Here is a list of all modules:

- MQTT Packet (MQP) Buffer structure
- Helper Macros for RX PUBLISH
- LIBRARY Generated Error Codes
- Information to establish a secure connection.
- Abstraction of Network Services on a platform
- Options for platform to configure network
- The Server Library API(s)
  - Helper Macros for RX CONNECT
- The Server Daemon API(s)
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>MQTT Packet (MQP)</strong></td>
<td></td>
<td>Buffer structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Classes

struct mqtt_packet
Detailed Description

The core construct to encapsulate, construct and process a message
Helper Macros for RX
PUBLISH
#Defines

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

```c
#define MQP_PUB_PAY_BUF (mqp) (mqp->pl_len? MQP_PAYLOAD_BUF(mqp) : NULL)
```
Yields pointer to payload data

```c
#define MQP_PUB_PAY_LEN (mqp) (mqp->pl_len)
```
Length or size of payload data

```c
#define MQP_PUB_TOP_BUF (mqp) (MQP_VHEADER_BUF(mqp) + 2)
```
Yields pointer to topic content

```c
#define MQP_PUB_TOP_LEN (mqp) (mqp->vh_len - 2 - (mqp->msg_id? 2 : 0))
```
Length or size of topic content
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY Generated Error Codes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Defines**

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define MQP_ERR_NETWORK</code></td>
<td>(-1)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_TIMEOUT</code></td>
<td>(-2)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NET_OPS</code></td>
<td>(-3)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_FNPARAM</code></td>
<td>(-4)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_PKT_AVL</code></td>
<td>(-5)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_PKT_LEN</code></td>
<td>(-6)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NOTCONN</code></td>
<td>(-7)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_BADCALL</code></td>
<td>(-8)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_CONTENT</code></td>
<td>(-9)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_LIBQUIT</code></td>
<td>(-10)</td>
</tr>
<tr>
<td><code>#define MQP_ERR_NOT_DEF</code></td>
<td>(-32)</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Definition</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define MQP_ERR_NOTCONN (-7)</code></td>
<td>Lib isn't CONNECTED to server</td>
</tr>
<tr>
<td><code>#define MQP_ERR_PKT_AVL (-5)</code></td>
<td>No pkts are available in pool</td>
</tr>
<tr>
<td><code>#define MQP_ERR_PKT_LEN (-6)</code></td>
<td>Inadequate free buffer in pkt</td>
</tr>
<tr>
<td><code>#define MQP_ERR_TIMEOUT (-2)</code></td>
<td>Net transaction has timed out</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Abstraction of Network Services on a platform
Classes

struct device_net_services
Detailed Description

Services to enable the MQTT Client-Server communication over network

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

<table>
<thead>
<tr>
<th>Define</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_NETCONN_OPT_TCP</td>
<td>0x01</td>
</tr>
<tr>
<td>DEV_NETCONN_OPT_UDP</td>
<td>0x02</td>
</tr>
<tr>
<td>DEV_NETCONN_OPT_IP6</td>
<td>0x04</td>
</tr>
<tr>
<td>DEV_NETCONN_OPT_URL</td>
<td>0x08</td>
</tr>
<tr>
<td>DEV_NETCONN_OPT_SEC</td>
<td>0x10</td>
</tr>
</tbody>
</table>
### Define Documentation

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define DEV_NETCONN_OPT_IP6 0x04</code></td>
<td>Assert for IPv6, otherwise it is IPv4</td>
<td></td>
</tr>
<tr>
<td><code>#define DEV_NETCONN_OPT_SEC 0x10</code></td>
<td>Assert to indicate a secure connection</td>
<td></td>
</tr>
<tr>
<td><code>#define DEV_NETCONN_OPT_TCP 0x01</code></td>
<td>Assert to indicate TCP net connection</td>
<td></td>
</tr>
<tr>
<td><code>#define DEV_NETCONN_OPT_UDP 0x02</code></td>
<td>Assert to create a local UDP port bind</td>
<td></td>
</tr>
<tr>
<td><code>#define DEV_NETCONN_OPT_URL 0x08</code></td>
<td>Assert if the network address is a URL</td>
<td></td>
</tr>
<tr>
<td>Main Page</td>
<td>Modules</td>
<td>Classes</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>The Server Library API(s)</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Classes | Modules | Defines | Functions
## Classes

