

Related Pages Modules Main Page Classes **Files Related Pages**

Here is a list of all related documentation pages:

• MQTT Client Library - App Notes

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XXYQ@n 1.7.4





Main Page

Related Pages

Modules

Classes

Files

MQTT Client Library - App Notes

Overview

This narrative outlines the usage / working apects 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 recepient. In other words, the message from the sender may or may not get delivered to the receipient 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 acknwoledged (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 iternation 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 ongoing 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 acknowleged by the remote entity. For transmission, the number of inflight 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 appliaction 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 compeltion 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 adequeate length.



Main Page Related Pages Modules Classes
Files
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

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



Main P	age	Related Pages	Modules	Classes	
Files					
MQTT Packet (MQP) Buffer structure			Clas	sses	

Classes

struct mqtt_packet

Detailed Description

The core construct to encapsulate, construct and process a message

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



Main P	age	Related Pages	Modules	Classes	
Files					
Helper Macros for RX PUBLISH			Def	fines	

Defines

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

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

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

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

Define Documentation

```
#define MQP_PUB_PAY_BUF ( mqp) (mqp->pl_len? MQP_PAYLC)

Yields pointer to payload data

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

Length or size of payload data

#define MQP_PUB_TOP_BUF ( mqp) (MQP_VHEADER_BUF(mq)

Yields pointer to topic content

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

Length or size of topic content
```

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



Main Page Related Pages Modules Classes

Files

LIBRARY Generated

Error Codes

Defines

```
#define MQP_ERR_NETWORK (-1)

#define MQP_ERR_TIMEOUT (-2)

#define MQP_ERR_NET_OPS (-3)

#define MQP_ERR_FNPARAM (-4)

#define MQP_ERR_PKT_AVL (-5)

#define MQP_ERR_PKT_LEN (-6)

#define MQP_ERR_NOTCONN (-7)

#define MQP_ERR_BADCALL (-8)

#define MQP_ERR_CONTENT (-9)

#define MQP_ERR_LIBQUIT (-10)

#define MQP_ERR_NOT_DEF (-32)
```

Detailed Description

Library provides these codes as return values in several routines

Define Documentation

#define MQP_ERR_BADCALL (-8)

Irrelevant call for LIB state

#define MQP_ERR_CONTENT (-9)

MSG / Data content has errors

#define MQP_ERR_FNPARAM (-4)

Invalid parameter(s) provided

#define MQP_ERR_LIBQUIT (-10)

Needs reboot library has quit

#define MQP_ERR_NET_OPS (-3)

Platform Net Ops un-available

#define MQP_ERR_NETWORK (-1)

Problem in network (sock err)

#define MQP_ERR_NOT_DEF (-32)

Value other than defined ones

#define MQP_ERR_NOTCONN (-7)

Lib isn't CONNECTED to server

#define MQP_ERR_PKT_AVL (-5)

No pkts are available in pool

#define MQP_ERR_PKT_LEN (-6)

Inadequate free buffer in pkt

#define MQP_ERR_TIMEOUT (-2)

Net transaction has timed out

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XYQ@M 1.7.4



Main Page Related Pages Modules Classes
Files

Classes

Information to establish a secure connection.

Classes

struct secure_conn

Detailed Description

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

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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @OXYQ@n 1.7.4





Main P	age	Related Pages	Modules	Classes	
Files					
Abstraction of Network Services on a platform			Clas	sses	

Classes

struct device_net_services

Detailed Description

Services to enable the MQTT Client-Server communication over network

These services are invoked by the MQTT Library.

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



Main Page	Related Pages	Modules	Classes	
Files				
Options for platform to configure network			Def	ines

Defines

```
#define DEV_NETCONN_OPT_TCP 0x01

#define DEV_NETCONN_OPT_UDP 0x02

#define DEV_NETCONN_OPT_IP6 0x04

#define DEV_NETCONN_OPT_URL 0x08

#define DEV_NETCONN_OPT_SEC 0x10
```

Define Documentation

#define DEV_NETCONN_OPT_IP6 0x04

Assert for IPv6, otherwise it is IPv4

#define DEV_NETCONN_OPT_SEC 0x10

Assert to indicate a secure connection

#define DEV_NETCONN_OPT_TCP 0x01

Assert to indicate TCP net connection

#define DEV_NETCONN_OPT_UDP 0x02

Assert to create a local UDP port bind

#define DEV_NETCONN_OPT_URL 0x08

Assert if the network address is a URL

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



Main P	age	Related Pages	Modules	Classes
Files				
The Client Library API(s)		Classes Defines Functions		

Classes

```
struct mqtt_client_ctx_cbs
struct mqtt_client_ctx_cfg
struct mqtt_client_lib_cfg
```

Defines

```
#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

u16	mqtt_client_new_msg_id (void)
bool	mqtt_client_is_connected (void *ctx)
i32	<pre>mqtt_connect_msg_send (void *ctx, bool clean_session, u16 ka_secs)</pre>
i32	<pre>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)</pre>
i32	<pre>mqtt_client_pub_dispatch (void *ctx, struct mqtt_packet *mqp, enum mqtt_qos qos, bool retain)</pre>
i32	<pre>mqtt_sub_msg_send (void *ctx, const struct utf8_strqos *qos_topics, u32 count)</pre>
i32	<pre>mqtt_sub_dispatch (void *ctx, struct mqtt_packet *mqp)</pre>
i32	<pre>mqtt_unsub_msg_send (void *ctx, const struct utf8_string *topics, u32 count)</pre>
i32	<pre>mqtt_unsub_dispatch (void *ctx, struct mqtt_packet *mqp)</pre>
i32	mqtt_pingreq_send (void *ctx)
i32	mqtt_disconn_send (void *ctx)
i32	mqtt_client_send_progress (void *ctx)
i32	<pre>mqtt_client_ctx_await_msg (void *ctx, u8 msg_type, struct mqtt_packet *mqp, u32 wait_secs)</pre>
i32	<pre>mqtt_client_ctx_run (void *ctx, u32 wait_secs)</pre>
i32	<pre>mqtt_client_await_msg (struct mqtt_packet *mqp, u32 wait_secs, void **app)</pre>
i32	mqtt_client_run (u32 wait_secs)
struct mqtt_packet *	mqp_client_alloc (u8 msg_type, u8 offset)
i32	<pre>mqtt_client_buffers_register (u32 num_mqp, struct mqtt_packet *mqp_vec, u32 buf_len, u8 *buf_vec)</pre>
	<pre>mqtt_client_ctx_info_register (void *ctx, const struct utf8_string *client_id, const struct</pre>

i32	<pre>utf8_string *user_name, const struct utf8_string *pass_word)</pre>
i32	<pre>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)</pre>
i32	<pre>mqtt_client_net_svc_register (const struct device_net_services *net)</pre>
i32	<pre>mqtt_client_ctx_create (const struct mqtt_client_ctx_cfg *ctx_cfg, const struct mqtt_client_ctx_cbs *ctx_cbs, void *app, void **ctx)</pre>
i32	mqtt_client_ctx_delete (void *ctx)
i32	mqtt_client_lib_init (const struct mqtt_client_lib_cfg *cfg)
i32	mqtt_client_lib_exit (void)

Define Documentation

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

CONNACK MQP:: Return Code

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

CONNACK MQP:: \ Session Bit

#define MQTT_CLIENT_VERSTR "1.0.0"

Version of Client LIB

#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

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

CONNACK VH:: Session Bit

Function Documentation

```
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

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.

```
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] mqp_vec An array of MQTT Packet Holder without a

buffer.

[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 peristent 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 * ctx_void * apr
```

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 refere 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_i const struct utf8 string * pass

Register application info and its credentials with the client library. This routine registers information for all the specificed parameters, therefore, an upate to single element would imply re-specification of the other paramters, 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:

mgtt 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.

i32 mgtt client ctx run (void * ctx,

```
u32 wait_secs
```

Run the context for the specificed 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 enable 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)

```
i32 mqtt_client_ctx_will_register ( void * ctx, const struct utf8_string * will_to const struct utf8 string * will m
```

```
enum mqtt_qos will_qc
bool retain
```

Register WILL information of the client application. This routine registers information for all the specificed parameters, therefore, an update to single element would imply re-specification of the other paramters, 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.

