0"
#define VHB_CONNACK_RC(vh_buf)  (vh_buf[1])
#define MQP_CONNACK_RC(mqp)  (mqp->buffer[3])
#define VHB_CONNACK_SP(vh_buf)  (vh_buf[0] & 0x1)
#define MQP_CONNACK_SP(mqp)  (mqp->buffer[2] & 0x1)
#define VHB_CONNACK_VH16(vh_buf)  ((vh_buf[0] << 8) | vh_buf[1])
#define MQP_CONNACK_VH16(mqp)  ((mqp->buffer[2] << 8) | mqp->buffer[3])
### Functions

<table>
<thead>
<tr>
<th>Argument Types</th>
<th>Function Name</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td><code>mqtt_client_new_msg_id</code></td>
<td>(void)</td>
</tr>
<tr>
<td>bool</td>
<td><code>mqtt_client_is_connected</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_connect_msg_send</code></td>
<td>(void *ctx, bool clean_session, u16 ka_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_pub_msg_send</code></td>
<td>(void *ctx, const struct utf8_string *topic, const u8 *data_buf, u32 data_len, enum mqtt_qos qos, bool retain)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_pub_dispatch</code></td>
<td>(void *ctx, struct mqtt_packet *mqp, enum mqtt_qos qos, bool retain)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_sub_msg_send</code></td>
<td>(void *ctx, const struct utf8_strqos *qos_topics, u32 count)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_sub_dispatch</code></td>
<td>(void *ctx, struct mqtt_packet *mqp)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_unsub_msg_send</code></td>
<td>(void *ctx, const struct utf8_string *topics, u32 count)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_unsub_dispatch</code></td>
<td>(void *ctx, struct mqtt_packet *mqp)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_pingreq_send</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_disconn_send</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_send_progress</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_await_msg</code></td>
<td>(void *ctx, u8 msg_type, struct mqtt_packet *mqp, u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_run</code></td>
<td>(void *ctx, u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_run</code></td>
<td>(void *ctx, u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_await_msg</code></td>
<td>(struct mqtt_packet *mqp, u32 wait_secs, void **app)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_run</code></td>
<td>(u32 wait_secs)</td>
</tr>
<tr>
<td>struct mqtt_packet *</td>
<td><code>mqp_client_alloc</code></td>
<td>(u8 msg_type, u8 offset)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_buffers_register</code></td>
<td>(u32 num_mqp, struct mqtt_packet *mqp_vec, u32 buf_len, u8 *buf_vec)</td>
</tr>
<tr>
<td></td>
<td><code>mqtt_client_ctx_info_register</code></td>
<td>(void *ctx, const struct utf8_string *client_id, const struct</td>
</tr>
<tr>
<td>i32</td>
<td>utf8_string *user_name, const struct utf8_string *pass_word</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_ctx_will_register (void *ctx, const struct utf8_string *will_top, const struct utf8_string *will_msg, enum mqtt_qos will_qos, bool retain)</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_net_svc_register (const struct device_net_services *net)</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_ctx_create (const struct mqtt_client_ctx_cfg *ctx_cfg, const struct mqtt_client_ctx_cbs *ctx_cbs, void *app, void **ctx)</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_ctx_delete (void *ctx)</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_lib_init (const struct mqtt_client_lib_cfg *cfg)</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td>mqtt_client_lib_exit (void)</td>
<td></td>
</tr>
</tbody>
</table>
Define Documentation

```c
#define MQPCONNACK_RC (mqp) (mqp->buffer[3])
```

CONNACK MQP:: Return Code

```c
#define MQP_CONNACK_SP (mqp) (mqp->buffer[2] & 0x1)
```

CONNACK MQP:: \ Session Bit

```c
#define MQTT_CLIENT_VERSTR "1.0.0"
```

Version of Client LIB

```c
#define VHB_CONNACK_RC (vh_buf) (vh_buf[1])
```

Helper functions & macros to derive 16 bit CONNACK Return Code from broker. CONNACK VH:: Return Code

```c
#define VHB_CONNACK_SP (vh_buf) (vh_buf[0] & 0x1)
```

CONNACK VH:: Session Bit
Function Documentation

```c
struct mqtt_packet* mqp_client_alloc ( u8 msg_type, u8 offset ) [read]
```

Allocates a free MQTT Packet Buffer. The pool that will be used by the library to allocate a free MQP buffer must be configured (i.e. registered) a-priori by the app.

The parameter 'offset' is used to specify the number of bytes that are reserved for the header in the buffer

**Parameters:**
- `[in]` **msg_type** Message Type for which MQP buffer is being assigned.
- `[in]` **offset** Number of bytes to be reserved for MQTT headers.

**Returns:**
- A NULL on error, otherwise a reference to a valid packet holder.

**See also:**
- `mqtt_client_register_buffers`

```c
i32 mqtt_clientAwait_msg ( struct mqtt_packet * mqp, u32 wait_secs, void ** app )
```

Block to receive any message for the grouped contexts within specified time. This service is valid only for the set-up, where the application has not configured the grouped contexts in the callback mode. The caller must provide a packet buffer of adequate size to hold the expected message from the server.
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:**
- [out] `mqp` packet buffer to hold the message received from the server.
- [in] `wait_secs` Maximum Time to wait for a message from the server.
- [out] `app` place holder to indicate application handle for the packet.

**Returns:**
- On success, the number of bytes received for 'msg_type' from server, otherwise LIB defined error values ([LIBRARY Generated Error Codes](#)).

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

```c
i32 mqtt_client_buffers_register ( u32 num_mqp,
                                 struct mqtt_packet * mqp_vec,
                                 u32 buf_len,
                                 u8 * buf_vec )
```

Create a pool of MQTT Packet Buffers for the client library. This routine creates a pool of free MQTT Packet Buffers by attaching a buffer (buf) to a packet holder (mqp). The count of mqp elements and buf elements in the routine are same. And the size of the buffer in constant across all the elements.

The MQTT Packet Buffer pool should support (a) certain number of
in-flight and stored packets that await ACK(s) from the server (b) certain number of packets from server whose processing would be deferred by the client app (to another context) (c) a packet to create a CONNECT message to re-establish transaction with the server.

A meaningful size of the pool is very much application specific and depends on the target functionality. For example, an application that intends to have only one in-flight message to the server would need atmost three MQP buffers (1 for TX (for Qos1 or 2 store), 1 for RX and 1 for CONNECT message). If the application sends only QoS0 messages to the server, then the number of MQP buffers would reduce to two (i.e. 1 Tx to support CONNECT / PUB out and 1 RX)

Parameters:

[in] num_mqp  Number or count of elements in mqp_vec and buf_vec.
[in] buf_len  The size or length of the buffer element in the 'buf_vec'
[in] buf_vec  An array of buffers. 0 on success otherwise -1 on error.

Note:
The parameters mqp_vec and buf_vec should be persistent entities.

See also:

mqtt_client_await_msg
mqtt_client_run

i32 mqtt_client_ctx_await_msg (void * ctx, u8 msg_type, struct mqtt_packet * mqp, u32 wait_secs)
Block on the 'context' to receive a message type with-in specified wait time. This service is valid only for the configuration, where the application has not provided the callbacks to the client LIB 'context'. The caller must provide a packet buffer of adequate size to hold the expected message from the server.

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 and these messages are not the one that client is waiting for.

Parameters:

[in] ctx handle to the underlying network context in the LIB

See also:

mqtt_client_ctx_create

Parameters:

[in] msg_type message type to receive. A value of 0 would imply that caller is ready to receive the next message, whatsoever, from the server.

[out] mqp packet buffer to hold the message received from the server.

[in] wait_secs maximum Time to wait for a message from the server.

Returns:

On success, the number of bytes received for 'msg_type' from server, otherwise LIB defined error values (LIBRARY Generated Error Codes)

```
i32 mqtt_client_ctx_create (const struct mqtt_client_ctx_cfg * ctx, const struct mqtt_client_ctx_cbs * cctx, void * app)
```
Create a Network Connection Context. This routine sets-up the parameters that are required to operate and manage the network connection with a MQTT server / broker. As part of the creation of a context, the implementation also records the handle, if provided, by the application. In addition, the invoker of the routine must facilitate a place holder to enable the client LIB to provision the reference to the 'context', so created.

Specifically, this routine associates or ties-up, in an one-to-one manner, the caller provided handle 'app' and the client LIB provisioned handle 'ctx'. The parameter 'app' is returned by the client LIB in certain other routines to indicate the underlying 'context' with which network transaction or event is associated. Similarly, the caller must specify the context handle 'ctx' for which the services are being invoked.

A user or a task prior to utilizing the services of the library to schedule MQTT transactions must create a 'context'. A client LIB 'context' can be operated in two modes: (a) sync-wait or explicit receive mode and (b) the callback mode. Provisioning or absence of the callback parameter in this routine defines the mode of operation of the 'context'.

Explicit receive mode is analogous to the paradigm of the socket programming in which an application utilize the recv() function call. It is anticipated that applications which would make use of limited set of MQTT messages may find this mode of operation useful. Applications which intend to operate the 'context' in this mode must not provision any callbacks.

On the other hand, certain applications, may prefer an asynchronous mode of operation and would want the client LIB 'context' to raise callbacks into the application, as and when, packets arrive from the server. And such applications must provide the callback routines.

Parameters:
[in]  **ctx_cfg**  configuration information for the Network Context.

[in]  **ctx_cbs**  callback routines. Must be set to NULL, if the application intends to operate the context in the sync-wait / explicit receive mode.

[in]  **app**  handle to application. Returned by LIB in other routines to refer to the underlying context.

[out]  **ctx**  reference to the context created and is provisioned by the implementation. (Valid only if routine returns a success)

**Returns:**
0 on success otherwise -1.

---

### i32 mqtt_client_ctx_delete ( void * ctx )

Delete a Network Connection Context. This routines destroys the previously created network 'context' and releases resources that would be assigned for maintaining the information about the 'context'.

A user or a task prior to deleting the 'context' must ensure that there is no active MQTT connection on this context.

**Parameters:**
- [in]  **ctx**  handle to network context to be deleted. The context must have been previously created.

**Returns:**
0 on success otherwise -1

---

### i32 mqtt_client_ctx_info_register ( void * ctx, const struct utf8_string * client, const struct utf8_string * user, const struct utf8_string * pass )


Register application info and its credentials with the client library. This routine registers information for all the specified parameters, therefore, an update to single element would imply re-specification of the other parameters, as well.

Note:
Contents embedded in the parameters is not copied by the routine, and instead a reference to the listed constructs is retained. Therefore, the app must enable the parameter contents for persistency.