<table>
<thead>
<tr>
<th>struct</th>
<th><code>mqtt_server_msg_cbs</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
</tbody>
</table>
Modules

| Helper Macros for RX CONNECT |
Defines

#define MQP_SERVER_RX_LEN 1024
Functions

<table>
<thead>
<tr>
<th>i32</th>
<th><code>mqtt_vh_msg_send</code> (void *ctx_cl, u8 msg_type, enum <code>mqtt_qos</code> qos, bool has_vh, u16 vh_data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td><code>mqtt_vh_msg_send_locked</code> (void *ctx_cl, u8 msg_type, enum <code>mqtt_qos</code> qos, bool has_vh, u16 vh_data)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_pub_dispatch</code> (void *ctx_cl, struct <code>mqtt_packet</code> *mqp, bool dup)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_pub_dispatch_locked</code> (void *ctx_cl, struct <code>mqtt_packet</code> *mqp, bool dup)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_run</code> (u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_register_net_svc</code> (const struct <code>device_net_services</code> *net)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_lib_init</code> (const struct <code>mqtt_server_lib_cfg</code> *cfg, const struct <code>mqtt_server_msg_cbs</code> *cbs)</td>
</tr>
</tbody>
</table>
Define Documentation

```
#define MQP_SERVER_RX_LEN 1024
```

Max size(B) of RX Buffer for MQTT Server
Function Documentation

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

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

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>This routine must be invoked only once in an run of the system.</td>
</tr>
</tbody>
</table>

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

See also:
struct mqtt_server_lib_cfg

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

See also:
mqtt_server_register_net_svc.

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

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

0 on success otherwise -1.

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

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

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

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

**Parameters:**

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

**Returns:**

on success, the number of bytes transferred. Otherwise, error defined in LIBRARY Generated Error Codes
i32 mqtt_server_pub_dispatch_locked ( void * ctx_c,  
  struct mqtt_packet * mqp,  
  bool dup )

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

Refer to `mqtt_server_pub_dispatch` for other details.

i32 mqtt_server_register_net_svc ( const struct device_net_services net_svc )

Abstraction for the device specific network services

Network services for communication with the clients

**Parameters:**

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

**Returns:**

- on success, 0, otherwise -1

**Abstraction of Network Services on a platform**

**Note:**

- all entries in net must be supported by the platform.

i32 mqtt_server_run ( u32 wait_secs )

Run the server packet LIB for the specified wait time. This routine
yields the control back to the application after the specified duration of wait time. Such an arrangement enable the application to make overall progress to meet it intended functionality.

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

**Parameters:**

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

**Note:**

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

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

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

**Parameters:**

[in] *ctx_cl*  handle to the underlying network context in the LIB. This handle is provided to the application by the LIB in the CONNECT callback.

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

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

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

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

Refer to **mqtt_vh_msg_send** for details
Helper Macros for RX CONNECT

The Server Library API(s)
Defines

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

```c
#define MQ_CONN_UTF8_BUF  (utf8)     ((utf8)? (utf8)->buffer : NULL)
```

Yields pointer to the UTF8 content

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

Length or size of the UTF8 content

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

Returns Client ID

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

Returns Pass Word

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

Returns User Name

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

Returns Will Data

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

Returns Will Topic
### The Server Daemon API(s)

- MQTT Server 1.0.0
### Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>mqtt_server_app_cbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>mqtt_server_app_cfg</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_topic_enroll</code> (const void *app_hnd, const struct utf8_string *topic, enum <code>mqtt_qos</code> qos)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_topic_disenroll</code> (const void *app_hnd, const struct utf8_string *topic)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_app_pub_send</code> (const struct utf8_string *topic, const u8 *data_buf, u32 data_len, enum <code>mqtt_qos</code> qos, bool retain)</td>
</tr>
<tr>
<td>void *</td>
<td><code>mqtt_server_app_register</code> (const struct mqtt_server_app_cbs *cbs, const i8 *name)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_init</code> (const struct <code>mqtt_server_lib_cfg</code> *lib_cfg, const struct <code>mqtt_server_app_cfg</code> *app_cfg)</td>
</tr>
</tbody>
</table>
Function Documentation

```c
i32 mqtt_server_app_pub_send ( const struct utf8_string * topic,
    const u8 * data_buf,
    u32 data_len,
    enum mqtt_qos qos,
    bool retain )
```

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

**Parameters:**

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

**Returns:**

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

```c
void* mqtt_server_app_register ( const struct mqtt_server_app_cb
    const i8 *
)
```

Register an application with the server. This routine makes known to the server about an application identified by its name and creates a context / reference for the application in the server.
An application intending to utilize the service of the MQTT server must be first registered with the server.

**Parameters:**

[in] **cbs**  
callback routines from the application to be invoked by the server

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

**Returns:**

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

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

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

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

**Note:**

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

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

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

Returns:

0 on success, otherwise -1 on error.

```c
i32 mqtt_server_topic_disenroll (const void *app_hnd,
const struct utf8_string *topic)
```

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

Parameters:

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

Returns:

0 on success, otherwise a negative value on error.