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:

true if connection is active otherwise false.

i32 mqtt_client_lib_exit (void)

Exit the MQTT client library.

Returns:

0 on success otherwise -1.

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 connetions. 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 connetions 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_service:

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

dropped.

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:

mgtt 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)

```
i32 mqtt_client_pub_msg_send ( void * ctx, const struct utf8_string * topic, const u8 * data_but u32 data_levenum 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:

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

See also:

mqtt_client_ctx_create

Parameters:

[in] topic UTF8 based Topic Name for which data is

being published.

[in] data_buf The binary data that is being published for the

topic.

[in] data len The length of the binary data.

[in] **qos** quality of service of the message

[in] **retain** should the server retain the message.

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 consercutive 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 backoff 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:

```
mgtt client ctx create
```

Returns:

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

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 internaly 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:

mgtt 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)

i32 mgtt 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)

i32 mqtt_pingreq_send (void * ctx)

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

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)

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)

```
const struct utf8_strqos * qos_topics, u32 count
)
```

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)

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:

mgtt 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)

```
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:

mgtt 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)



MQTT Client 1.0.0

Main Page		Related Pages	Modules	Classes	
Files					
Options for application to config CTX					

Defines

```
#define MQTT_CFG_PROTOCOL_V31 0x0001
#define MQTT_CFG_APP_HAS_RTSK 0x0002
#define MQTT_CFG_MK_GROUP_CTX 0x0004
```

Define Documentation

#define MQTT_CFG_APP_HAS_RTSK 0x0002

Assert if APP has dedicated RX Task

#define MQTT_CFG_MK_GROUP_CTX 0x0004

Assert if task manages > 1 contexts

#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 @@XYQ@n 1.7.4



MQTT Client 1.0.0

Main Page Related Pages Modules Classes

Files

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 DEV_NETCONN_OPT_IP6

Options for platform to configure network

Options for platform to configure network

#define MQTT_NETCONN_OPT_URL DEV_NETCONN_OPT_URL

Options for platform to configure network

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @OXXV@@in 1.7.4



MQTT Client 1.0.0

Main Pa	ge Relat	ed Pages	Modules	Classes		
Files						
Class List	Class Index	Class Membe	rs			
Class List						

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

acscriptions.	
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	



MQTT Client 1.0.0

Main Page F		Relate	Related Pages		Modules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
client_ctx Struct Reference				Public Attrib	utes		

List of all members.

Public Attributes

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

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

• D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_commc

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



MQTT Client 1.0.0

Main Page Relate		ed Pages	N	Modules	Classes		
Files							
Class List	Cla	ss Index	Class Membe	rs			
device_net_services Struct Reference Abstraction of Network Services on a platform						Public Attribut	es

List of all members.

```
i32(* open )(u32 nwconn_opts, const i8 *server_addr, u16
      port number, const struct secure_conn *nw_security)
      send )(i32 comm, const u8 *buf, u32 len, void *ctx)
i32(*
      recv )(i32 comm, u8 *buf, u32 len, u32 wait secs, bool
i32(*
      *err timeo, void *ctx)
      send_dest )(i32 comm, const u8 *buf, u32 len, u16 dest port,
i32(*
      const i8 *dest ip, u32 ip len)
      recv from )(i32 comm, u8 *buf, u32 len, u16 *from port, i8
i32(*
      *from_ip, u32 *ip_len)
      close )(i32 comm)
i32(*
      listen )(u32 nwconn opts, u16 port number, const struct
i32(*
      secure_conn *nw security)
      accept )(i32 listen, i8 *client ip, u32 *ip length)
i32(*
      io mon )(i32 *recv cvec, i32 *send cvec, i32 *rsvd cvec, u32
i32(*
      wait secs)
u32(* time )(void)
```

Member Data Documentation

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

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

Parameters:

[in] **listen** handle to listen for the incoming

connection requests from the remote

clients

[out] **client** ip IP address of the connected client. This

value is valid only on successful return of the routine. The place holder must provide memory to store atleast 16

bytes.

[in, out] ip_length Length of IP address. It is provided by

the caller to declare the length of the place holder and updated by routine to indicate the length of the connected

client's IP address.

Returns:

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

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

Close communication connection

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

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

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

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

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

Parameters:

Returns:

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

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

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

Parameters:

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

enumerate server address and / or

connection related details

[in] port_number Network port number, typically, 1883 or

8883

[in] **nw_security** Information to establish a secure

connection with client. Set it to NULL, if not used. **Information to establish a**

secure connection.

Returns:

a valid handle to listening contruct, otherwise NULL

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

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

Parameters:

[in] nwconn_opts Implementation specific construct to

enumerate server address and / or

connection related details

[in] server_addr URL or IP address (string) or other

server reference. For setting up a local

(UDP) port, set it to NULL.

[in] port_number Network port number, typically, 1883 or

8883 for remote severs. For setting up a local (UDP) port, use an intended port

number.

[in] nw_security Information to establish a secure

connection with server. Set it to NULL, if

not used. Information to establish a

secure connection.

Returns:

a valid handle to connection, otherwise NULL

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

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 MOTT connection context

Returns:

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

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

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

Parameters:

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

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

indicate the length of the remote

network entity's IP address.

Returns:

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

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

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

Parameters:

[in] **comm** handle to network connection as returned by **open()**.

[in] **buf** refers to the data that is intended to be sent

[in] len length of the data

[in] ctx reference to the MQTT connection context

Returns:

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

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

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

Parameters:

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

[in] ip len length of the destination IP address.

Returns:

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

u32(* device_net_services::time)(void)

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

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

It is the 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_commc

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





Main Pa	Page Related Pages		ed Pages	N	Modules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
mqtt_a Refere		wlist S	Struct			Public Attrib	utes

```
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_commc

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XXY@@M 1.7.4



Main Pa	Page Related Pages		N	Modules	Classes		
Files							
Class List	Cla	ss Index	Class Membe	rs			
mqtt_c Struct I	Refe	erence	:			Public Attribute	S

#include <mqtt_client.h>