Parameters:
- [in] `ctx` handle to the underlying network context in the LIB

See also:
- `mqtt_client_ctx_create`

Parameters:
- [in] `client_id` MQTT UTF8 identifier of the client. If set to NULL, then the client will be treated as zero length entity.
- [in] `user_name` MQTT UTF8 user name for the client. If not used, set it to NULL. If used, then it can't be of zero length.
- [in] `pass_word` MQTT UTF8 pass word for the client. If not used, set it to NULL, If used, then it can't be of zero length.

Returns:
- 0 on success otherwise -1

User name without a pass word is a valid configuration. A pass word won't be processed if it is not associated with a valid user name.
u32 wait_secs
)

Run the context for the specified wait time. This service is valid only for the configuration, where the application has populated the callbacks that can be invoked by the client LIB 'context'.

This routine yields the control back to the application after the duration of the wait time. Such an arrangement enables the application to make overall progress to meet its 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 the wait time.

**Parameters:**

- **[in]** ctx handle to the underlying network context in the LIB

**See also:**

mqtt_client_ctx_create

**Parameters:**

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

**Returns:**

- MQP_ERR_NOTCONN if MQTT connection is closed by the application, MQP_ERR_TIMEOUT if there was no MQTT transaction in the interval of wait time and other values (LIBRARY Generated Error Codes)

```c
i32 mqtt_client_ctx_will_register ( void * ctx,
const struct utf8_string * will_to
const struct utf8_string * will_m
```
Register WILL information of the client application. This routine registers information for all the specified parameters, therefore, an update to a single element would imply re-specification of the other parameters, as well.

**Note:**
Contents embedded in the parameters is not copied by the routine, and instead a reference to the listed constructs is retained. Therefore, the app must enable the parameter contents for persistency.

**Parameters:**
- `[in] ctx` handle to the underlying network context in the LIB

**See also:**
- `mqtt_client_ctx_create`

**Parameters:**
- `[in] will_top` UTF8 WILL Topic on which WILL message is to be published.
- `[in] will_msg` UTF8 WILL message.
- `[in] will_qos` QOS for the WILL message
- `[in] retain` asserted to indicate that published WILL must be retained

**Returns:**
- 0 on success otherwise -1.

Both `will_top` and `will_msg` should be either present or should be NULL. `will_qos` and `retain` are relevant only for a valid Topic and Message combo.

```c
bool mqtt_client_is_connected ( void * ctx )
```
Ascertain whether connection / session with the server is active or not. Prior to sending out any information any message to server, the application can use this routine to check the status of the connection. If connection does not exist, then client should first CONNECT to the broker.

A connection to server could have been closed unsolicitedly either due to keep alive time-out or due to error in RX / TX transactions.

**Note:**
this API does not refer to network layer connection

**Parameters:**

> [in] `ctx` handle to the underlying network context in the LIB

**See also:**

`mqtt_client_ctx_create`

**Returns:**

ture if connection is active otherwise false.

```c
i32 mqtt_client_lib_exit ( void )
```

Exit the MQTT client library.

**Returns:**

0 on success otherwise -1.

```c
i32 mqtt_client_lib_init ( const struct mqtt_client_lib_cfg * cfg )
```

Initialize the MQTT client library. This routine initializes all the common constructs that are required to manage the multiple network connections. The client 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. Depending upon the deployment needs, this routine can be invoked either as part of the platform initialization sequence or as part of the application. Deployments that have more than one application utilizing the services of the client LIB should try to invoke the routine from the initialization sequence of the platform.

In addition, if an application has to manage more than one network connections (i.e. in other words, if the application has to handle a group of connections), then certain configuration must be set in the LIB

See also:  
struct mqtt_client_lib_cfg

Note:  
There must be only one group of network connections in the system.

Parameters:  
[in] cfg Configuration information for the MQTT client Library.

Returns:  
0 on success otherwise -1.

i32 mqtt_client_net_svc_register (const struct device_net_services

Abstraction for the device specific network services. Network services for communication with the server

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.

---

**u16 mqtt_client_new_msg_id (void )**

Provides a new MSG Identifier for a packet dispatch to server

**Returns:**

MSG / PKT Transaction identifier

---

**i32 mqtt_client_pub_dispatch ( void * ctx,**

```
struct mqtt_packet * mqp,
enum mqtt_qos qos,
bool retain
```

) Dispatch application constructed PUBLISH message to the server. Prior to sending the message to the server, this routine will prepare a fixed header to account for the size of the contents and the flags that have been indicated by the caller.

After the packet has been sent to the server, if the associated QoS of the dispatched packet is ether level 1 or 2, the client LIB 'context' will then store the packet until the time, a corresponding PUB-ACK (for QoS1) or PUB-REC (QoS2) message is received from the server.

If the client LIB 'context' has been configured to assert 'clean session', then the references to all the stored and unacknowledged PUBLISH messages are dropped at the time of MQTT disconnection (or network disconnection). Otherwise, these unacknowledged packets continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored PUBLISH messages, if any, are
The caller must populate the payload information with topic and data before invoking this service.

This service facilitates direct writing of topic and (real-time) payload data into the buffer, thereby, avoiding power consuming and wasteful intermediate data copies.

In case, the routine returns an error, the caller is responsible for freeing up or re-using the packet buffer. For all other cases, the client library will manage the return of the packet buffer to the pool.

**Parameters:**

- `[in] ctx` handle to the underlying network context in the LIB

**See also:**

- `mqtt_client_ctx_create`

**Parameters:**

- `[in] mqp` app created PUBLISH message without the fixed header
- `[in] qos` QoS with which the message needs to send to server
- `[in] retain` Asserted if the message is to be retained by server.

**Returns:**

- on success, the transaction Message ID, otherwise LIB defined errors (*LIBRARY Generated Error Codes*)

```c
i32 mqtt_client_pub_msg_send ( void * ctx, const struct utf8_string * topic, const u8 * data_buf, u32 data_len, enum mqtt_qos qos, bool retain );
```
Send a PUBLISH message to the server (don't wait for PUBACK / PUBREC). This routine creates a PUBLISH message in an internally allocated packet buffer by embedding the 'topic' and 'data' contents, then prepares the packet header and finally, dispatches the message to the server.

After the packet has been sent to the server, if the associated QoS of the dispatched packet is ether level 1 or 2, the client LIB 'context' will then store the packet until the time, a corresponding PUB-ACK (for QoS1) or PUB-REC (QoS2) message is received from the server.

If the client LIB 'context' has been configured to assert 'clean session', then the references to all the stored and unacknowledged PUBLISH messages are dropped at the time of MQTT disconnection (or network disconnection). Otherwise, these unacknowledged packets continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored PUBLISH messages, if any, are dropped.

**Parameters:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>ctx</td>
<td>handle to the underlying network context in the LIB</td>
</tr>
</tbody>
</table>

**See also:**

mqtt_client_ctx_create

**Parameters:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>topic</td>
<td>UTF8 based Topic Name for which data is being published.</td>
</tr>
<tr>
<td>[in]</td>
<td>data_buf</td>
<td>The binary data that is being published for the topic.</td>
</tr>
<tr>
<td>[in]</td>
<td>data_len</td>
<td>The length of the binary data.</td>
</tr>
<tr>
<td>[in]</td>
<td>qos</td>
<td>quality of service of the message</td>
</tr>
<tr>
<td>[in]</td>
<td>retain</td>
<td>should the server retain the message.</td>
</tr>
</tbody>
</table>
**Returns:**

on success, a transaction message id otherwise, LIB defined errors (*LIBRARY Generated Error Codes*)

---

**i32 mqtt_client_run ( u32 wait_secs )**

Run the LIB for the specified wait time. This service is valid only for the set-up of grouped contexts, where the application has populated the callbacks that can be invoked by the LIB.

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

**Returns:**

on connection close by client app, number of bytes received for the last msg from broker, otherwise LIB defined error values.

**Note:**

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

---

**i32 mqtt_client_send_progress ( void * ctx )**

Send remaining data or contents of the scheduled message to the server. This routine tries to send the remaining data in an active
transfer of a message to the server. This service is valid, only if the network layer of the platform is not able to send out the entire message in one TCP packet and has to "back-off" or "give up the control" before it can schedule or dispatch the next packet. In such a scenario, the network layer sends the first part (segment) of the scheduled message in the mqtt_xxx_send() API and the subsequent parts or the segments are sent using this routine.

This routine is not applicable to the platforms or the scenarios, where the implementation of the platform can segment the MQTT message in a manner to schedule consecutive or back-to-back blocking socket transactions. Specifically, this API must be used by an application, only if the network layer can indicate in an asynchronous manner, its readiness to send the next packet to the server. And the mechanism to indicate readiness of the network layer for the next send transaction is out of band and out of scope for the Client LIB and depends on the platform.

A platform that resorts to partial send of a message and has to back-off from transmission implies the following the considerations on to the application. (a) The next segment / part of the currently active MQTT packet can be sent or scheduled only after receiving the indication from the network layer to do so. (b) The next or new MQTT message (or its first segment) can be scheduled for transmission only after receiving the indication for completion of handling of the last segment of currently active message.

**Note:**

The application developer should refer to the platform specific network implementation for details.

The routine returns the number of remaining bytes in the message to be sent. However, as described earlier, the application is expected to wait for an indication about the readiness of the network layer prior to sending or scheduling another segment, if so available, to the server. Now, the new segment can be a next part of the currently active message or it can be the first segment of a new message. A return value of zero means that there is no more data left in the scheduled message to be sent to the server and the application should wait for
an appropriate event to indicate the transmission of the last segment.

In case of an error, the transfer of the remaining segments or parts of the scheduled message is aborted. Depending on the configuration of the 'clean session' in-conjunction with the revision of the MQTT protocol, the active message would be stored for re-transmission, MQTT connection is established again. To store a message for re-transmission, at least one segment of the message must have been successfully sent to the server.

**Note:**
This API must be used by the application only if the platform has the capability to indicate the completion of the sending of an active segment.

**Parameters:**
- `[in] ctx` handle to the underlying network context in the LIB

**See also:**
- `mqtt_client_ctx_create`

**Returns:**
- the number of bytes remaining to be sent in the message. Otherwise, LIB defined errors (**LIBRARY Generated Error Codes**)

```c
i32 mqtt_connect_msg_send ( void * ctx,
                           bool clean_session,
                           u16 ka_secs
                        )
```

Send the CONNECT message to the server (and don't wait for CONNACK). This routine accomplishes multiple sequences. As a first step, it tries to establish a network connection with the server. Then, it populates an internally allocated packet buffer with all the previously provided payload data information, prepares the requisite headers and finally, dispatches the constructed message to the server.
Prior to invoking this service, the client application should provision the intended payload contents of the CONNECT message by using the API(s) `mqtt_client_ctx_info_register` and `mqtt_client_ctx_will_register`. And information about the server of interest must be provided in the client LIB 'context' creation (`mqtt_client_ctx_create`).