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

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

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

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

Returns:
- 0 on success, otherwise a negative value on error.
Here are the classes, structs, unions and interfaces with brief descriptions:

<table>
<thead>
<tr>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_ctx</td>
</tr>
<tr>
<td>device_net_services</td>
</tr>
<tr>
<td>mqtt_ack_wlist</td>
</tr>
<tr>
<td>mqtt_packet</td>
</tr>
<tr>
<td>mqtt_server_app_cbs</td>
</tr>
<tr>
<td>mqtt_server_app_cfg</td>
</tr>
<tr>
<td>mqtt_server_lib_cfg</td>
</tr>
<tr>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>pub_qos2_cq</td>
</tr>
<tr>
<td>secure_conn</td>
</tr>
<tr>
<td>utf8_string</td>
</tr>
<tr>
<td>utf8_strqos</td>
</tr>
</tbody>
</table>
### client_ctx Struct Reference

List of all members.
Public Attributes

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

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
### device_net_services

**Struct Reference**

Abstraction of Network Services on a platform

List of all members.
Public Attributes

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

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

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

**Parameters:**

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

**Returns:**

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

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

Close communication connection

**i32(device_net_services::io_mon)(i32 *recv_cvec, i32 *send_cvec, i32 *rsvd_cvec, u32 wait_secs)**

Monitor activity on communication handles. The routine blocks for the specified period of time to monitor activity, if any, on each of the
communication handle that has been provided in one or more vector sets. At the expiry of the wait time, this function must identify the handles, on which, activities were observed.

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

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

**Parameters:**

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

**Returns:**

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

```markdown
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
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 servers. For setting up a local (UDP) port, use an intended port number.
- **[in] nw_security** Information to establish a secure connection with server. Set it to NULL, if not used. **Information to establish a secure connection.**

**Returns:**

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

Parameters:

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

Returns:

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

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

Parameters:

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

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

Returns:

- on success, the number of bytes sent, 0 on connection reset, otherwise -1
Send data to particular port on the specified network element. The routine blocks till the time, the data has been copied into the network stack for dispatch to the "specified" network entity.

**Parameters:**
- **[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.

```c
u32(* device_net_services::time)(void)
```

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

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

It is the responsibility of the implementation to manage the roll-over problem of the hardware and ensure the integrity of the time value is maintained.
**Returns:**

* time in seconds

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
### mqtt_ack_wlist Struct Reference

List of all members.
### Public Attributes

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

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

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

---

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
### mqtt_packet Struct Reference

**MQTT Packet (MQP) Buffer structure**

List of all members.
### Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u8</td>
<td>msg_type</td>
<td></td>
</tr>
<tr>
<td>u8</td>
<td>fh_byte1</td>
<td></td>
</tr>
<tr>
<td>u16</td>
<td>msg_id</td>
<td></td>
</tr>
<tr>
<td>u8</td>
<td>n.refs</td>
<td></td>
</tr>
<tr>
<td>u8</td>
<td>pad [3]</td>
<td></td>
</tr>
<tr>
<td>u8</td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td>u8</td>
<td>fh_len</td>
<td></td>
</tr>
<tr>
<td>u16</td>
<td>vh_len</td>
<td></td>
</tr>
<tr>
<td>u32</td>
<td>pl_len</td>
<td></td>
</tr>
<tr>
<td>u32</td>
<td>private</td>
<td></td>
</tr>
<tr>
<td>u32</td>
<td>maxlen</td>
<td></td>
</tr>
<tr>
<td>u8 *</td>
<td>buffer</td>
<td></td>
</tr>
<tr>
<td>void(*)</td>
<td>free</td>
<td>(struct mqtt_packet *mqp)</td>
</tr>
</tbody>
</table>

```
struct mqtt_packet * next
```
### Member Data Documentation

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

# users of this msg

u8 `mqtt_packet::offset`

Start of data index

u32 `mqtt_packet::pl_len`

Pay Load Length

u16 `mqtt_packet::vh_len`

Var Header Length

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
### mqtt_server_app_cbs

#### Struct Reference

The Server Daemon API(s)

List of all members.
## Public Attributes

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connect</code></td>
<td>u16(* connect)(const struct utf8_string *client_id, const struct utf8_string *user_name, const struct utf8_string *pass_word, void **app_usr)</td>
</tr>
<tr>
<td><code>publish</code></td>
<td>void(* publish)(const struct utf8_string *topic, const u8 *payload, u32 pay_len, bool dup, u8 qos, bool retain)</td>
</tr>
<tr>
<td><code>disconn</code></td>
<td>void(* disconn)(const void *app_usr, bool due2err)</td>
</tr>
</tbody>
</table>
Member Data Documentation

\texttt{u16(* mqtt_server_app_cbs::connect)(const struct utf8_string *client_id, const struct Connection request from remote client - assess credentials. This routine presents, to the application, the information about the credentials of the remote client that is trying to establish a MQTT connection with the server. The application shall utilize these data in conjunction with its policy to either allow or deny connection to the server.

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

Parameters:

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

Returns:

16bit value for the variable header in the CONNACK message. The MSB in the return value refers to the 8bit parameter of the acknowledge flags and must be set 0. The LSB in the return value refers to the 8bit 'return code' parameter in the CONNACK message and must be set accordingly.
void(* mqtt_server_app_cbs::disconn)(const void *app_usr, bool due2err)

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

**Parameters:**

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

**Returns:**
none

void(* mqtt_server_app_cbs::publish)(const struct utf8_string *topic, const u8 *payload, u32 pay_len, bool dup, u8 qos, bool retain)

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

**Parameters:**

- **[in] topic** UTF8 topic Name for which data has been published
- **[in] data_buf** the published binary data for the topic
- **[in] data_len** the length of the binary data
- **[in] dup** is this a duplicate data from remote client?
- **[in] qos** quality of service of the message / data
- **[in] retain** should the server retain the data?
Returns:
none

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
#include <server_core.h>

List of all members.
### Public Attributes

| void * place_holder |
Detailed Description

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