```
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 cause)
```

Detailed Description

Callbacks to be invoked by MQTT Client library onto Client application

Member Data Documentation

void(* mqtt_client_ctx_cbs::ack_notify)(void *app, u8 msg_type, u1

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 messsage

[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.

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:

[in] app application whose connection got terminated

See also:

mqtt_client_ctx_create

Parameters:

[in] cause reason that created disconnection (LIBRARY Generated Error Codes)

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 correponding 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 owernship 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] qos quality of Service of the PUBLISH message

[in] retain Asserted to indicate message at new

subscription

[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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XYQ@n 1.7.4



Main Pa	Page Related Pages			N	Modules	Classes	
Files							
Class List	Cla	ss Index					
mqtt_cl Struct I The Clien	Ref€	erence	;			Public Attributes	3

u16	config_opts
u32	nwconn_opts
i8 *	server_addr
u16	port_number
struct secure_conn *	nw_security

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



Main Pa	Page Related Pages		N	Modules	Classes		
Files							
Class List	Cla	ss Index	Class Membe	rs			
mqtt_c Struct I	Refe	erence	;			Public Attributes)

#include <mqtt_client.h>

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

Detailed Description

Contruct / 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_cbfn

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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @OXXYQ@n 1.7.4





Main Pa	ge	Related Pages		١	Modules	Classes
Files						
Class List	Cla	ss Index	Class Membe	rs		
mqtt_pa	nce		uct uffer structu	re		Public Attributes

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

Member Data Documentation

u8* mqtt_packet::buffer
The attached buffer
u8 mqtt_packet::fh_byte1
Fixed Header: Byte1
u8 mqtt_packet::fh_len
Fix Header Length
void(* mqtt_packet::free)(struct mqtt_packet *mqp)
Method to free this packet to a particular pool
u32 mqtt_packet::maxlen
Maximum buffer size
u16 mqtt_packet::msg_id
Msg transaction ID
u8 mqtt_packet::msg_type
MQTT Message Type

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

• D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_commc

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



Main Page		Related Pages			lodules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
pub_qos2_cq Struct Reference						Public Attrib	utes

u16	id_vec [MAX_PUBREL_INFLT]
u8	n_free
u8	rd_idx
u8	wr_idx

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

• D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_commc

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



Main Page		Related Pages		N	Modules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
secure_conn Struct Reference Information to establish a secure connection.						Public Attrib	utes

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

Member Data Documentation

void* secure_conn::cipher

Reference to information about cryptograph ciphers

char** secure_conn::files

Reference to array of file-names used for security

void* secure conn::method

Reference to information about protocol or methods

u32 secure_conn::n_file

Count of secure connection related files, certs...

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

• D:/my data/GIT/network apps/netapps/mqtt/common/mqtt_commo

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



Main Page		Related Pages			lodules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
utf8_string Struct Reference						Public Attrib	utes

#include <mqtt_common.h>

i8 * buffer u16 length

Detailed Description

Description of UTF8 information as used by MQTT Library.

Member Data Documentation



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

• D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_commc

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XXYG@M 1.7.4



MQTT Client 1.0.0

Main Pa	age	Related Pages			lodules	Classes	
Files							
Class List	Cla	ss Index	Class Members				
utf8_st Refere	-	s Stru	ct			Public Attrib	utes

#include <mqtt_common.h>

List of all members.

Public Attributes

i8 *	buffer
u16	length
enum mqtt_qos	qosreq

Detailed Description

Construct to create Topic to SUBSCRIBE

Member Data Documentation



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

• D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_commc

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XYQ@M 1.7.4



MQTT Client 1.0.0

Main Page Rela		Relat	ed Pages	d Pages Modu		Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
Class Ir	nde	X					
			C D M F	P S U			
C			M		mqtt	_client_lib_cf	fg p
client_ctx			mqtt_ack_	wlist	n	nqtt_packet	
D		I	mqtt_client_c	ctx_cbs	P		
device_net	t_ser	vices	mqtt_client_	ctx_cfg			
			C D M F	P S U			

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



MQTT Client 1.0.0

Main Page				Related Pages					M	od	ule	es	Classes					
F	Files																	
Class List Clas				class	s Index Class Members													
All Variables				S														
a	b	С	d	f	g	i	I	m	n	0	р	q	r	S	t	V		

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_packetfiles: secure_connfree: mqtt_packet
- g -
 - grp_uses_cbfn : mqtt_client_lib_cfg
- i -
 - io mon: device_net_services
- 1 -
 - 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
- **0 -**
 - 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_servicesrecv from : device_net_services
- S
 - send : device_net_servicessend_dest : device_net_servicesserver_addr : mqtt_client_ctx_cfg
- time : device_net_services
- vh_len : mqtt_packet

- t -

- V -



MQTT Client 1.0.0

Main Page
Related Pages
Modules
Classes

Files
File List
File Members

File List

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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @OXXVQ@n 1.7.4



MQTT Client 1.0.0

Main Page
Related Pages
Modules
Classes

Files

File List
File Members

Classes | Defines | Functions

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

#include "mqtt_common.h"

Go to the source code of this file.

Classes

```
struct mqtt_client_ctx_cbs
struct mqtt_client_ctx_cfg
struct mqtt_client_lib_cfg
```

Defines

```
#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)
       VHB CONNACK VH16(vh buf) ((vh buf[0] << 8) |
#define
       vh buf[1])
       MQP_CONNACK_VH16(mqp) ((mqp->buffer[2] << 8) | mqp-
#define
       >buffer[3])
#define MQTT CFG PROTOCOL V31 0x0001
#define MQTT CFG APP HAS RTSK 0x0002
#define MQTT CFG MK GROUP CTX 0x0004
#define MQTT NETCONN OPT IP6 DEV NETCONN OPT IP6
#define MQTT NETCONN OPT URL DEV NETCONN OPT URL
#define MOTT NETCONN OPT SEC DEV NETCONN OPT SEC
```

Functions

u16	mqtt_client_new_msg_id (void)
bool	mqtt_client_is_connected (void *ctx)
i32	<pre>mqtt_connect_msg_send (void *ctx, bool clean_session, u16 ka_secs)</pre>
i32	<pre>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)</pre>
i32	<pre>mqtt_client_pub_dispatch (void *ctx, struct mqtt_packet *mqp, enum mqtt_qos qos, bool retain)</pre>
i32	<pre>mqtt_sub_msg_send (void *ctx, const struct utf8_strqos *qos_topics, u32 count)</pre>
i32	<pre>mqtt_sub_dispatch (void *ctx, struct mqtt_packet *mqp)</pre>
i32	<pre>mqtt_unsub_msg_send (void *ctx, const struct utf8_string *topics, u32 count)</pre>
i32	<pre>mqtt_unsub_dispatch (void *ctx, struct mqtt_packet *mqp)</pre>
i32	mqtt_pingreq_send (void *ctx)
i32	mqtt_disconn_send (void *ctx)
i32	mqtt_client_send_progress (void *ctx)
i32	<pre>mqtt_client_ctx_await_msg (void *ctx, u8 msg_type, struct mqtt_packet *mqp, u32 wait_secs)</pre>
i32	<pre>mqtt_client_ctx_run (void *ctx, u32 wait_secs)</pre>
i32	<pre>mqtt_client_await_msg (struct mqtt_packet *mqp, u32 wait_secs, void **app)</pre>
i32	mqtt_client_run (u32 wait_secs)
struct mqtt_packet *	mqp_client_alloc (u8 msg_type, u8 offset)
i32	<pre>mqtt_client_buffers_register (u32 num_mqp, struct mqtt_packet *mqp_vec, u32 buf_len, u8 *buf_vec)</pre>
	<pre>mqtt_client_ctx_info_register (void *ctx, const struct utf8_string *client_id, const struct</pre>

i32	<pre>utf8_string *user_name, const struct utf8_string *pass_word)</pre>
i32	<pre>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)</pre>
i32	<pre>mqtt_client_net_svc_register (const struct device_net_services *net)</pre>
i32	<pre>mqtt_client_ctx_create (const struct mqtt_client_ctx_cfg *ctx_cfg, const struct mqtt_client_ctx_cbs *ctx_cbs, void *app, void **ctx)</pre>
i32	mqtt_client_ctx_delete (void *ctx)
i32	mqtt_client_lib_init (const struct mqtt_client_lib_cfg *cfg)
i32	mqtt_client_lib_exit (void)

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 unrestrictive 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

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



MQTT Client 1.0.0

Main Page		Relat	ed Pages	Modules	Classes
Files					
File List	File N	Members			
				Enume	efines Typedefs rations Functions
D·/my	data	a <i>lC</i> IT/i	network	anns/netai	nns/matt/con

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

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

Go to the source code of this file.

Classes

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

Defines