The client application must invoke an appropriate receive routine to know about the corresponding response CONNACK from the server. The client LIB will close the network connection to the server, if the server happens to refuse the CONNECT request.

**Parameters:**

- `[in] ctx` handle to the underlying network context in the LIB

**See also:**

- `mqtt_client_ctx_create`

**Parameters:**

- `[in] clean_session` asserted to delete references to previous session at both server and client
- `[in] ka_secs` Keep Alive Time

**Returns:**

- number of bytes sent or LIB defined errors ([LIBRARY Generated Error Codes](#))

```c
i32 mqtt_disconn_send ( void * ctx )
```

Send a DISCONNECT message to the server.

**Parameters:**

- `[in] ctx` handle to the underlying network context in the LIB

**See also:**

- `mqtt_client_ctx_create`
Returns:
    number of bytes sent or Lib define errors (LIBRARY Generated Error Codes)

\textbf{i32 \texttt{mqtt\_pingreq\_send ( void * \texttt{ctx} )}}

Send a PINGREQ message to the server (and don't wait for PINGRSP).

\textbf{Parameters:}
    \texttt{[in] \texttt{ctx}} handle to the underlying network context in the LIB

\textbf{See also:}
    \texttt{mqtt\_client\_ctx\_create}

\textbf{Returns:}
    number of bytes sent or Lib define errors (LIBRARY Generated Error Codes)

\textbf{i32 \texttt{mqtt\_sub\_dispatch ( void * \texttt{ctx},
                     \texttt{struct mqtt\_packet * \texttt{mqp}}
                   )}}

Dispatch application constructed SUSBSCRIBE message to the server. Prior to sending the message to the server, this routine will prepare a fixed header to account for the size of the size of the contents.

After the packet has been dispatched to the server, the library will store the packet until the time, a corresponding SUB-ACK has been received from the server. This mechanism enables the client LIB 'context' to trace the sequence of the message-ID and / or resend the SUB packets to the server.

The client LIB 'context', if configured to operate in the MQTT 3.1.1 mode will drop or remove the un-acknowledged SUB messages at
the time of the termination of the network connection.

In the MQTT 3.1 mode, the client LIB 'context' will remove the unacknowledged SUB messages at the time of the termination of the network connection, if the 'clean session' has been asserted. In case, the 'clean session' has not been asserted, the stored SUB messages will continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored SUBSCRIBE messages, if any, are dropped.

The caller must populate the payload information of topic along with qos before invoking this service.

This service facilitates direct writing of topic and (real-time) payload data into the buffer, thereby, avoiding power consuming and wasteful intermediate data copies.

In case, the routine returns an error, the caller is responsible for freeing up or re-using the packet buffer. For all other cases, the client library will manage the return of the packet buffer to the pool.

Parameters:

[in] ctx handle to the underlying network context in the LIB

See also:

mqtt_client_ctx_create

Parameters:

[in] mqp app created SUBSCRIBE message without the fixed header.

Returns:

on success, the transaction Message ID, otherwise Lib defined errors (LIBRARY Generated Error Codes)

i32 mqtt_sub_msg_send ( void * ctx,
Send a SUBSCRIBE message to the server (and don't wait for SUBACK). This routine creates a SUBSCRIBE message in an internally allocated packet buffer by embedding the 'qos_topics', then prepares the message header and finally, dispatches the packet to the server.

After the packet has been dispatched to the server, the library will store the packet until the time, a corresponding SUB-ACK has been received from the server. This mechanism enables the client LIB 'context' to trace the sequence of the message-ID and / or resend the SUB packets to the server.

The client LIB 'context', if configured to operate in the MQTT 3.1.1 mode will drop or remove the un-acknowledged SUB messages at the time of the termination of the network connection.

In the MQTT 3.1 mode, the client LIB 'context' will remove the unacknowledged SUB messages at the time of the termination of the network connection, if the 'clean session' has been asserted. In case, the 'clean session' has not been asserted, the stored SUB messages will continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored SUBSCRIBE messages, if any, are dropped.

**Parameters:**

- **[in]** `ctx` handle to the underlying network context in the LIB

See also:

- `mqtt_client_ctx_create`

**Parameters:**

- **[in]** `qos_topics` an array of topic along-with its qos
[in] count the number of elements in the array

Returns:
on success, the transaction Message ID, otherwise Lib defined errors (LIBRARY Generated Error Codes)

i32 mqtt_unsub_dispatch ( void * ctx,
                        struct mqtt_packet * mqp )

Dispatch application constructed UNSUSBScribe message to the server. Prior to sending the message to the server, this routine will prepare a fixed header to account for the size of the size of the contents.

After the packet has been dispatched to the server, the library will store the packet until the time, a corresponding UNSUB-ACK has been received from the server. This mechanism enables the client LIB 'context' to trace the sequence of the message-ID and / or resend the UNSUB packets to the server.

The client LIB 'context', if configured to operate in the MQTT 3.1.1 mode will drop or remove the un-acknowledged SUB messages at the time of the termination of the network connection.

In the MQTT 3.1 mode, the client LIB 'context' will remove the unacknowledged UNSUB messages at the time of the termination of the network connection, if the 'clean session' has been asserted. In case, the 'clean session' has not been asserted, the stored UNSUB messages will continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored UNSUBSCRIBE messages, if any, are dropped.

The caller must populate the payload information of topics before invoking this service.
This service facilitates direct writing of topic and (real-time) payload data into the buffer, thereby, avoiding power consuming and wasteful intermediate data copies.

In case, the routine returns an error, the caller is responsible for freeing up or re-using the packet buffer. For all other cases, the client library will manage the return of the packet buffer to the pool.

**Parameters:**

[in] `ctx` handle to the underlying network context in the LIB

**See also:**

`mqtt_client_ctx_create`

**Parameters:**

[in] `Packet` Buffer that holds UNSUBSCRIBE message without a fixed header

**Returns:**

on success, the transaction Message ID, otherwise LIB defined errors *(LIBRARY Generated Error Codes)*

```c
i32 mqtt_unsub_msg_send ( void * ctx, const struct utf8_string * topics, u32 count )
```

Send an UNSUBSCRIBE message to the server (and don't wait for UNSUBACK). This routine creates an UNSUBSCRIBE message in an internally allocated packet buffer by embedding the 'topics', then prepares the message header and finally, dispatches the packet to the server.

After the packet has been dispatched to the server, the library will store the packet until the time, a corresponding UNSUB-ACK has been received from the server. This mechanism enables the client LIB 'context' to trace the sequence of the message-ID and / or resend the
UNSUB packets to the server.

The client LIB 'context', if configured to operate in the MQTT 3.1.1 mode will drop or remove the un-acknowledged SUB messages at the time of the termination of the network connection.

In the MQTT 3.1 mode, the client LIB 'context' will remove the unacknowledged UNSUB messages at the time of the termination of the network connection, if the 'clean session' has been asserted. In case, the 'clean session' has not been asserted, the stored UNSUB messages will continue to be available for the next iteration of the MQTT connection. However, if the client application asserts the 'clean session' parameter in the next iteration of the CONNECT operation, then references to all the stored UNSUBSCRIBE messages, if any, are dropped.

**Parameters:**

- [in] **ctx** handle to the underlying network context in the LIB

**See also:**

`mqtt_client_ctx_create`

**Parameters:**

- [in] **topics** an array of topic to unsubscribe
- [in] **count** the number of elements in the array

**Returns:**

- on success, the transaction Message ID, otherwise Lib defined errors ([LIBRARY Generated Error Codes](#))
<table>
<thead>
<tr>
<th>Files</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Options for application to config CTX</strong></td>
<td></td>
</tr>
</tbody>
</table>
Defines

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT_CFG_PROTOCOL_V31</td>
<td>0x0001</td>
</tr>
<tr>
<td>MQTT_CFG_APP_HAS_RTSK</td>
<td>0x0002</td>
</tr>
<tr>
<td>MQTT_CFG_MK_GROUP_CTX</td>
<td>0x0004</td>
</tr>
</tbody>
</table>
### Define Documentation

```c
#define MQTT_CFG_APP_HAS_RTSK  0x0002

Assert if APP has dedicated RX Task
```

```c
#define MQTT_CFG_MK_GROUP_CTX  0x0004

Assert if task manages > 1 contexts
```

```c
#define MQTT_CFG_PROTOCOL_V31  0x0001

Assert for ver3.1, default is 3.1.1
```

---

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
<table>
<thead>
<tr>
<th>Files</th>
<th>Defines</th>
</tr>
</thead>
</table>
| Options for App to configure network | }
Defines

#define MQTT_NETCONN_OPT_IP6 DEV_NETCONN_OPT_IP6
#define MQTT_NETCONN_OPT_URL DEV_NETCONN_OPT_URL
#define MQTT_NETCONN_OPT_SEC DEV_NETCONN_OPT_SEC
Define Documentation

```
#define MQTT_NETCONN_OPT_IP6  DE.TemplateNETCONN_OPT_IP6

Options for platform to configure network
```

```
#define MQTT_NETCONN_OPT_SEC  DE.TemplateNETCONN_OPT_SEC

Options for platform to configure network
```

```
#define MQTT_NETCONN_OPT_URL  DE.TemplateNETCONN_OPT_URL

Options for platform to configure network
```
Class List

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

- client_ctx
- device_net_services
- mqtt_ack_wlist
- mqtt_client_ctx_cbs
- mqtt_client_ctx_cfg
- mqtt_client_lib_cfg
- mqtt_packet
- pub_qos2_cq
- secure_conn
- utf8_string
- utf8_strqos
client_ctx Struct Reference

List of all members.
Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *</td>
<td>usr</td>
</tr>
<tr>
<td>i32</td>
<td>net</td>
</tr>
<tr>
<td>i8</td>
<td>remote_ip [16]</td>
</tr>
<tr>
<td>u32</td>
<td>ip_length</td>
</tr>
<tr>
<td>u32</td>
<td>timeout</td>
</tr>
<tr>
<td>u16</td>
<td>ka_secs</td>
</tr>
<tr>
<td>u32</td>
<td>flags</td>
</tr>
<tr>
<td>struct client_ctx *</td>
<td>next</td>
</tr>
</tbody>
</table>