```c
void* mqtt_server_app_cfg::place_holder
```

Dummy, not used as of now

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
<table>
<thead>
<tr>
<th>Class List</th>
<th>Class Index</th>
<th>Class Members</th>
<th>Public Attributes</th>
</tr>
</thead>
</table>

**mqtt_server_lib_cfg**

**Struct Reference**

The Server Library API(s)

List of all members.
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td><strong>listener_port</strong></td>
</tr>
<tr>
<td>u16</td>
<td><strong>loopback_port</strong></td>
</tr>
<tr>
<td>void *</td>
<td><strong>mutex</strong></td>
</tr>
<tr>
<td>void(*)</td>
<td><strong>mutex_lockin</strong>(void *mutex)</td>
</tr>
<tr>
<td>void(*)</td>
<td><strong>mutex_unlock</strong>(void *mutex)</td>
</tr>
<tr>
<td>i32(*)</td>
<td><strong>debug_printf</strong>(const i8 *format,...)</td>
</tr>
<tr>
<td>bool</td>
<td><strong>aux_debug_en</strong></td>
</tr>
</tbody>
</table>
## Member Data Documentation

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bool</strong></td>
<td><code>mqtt_server_lib_cfg::aux_debug_en</code> Assert to indicate additional debug info</td>
</tr>
<tr>
<td><strong>i32</strong></td>
<td><code>(const i8 *format,...)</code> Debug, mandatory</td>
</tr>
<tr>
<td><strong>u16</strong></td>
<td><code>mqtt_server_lib_cfg::listener_port</code> Port to listen to incoming network connections from the clients</td>
</tr>
<tr>
<td><strong>u16</strong></td>
<td><code>mqtt_server_lib_cfg::loopback_port</code> If the server application has more than one task and / or supports at-least one plug-in, then a non-zero value must be provided. Otherwise, this parameter must be set to zero. This parameter is used by the implementation to synchronize multiple tasks for the network connection.</td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><code>mqtt_server_lib_cfg::mutex</code> For a multi-task enviroment, provide a handle to platform mutex</td>
</tr>
</tbody>
</table>

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/server_pkts.h
#include <server_pkts.h>

List of all members.
Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>u16</td>
<td><code>connect_rx</code></td>
<td><code>(void *ctx_cl, u8 conn_flags, struct utf8_string *const *utf8_vec, void **usr)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>sub_msg_rx</code></td>
<td><code>(void *usr_cl, const struct utf8_strqos *qos_topics, u32 n_topics, u16 msg_id, u8 *acks)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>un_sub_msg</code></td>
<td><code>(void *usr_cl, const struct utf8_string *topics, u32 n_topics, u16 msg_id)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>pub_msg_rx</code></td>
<td><code>(void *usr_cl, const struct utf8_string *topic, const u8 *data_buf, u32 data_len, u16 msg_id, bool dup, enum mqtt_qos qos, bool retain)</code></td>
</tr>
<tr>
<td>bool</td>
<td><code>ack_notify</code></td>
<td><code>(void *usr_cl, u8 msg_type, u16 msg_id)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>on_cl_net_close</code></td>
<td><code>(void *usr_cl, bool due2err)</code></td>
</tr>
<tr>
<td>void</td>
<td><code>on_connack_send</code></td>
<td><code>(void *usr_cl, bool clean_session)</code></td>
</tr>
</tbody>
</table>
**Detailed Description**

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

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

```c
bool(* mqtt_server_msg_cbs::ack_notify)(void *usr_cl, u8 msg_type);
```

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

PUBACK, PUBREC, PUBREL, PUBCOMP

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

### Parameters:

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

### Returns:

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

```c
u16(* mqtt_server_msg_cbs::connect_rx)(void *ctx_cl, u8 connフラg);
```

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

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

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

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

Parameters:

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

Returns:

16bit value for the variable header of the CONNACK message

```c
void(* mqtt_server_msg_cbs::on_cl_net_close)(void *usr_cl, bool due2err)
```

Notify that network connection to client has been closed. This routine is invoked by the LIB to declare to the application that the network connection to a particular client has been terminated and follow-up, if
needed, can now commence. If configured, removal of the client session and / or dispatch of the WILL message, will be typical aspects, among others, to follow-up. The routine aids the application by indicating if an error condition had caused the closure.

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

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

```c
void(* mqtt_server_msg_cbs::on_connack_send)(void *usr_cl, bool clean_session)
```

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

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

```c
bool(* mqtt_server_msg_cbs::pub_msg_rx)(void *usr_cl, const struct Indicate the PUBLISH Message from the client. This routine provides, to the application, the binary data along-with its qualifiers and the topic to which a remote client has published data.