```
#define MQTT_COMMON_VERSTR "1.0.0"
#define MIN(a, b) ((a > b)? b : a)
#define MOTT CONNECT 0x01
#define MOTT 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 MOTT PINGREO 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)
       MAKE FH FLAGS(bool dup, enum gos, bool retain) (u8)((
#define
       3) | (enum gos << 1) | bool retain) & 0xF)
#define QOS_VALUE(enum gos) (u8)(enum gos & 0x3)
#define OFL VALUE 0x80
#define DUP_FLAG_VAL(bool val) (u8)(bool val << 3)
#define BOOL_RETAIN(fh byte1) ((fh byte1 & 0x1)? true : false)
#define BOOL_DUP(fh byte1) ((fh byte1 & 0x8)? true : false)
#define ENUM_QOS(fh byte1) (enum mqtt_qos)((fh byte1 & 0x6) >:
#define MSG_TYPE(fh_byte1) (u8)((fh_byte1 & 0xf0) >> 4)
#define MQP FHEADER BUF(mgp) (mgp->buffer + mgp->offset)
       MQP_VHEADER_BUF(mqp) (MQP FHEADER BUF(mqp) +
#define
       >fh_len)
       MQP_PAYLOAD_BUF(mqp) (MQP VHEADER BUF(mqp) +
#define
       >vh len)
```

```
#define MQP_CONTENT_LEN(mqp)
                               (mqp->fh len + mqp->vh len +
#define MQP_FREEBUF_LEN(mqp)
#define MQP FHEADER VAL(mqp)
                               (mqp->fh byte1)
#define MQP_FHEADER_MSG(mqp) (MSG_TYPE(MQP_FHEADER
#define MQP_FHEADER_FLG(mqp)
                               (MSG FLAGS(MQP FHEADEF
       DEFINE MQP VEC(num mqp, mqp vec) static struct mqtt
#define
       mqp vec[num mqp];
#define DEFINE_MQP_BUF_VEC(num mqp, mqp vec, buf len, buf v
#define MQP_PUB_TOP_BUF(mqp)
                               (MQP VHEADER BUF(mqp) +
#define MQP_PUB_TOP_LEN(mqp)
                              (mqp->vh len - 2 - (mqp->msg
       MQP PUB PAY BUF(mqp)
                              (mqp->pl len? MQP PAYLOAD
#define
       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)
                          (-4)
#define MQP ERR FNPARAM
#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
                               80x0
#define DEV NETCONN OPT SEC
                               0x10
#define MAX PUBREL INFLT 8
#define KA TIMEOUT NONE 0xffffffff
```

Typedefs

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

Enumerations

enum mqtt_qos { MQTT_QOS0, MQTT_QOS1, MQTT_QOS2 }

Functions

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

bool	<pre>qos2_pub_cq_check (struct pub_qos2_cq *cq, u16 msg_id)</pre>
void	<pre>cl_ctx_reset (struct client_ctx *cl_ctx)</pre>
void	<pre>cl_ctx_timeout_insert (struct client_ctx **head, struct client_ctx *elem)</pre>
void	<pre>cl_ctx_remove (struct client_ctx **head, structiont_ctx *elem)</pre>
	<pre>cl_ctx_timeout_update (struct client_ctx *cl ctx, u32 now secs)</pre>

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(mq
p))
#define MQTT_COMMON_VERSTR "1.0.0"
Version of Common LIB
```

#define MQTT_CONNECT 0x01

MQTT Message Types

#define QFL_VALUE 0x80

QOS Failure value (SUBACK)

Enumeration Type Documentation

enum mqtt_qos

MQTT Quality of Service

Enumerator:

MQTT_QOS0 QoS Level 0

MQTT_QOS1 QoS Level 1

MQTT_QOS2 QoS Level 2

Function Documentation

```
i32 mqp_buf_rd_remlen ( u8 * buf, u32 * remlen )
```

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

Parameters:

[in] **buf** refers to memory to head-read 'Remaining

Length' from

[in] remlen place-holder for The 'Remaining Length' value

Returns:

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

```
i32 mqp_buf_tail_wr_remlen ( u8 * buf, u32 remlen )
```

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

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

Parameters:

[in] **buf** refers to memory to tail-write 'Remaining

Length' into

[in] remlen The 'Remaining Length' value

Returns:

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

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

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

Warning:

The routine does not check for correctness of the paramters.

Parameters:

[in] **buf** refers to memory to write UTF8 information into

[in] utf8 contains UTF8 information to be written

Returns:

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

void mqp_free (struct mqtt_packet * mqp)

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

Parameters:

[in] mqp packet buffer to be freed

Returns:

none

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

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

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

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

```
bool mqp_proc_msg_id_ack_rx ( struct mqtt_packet * mqp_raw, bool has_payloac )
```

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

Warning:

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

Parameters:

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

Returns:

on success, true, otherwise false

bool mqp_proc_pub_rx (struct mqtt_packet * mqp_raw)

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

Warning:

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

Parameters:

[in] mqp_raw holds a raw buffer from the network

Returns:

on success, true, other wise false

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

Include payload data for publishing The payload data is associated

with a topic.

Warning:

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

Parameters:

```
[in] mqp packet buffer in which payload data must be included.
```

[in] data_buf data to be included in the packet buffer
[in] data_len length of the data to be included in the packet.

Returns:

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

Note:

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

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

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

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

Warning:

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

Parameters:

[in] mqp packet buffer in which topic must be included.

[in] **topic** UTF8 information

[in] msg_id Message or Packet transaction ID

Returns:

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

Note:

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

void mqp_reset (struct mqtt_packet * mqp)

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

Parameters:

[in] mqp packet buffer to be reset

Returns:

none

See also:

mqp_init

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



MQTT Client 1.0.0

Maii	n P	age	Related Pages			Modules CI			asses	
Files										
File Li	ist	File Me	mbers							
All	Functions En		Enumerations Enu		Enu	umerator Define		nes		
d m	q	v								

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
- mgp buf rd remlen(): mgtt_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
- MOP ERR BADCALL: mgtt common.h
- MQP ERR CONTENT: mqtt_common.h
- MOP ERR FNPARAM: mgtt 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
- mgtt client await msg(): mgtt_client.h
- mqtt_client_buffers_register(): mqtt_client.h
- mqtt_client_ctx_await_msg(): mqtt_client.h
- mgtt client ctx create(): mgtt_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

- MQTT_COMMON_VERSTR : mqtt_common.h
- MQTT CONNECT: mqtt_common.h
- mqtt_connect_msg_send(): mqtt_client.h
- mqtt_disconn_send(): mqtt_client.h
- MQTT_NETCONN_OPT_IP6: mqtt_client.h
- MQTT_NETCONN_OPT_SEC: mqtt_client.h
- MQTT NETCONN OPT URL: mqtt_client.h
- mqtt_pingreq_send(): mqtt_client.h
- mqtt_qos : mqtt_common.h
- MQTT QOS0: mqtt_common.h
- MQTT QOS1: mqtt_common.h
- MQTT QOS2: mqtt_common.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

- q -

• QFL_VALUE : mqtt_common.h

- V -

- VHB_CONNACK_RC: mqtt_client.h
- VHB_CONNACK_SP : mqtt_client.h



Main Pa	Main Page Relate		Modules	Classes
Files				
Class List	Class Index	Class Membe	rs	
al!a.a.k a	· · 4 · · · · · · · · · · · · · · · · ·			

client_ctx Member List

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

flags (defined in client_ctx)	client_ctx
<pre>ip_length (defined in client_ctx)</pre>	client_ctx
ka_secs (defined in client_ctx)	client_ctx
net (defined in client_ctx)	client_ctx
next (defined in client_ctx)	client_ctx
<pre>remote_ip (defined in client_ctx)</pre>	client_ctx
timeout (defined in client_ctx)	client_ctx
usr (defined in client_ctx)	client_ctx



Main P	age	Relat	ed Pages	Modu	ıles	Classes	
Files							
File List	File N	/lembers					

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

Go to the documentation of this file.