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
device_net_services

Struct Reference

Abstraction of Network Services on a platform

List of all members.
### Public Attributes

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

The `time` function takes a void argument, while all other functions take a variable number of arguments that include memory pointers, struct references, and various types.
### Member Data Documentation

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

Accept an incoming 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 at least 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, u32 wait_secs)**

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

```c
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 contract, otherwise NULL

```c
i32(* device_net_services::open)(u32 nwconn_opts, const i8 *server_addr, u16 port_number, const struct
``` 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 sever. 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
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.

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.
maximum length of 'buf'
place-holder for the port of the sender network entity
place-holder to retrieve the IP address of the sender network entity. The memory space must be provisioned to store at least 16 bytes.
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.

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
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:**

<table>
<thead>
<tr>
<th>Param</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>comm</td>
<td>handle to network connection as returned by open().</td>
</tr>
<tr>
<td>[in]</td>
<td>buf</td>
<td>refers to data that is intended to be sent</td>
</tr>
<tr>
<td>[in]</td>
<td>len</td>
<td>length of the data</td>
</tr>
<tr>
<td>[in]</td>
<td>dest_port</td>
<td>network port to which data is to be sent.</td>
</tr>
<tr>
<td>[in]</td>
<td>dest_ip</td>
<td>IP address of the entity to which data is to be sent.</td>
</tr>
<tr>
<td>[in]</td>
<td>ip_len</td>
<td>length of the destination IP address.</td>
</tr>
</tbody>
</table>

**Returns:**

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

```c
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 responsibility 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_common.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
mqtt_ack_wlist Struct Reference

List of all members.
Public Attributes

```c
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_common.h`
#include `<mqtt_client.h>`

List of all members.
### Public Attributes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool(* publish_rx)(void *app, bool dup, enum mqtt_qos qos, bool retain, struct mqtt_packet *mqp)</code></td>
<td>Function to publish a message with various attributes.</td>
</tr>
<tr>
<td><code>void(* ack_notify)(void *app, u8 msg_type, u16 msg_id, u8 *buf, u32 len)</code></td>
<td>Function to handle acknowledgment messages.</td>
</tr>
<tr>
<td><code>void(* disconn_cb)(void *app, i32 cause)</code></td>
<td>Function to handle disconnection events.</td>
</tr>
</tbody>
</table>
Detailed Description

Callbacks to be invoked by MQTT Client library onto Client application
Member Data Documentation

```c
void(* mqtt_client_ctx_cbs::ack_notify)(void *app, u8 msg_type, u16 msg_id, u8 *buf, u32 len)
```

Notifies the client application about an ACK or a response from the server. Following are the messages that are notified by the client LIB to the application.

CONNACK, PINGRSP, PUBACK, PUBCOMP, SUBACK, UNSUBACK

**Parameters:**

- `[in] app` application to which this ACK message is targeted

**See also:**

- `mqtt_client_ctx_create`

**Parameters:**

- `[in] msg_type` Type of the MQTT message
- `[in] msg_id` transaction identity of the message
- `[in] buf` refers to contents of message and depends on msg_type
- `[in] len` length of the buf

**Returns:**

- none

**Note:**

The size of the buf parameter i.e len is non-zero for the SUBACK and CONNACK messages. For SUBACK the buf carries an array of QOS responses provided by the server. For CONNACK, the buf carries variable header contents. Helper macro `VHB_CONNACK_RC()` and `VHB_CONNACK_SP()` can be used to access contents provided by the server. For all other messages, the value of len parameter is zero. The parameter msg_id is not relevant for the messages CONNACK and PINGRSP and is set to zero.
### `disconn_cb` Callback

```c
void(* mqtt_client_ctx_cbs::disconn_cb)(void *app, i32 cause)
```

Notifies the client application about the termination of connection with the server. After servicing this callback, the application can destroy associated context if it no longer required.

**Parameters:**
- **app** application whose connection got terminated

**See also:**
- `mqtt_client_ctx_create`

**Parameters:**
- **cause** reason that created disconnection

**See also:**
- `mqtt_client_ctx_create`

### `publish_rx` Callback

```c
bool(* mqtt_client_ctx_cbs::publish_rx)(void *app, bool dup, enum)
```

Provides a PUBLISH message from server to the client application. The application can utilize the associated set of helper macros to get references to the topic and the data information contained in the message. **Helper Macros for RX PUBLISH**

Depending upon the QoS level of the message, the MQTT client library shall dispatch the corresponding acknowledgement (PUBACK or PUBREC) to the server, thereby, relieving application of this support.

If the application completes the processing of the packet within the implementation of this callback routine, then a value of 'true' must be returned to the client LIB 'context'. The library, in this case, does not handover the packet to application and instead, frees it up on return from this routine.

If the application intends to defer the processing of the PUBLISH message to a different execution task, then it must takeover the ownership of the packet by returning a value of 'false' to the client LIB 'context. In this arrangement, the client LIB 'context' hands over the...
packet to the application. The responsibility of storing, managing and eventually freeing up the packet back to the pool, now, lies with the app.

**Parameters:**

[in] **app** application to which this PUBLISH message is targeted

**See also:**

`mqtt_client_ctx_create`

**Parameters:**

[in] **dup** asserted to indicate a DUPLICATE PUBLISH  
in [in] **qos** quality of Service of the PUBLISH message  
in [in] **retain** Asserted to indicate message at new subscription  
in [in] **mqp** Packet Buffer that holds the PUBLISH message

**Returns:**

true to indicate that processing of the packet has been completed and it can freed-up and returned back to the pool by the library. Otherwise, false.

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

- D:/my_data/GIT/network_apps/netapps/mqtt/client/mqtt_client.h
**mqtt_client_ctx_cfg**

**Struct Reference**

The Client Library API(s)

List of all members.
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td>config_opts</td>
</tr>
<tr>
<td>u32</td>
<td>nwconn_opts</td>
</tr>
<tr>
<td>i8 *</td>
<td>server_addr</td>
</tr>
<tr>
<td>u16</td>
<td>port_number</td>
</tr>
<tr>
<td>struct secure_conn *</td>
<td>nw_security</td>
</tr>
</tbody>
</table>
Member Data Documentation

**u16** `mqtt_client_ctx_cfg::config_opts`

Context config Opt, Options for application to config CTX

**struct secure_conn** `*mqtt_client_ctx_cfg::nw_security`

Refer to Information to establish a secure connection.

**u32** `mqtt_client_ctx_cfg::nwconn_opts`

Network Options, Options for App to configure network

**u16** `mqtt_client_ctx_cfg::port_number`

Network Listening Port number of the server

**i8** `*mqtt_client_ctx_cfg::server_addr`

Reference to '\0' terminated address string

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

- D:/my_data/GIT/network_apps/netapps/mqtt/client/mqtt_client.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
Struct Reference

The Client Library API(s)