On return from this routine, if the application has found a problem in
processing of the contents of the PUBLISH message, the LIB will simply terminate the associated network connection. Otherwise, depending upon the QoS level of the PUBLISH message, the LIB shall dispatch the ACK (PUBACK or PUBREC) to the client, thereby, relieving the application from this support.

**Parameters:**

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

**Returns:**

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

```cpp
bool(* mqtt_server_msg_cbs::sub_msg_rx)(void *usr_cl, const struct
```

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

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

**Parameters:**
bool(* mqtt_server_msg_cbs::un_sub_msg)(void *usr_cl, const struct UnsubMessage)

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

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

**Parameters:**

- **[in]** `usr_cl` handle to connection context in the application
- **[in]** `topics` an array of topic in the message
- **[in]** `n_topics` the count / number of elements in the array
[in] msg_id  the associated message ID provided by the client

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

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

- D:/my_data/GIT/network_apps/netapps/mqtt/server/server_pkts.h
pub_qos2_cq Struct Reference

List of all members.
### Public Attributes

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

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

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

---

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
### secure_conn Struct

**Reference**

Information to establish a secure connection.

List of all members.
## Public Attributes

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

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

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by [Doxygen](http://www.stackoverflow.com) 1.7.4
utf8_string Struct Reference

#include <mqtt_common.h>

List of all members.
## Public Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>i8 *</td>
<td>buffer</td>
</tr>
<tr>
<td>u16</td>
<td>length</td>
</tr>
</tbody>
</table>
Detailed Description

Description of UTF8 information as used by MQTT Library.
Member Data Documentation

**i8* utf8_string::buffer**

Refers to UTF8 content

**u16 utf8_string::length**

Length of UTF8 content

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

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

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
utf8_strqos Struct

Reference

#include <mqtt_common.h>

List of all members.
<table>
<thead>
<tr>
<th>Public Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>i8 * buffer</td>
</tr>
<tr>
<td>u16 length</td>
</tr>
<tr>
<td>enum mqtt_qos qosreq</td>
</tr>
</tbody>
</table>
Detailed Description

Construct to create Topic to SUBSCRIBE
### Member Data Documentation

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i8* <code>utf8_strqos::buffer</code></td>
<td>Refers to UTF8 content</td>
</tr>
<tr>
<td>u16 <code>utf8_strqos::length</code></td>
<td>Length of UTF8 content</td>
</tr>
<tr>
<td><code>enum mqtt_qos utf8_strqos::qosreq</code></td>
<td>QoS Level for content</td>
</tr>
</tbody>
</table>

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

- D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_comm

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by [doxygen](http://www.doxygen.org) 1.7.4
Class Index

C | D | M | P | S | U

C | client_ctx | mqtt_ack_wlist | mqtt_server_app_cf

D | mqtt_packet | mqtt_server_lib_cf

P | device_net_services | mqtt_server_app_cbs

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
Here is a list of all documented class members with links to the class documentation for each member:

- **a** -
  - accept : [device_net_services](#)
  - ack_notify : [mqtt_server_msg_cbs](#)
  - aux_debug_en : [mqtt_server_lib_cfg](#)

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

- **c** -
  - cipher : [secure_conn](#)
  - close : [device_net_services](#)
  - connect : [mqtt_server_app_cbs](#)
  - connect_rx : [mqtt_server_msg_cbs](#)

- **d** -
  - debug_printf : [mqtt_server_lib_cfg](#)
  - disconn : [mqtt_server_app_cbs](#)

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

- i -

• io_mon: device_net_services

- l -

• length: utf8_string, utf8_strqos
• listen: device_net_services
• listener_port: mqtt_server_lib_cfg
• loopback_port: mqtt_server_lib_cfg

- m -

• maxlen: mqtt_packet
• method: secure_conn
• msg_id: mqtt_packet
• msg_type: mqtt_packet
• mutex: mqtt_server_lib_cfg

- n -

• n_file: secure_conn
• n_refs: mqtt_packet

- o -

• offset: mqtt_packet
• on_cl_net_close: mqtt_server_msg_cbs
• on_connack_send: mqtt_server_msg_cbs
• open: device_net_services