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

```
00016 /*
00017
        mqtt_common.h
00018
        This module outlines the interfaces that a
00019
re common to both client amd
        server components. The applications are no
00020
t expected to utilize the
00021 services outlined in this module.
00022 */
00023
00024 #ifndef __MQTT_COMMON_H__
00025 #define MOTT COMMON H
00026
00042 #include <stdbool.h>
00043 #include <stdlib.h>
00044 #include <stdio.h>
00045
00046 #define MQTT_COMMON_VERSTR "1.0.0"
00048 typedef int
                               i32;
00049 typedef unsigned int
                               u32;
00050 typedef unsigned char
                              u8;
00051 typedef char
                                i8;
00052 typedef unsigned short u16;
00053 typedef short
                               i16;
00054
00055 \text{ #define MIN}(a, b) ((a > b)? b : a)
00056
00058 #define MQTT_CONNECT
                                  0x01
00059 #define MQTT_CONNACK
                                  0 \times 02
00060 #define MQTT_PUBLISH
                                  0 \times 03
00061 #define MQTT_PUBACK
                                  0 \times 04
00062 #define MQTT_PUBREC
                                  0 \times 05
00063 #define MQTT_PUBREL
                                  0x06
00064 #define MQTT_PUBCOMP
                                  0 \times 07
00065 #define MQTT_SUBSCRIBE
                                  0x08
00066 #define MQTT_SUBACK
                                  0 \times 09
00067 #define MOTT UNSUBSCRIBE
                                  0x0A
```

```
00068 #define MQTT_UNSUBACK
                               0x0B
00069 #define MQTT_PINGREQ
                                0 \times 0 C
00070 #define MQTT_PINGRSP
                                0 \times 0 D
00071 #define MQTT_DISCONNECT
                                0x0E
00072
00073 #define MAX FH LEN
                               0x05
00076 #define MAX_REMLEN_BYTES (MAX_FH_LEN - 1)
00077
00078 #define MAKE_FH_BYTE1(msg_type, flags) (u8)
((msg_type << 4) | flags)</pre>
00079
00080 #define MAKE FH_FLAGS(bool_dup, enum_qos, bo
ol retain)
              (u8)(((bool_dup << 3) | (enum_qos <<
00081
 1) | bool retain) & 0xF)
00082
00083 #define QOS_VALUE(enum_qos) (u8)(enum_qos &
0x3)
00084 #define OFL VALUE
                                  0x80
00086 #define DUP FLAG VAL(bool val) (u8)(bool val
<< 3)
00087
00088 #define BOOL_RETAIN(fh_byte1) ((fh_byte1 &
0x1)? true : false)
00089 #define BOOL_DUP(fh_byte1) ((fh_byte1 &
0x8)? true : false)
00090 #define ENUM_QOS(fh_byte1) (enum mqtt_qo
s)((fh_byte1 \& 0x6) >> 1)
00091
00092 #define MSG_TYPE(fh_byte1) (u8)((fh_byte1 &
0xf0) >> 4)
00093
00094 static inline u32 buf_wr_nbytes(u8 *dst, con
st u8 *src, u32 n)
00095 {
00096
             u32 c = n;
00097
      while(c--)
```

```
00098
                       *dst++ = *src++;
00099
00100
             return n;
00101 }
00102
00103 static inline u32 buf_set(u8 *dst, u8 val, u
32 n)
00104 {
00105
              u32 c = n;
              while(c--)
00106
00107
                      *dst++ = val;
00108
00109
             return n;
00110 }
00111
00113 static inline u32 buf_wr_nbo_2B(u8 *buf, u16
val)
00114 {
              buf[0] = (u8)((val >> 8) \& 0xFF); /*
00115
MSB */
              buf[1] = (u8)((val) & 0xFF); /*
00116
LSB */
00117
             return 2;
00118 }
00119
00121 static inline u32 buf_rd_nbo_2B(const u8 *bu
f, u16 *val)
00122 {
             *val = (u16)((buf[0] << 8) | (buf[1]
00123
));
00124
            return 2;
00125 }
00126
00132 struct mgtt_packet {
00133
00134
              u8
                                      msg_type;
                                      fh_byte1;
00135
              u8
```

```
00137
              u16
                                       msg_id;
                                       n_refs;
00139
              u8
                                       pad[3];
00140
              u8
00141
00142
                                       offset:
              u8
                                       fh_len;
00143
              u8
                                       vh_len;
00144
              u16
                                       pl_len;
00145
              u32
00147
              u32
                                       private;
00148
00149
              u32
                                       maxlen;
                                      *buffer;
00150
              u8
                                      (*free)(struct
00153
              void
mqtt_packet *mqp);
00154
              struct mqtt_packet
00155
                                      *next;
00156 };
00157
00160 #define MOP FHEADER BUF(mgp)
                                      (mqp->buffer +
 mgp->offset)
00161 #define MQP_VHEADER_BUF(mqp)
                                      (MQP_FHEADER_B
UF(mqp) + mqp->fh_len)
00162 #define MQP_PAYLOAD_BUF(mqp)
                                      (MQP_VHEADER_B
UF(mqp) + mqp->vh_len)
00163
00164 #define MQP_CONTENT_LEN(mqp)
                                      (mqp->fh_len +
 mqp->vh_len + mqp->pl_len)
00165 #define MQP_FREEBUF_LEN(mgp)
                                      (mqp->maxlen -
 mqp->offset -
00166
                                       MQP_CONTENT_L
EN(mqp))
00167
                                      (mqp->fh_byte1)
00168 #define MQP_FHEADER_VAL(mgp)
00169 #define MQP_FHEADER_MSG(mgp)
                                      (MSG_TYPE(MQP_
FHEADER_VAL(mqp)))
00170 #define MQP_FHEADER_FLG(mqp)
                                     (MSG_FLAGS(MQP
```

```
_FHEADER_VAL(mqp)))
00171
00172 #define DEFINE_MQP_VEC(num_mqp, mqp_vec)
              static struct mqtt_packet mqp_vec[nu
00173
m_mqp];
00174
00175 #define DEFINE_MQP_BUF_VEC(num_mqp, mqp_vec,
buf_len, buf_vec)
              DEFINE_MQP_VEC(num_mqp, mqp_vec);
00176
              static u8 buf vec[num mgp][buf len];
00177
00178
00179 /*----
00180 * Heleper MACROS for PUBLISH-RX Message Pro
cessing
00181 *----
00182 */
00183
00189 #define MOP PUB TOP BUF(mgp) (MOP VHEADER BU
F(mqp) + 2
00190
00192 #define MQP_PUB_TOP_LEN(mgp) (mgp->vh_len -
2 - (mqp->msg_id? 2 : 0))
00193
00195 #define MQP_PUB_PAY_BUF(mgp) (mgp->pl_len? M
QP_PAYLOAD_BUF(mqp) : NULL)
00196
00198 #define MQP_PUB_PAY_LEN(mgp) (mgp->pl_len)
00199
00205 #define WILL RETAIN VAL 0x20
00206 #define WILL CONFIG VAL 0x04
00207 #define CLEAN START VAL 0x02
00208 #define USER_NAME_OPVAL 0x80
00209 #define PASS WORD OPVAL 0x40
```

```
00210
00219 #define MQP_ERR_NETWORK
                              (-1)
00220 #define MQP_ERR_TIMEOUT
                              (-2)
00221 #define MQP_ERR_NET_OPS (-3)
00222 #define MOP ERR FNPARAM
                              (-4)
00223 #define MQP_ERR_PKT_AVL (-5)
00224 #define MQP_ERR_PKT_LEN (-6)
00225 #define MQP_ERR_NOTCONN
                              (-7)
00226 #define MQP_ERR_BADCALL (-8)
00227 #define MQP_ERR_CONTENT (-9)
00228 #define MQP_ERR_LIBQUIT (-10)
00231 #define MOP ERR NOT DEF (-32)
00235 /*-----
00236 * Common Operations
00237 *-----
00238 */
00239
00246 void mqp_free(struct mqtt_packet *mqp);
00247
00257 void mqp_reset(struct mqtt_packet *mqp);
00258
00268 void mgp_init(struct mgtt_packet *mgp, u8 of
fset);
00269
00271 static
00272 inline void mgp_buffer_attach(struct mgtt_pa
cket *mgp, u8 *buffer, u32 length,
                                  u8 offset)
00273
00274 {
             mqp_init(mqp, offset);
00275
00276
00277
             mqp->buffer = buffer;