```c
#include <mqtt_client.h>
```

List of all members.
### Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td>loopback_port</td>
</tr>
<tr>
<td>bool</td>
<td>grp_uses_cbfn</td>
</tr>
<tr>
<td>void *</td>
<td>mutex</td>
</tr>
<tr>
<td>void (*)</td>
<td>mutex_lockin( void *mutex )</td>
</tr>
<tr>
<td>void (*)</td>
<td>mutex_unlock( void *mutex )</td>
</tr>
<tr>
<td>i32 (*)</td>
<td>debug_printf( const i8 *format,... )</td>
</tr>
<tr>
<td>bool</td>
<td>aux_debug_en</td>
</tr>
</tbody>
</table>
Detailed Description

Contract / Data to initialize MQTT Client Library
Member Data Documentation

**bool mqtt_client_lib_cfg::aux_debug_en**

Assert to indicate additional debug info

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

Debug, mandatory

**bool mqtt_client_lib_cfg::grp_uses_cbf**

Assert if grouped contexts use call-backs

**u16 mqtt_client_lib_cfg::loopback_port**

If an application has more than one contexts (i.e. grouped contexts) to manage in a single task, then a non-zero value must be provided. Otherwise, this parameter must be set to zero.

**void* mqtt_client_lib_cfg::mutex**

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

**void(** *mqtt_client_lib_cfg::mutex_lockin)*(**void **mutex**)

Take platform mutex function

**void(** *mqtt_client_lib_cfg::mutex_unlock)*(**void **mutex**)


Give platform mutex function

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

- D:/my_data/GIT/network_apps/netapps/mqtt/client/mqtt_client.h
mqtt_packet Struct Reference

MQTT Packet (MQP) Buffer structure

List of all members.
### Public Attributes

<table>
<thead>
<tr>
<th>Public Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>u8 msg_type</td>
</tr>
<tr>
<td>u8 fh_byte1</td>
</tr>
<tr>
<td>u16 msg_id</td>
</tr>
<tr>
<td>u8 n.refs</td>
</tr>
<tr>
<td>u8 pad [3]</td>
</tr>
<tr>
<td>u8 offset</td>
</tr>
<tr>
<td>u8 fh_len</td>
</tr>
<tr>
<td>u16 vh_len</td>
</tr>
<tr>
<td>u32 pl_len</td>
</tr>
<tr>
<td>u32 private</td>
</tr>
<tr>
<td>u32 maxlen</td>
</tr>
<tr>
<td>u8 * buffer</td>
</tr>
<tr>
<td>void(* free )(struct mqtt_packet *mpq)</td>
</tr>
<tr>
<td>struct mqtt_packet * next</td>
</tr>
</tbody>
</table>
### Member Data Documentation

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>u8* mqtt_packet::buffer</code></td>
<td>The attached buffer</td>
</tr>
<tr>
<td><code>u8 mqtt_packet::fh_byte1</code></td>
<td>Fixed Header: Byte1</td>
</tr>
<tr>
<td><code>u8 mqtt_packet::fh_len</code></td>
<td>Fix Header Length</td>
</tr>
<tr>
<td><code>void(* mqtt_packet::free)(struct mqtt_packet *mqp)</code></td>
<td>Method to free this packet to a particular pool</td>
</tr>
<tr>
<td><code>u32 mqtt_packet::maxlen</code></td>
<td>Maximum buffer size</td>
</tr>
<tr>
<td><code>u16 mqtt_packet::msg_id</code></td>
<td>Msg transaction ID</td>
</tr>
<tr>
<td><code>u8 mqtt_packet::msg_type</code></td>
<td>MQTT Message Type</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>u8 <code>mqtt_packet::n.refs</code></td>
<td></td>
</tr>
</tbody>
</table>

# users of this msg

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| u8 `mqtt_packet::offset` | Start of data index

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| u32 `mqtt_packet::pl.len` | Pay Load Length

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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_common.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
## pub_qos2_cq Struct Reference

List of all members.
Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td>id_vec [MAX_PUBREL_INFLT]</td>
</tr>
<tr>
<td>u8</td>
<td>n_free</td>
</tr>
<tr>
<td>u8</td>
<td>rd_idx</td>
</tr>
<tr>
<td>u8</td>
<td>wr_idx</td>
</tr>
</tbody>
</table>

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
### secure_conn Struct

**Reference**

*Information to establish a secure connection.*

List of all members.
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *</td>
<td>method</td>
</tr>
<tr>
<td>void *</td>
<td>cipher</td>
</tr>
<tr>
<td>u32</td>
<td>n_file</td>
</tr>
<tr>
<td>char **</td>
<td>files</td>
</tr>
</tbody>
</table>
## Member Data Documentation

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void* secure_conn::cipher</code></td>
<td>Reference to information about cryptograph ciphers</td>
</tr>
<tr>
<td><code>char** secure_conn::files</code></td>
<td>Reference to array of file-names used for security</td>
</tr>
<tr>
<td><code>void* secure_conn::method</code></td>
<td>Reference to information about protocol or methods</td>
</tr>
<tr>
<td><code>u32 secure_conn::n_file</code></td>
<td>Count of secure connection related files, certs...</td>
</tr>
</tbody>
</table>

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h
utf8_string Struct Reference

#include <mqtt_common.h>

List of all members.
Public Attributes

<table>
<thead>
<tr>
<th>i8  * buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16 length</td>
</tr>
</tbody>
</table>
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_common.h

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by [doxygen](http://www.doxygen.org) 1.7.4
utf8_strqos Struct
Reference

#include <mqtt_common.h>

List of all members.
Public Attributes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i8 *</td>
<td>buffer</td>
</tr>
<tr>
<td>u16</td>
<td>length</td>
</tr>
<tr>
<td>enum</td>
<td>mqtt_qos qosreq</td>
</tr>
</tbody>
</table>
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_common.h

---

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4
## Class Index

<table>
<thead>
<tr>
<th>File</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_ctx</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>mqtt_ack_wlist</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>mqtt_client_ctx_cbs</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>device_net_services</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by [doxygen](http://www.doxygen.org) 1.7.4
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_client_ctx_cbs](#)
  - aux_debug_en : [mqtt_client_lib_cfg](#)

- **b** -
  - buffer : [mqtt_packet](#), [utf8_strqos](#), [utf8_string](#)

- **c** -
  - cipher : [secure_conn](#)
  - close : [device_net_services](#)
  - config_opts : [mqtt_client_ctx_cfg](#)

- **d** -
  - debug_printf : [mqtt_client_lib_cfg](#)
  - disconn_cb : [mqtt_client_ctx_cbs](#)

- **f** -
  - fh_byte1 : [mqtt_packet](#)
- **fh_len**: `mqtt_packet`
- **files**: `secure_conn`
- **free**: `mqtt_packet`

- **g**
  - **grp_uses_cbfn**: `mqtt_client_lib_cfg`

- **i**
  - **io_mon**: `device_net_services`

- **l**
  - **length**: `utf8_string, utf8_strqos`
  - **listen**: `device_net_services`
  - **loopback_port**: `mqtt_client_lib_cfg`

- **m**
  - **maxlen**: `mqtt_packet`
  - **method**: `secure_conn`
  - **msg_id**: `mqtt_packet`
  - **msg_type**: `mqtt_packet`
  - **mutex**: `mqtt_client_lib_cfg`
  - **mutex_lockin**: `mqtt_client_lib_cfg`
  - **mutex_unlock**: `mqtt_client_lib_cfg`

- **n**
  - **n_file**: `secure_conn`
  - **n_refs**: `mqtt_packet`
  - **nw_security**: `mqtt_client_ctx_cfg`
  - **nwconn_opts**: `mqtt_client_ctx_cfg`

- **o**
  - **offset**: `mqtt_packet`
  - **open**: `device_net_services`
- p -
  - pl_len : mqtt_packet
  - port_number : mqtt_client_ctx_cfg
  - publish_rx : mqtt_client_ctx_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
  - server_addr : mqtt_client_ctx_cfg

- t -
  - time : device_net_services

- v -
  - vh_len : mqtt_packet
Here is a list of all documented files with brief descriptions:

- **client_notes.h** [code]
- **MainPage.h** [code]
- D:/my_data/GIT/network_apps/netapps/mqtt/client/mqtt_client.h [code]
- D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h [code]
```
#include "mqtt_common.h"

Go to the source code of this file.
```
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mqtt_client_ctx_cbs</td>
<td></td>
</tr>
<tr>
<td>mqtt_client_ctx_cfg</td>
<td></td>
</tr>
<tr>
<td>mqtt_client_lib_cfg</td>
<td></td>
</tr>
</tbody>
</table>
### Defines

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MQTT_CLIENT_VERSTR</code></td>
<td>&quot;1.0.0&quot;</td>
</tr>
<tr>
<td><code>VHB_CONNACK_RC</code> (vh_buf)</td>
<td>vh_buf[1]</td>
</tr>
<tr>
<td><code>MQP_CONNACK_RC</code> (mqp)</td>
<td>mqp-&gt;buffer[3]</td>
</tr>
<tr>
<td><code>VHB_CONNACK_SP</code> (vh_buf)</td>
<td>vh_buf[0] &amp; 0x1</td>
</tr>
<tr>
<td><code>MQP_CONNACK_SP</code> (mqp)</td>
<td>mqp-&gt;buffer[2] &amp; 0x1</td>
</tr>
<tr>
<td><code>VHB_CONNACK_VH16</code> (vh_buf)</td>
<td>((vh_buf[0] &lt;&lt; 8)</td>
</tr>
<tr>
<td><code>MQP_CONNACK_VH16</code> (mqp)</td>
<td>((mqp-&gt;buffer[2] &lt;&lt; 8)</td>
</tr>
<tr>
<td><code>MQTT_CFG_PROTOCOL_V31</code></td>
<td>0x0001</td>
</tr>
<tr>
<td><code>MQTT_CFG_APP_HAS_RTSK</code></td>
<td>0x0002</td>
</tr>
<tr>
<td><code>MQTT_CFG_MK_GROUP_CTX</code></td>
<td>0x0004</td>
</tr>
<tr>
<td><code>MQTT_NETCONN_OPT_IP6</code></td>
<td>DEV_NETCONN_OPT_IP6</td>
</tr>
<tr>
<td><code>MQTT_NETCONN_OPT_URL</code></td>
<td>DEV_NETCONN_OPT_URL</td>
</tr>
<tr>
<td><code>MQTT_NETCONN_OPT_SEC</code></td>
<td>DEV_NETCONN_OPT_SEC</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Function Name</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td><code>mqtt_client_new_msg_id</code></td>
<td>(void)</td>
</tr>
<tr>
<td>bool</td>
<td><code>mqtt_client_is_connected</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_connect_msg_send</code></td>
<td>(void *ctx, bool clean_session, u16 ka_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_pub_msg_send</code></td>
<td>(void *ctx, const struct <code>utf8_string</code> *topic, const u8 *data_buf, u32 data_len, enum <code>mqtt_qos</code> qos, bool retain)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_pub_dispatch</code></td>
<td>(void *ctx, struct <code>mqtt_packet</code> *mqp, enum <code>mqtt_qos</code> qos, bool retain)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_sub_msg_send</code></td>
<td>(void *ctx, const struct <code>utf8_strqos</code> *qos_topics, u32 count)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_sub_dispatch</code></td>
<td>(void *ctx, struct <code>mqtt_packet</code> *mqp)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_unsub_msg_send</code></td>
<td>(void *ctx, const struct <code>utf8_string</code> *topics, u32 count)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_unsub_dispatch</code></td>
<td>(void *ctx, struct <code>mqtt_packet</code> *mqp)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_pingreq_send</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_disconn_send</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_send_progress</code></td>
<td>(void *ctx)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_await_msg</code></td>
<td>(void *ctx, u8 msg_type, struct <code>mqtt_packet</code> *mqp, u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_run</code></td>
<td>(void *ctx, u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_await_msg</code></td>
<td>(struct <code>mqtt_packet</code> *mqp, u32 wait_secs, void **app)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_run</code></td>
<td>(u32 wait_secs)</td>
</tr>
<tr>
<td>struct <code>mqtt_packet</code> *</td>
<td><code>mqp_client_alloc</code></td>
<td>(u8 msg_type, u8 offset)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_buffers_register</code></td>
<td>(u32 num_mqp, struct <code>mqtt_packet</code> *mqp_vec, u32 buf_len, u8 *buf_vec)</td>
</tr>
<tr>
<td></td>
<td><code>mqtt_client_ctx_info_register</code></td>
<td>(void *ctx, const struct <code>utf8_string</code> *client_id, const struct</td>
</tr>
<tr>
<td>i32</td>
<td><code>utf8_string *user_name, const struct utf8_string *pass_word)</code></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_will_register (void *ctx, const struct utf8_string *will_top, const struct utf8_string *will_msg, enum mqtt_qos will_qos, bool retain)</code></td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_net_svc_register (const struct device_net_services *net)</code></td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_create (const struct mqtt_client_ctx_cfg *ctx_cfg, const struct mqtt_client_ctx_cbs *ctx_cbs, void *app, void **ctx)</code></td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_ctx_delete (void *ctx)</code></td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_lib_init (const struct mqtt_client_lib_cfg *cfg)</code></td>
<td></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_client_lib_exit (void)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

This C library provisions the interface / API(s) for the MQTT Client.

This is a light-weight library to enable the services of the MQTT protocol for one or more client applications (that would typically run on a resource constrained system). The key consideration in the design of small footprint library has been the abstraction of the low level details of the message transactions with the server and yet, provide rudimentary API(s), such that, the capabilities and features of the protocol are available to be utilized by existing and new applications in an un-restrictive manner.

The library is targeted to conform to MQTT 3.1.1 specification.

The MQTT Client library is a highly portable software and implies a very limited set of dependencies on a platform. Importantly, these limited dependencies are the commonly used 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 client library supports multiple and simultaneous MQTT connections to one or more servers. In this client LIB, the reference to an individual connection in conjunction with the associated configuration parameters has been termed as a 'context'. Therefore, the client library supports multiple 'contexts'. An application that intends to make use of the client library must set-up or create a 'context' prior to establishing a connection with the server. The application can choose to destroy the 'context' after the connection with the server has been terminated. The client LIB can only support a finite set of 'contexts' and the number can be configured by using a compiler line option / flag -DCFG_CL_MQTT_CTXS.

Once, the 'context' is set-up, the application can send and receive the
MQTT packets to / from the server. Among several features supported by the client LIB, the 'context' manages the keep-alive mechanism by automatically sending PING request to the server, if there has been no other packet send to the server with the keep-alive duration.

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

Go to the source code of this file.
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>mqtt_packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>utf8_string</td>
</tr>
<tr>
<td>struct</td>
<td>mqtt_ack_wlist</td>
</tr>
<tr>
<td>struct</td>
<td>utf8_strqos</td>
</tr>
<tr>
<td>struct</td>
<td>secure_conn</td>
</tr>
<tr>
<td>struct</td>
<td>device_net_services</td>
</tr>
<tr>
<td>struct</td>
<td>pub_qos2_cq</td>
</tr>
<tr>
<td>struct</td>
<td>client_ctx</td>
</tr>
</tbody>
</table>
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) | flags)
#define MAKE_FH_FLAGS(bool_dup, enum_qos, bool_retain) (u8)(((bool_dup << 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) >> 1)
#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) + mqp->fh_len)
#define MQP_PAYLOAD_BUF(mqp) (MQP_VHEADER_BUF(mqp) + mqp->vh_len)
#define MQP_CONTENT_LEN(mqp) (mqp->fh_len + mqp->vh_len + mqp->pl_len)
#define MQP_FREEBUF_LEN(mqp)
#define MQP_FHEADER_VAL(mqp) (mqp->fh_byte1)
#define MQP_FHEADER_MSG(mqp) (MSG_TYPE(MQP_FHEADER_VAL(mqp)))
#define MQP_FHEADER_FLG(mqp) (MSG_FLAGS(MQP_FHEADER_VAL(mqp)))

#define DEFINE_MQP_VEC(num_mqp, mqp_vec) static struct mqtt_packet mqp_vec[num_mqp];
#define DEFINE_MQP_BUF_VEC(num_mqp, mqp_vec, buf_len, buf_vec)
#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 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**

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>i32</td>
</tr>
<tr>
<td>unsigned int</td>
<td>u32</td>
</tr>
<tr>
<td>unsigned char</td>
<td>u8</td>
</tr>
<tr>
<td>char</td>
<td>i8</td>
</tr>
<tr>
<td>unsigned short</td>
<td>u16</td>
</tr>
<tr>
<td>short</td>
<td>i16</td>
</tr>
</tbody>
</table>
Enumerations