- p -

• pl_len: mqtt_packet
• place_holder: mqtt_server_app_cfg
• pub_msg_rx: mqtt_server_msg_cbs
• publish: mqtt_server_app_cbs
- q -
  - qosreq : utf8_strqos

- r -
  - recv : device_net_services
  - recv_from : device_net_services

- s -
  - send : device_net_services
  - send_dest : device_net_services
  - sub_msg_rx : mqtt_server_msg_cbs

- t -
  - time : device_net_services

- u -
  - un_sub_msg : mqtt_server_msg_cbs

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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MainPage.h</td>
<td></td>
</tr>
<tr>
<td><code>D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h</code></td>
<td></td>
</tr>
<tr>
<td><code>D:/my_data/GIT/network_apps/netapps/mqtt/server/server_core.h</code></td>
<td></td>
</tr>
<tr>
<td><code>D:/my_data/GIT/network_apps/netapps/mqtt/server/server_pkts.h</code></td>
<td></td>
</tr>
</tbody>
</table>

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by [doxygen](http://www.doxygen.nl) 1.7.4
D:/my_data/GIT/network_apps/netapps/mqtt/common/mqtt_common.h

File Reference

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

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

<table>
<thead>
<tr>
<th>struct</th>
<th>mqtt_packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>utf8_string</td>
</tr>
<tr>
<td>struct</td>
<td>mqtt_ack_wlist</td>
</tr>
<tr>
<td>struct</td>
<td>utf8_strqos</td>
</tr>
<tr>
<td>struct</td>
<td>secure_conn</td>
</tr>
<tr>
<td>struct</td>
<td>device_net_services</td>
</tr>
<tr>
<td>struct</td>
<td>pub_qos2_cq</td>
</tr>
<tr>
<td>struct</td>
<td>client_ctx</td>
</tr>
</tbody>
</table>
Defines

#define MQTT_COMMON_VERSTR "1.0.0"
#define MIN(a, b) ((a > b)? b : a)
#define MQTT_CONNECT 0x01
#define MQTT_CONNACK 0x02
#define MQTT_PUBLISH 0x03
#define MQTT_PUBACK 0x04
#define MQTT_PUBREC 0x05
#define MQTT_PUBREL 0x06
#define MQTT_PUBCOMP 0x07
#define MQTT_SUBSCRIBE 0x08
#define MQTT_SUBACK 0x09
#define MQTT_UNSUBSCRIBE 0x0A
#define MQTT_UNSUBACK 0x0B
#define MQTT_PINGREQ 0x0C
#define MQTT_PINGRSP 0x0D
#define MQTT_DISCONNECT 0x0E
#define MAX_FH_LEN 0x05
#define MAX_REMLEN_BYTES (MAX_FH_LEN - 1)
#define MAKE_FH_BYTE1(msg_type,flags) ((u8)((msg_type << 4) | flags))
#define MAKE_FH_FLAGS(bool_dup, enum_qos, bool_retain) ((u8)((bool_dup << 3) | (enum_qos << 1) | bool_retain) & 0xF)
#define QOS_VALUE(enum_qos) (u8)(enum_qos & 0x3)
#define QFL_VALUE 0x80
#define DUP_FLAG_VAL(bool_val) (u8)(bool_val << 3)
#define BOOL_RETAIN(fh_byte1) ((fh_byte1 & 0x1)? true : false)
#define BOOL_DUP(fh_byte1) ((fh_byte1 & 0x8)? true : false)
#define ENUM_QOS(fh_byte1) (enum mqtt_qos)((fh_byte1 & 0x6) >> 1)
#define MSG_TYPE(fh_byte1) (u8)((fh_byte1 & 0xf0) >> 4)
#define MQP_FHEADER_BUF(mqp) (mqp->buffer + mqp->offset)
#define MQP_VHEADER_BUF(mqp) (MQP_FHEADER_BUF(mqp) + mqp->fh_len)
#define MQP_PAYLOAD_BUF(mqp) (MQP_VHEADER_BUF(mqp) + mqp->vh_len)
#define MQP_CONTENT_LEN(mqp) (mqp->fh_len + mqp->vh_len + mqp->pl_len)
#define MQP_FREEBUF_LEN(mqp)
#define MQP_FHEADER_VAL(mqp) (mqp->fh_byte1)
#define MQP_FHEADER_MSG(mqp) (MSG_TYPE(MQP_FHEADER_VAL(mqp))
#define MQP_FHEADER_FLG(mqp) (MSG_FLAGS(MQP_FHEADER_VAL(mqp))
#define DEFINE_MQP_VEC(num_mqp, mqp_vec) static struct mqtt_packet mqp_vec[num_mqp];
#define DEFINE_MQP_BUF_VEC(num_mqp, mqp_vec, buf_len, buf_vec)
#define MQP_PUB_TOP_BUF(mqp) (MQP_VHEADER_BUF(mqp) + 2)
#define MQP_PUB_TOP_LEN(mqp) (mqp->vh_len - 2 - (mqp->msg_id? 2 : 0))
#define MQP_PUB_PAY_BUF(mqp) (mqp->pl_len? MQP_PAYLOAD_BUF(mqp) : NULL)
#define MQP_PUB_PAY_LEN(mqp) (mqp->pl_len)
#define MQP_ERR_NETWORK (-1)
#define MQP_ERR_TIMEOUT (-2)
#define MQP_ERR_NET_OPS (-3)
#define MQP_ERR_FNPARAM (-4)
#define MQP_ERR_PKT_AVL (-5)
#define MQP_ERR_PKT_LEN (-6)
#define MQP_ERR_NOTCONN (-7)
#define MQP_ERR_BADCALL (-8)
#define MQP_ERR_CONTENT (-9)
#define MQP_ERR_LIBQUIT (-10)
#define MQP_ERR_NOT_DEF (-32)
#define DEV_NETCONN_OPT_TCP 0x01
#define DEV_NETCONN_OPT_UDP 0x02
#define DEV_NETCONN_OPT_IP6 0x04
#define DEV_NETCONN_OPT_URL 0x08
#define DEV_NETCONN_OPT_SEC 0x10
#define MAX_PUBREL_INFLT 8
#define KA_TIMEOUT_NONE 0xffffffff
## Typedefs