             mqp->maxlen = length;
00278
             mqp->free = NULL;
00279
00280
```

```
00281
      return;
00282 }
00283
00285 struct utf8_string {
00286
00287
              i8
                  *buffer;
                    length;
00288
              u16
00289 };
00290
00300 i32 mqp_buf_wr_utf8(u8 *buf, const struct ut
f8_string *utf8);
00301
00314 i32 mqp_buf_tail_wr_remlen(u8 *buf, u32 reml
en);
00315
00324 i32 mgp_buf_rd_remlen(u8 *buf, u32 *remlen);
00325
00343 i32
00344 mgp pub append topic(struct mgtt packet *mgp
, const struct utf8_string *topic,
00345
                           u16 msg_id);
00346
00360 i32 mgp_pub_append_data(struct mgtt_packet *
mqp, const u8 *data_buf,
                              u32 data_len);
00361
00362
00373 bool mgp_proc_msg_id_ack_rx(struct mgtt_pack
et *mgp_raw, bool has_payload);
00374
00384 bool mgp_proc_pub_rx(struct mgtt_packet *mgp
raw);
00385
00386 /*
         Wait-List of MQTT Messages for which ackn
oledge is pending from remote node.
00388 */
00389 struct mqtt_ack_wlist {
```

```
00390
              struct mqtt_packet *head; /* Points
00391
to head of single linked-list. */
              struct mqtt_packet *tail; /* Points
00392
to tail of single linked-list. */
00393 };
00394
00395 static inline bool mgp_ack_wlist_is_empty(st
ruct mgtt_ack_wlist *list)
00396 {
             return (NULL == list->head) ? true :
00397
false:
00398 }
00399
00400 /*
00401
        Add specified element into trailing end o
f list.
00402
00403
         Returns, on success, true, otherwise fals
e.
00404 */
00405 bool mgp_ack_wlist_append(struct mgtt_ack_wl
ist *list,
00406
                                struct mgtt packet
    *elem);
00407
00408 /*
         Removes element that has specified msg_id
00409
from list.
00410
         Returns, on success, pointer to removed e
00411
lement, otherwise NULL.
00412 */
00413 struct mgtt_packet *mgp_ack_wlist_remove(str
uct mqtt_ack_wlist *list,
00414
                                                 u1
6 msg_id);
```

```
00415 /*
         Removes and frees all elements in list.
00416
00417 */
00418 void mgp_ack_wlist_purge(struct mgtt_ack_wli
st *list);
00419
00420 static inline bool is_wlist_empty(const stru
ct mgtt_ack_wlist *list)
00421 {
             return list->head? false : true;
00422
00423 }
00424
00439 i32 mqp_prep_fh(struct mqtt_packet *mqp, u8
flags);
00440
00442 enum mqtt_qos {
00443
              MQTT_QOSO,
00444
              MQTT_QOS1,
00445
00446
              MOTT QOS2
00447 };
00448
00450 struct utf8_strqos {
00451
00452
              i8
                              *buffer;
              u16
                              length;
00453
00454
              enum mqtt_qos qosreq;
00455 };
00456
00457
00467 struct secure_conn {
00468
              void *method;
00469
              void *cipher;
00470
00471
              u32 n_{file};
              char **files;
00472
00473 };
```

```
00474
00484 struct device net services {
00485
00489 #define DEV_NETCONN_OPT_TCP
                                   0x01
00490 #define DEV NETCONN OPT UDP
                                   0x02
00491 #define DEV NETCONN OPT IP6
                                   0 \times 04
00492 #define DEV NETCONN OPT URL
                                   0x08
00493 #define DEV NETCONN OPT SEC 0x10
              i32 (*open)(u32 nwconn_opts, const i
00512
8 *server_addr, u16 port_number,
                          const struct secure_conn
00513
 *nw security);
00514
              i32 (*send)(i32 comm, const u8 *bu
00526
f, u32 len, void *ctx);
00527
              i32 (*recv)(i32 comm, u8 *buf, u32
00548
len, u32 wait_secs,
00549
                            bool *err timeo, void
*ctx);
00550
              i32 (*send_dest)(i32 comm, const u
00565
8 *buf, u32 len, u16 dest_port,
00566
                                 const i8 *dest ip
, u32 ip_len);
00567
              i32 (*recv_from)(i32 comm, u8 *buf
00588
, u32 len, u16 *from_port,
                                 i8 *from_ip, u32
00589
*ip_len);
00590
              i32 (*close)(i32 comm);
00592
00593
00605
       i32 (*listen)(u32 nwconn_opts, u16 p
ort_number,
00606
                            const struct secure co
nn *nw security);
```

```
00607
           i32 (*accept)(i32 listen, i8 *client
00623
_ip, u32 *ip_length);
00624
            i32 (*io_mon)(i32 *recv_cvec, i32
00653
*send cvec,
00654
                          i32 *rsvd_cvec, u32
wait_secs);
00655
            u32 (*time)(void);
00677
00678 };
00679 /* device_net_services */
00681
00682 /* Receive data from the specified network a
nd read into the 'mqp' */
00683 i32 mqp_recv(i32 net, const struct devi
ce_net_services *net_ops,
                struct mqtt_packet *mqp, u32 wa
00684
it_secs, bool *timed_out,
                void *ctx);
00685
00686
00687 /*-------
_____
00688 * Data structure for managing the QoS2 PUB
RX packets and follow-ups
00689 *------
*/
00690
00691 #define MAX_PUBREL_INFLT 8 /* Must be kept a
s a value of 2<sup>n</sup> */
00692
00693 struct pub_qos2_cq { /* Circular Queue CQ to
track QOS2 PUB RX messages */
00694
           u16 id_vec[MAX_PUBREL_INFLT]; /* Ve
00695
ctor to store RX Message-IDs */
00696
            u8 n free;
                                       /* Nu
```

```
m of free elements in vector */
00697
                                              /* In
              u8 rd_idx;
dex to Read next Message-ID */
                                              /* In
00698
              u8 wr_idx;
dex to Write next Message-ID */
00699 };
00700
00701 /* Reset the specified Circular Queue (CQ) */
00702 void qos2_pub_cq_reset(struct pub_qos2_cq *c
q);
00703
00704 /* Append the message-id into the CQ tail. R
eturn true on success, else false */
00705 bool qos2_pub_cq_logup(struct pub_qos2_cq *c
q, u16 msg_id);
00706
00707 /* Remove the message-id from the CO head. R
eturn true on success, else false */
00708 bool gos2 pub cg unlog(struct pub gos2 cg *c
q, u16 msg_id);
00709
00710 /* Is the message-id available in the CQ ? R
eturn true on success, else false */
00711 bool qos2_pub_cq_check(struct pub_qos2 cq *c
q, u16 msg_id);
00712
00713 /* Get the count of message-ID(s) available
in the CO */
00714 static inline i32 qos2_pub_cq_count(struct p
ub_qos2_cq *cq)
00715 {
00716
             return MAX_PUBREL_INFLT - cq->n_free
00717 }
00718
00719 struct client ctx {
```

```
00720
                           *usr; /* Client Usr */
00721
              void
                            net; /* Socket HND */
              i32
00722
00723
00724
              i8
                            remote_ip[16];
                            ip_length;
00725
              u32
00726
00727
              u32
                            timeout;
00728
              u16
                            ka_secs;
00729
00730
              u32
                            flags;
00731
00732
              struct client ctx *next;
00733 };
00734
00735 void cl_ctx_reset(struct client_ctx *cl_ctx)
00736 void cl_ctx_timeout_insert(struct client_ctx
 **head,
00737
                                  struct client ctx
 *elem);
00738
00739 void cl_ctx_remove(struct client_ctx **head,
                          struct client ctx *elem);
00740
00741
00742 #define KA_TIMEOUT_NONE 0xffffffff /* Diffe
rent than KA SECS = 0 * /
00743 void cl_ctx_timeout_update(struct client_ctx
 *cl_ctx, u32 now_secs);
00744
00745 #endif
                                                    |
```