```c
enum mqtt_qos { MQTT_QOS0, MQTT_QOS1, MQTT_QOS2 }
```
### Functions

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Function Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>mqp_free</code></td>
<td><code>(struct mqtt_packet *mqp)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>mqp_reset</code></td>
<td><code>(struct mqtt_packet *mqp)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>mqp_init</code></td>
<td><code>(struct mqtt_packet *mqp, u8 offset)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.buf_wr_utf8</code></td>
<td><code>(u8 *buf, const struct utf8_string *utf8)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.buf_tail_wr_remlen</code></td>
<td><code>(u8 *buf, u32 remlen)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.buf_rd_remlen</code></td>
<td><code>(u8 *buf, u32 *remlen)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.pub_append_topic</code></td>
<td><code>(struct mqtt_packet *mqp, const struct utf8_string *topic, u16 msg_id)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.pub_append_data</code></td>
<td><code>(struct mqtt_packet *mqp, const u8 *data_buf, u32 data_len)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>mqp.proc_msg_id_ack_rx</code></td>
<td><code>(struct mqtt_packet *mqp_raw, bool has_payload)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>mqp.proc_pub_rx</code></td>
<td><code>(struct mqtt_packet *mqp_raw)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>mqp.ack_wlist_append</code></td>
<td><code>(struct mqtt_ack_wlist *list, struct mqtt_packet *elem)</code></td>
</tr>
<tr>
<td>struct</td>
<td><code>mqtt_packet</code></td>
<td><code>*</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>mqp.ack_wlist_remove</code></td>
<td><code>(struct mqtt_ack_wlist *list, u16 msg_id)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>mqp.ack_wlist_purge</code></td>
<td><code>(struct mqtt_ack_wlist *list)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.prep_fh</code></td>
<td><code>(struct mqtt_packet *mqp, u8 flags)</code></td>
</tr>
<tr>
<td>i32</td>
<td><code>mqp.recv</code></td>
<td><code>(i32 net, const struct device_net_services *net_ops, struct mqtt_packet *mqp, u32 wait_secs, bool *timed_out, void *ctx)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>qos2.pub_cq_reset</code></td>
<td><code>(struct pub_qos2_cq *cq)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>qos2.pub_cq.logup</code></td>
<td><code>(struct pub_qos2_cq *cq, u16 msg_id)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>qos2.pub_cq.unlog</code></td>
<td><code>(struct pub_qos2_cq *cq, u16 msg_id)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>bool  qos2_pub_cq_check</td>
<td>(struct *<em>pub_qos2_cq</em> cq, u16 msg_id)</td>
<td></td>
</tr>
<tr>
<td>void  cl_ctx_reset</td>
<td>(struct *<em>client_ctx</em> cl_ctx)</td>
<td></td>
</tr>
<tr>
<td>void  cl_ctx_timeout_insert</td>
<td>(struct <strong>client_ctx</strong> head, struct client_ctx *elem)</td>
<td></td>
</tr>
<tr>
<td>void  cl_ctx_remove</td>
<td>(struct <strong>client_ctx</strong> head, struct client_ctx *elem)</td>
<td></td>
</tr>
<tr>
<td>void  cl_ctx_timeout_update</td>
<td>(struct <strong>client_ctx</strong> cl_ctx, u32 now_secs)</td>
<td></td>
</tr>
</tbody>
</table>
**Detailed Description**

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

The applications are not expected to utilize 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(mqp))

#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

```c
enum mqtt_qos
```

<table>
<thead>
<tr>
<th>MQTT Quality of Service</th>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>MQTT_QOS0</code></td>
</tr>
<tr>
<td></td>
<td>QoS Level 0</td>
</tr>
<tr>
<td></td>
<td><code>MQTT_QOS1</code></td>
</tr>
<tr>
<td></td>
<td>QoS Level 1</td>
</tr>
<tr>
<td></td>
<td><code>MQTT_QOS2</code></td>
</tr>
<tr>
<td></td>
<td>QoS Level 2</td>
</tr>
</tbody>
</table>
Function Documentation

```c
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

```c
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

```c
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 parameters.

**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.

```c
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

```c
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

```c
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 composition of MQTT packet prior to its dispatch to the server.

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

```c
bool mqp_proc_msg_id_ack_rx (struct mqtt_packet * mqp_raw,
                              bool has_payload)
```

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

```c
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, otherwise false

```c
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 published data.

```c
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:**
### mqp_append

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] <code>mqp</code></td>
<td>packet buffer in which topic must be included.</td>
</tr>
<tr>
<td>[in] <code>topic</code></td>
<td>UTF8 information</td>
</tr>
<tr>
<td>[in] <code>msg_id</code></td>
<td>Message or Packet transaction ID</td>
</tr>
</tbody>
</table>

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

**Note:**
- A 'topic' must be appended prior to inclusion of published 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`
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
- mqp_buf_rd_remlen() : mqtt_common.h
- mqp_buf_tail_wr_remlen() : mqtt_common.h
- mqp_buf_wr_utf8() : mqtt_common.h
- mqp_client_alloc() : mqtt_client.h
- MQP_CONNACK_RC : mqtt_client.h
- MQP_CONNACK_SP : mqtt_client.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
MQTT_CFG_APP_HAS_RTSK : mqtt_client.h
MQTT_CFG_MK_GROUP_CTX : mqtt_client.h
MQTT_CFG_PROTOCOL_V31 : mqtt_client.h
mqtt_client_await_msg() : mqtt_client.h
mqtt_client_buffers_register() : mqtt_client.h
mqtt_client_ctx_await_msg() : mqtt_client.h
mqtt_client_ctx_create() : mqtt_client.h
mqtt_client_ctx_delete() : mqtt_client.h
mqtt_client_ctx_info_register() : mqtt_client.h
mqtt_client_ctx_run() : mqtt_client.h
mqtt_client_ctx_will_register() : mqtt_client.h
mqtt_client_is_connected() : mqtt_client.h
mqtt_client_lib_exit() : mqtt_client.h
mqtt_client_lib_init() : mqtt_client.h
mqtt_client_net_svc_register() : mqtt_client.h
mqtt_client_new_msg_id() : mqtt_client.h
mqtt_client_pub_dispatch() : mqtt_client.h
mqtt_client_pub_msg_send() : mqtt_client.h
mqtt_client_run() : mqtt_client.h
mqtt_client_send_progress() : mqtt_client.h
MQTT_CLIENT_VERSTR : mqtt_client.h
- oq -
  - QFL_VALUE : mqtt_common.h

- v -
  - VHB_CONNACK_RC : mqtt_client.h
  - VHB_CONNACK_SP : mqtt_client.h
**client_ctx Member List**

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

<table>
<thead>
<tr>
<th>Member</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>ip_length</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>ka_secs</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>net</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>next</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>remote_ip</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>timeout</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
<tr>
<td>usr</td>
<td>(defined in <code>client_ctx</code>)</td>
</tr>
</tbody>
</table>
D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h

Go to the documentation of this file.

00001 /************************************************************************************
* ********************************************************************************
00002 *
00003 * Copyright (C) 2014 Texas Instruments Incorporated
00004 *
00005 * All rights reserved. Property of Texas Instruments Incorporated.
00006 * Restricted rights to use, duplicate or disclose 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 against the terms and conditions
00011 * stipulated in the agreement under which this program has been supplied,
00012 * and under no circumstances can it be used with non-TI connectivity device.
00013 *
00014 **********************************************************************************
00015
This module outlines the interfaces that are common to both client and server components. The applications are not expected to utilize the services outlined in this module.