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void mqp_free (struct mqtt_packet *mqp)</td>
<td>Frees the allocated memory for the MQTT packet.</td>
</tr>
<tr>
<td>void mqp_reset (struct mqtt_packet *mqp)</td>
<td>Resets the MQTT packet to default values.</td>
</tr>
<tr>
<td>void mqp_init (struct mqtt_packet *mqp, u8 offset)</td>
<td>Initializes the MQTT packet with an offset.</td>
</tr>
<tr>
<td>i32 mqp_buf_wr_utf8 (u8 *buf, const struct utf8_string *utf8)</td>
<td>Writes a UTF-8 string to the MQTT packet buffer.</td>
</tr>
<tr>
<td>i32 mqp_buf_tail_wr_remlen (u8 *buf, u32 remlen)</td>
<td>Writes a tail of the MQTT packet buffer with a specified remaining length.</td>
</tr>
<tr>
<td>i32 mqp_buf_rd_remlen (u8 *buf, u32 *remlen)</td>
<td>Reads a tail of the MQTT packet buffer and returns its remaining length.</td>
</tr>
<tr>
<td>i32 mqp_pub_append_topic (struct mqtt_packet *mqp, const struct utf8_string *topic, u16 msg_id)</td>
<td>Appends a topic to the MQTT packet.</td>
</tr>
<tr>
<td>i32 mqp_pub_append_data (struct mqtt_packet *mqp, const u8 *data_buf, u32 data_len)</td>
<td>Appends data to the MQTT packet.</td>
</tr>
<tr>
<td>bool mqp_proc_msg_id_ack_rx (struct mqtt_packet *mqp_raw, bool has_payload)</td>
<td>Processes an MQTT message ID acknowledgment.</td>
</tr>
<tr>
<td>bool mqp_proc_pub_rx (struct mqtt_packet *mqp_raw)</td>
<td>Processes an MQTT publication.</td>
</tr>
<tr>
<td>bool mqp_ack_wlist_append (struct mqtt_ack_wlist *list, struct mqtt_packet *elem)</td>
<td>Appends a MQTT packet to the acknowledgment list.</td>
</tr>
<tr>
<td>struct mqtt_packet * mqp_ack_wlist_remove (struct mqtt_ack_wlist *list, u16 msg_id)</td>
<td>Removes a MQTT packet from the acknowledgment list by ID.</td>
</tr>
<tr>
<td>void mqp_ack_wlist_purge (struct mqtt_ack_wlist *list)</td>
<td>Purges the entire acknowledgment list.</td>
</tr>
<tr>
<td>i32 mqp_prep_fh (struct mqtt_packet *mqp, u8 flags)</td>
<td>Prepares a MQTT packet for transmission with specified flags.</td>
</tr>
<tr>
<td>i32 mqp_recv (i32 net, const struct device_net_services *net_ops, struct mqtt_packet *mqp, u32 wait_secs, bool *timed_out, void *ctx)</td>
<td>Receives an MQTT message from the network.</td>
</tr>
<tr>
<td>void qos2_pub_cq_reset (struct pub_qos2_cq *cq)</td>
<td>Resets a publication quality of service (QoS) 2 control queue.</td>
</tr>
<tr>
<td>bool qos2_pub_cq_logup (struct pub_qos2_cq *cq, u16 msg_id)</td>
<td>Logs up a publication to the QoS 2 control queue.</td>
</tr>
<tr>
<td>bool qos2_pub_cq_unlog (struct pub_qos2_cq *cq, u16 msg_id)</td>
<td>Unlogs a publication from the QoS 2 control queue.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bool qos2_pub_cq_check</td>
<td>(struct pub_qos2_cq *cq, u16 msg_id)</td>
</tr>
<tr>
<td>void cl_ctx_reset</td>
<td>(struct client_ctx *cl_ctx)</td>
</tr>
<tr>
<td>void cl_ctx_timeout_insert</td>
<td>(struct client_ctx **head, struct client_ctx *elem)</td>
</tr>
<tr>
<td>void cl_ctx_remove</td>
<td>(struct client_ctx **head, struct client_ctx *elem)</td>
</tr>
<tr>
<td>void cl_ctx_timeout_update</td>
<td>(struct client_ctx **cl_ctx, u32 now_secs)</td>
</tr>
</tbody>
</table>
Detailed Description