Files	
Class List Class Index Class	Members

device_net_services Member List

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

```
accept
          device net services
         device net services
close
io mon device net services
listen device net services
          device net services
open
          device net services
recv
recv from device net services
send
          device net services
send dest device net services
time
          device net services
```



Main Pa	ge	Relat	ed Pages	I	Modules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
mqtt_ac	ck_	wlist N	/lember L	ist	t		

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

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



Main P	age	Relat	ed Pages	Module	es	Classes	
Files							
File List	File N	/lembers					

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

Go to the documentation of this file.

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

```
00016
00017 /*
00018 mqtt_client.h
00019
00020 This module enumerates the public interfac
es / API of the MOTT Client
00021
     Library.
00022 */
00023
00024
00025 #ifndef __MQTT_CLIENT_H__
00026 #define MOTT CLIENT H
00027
00076 #include "mgtt common.h"
00077
00079 * MQTT Client Messaging Routines / Services
00080 *-----
00081 */
00082
00087 #define MOTT CLIENT VERSTR "1.0.0"
00093 u16 mgtt_client_new_msg_id(void);
00094
00110 bool mqtt_client_is_connected(void *ctx);
00111
00137 i32 mgtt_connect_msg_send(void *ctx, bool cl
ean_session, u16 ka_secs);
00138
00168 i32 mgtt_client_pub_msg_send(void *ctx,
00169
                              const struct u
tf8_string *topic,
                              const u8 *data
00170
buf, u32 data_len,
00171
                              enum mqtt_qos q
```

```
os, bool retain);
00172
00211 i32 mgtt_client_pub_dispatch(void *ctx, stru
ct mgtt_packet *mgp,
00212
                                    enum matt aos a
os, bool retain);
00213
00243 i32 mgtt_sub_msg_send(void *ctx, const struct
utf8_strgos *gos_topics, u32 count);
00244
00283 i32 mgtt_sub_dispatch(void *ctx, struct mgtt
packet *mqp);
00284
00315 i32 mgtt unsub msg send(void *ctx, const str
uct utf8_string *topics, u32 count);
00316
00356 i32 mgtt_unsub_dispatch(void *ctx, struct mg
tt packet *mqp);
00357
00365 i32 mgtt pingreg send(void *ctx);
00366
00373 i32 mgtt disconn send(void *ctx);
00374
00375
00431 i32 mgtt_client_send_progress(void *ctx);
00432
00455 i32 mgtt_client_ctx_await_msg(void *ctx, u8
msg_type, struct mgtt_packet *mgp,
00456
                                     u32 wait_secs)
00457
00462 static inline
00463 i32 mgtt_client_ctx_recv(void *ctx, struct m
qtt_packet *mqp, u32 wait_secs)
00464 {
00465
              /* Receive next and any MQTT Message
 from the broker */
```

```
00466
      return mqtt_client_ctx_await_msg(ctx
, 0x00, mqp, wait_secs);
00467 }
00468
00491 i32 mgtt_client_ctx_run(void *ctx, u32 wait_
secs);
00492
00513 i32 mgtt_client_await_msg(struct mgtt_packet
 *mqp, u32 wait_secs, void **app);
00514
00537 i32 mgtt_client_run(u32 wait_secs);
00538
00539 /*------
00540 * MQTT Client Library: Packet Buffer Pool
and its management
00541 *--------
00542 */
00543
00557 struct mqtt_packet *mqp_client_alloc(u8 msg_
type, u8 offset);
00558
00564 static inline struct mgtt_packet *mgp_client
_send_alloc(u8 msg_type)
00565 {
00566
            return mgp_client_alloc(msg_type, MA
X_FH_LEN);
00567 }
00568
00574 static inline struct mgtt_packet *mgp_client
_recv_alloc(u8 msg_type)
00575 {
00576
          return mgp_client_alloc(msg_type, 0)
00577 }
00578
```

```
00609 i32 mgtt_client_buffers_register(u32 num_mgp
, struct mqtt_packet *mqp_vec,
00610
                                      u32 buf len
, u8 *buf_vec);
00611
00612 /*------
00613 * MQTT Client Library: Register applicatio
n, platform information and services.
00615 */
00616
00639 i32 mqtt_client_ctx_info_register(void *ctx,
00640
                                       const stru
ct utf8_string *client_id,
00641
                                     const stru
ct utf8_string *user_name,
00642
                                       const stru
ct utf8 string *pass word);
00643
00664 i32 mqtt_client_ctx_will_register(void *ctx,
00665
                                      const stru
ct utf8_string *will_top,
00666
                                      const stru
ct utf8_string *will_msg,
                                       enum mqtt
00667
gos will_gos, bool retain);
00668
00678 i32 mqtt_client_net_svc_register(const struct
device_net_services *net);
00679
00682 #define VHB_CONNACK_RC(vh_buf) (vh_buf[1])
00683 #define MQP_CONNACK_RC(mqp) (mqp->buffer[
3])
00685 #define VHB_CONNACK_SP(vh_buf) (vh_buf[0] &
```

```
0x1)
00686 #define MQP_CONNACK_SP(mqp) (mqp->buffer[
2] & 0x1)
00689 #define VHB_CONNACK_VH16(vh_buf)((vh_buf[0]
<< 8) | vh_buf[1])
00690 #define MQP_CONNACK_VH16(mgp) ((mgp->buffe
r[2] \ll 8) \mid mqp \rightarrow buffer[3])
00691
00693 struct mqtt_client_ctx_cbs {
00694
00729
              bool (*publish_rx)(void *app,
00730
                                  bool dup, enum mg
tt_qos qos, bool retain,
00731
                                  struct mgtt packet
*mqp);
00732
              void (*ack_notify)(void *app, u8 msg
00757
_type, u16 msg_id, u8 *buf, u32 len);
00758
00768
              void (*disconn cb)(void *app, i32 ca
use);
00769 };
00770
00771 struct mgtt client ctx cfg {
00772
00776 #define MQTT_CFG_PROTOCOL_V31
                                      0x0001
00777 #define MOTT CFG APP HAS RTSK
                                      0x0002
00778 #define MQTT_CFG_MK_GROUP_CTX
                                      0 \times 0004
00781
              u16 config_opts;
00786 #define MQTT_NETCONN_OPT_IP6 DEV_NETCONN_OP
T IP6
00787 #define MQTT_NETCONN_OPT_URL DEV_NETCONN_OP
T URL
00788 #define MQTT_NETCONN_OPT_SEC DEV_NETCONN_OP
T SEC
00791
              u32
                   nwconn_opts;
              i8 *server addr;
00793
```

```
00794
              u16    port_number;
              struct secure_conn *nw_security;
00795
00796 };
00797
00840 i32 mgtt_client_ctx_create(const struct mgtt
_client_ctx_cfg *ctx_cfg,
00841
                                  const struct mqtt
_client_ctx_cbs *ctx_cbs,
                                 void *app, void *
00842
*ctx);
00843
00858 i32 mqtt_client_ctx_delete(void *ctx);
00859
00861 struct mqtt_client_lib_cfg {
00862
00867
              u16
                    loopback_port;
00868
              bool
                    grp_uses_cbfn;
00871
              void
                    *mutex;
              void (*mutex lockin)(void *mutex);
00872
              void (*mutex unlock)(void *mutex);
00873
                   (*debug printf)(const i8 *forma
00875
              i32
t, ...);
              bool aux_debug_en;
00876
00877 };
00878
00904 i32 mgtt_client_lib_init(const struct mgtt_c
lient_lib_cfg *cfg);
00905
00909 i32 mgtt_client_lib_exit(void);
00910 /* End group client_api */
00912
00913 #endif
                                                   F
```