#ifndef __MQTT_COMMON_H__
define __MQTT_COMMON_H__

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

#define MQTT_COMMON_VERSTR "1.0.0"

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

#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) | flags)
#define MAKE_FH_FLAGS(bool_dup, enum_qos, bool_retain) (u8)(((bool_dup << 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) >> 1)
#define MSG_TYPE(fh_byte1) (u8)((fh_byte1 & 0xf0) >> 4)

static inline u32 buf_wr_nbytes(u8 *dst, const u8 *src, u32 n)
{
    u32 c = n;
    while(c--)
```
*dst++ = *src++;
return n;
}

static inline u32 buf_set(u8 *dst, u8 val, u32 n) {
    u32 c = n;
    while(c--)
        *dst++ = val;
    return n;
}

static inline u32 buf_wr_nbo_2B(u8 *buf, u16 val) {
    buf[0] = (u8)((val >> 8) & 0xFF); /* MSB */
    buf[1] = (u8)((val) & 0xFF); /* LSB */
    return 2;
}

static inline u32 buf_rd_nbo_2B(const u8 *buf, u16 *val) {
    *val = (u16)((buf[0] << 8) | (buf[1]));
    return 2;
}

struct mqtt_packet {
    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;

#define MQP_FHEADER_BUF(mqp) (mqp->buffer + mqp->offset)

#define MQP_VHEADER_BUF(mqp) (MQP_FHEADER_BUF(mqp) + mqp->fh_len)

#define MQP_PAYLOAD_BUF(mqp) (MQP_VHEADER_BUF(mqp) + mqp->vh_len)

#define MQP_CONTENT_LEN(mqp) (mqp->fh_len + mqp->vh_len + mqp->pl_len)

#define MQP_FREEBUF_LEN(mqp) (mqp->maxlen - MQP_CONTENT_LEN(mqp))

#define MQP_FHEADER_VAL(mqp) (mqp->fh_byte1)

#define MQP_FHEADER_MSG(mqp) (MSG_TYPE(MQP_FHEADER_VAL(mqp)))

#define MQP_FHEADER_FLG(mqp) (MSG_FLAGS(MQP_FHEADER_VAL(mqp)))
#define DEFINE_MQP_VEC(num_mqp, mqp_vec) 
  static struct mqtt_packet mqp_vec[num_mqp];

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

static u8 buf_vec[num_mqp][buf_len];

/*------------------------------------------
---------------------------
* Heleper MACROS for PUBLISH-RX Message Processing
---------------------------
*/

#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 WILL_RETAIN_VAL 0x20
#define WILL_CONFIG_VAL 0x04
#define CLEAN_START_VAL 0x02
#define USER_NAME_OPVAL 0x80
#define PASS_WORD_OPVAL 0x40
#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)

/*------------------------------------------
   * Common Operations
   *------------------------------------------
   */

void mqp_free(struct mqtt_packet *mqp);

void mqp_reset(struct mqtt_packet *mqp);

void mqp_init(struct mqtt_packet *mqp, u8 offset);

static inline void mqp_buffer_attach(struct mqtt_packet *mqp, u8 *buffer, u32 length, u8 offset) {
    mqp_init(mqp, offset);
    mqp->buffer = buffer;
    mqp->maxlen = length;
    mqp->free = NULL;
struct utf8_string {
    i8  *buffer;
    u16 length;
};

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);

/*
 Wait-List of MQTT Messages for which acknowledge is pending from remote node.
 */

struct mqtt_ack_wlist {
struct mqtt_packet *head; /* Points to head of single linked-list. */
struct mqtt_packet *tail; /* Points to tail of single linked-list. */
};

static inline bool mqp_ack_wlist_is_empty(struct mqtt_ack_wlist *list) {
    return (NULL == list->head) ? true : false;
}

/*
Add specified element into trailing end of list.
Returns, on success, true, otherwise false.
*/
bool mqp_ack_wlist_append(struct mqtt_ack_wlist *list,
    struct mqtt_packet *elem);

/*
Removes element that has specified msg_id from list.
Returns, on success, pointer to removed element, otherwise NULL.
*/
struct mqtt_packet *mqp_ack_wlist_remove(struct mqtt_ack_wlist *list,
    u16 msg_id);
Removes and frees all elements in list.

```c
void mqp_ack_wlist_purge(struct mqtt_ack_wlist *list);

static inline bool is_wlist_empty(const struct mqtt_ack_wlist *list) {
    return list->head? false : true;
}

i32 mqp_prep_fh(struct mqtt_packet *mqp, u8 flags);

textured, .

enum mqtt_qos {
    MQTT_QOS0,
    MQTT_QOS1,
    MQTT_QOS2
};

struct utf8_strqos {
    u8 *buffer;
    u16 length;
    enum mqtt_qos qosreq;
};

struct secure_conn {
    void *method;
    void *cipher;
    u32 n_file;
    char **files;
};
```
struct device_net_services {
#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

    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);

/* device_net_services */

/* Receive data from the specified network and read into the 'mqp' */
i32 mqp_recv(i32 net, const struct device_net_services *net_ops,
struct mqtt_packet *mqp, u32 wait_secs, bool *timed_out,
void *ctx);

#define MAX_PUBREL_INFLT 8 /* Must be kept as a value of 2^n */

struct pub_qos2_cq { /* Circular Queue CQ to track QoS2 PUB RX messages */
u16 id_vec[MAX_PUBREL_INFLT]; /* Vector to store RX Message-IDs */
u8 n_free; /* Num...
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 /* Reset the specified Circular Queue (CQ) */
00701
00702 void qos2_pub_cq_reset(struct pub_qos2_cq *cq);
00703
00704 /* Append the message-id into the CQ tail. Return true on success, else false */
00705 bool qos2_pub_cq_logup(struct pub_qos2_cq *cq, u16 msg_id);
00706
00707 /* Remove the message-id from the CQ head. Return true on success, else false */
00708 bool qos2_pub_cq_unlog(struct pub_qos2_cq *cq, u16 msg_id);
00709
00710 /* Is the message-id available in the CQ ? Return true on success, else false */
00711 bool qos2_pub_cq_check(struct pub_qos2_cq *cq, u16 msg_id);
00712
00713 /* Get the count of message-ID(s) available in the CQ */
00714 static inline i32 qos2_pub_cq_count(struct pub_qos2_cq *cq)
00715 {
00716    return MAX_PUBREL_INFLT - cq->n_free;
00717 }
00718
00719 struct client_ctx {
00720
void *usr; /* Client Usr */
i32 net; /* Socket HND */
i8 remote_ip[16];
u32 ip_length;
u32 timeout;
u16 ka_secs;
u32 ip_length;
u32 flags;
struct client_ctx *next;
};

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);
#define KA_TIMEOUT_NONE 0xffffffff /* Different than KA SECS = 0 */
void cl_ctx_timeout_update(struct client_ctx *cl_ctx, u32 now_secs);
# device_net_services Member List

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>close</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>io_mon</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>listen</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>open</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>recv</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>recv_from</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>send</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>send_dest</td>
<td><code>device_net_services</code></td>
</tr>
<tr>
<td>time</td>
<td><code>device_net_services</code></td>
</tr>
</tbody>
</table>

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4.
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"
- **tail** (defined in `mqtt_ack_wlist"

---

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by <a href="https://www.doxygen.org">doxygen</a> 1.7.4
Go to the documentation of this file.

```
00001
00002  /********************************************************************************
00003  ******************************************************************************
00004  *
00005  *
00006  *      Copyright (C) 2014 Texas Instruments Incorporated
00007  *
00008  *      All rights reserved. Property of Texas Instruments Incorporated.
00009  *
00010  *      Restricted rights to use, duplicate or disclose this code are
00011  *
00012  *      granted through contract.
00013  *
00014  *
00015  *      The program may not be used without the written permission of
00016  *
00017  *      Texas Instruments Incorporated or against the terms and conditions
00018  *
00019  *      stipulated in the agreement under which this program has been supplied,
00020  *
00021  *      and under no circumstances can it be used with non-TI connectivity device.
00022  *
00023  *
00024  ***************************************************************************/
```
This module enumerates the public interfaces / API of the MQTT Client Library.

#ifndef __MQTT_CLIENT_H__
#define __MQTT_CLIENT_H__
#include "mqtt_common.h"

#ifndef __MQTT_CLIENT_VERSTR
#define MQTT_CLIENT_VERSTR "1.0.0"
#endif

u16 mqtt_client_new_msg_id(void);

bool mqtt_client_is_connected(void *ctx);

i32 mqtt_connect_msg_send(void *ctx, bool clean_session, u16 ka_secs);

i32 mqtt_client_pub_msg_send(void *ctx, const struct utf8_string *topic,
                             const u8 *data_buf, u32 data_len,
                             enum mqtt_qos q...
os, bool retain);
00172
00211 i32 mqtt_client_pub_dispatch(void *ctx, struct mqtt_packet *mqp,
00212 enum mqtt_qos q
00213 os, bool retain);
00213
00243 i32 mqtt_sub_msg_send(void *ctx, const struct utf8_strqos *qos_topics, u32 count);
00244
00283 i32 mqtt_sub_dispatch(void *ctx, struct mqtt_packet *mqp);
00284
00315 i32 mqtt_unsub_msg_send(void *ctx, const struct utf8_string *topics, u32 count);
00316
00356 i32 mqtt_unsub_dispatch(void *ctx, struct mqtt_packet *mqp);
00357
00365 i32 mqtt_pingreq_send(void *ctx);
00366
00373 i32 mqtt_disconn_send(void *ctx);
00374
00375
00431 i32 mqtt_client_send_progress(void *ctx);
00432
00455 i32 mqtt_client_ctx_await_msg(void *ctx, u8 msg_type, struct mqtt_packet *mqp,
00456 u32 wait_secs) ;
00457
00462 static inline
00463 i32 mqtt_client_ctx_recv(void *ctx, struct mqtt_packet *mqp, u32 wait_secs)
00464 {
00465 /* Receive next and any MQTT Message
from the broker */
return mqtt_client_ctxAwaitMsg(ctx, 0x00, mqp, wait_secs);
}

mqtt_client_ctxRun(void *ctx, u32 wait_secs);

mqtt_clientAwaitMsg(struct mqtt_packet *mqp, u32 wait_secs, void **app);

mqtt_clientRun(u32 wait_secs);

/*------------------------------------------
------------------------------------
MQTT Client Library: Packet Buffer Pool
and its management
------------------------------------------
------------------------------------*/

mqtt_packet *mqp_client_alloc(u8 msg_type, u8 offset);

static inline struct mqtt_packet *mqp_client_send_alloc(u8 msg_type)
{
    return mqp_client_alloc(msg_type, MAX_FH_LEN);
}

static inline struct mqtt_packet *mqp_client_recv_alloc(u8 msg_type)
{
    return mqp_client_alloc(msg_type, 0);
}
mqtt_client_buffers_register(u32 num_mqp, struct mqtt_packet *mqp_vec, u32 buf_len, u8 *buf_vec);

/*------------------------------------------
------------------------------------
* MQTT Client Library: Register application, platform information and services.
*------------------------------------------
------------------------------------
*/

mqtt_client_ctx_info_register(void *ctx, const struct utf8_string *client_id, const struct utf8_string *user_name, const struct utf8_string *pass_word);

mqtt_client_ctx_will_register(void *ctx, const struct utf8_string *will_top, const struct utf8_string *will_msg, enum mqtt_qos will_qos, bool retain);

mqtt_client_net_svc_register(const struct device_net_services *net);

#define VHB_CONNACK_RC(vh_buf) (vh_buf[1])
#define MQP_CONNACK_RC(mqp) (mqp->buffer[3])
#define VHB_CONNACK_SP(vh_buf) (vh_buf[0] &
```c
#define MQP_CONNACK_SP(mqp)  (mqp->buffer[2] & 0x1)
#define VHB_CONNACK_VH16(vh_buf)((vh_buf[0] << 8) | vh_buf[1])
#define MQP_CONNACK_VH16(mqp)  ((mqp->buffer[2] << 8) | mqp->buffer[3])

struct mqtt_client_ctx_cbs {
    bool (*publish_rx)(void *app, bool dup, enum mqtt_qos qos, bool retain, struct mqtt_packet *mqp);
    void (*ack_notify)(void *app, u8 msg_type, u16 msg_id, u8 *buf, u32 len);
    void (*disconn_cb)(void *app, i32 ca use);
};

struct mqtt_client_ctx_cfg {
    #define MQTT_CFG_PROTOCOL_V31 0x0001
    #define MQTT_CFG_APP_HAS_RTSK 0x0002
    #define MQTT_CFG_MK_GROUP_CTX 0x0004
    u16 config_opts;
    #define MQTT_NETCONN_OPT_IP6 DEV_NETCONN_OPT_IP6
    #define MQTT_NETCONN_OPT_URL DEV_NETCONN_OPT_URL
    #define MQTT_NETCONN_OPT_SEC DEV_NETCONN_OPT_SEC
    u32 nwconn_opts;
    i8 *server_addr;
};
```
struct secure_conn *nw_security;

struct mqtt_client_lib_cfg {
    u16     loopback_port;
    bool    grp_uses_cfn;
    void    *mutex;
    void (*mutex_lockin)(void *mutex);
    void (*mutex_unlock)(void *mutex);
    i32     (*debug_printf)(const i8 *format, ...);
    bool    aux_debug_en;
};

mqtt_client_ctx_create(const struct mqtt_client_ctx_cfg *ctx_cfg,
                        const struct mqtt_client_ctx_cbs *ctx_cbs,
                        void *app, void *ctx);

mqtt_client_ctx_delete(void *ctx);

struct mqtt_client_lib_cfg {
    u16     port_number;
    struct secure_conn *nw_security;
};

mqtt_client_ctx_create(const struct mqtt_client_ctx_cfg *ctx_cfg,
                        const struct mqtt_client_ctx_cbs *ctx_cbs,
                        void *app, void *ctx);

mqtt_client_ctx_delete(void *ctx);

struct mqtt_client_lib_cfg {
    u16     loopback_port;
    bool    grp_uses_cfn;
    void    *mutex;
    void (*mutex_lockin)(void *mutex);
    void (*mutex_unlock)(void *mutex);
    i32     (*debug_printf)(const i8 *format,

mqtt_client_lib_init(const struct mqtt_client_lib_cfg *cfg);

mqtt_client_lib_exit(void);

/* End group client_api */
mqtt_client_ctx_cbs Member List

This is the complete list of members for `mqtt_client_ctx_cbs`, including all inherited members.

- `ack_notify` `mqtt_client_ctx_cbs`
- `disconn_cb` `mqtt_client_ctx_cbs`
- `publish_rx` `mqtt_client_ctx_cbs`
### mqtt_client_ctx_cfg Member List

This is the complete list of members for `mqtt_client_ctx_cfg`, including all inherited members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>config_opts</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>nw_security</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>nwconn_opts</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>port_number</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
<tr>
<td>server_addr</td>
<td>mqtt_client_ctx_cfg</td>
</tr>
</tbody>
</table>
### mqtt_client_lib_cfg Member List

This is the complete list of members for `mqtt_client_lib_cfg`, including all inherited members.

<table>
<thead>
<tr>
<th>aux_debug_en</th>
<th>mqtt_client_lib_cfg</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug_printf</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>grp_uses_cbfnn</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>loopback_port</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>mutex</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>mutex_lockin</td>
<td>mqtt_client_lib_cfg</td>
</tr>
<tr>
<td>mutex_unlock</td>
<td>mqtt_client_lib_cfg</td>
</tr>
</tbody>
</table>

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by **doxygen** 1.7.4
### mqtt_packet Member List

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

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>fh_byte1</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>fh_len</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>free</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>maxlen</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>msg_id</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>msg_type</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>n.refs</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>next</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>offset</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>pad</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>pl_len</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>private</td>
<td>mqtt_packet</td>
</tr>
<tr>
<td>vh_len</td>
<td>mqtt_packet</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipher</td>
<td>secure_conn</td>
</tr>
<tr>
<td>files</td>
<td>secure_conn</td>
</tr>
<tr>
<td>method</td>
<td>secure_conn</td>
</tr>
<tr>
<td>n_file</td>
<td>secure_conn</td>
</tr>
</tbody>
</table>
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_client_ctx_cbs
  - aux_debug_en : mqtt_client_lib_cfg

- b -

  - buffer : mqtt_packet, utf8_strqos, utf8_string

- c -

  - cipher : secure_conn
  - close : device_net_services
  - config_opts : mqtt_client_ctx_cfg

- d -

  - debug_printf : mqtt_client_lib_cfg
  - disconn_cb : mqtt_client_ctx_cbs

- f -

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

- **g** -

  - **grpUsesCbfn** : `mqtt_client_lib_cfg`

- **i** -

  - **io_mon** : `device_net_services`

- **l** -

  - **length** : `utf8_string`, `utf8_strqos`
  - **listen** : `device_net_services`
  - **loopback_port** : `mqtt_client_lib_cfg`

- **m** -

  - **maxlen** : `mqtt_packet`
  - **method** : `secure_conn`
  - **msg_id** : `mqtt_packet`
  - **msg_type** : `mqtt_packet`
  - **mutex** : `mqtt_client_lib_cfg`
  - **mutex_lockin** : `mqtt_client_lib_cfg`
  - **mutex_unlock** : `mqtt_client_lib_cfg`

- **n** -

  - **n_file** : `secure_conn`
  - **n_refs** : `mqtt_packet`
  - **nw_security** : `mqtt_client_ctx_cfg`
  - **nwconn_opts** : `mqtt_client_ctx_cfg`

- **o** -

  - **offset** : `mqtt_packet`
  - **open** : `device_net_services`

- **p** -
- pl_len : mqtt_packet
- port_number : mqtt_client_ctx_cfg
- publish_rx : mqtt_client_ctx_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
- server_addr : mqtt_client_ctx_cfg

- t -
- time : device_net_services

- v -
- vh_len : mqtt_packet
# Files

## File List

<table>
<thead>
<tr>
<th>File Members</th>
<th>client_notes.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td></td>
</tr>
</tbody>
</table>

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by [doxygen](http://www.stackoverflow.com) 1.7.4
# Files

<table>
<thead>
<tr>
<th>File List</th>
<th>File Members</th>
</tr>
</thead>
</table>

## MainPage.h

00001

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by [doxygen](http://doxygen.tin.it) 1.7.4
### Files

<table>
<thead>
<tr>
<th>File List</th>
<th>File Members</th>
</tr>
</thead>
</table>

| All | Functions | Enumerations | Enumerator | Defines |

- **m** -

- `mqp_buf_rd_remlen()` : `mqtt_common.h`
- `mqp_buf_tail_wr_remlen()` : `mqtt_common.h`
- `mqp_buf_wr_utf8()` : `mqtt_common.h`
- `mqp_client_alloc()` : `mqtt_client.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_client_await_msg()` : `mqtt_client.h`
- `mqtt_client_buffers_register()` : `mqtt_client.h`
- `mqtt_client_ctx_await_msg()` : `mqtt_client.h`
- `mqtt_client_ctx_create()` : `mqtt_client.h`
- `mqtt_client_ctx_delete()` : `mqtt_client.h`
- `mqtt_client_ctx_info_register()` : `mqtt_client.h`
- `mqtt_client_ctx_run()` : `mqtt_client.h`
- `mqtt_client_ctx_will_register()` : `mqtt_client.h`
- `mqtt_client_is_connected()` : `mqtt_client.h`
- `mqtt_client_lib_exit()` : `mqtt_client.h`
- `mqtt_client_lib_init()` : `mqtt_client.h`
- `mqtt_client_net_svc_register()` : `mqtt_client.h`
- `mqtt_client_new_msg_id()` : `mqtt_client.h`
- `mqtt_client_pub_dispatch()` : `mqtt_client.h`
- `mqtt_client_pub_msg_send()` : `mqtt_client.h`
- `mqtt_client_run()` : `mqtt_client.h`
- `mqtt_client_send_progress()` : `mqtt_client.h`
- `mqtt_connect_msg_send()` : `mqtt_client.h`
- `mqtt_disconn_send()` : `mqtt_client.h`
- `mqtt_pingreq_send()` : `mqtt_client.h`
- `mqtt_sub_dispatch()` : `mqtt_client.h`
- `mqtt_sub_msg_send()` : `mqtt_client.h`
- `mqtt_unsub_dispatch()` : `mqtt_client.h`
- `mqtt_unsub_msg_send()` : `mqtt_client.h`
- `mqtt_qos`: `mqtt_common.h`
<table>
<thead>
<tr>
<th>File List</th>
<th>File Members</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All</strong></td>
<td><strong>Functions</strong></td>
</tr>
<tr>
<td><strong>Enumerations</strong></td>
<td><strong>Enumerators</strong></td>
</tr>
<tr>
<td><strong>Defines</strong></td>
<td></td>
</tr>
</tbody>
</table>

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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by **doxygen** 1.7.4
### Defines

- 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`

- MAX_FH_LEN : `mqtt_common.h`
- MAX_REMLEN_BYTES : `mqtt_common.h`
- MQP_CONNACK_RC : `mqtt_client.h`
- MQP_CONNACK_SP : `mqtt_client.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`
- q -

- v -

- q -

- v -

- q -

- v -

- q -

- v -

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by doxygen 1.7.4