Main Pa	ge F	elated Pages	Mod	dules	Classes	
Files						
Class List	Class In	dex Class Memb	ers			
mqtt_cl	ient_c	ctx_cbs Men	nber l	_ist		

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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XXYG@M 1.7.4



Main Pa	Main Page Related Page		ed Pages	N	lodules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
mqtt_c	lien	t_ctx_	cfg Mem	be	r List		

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

```
config_opts mqtt_client_ctx_cfg
nw_security mqtt_client_ctx_cfg
nwconn_opts mqtt_client_ctx_cfg
port_number mqtt_client_ctx_cfg
server_addr mqtt_client_ctx_cfg
```



Main Pag	ge Rela	ted Pages	M	lodules	Classes	
Files						
Class List	Class Index	Class Membe	rs			
mqtt_cl	ient_lib_	cfg Memb	er	List		

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

```
aux_debug_en mqtt_client_lib_cfg
debug_printf mqtt_client_lib_cfg
grp_uses_cbfn mqtt_client_lib_cfg
loopback_port mqtt_client_lib_cfg
mutex mqtt_client_lib_cfg
mutex_lockin mqtt_client_lib_cfg
mutex_unlock mqtt_client_lib_cfg
```



Main Pa	ge Relat	ed Pages	Modules	Classes
Files				
Class List	Class Index	Class Membe	rs	
matt n	acket Mei	mhar List		

mqtt_packet Member List

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

buffer	mqtt_packet
fh_byte1	mqtt_packet
fh_len	mqtt_packet
free	mqtt_packet
maxlen	mqtt_packet
msg_id	mqtt_packet
msg_type	mqtt_packet
n_refs	mqtt_packet
next (defined in mqtt_packet)	mqtt_packet
offset	mqtt_packet
<pre>pad (defined in mqtt_packet)</pre>	mqtt_packet
pl_len	mqtt_packet
<pre>private (defined in mqtt_packet)</pre>	mqtt_packet
vh_len	mqtt_packet

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @OXXV@@M 1.7.4



Main Pa	ge	Relate	ed Pages	N	Modules	Classes	
Files							
Class List	Cla	ss Index	Class Membe	rs			
pub_qc)s2_	_cq Me	ember Lis	t			

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
```

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



Main Pa	ge	Related Pages			lodules	Classes	
Files							
Class List	Clas	s Index	Class Membe	rs			
secure_	_con	n Mei	mber Lis	t			

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

cipher secure_conn files secure_conn method secure_conn n file secure_conn

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





Main Pa	ge Rela	ted Pages	Modules	Classes
Files				
Class List	Class Index	Class Membe	rs	
utf8_sti	ring Mem	ber List		

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

buffer utf8_string length utf8_string

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



Main Pa	ge	Relate	ed Pages	N	Modules	Classes	
Files							
Class List	Class List Cla		Class Membe	rs			
utf8_sti	rqos	s Mem	ber List				

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

buffer utf8_strqos length utf8_strqos qosreq utf8_strqos

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XXYG@M 1.7.4



ľ	Main Page Files Class List Class				Related Pages					M	od	ule	es	Classes			
F																	
C	Class List Class Index								Cla	ss IV	lemb	ers					
P	All	V	⁄aria	ble	S												
a	b	С	d	f	g	i	1	m	n	0	р	q	r	S	t	V	

- 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_connfree : mqtt_packet
- g -
 - grp_uses_cbfn : mqtt_client_lib_cfg
- i -
 - io mon: device net services
- | -
 - 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
- **0 -**
 - 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



Main P	age	Relat	ed Pages	Modules	Classes	
Files						
File List	File N	/lembers				
client_	_note	es.h				
00001						



Main P	Main Page		ed Pages	Modules	Classes	
Files						
File List	File N	Members				
MainP	age.	h				
00001						



Mai	n Pa	age	Relat	ed Pag	es	Modu	les	Cla	asses	
File	S									
File L	ist	File Mei	mbers							
All	Fu	inctions Enu		erations	ations Enumerator		Defines			
m										

- 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
- mgtt client await msg(): mgtt_client.h
- mqtt_client_buffers_register(): mqtt_client.h
- mqtt_client_ctx_await_msg(): mqtt_client.h
- mgtt client ctx create(): mgtt client.h
- mqtt_client_ctx_delete(): mqtt_client.h
- mqtt client ctx info register(): mqtt_client.h
- mgtt client ctx run(): mgtt 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

Generated on Mon Nov 17 2014 12:11:04 for MQTT Client by @@XYQQ@1 1.7.4



Mai	9		ed Page	es Mod	lules	Cla	asses		
File	s								
File L	.ist	File Me	mbers						
All	Fu	ınctions	Enum	erations	Enumerator	Defi	nes		

MQTT_QOS0: mqtt_common.h
 MQTT_QOS1: mqtt_common.h
 MQTT_QOS2: mqtt_common.h



Ma	Main Page Rela					elat	ed Page	ed Pages Modules				asses	
Fil	les												
File	Lis	st	F	ile Me	embe	ers							
All		Fu	ınctions		Enumerations		En	Enumerator		Defines			
d n	n	q	v										

- 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: mgtt 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
- MOP ERR NETWORK: mgtt common.h
- MQP ERR NOT DEF: mqtt_common.h
- MOP ERR NOTCONN: mgtt common.h
- MQP ERR PKT AVL: mqtt_common.h
- MQP ERR PKT LEN: mqtt_common.h

- MQP ERR TIMEOUT: mqtt common.h
- MQP PUB PAY BUF: mqtt common.h
- MQP_PUB_PAY_LEN: mqtt_common.h
- MQP_PUB_TOP_BUF: mqtt_common.h
- MQP_PUB_TOP_LEN: mqtt_common.h
- 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_VERSTR: mqtt_client.h
- MQTT_COMMON_VERSTR: mqtt_common.h
- MQTT CONNECT: mqtt_common.h
- MQTT_NETCONN_OPT_IP6: mqtt_client.h
- MQTT_NETCONN_OPT_SEC: mqtt_client.h
- MQTT_NETCONN_OPT_URL: mqtt_client.h

- q -

QFL VALUE : mqtt_common.h

- V -

- VHB_CONNACK_RC : mqtt_client.h
- VHB_CONNACK_SP: mqtt_client.h