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

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

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

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

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

Value:

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

```c
#define MAX_FH_LEN 0x05
```

MAX Length of Fixed Header

```c
#define MAX_REMLEN_BYTES (MAX_FH_LEN - 1)
```

Max number of bytes in remaining length field

```c
#define MQP_FREEBUF_LEN(mqp)
```

Value:

```c
(mqp->maxlen - mqp->offset -  
MQP_CONTENT_LEN(mqp))
```

```c
#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

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

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

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

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

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

```c
void mqp_free (struct mqtt_packet * mqp)
```

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

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

**Returns:**
none

```c
void mqp_init (struct mqtt_packet * mqp,
               u8 offset)
```

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

Parameters:
- [in] `mqp` packet buffer to be initialized
- [in] `offset` index in buffer to indicate start of the contents

Returns:
none

```c
i32 mqp_prep_fh (struct mqtt_packet * mqp,
                  u8 flags)
```

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

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

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

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

Construct a packet for Message ID enabled ACK received from network Process the raw ACK message information to update the
Warning: This routine does not check for correctness of the input parameters.

Parameters:
- \texttt{mqp\_raw} holds a raw buffer from the network
- \texttt{has\_payload} asserted, if ACK message should have a payload

Returns:
on success, true, otherwise false

\begin{verbatim}
bool mqp\_proc\_pub\_rx ( struct mqtt\_packet * mqp\_raw )
\end{verbatim}

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:
- \texttt{mqp\_raw} holds a raw buffer from the network

Returns:
on success, true, otherwise false

\begin{verbatim}
i32 mqp\_pub\_append\_data ( struct mqtt\_packet * mqp, const u8 * data\_buf, u32 data\_len )
\end{verbatim}

Include payload data for publishing. The payload data is associated
with a topic.

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

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

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

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

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

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

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

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

**Parameters:**
[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 published data.

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

Go to the source code of this file.
Classes

<table>
<thead>
<tr>
<th>struct</th>
<th>mqtt_server_app_cbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>mqtt_server_app_cfg</td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>i32</th>
<th><code>mqtt_server_topic_enroll</code> (const void *app_hnd, const struct utf8_string *topic, enum mqtt_qos qos)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td><code>mqtt_server_topic_disenroll</code> (const void *app_hnd, const struct utf8_string *topic)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_app_pub_send</code> (const struct utf8_string *topic, const u8 *data_buf, u32 data_len, enum mqtt_qos qos, bool retain)</td>
</tr>
<tr>
<td>void *</td>
<td><code>mqtt_server_app_register</code> (const struct mqtt_server_app_cbs *cbs, const i8 *name)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_init</code> (const struct mqtt_server_lib_cfg *lib_cfg, const struct mqtt_server_app_cfg *app_cfg)</td>
</tr>
</tbody>
</table>
Detailed Description

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

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

The server is targeted to conform to MQTT 3.1.1 specification.

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

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

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

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

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

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

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

**Note:**

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

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

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct mqtt_server_msg_cbs</td>
<td></td>
</tr>
<tr>
<td>struct mqtt_server_lib_cfg</td>
<td></td>
</tr>
</tbody>
</table>
Defines

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

<table>
<thead>
<tr>
<th>i32</th>
<th><code>mqtt_vh_msg_send</code></th>
<th>(void *ctx_cl, u8 msg_type, enum <code>mqtt_qos</code> qos, bool has_vh, u16 vh_data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td><code>mqtt_vh_msg_send_locked</code></td>
<td>(void *ctx_cl, u8 msg_type, enum <code>mqtt_qos</code> qos, bool has_vh, u16 vh_data)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_pub_dispatch</code></td>
<td>(void *ctx_cl, struct <code>mqtt_packet</code> *mqp, bool dup)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_pub_dispatch_locked</code></td>
<td>(void *ctx_cl, struct <code>mqtt_packet</code> *mqp, bool dup)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_run</code></td>
<td>(u32 wait_secs)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_register_net_svc</code></td>
<td>(const struct <code>device_net_services</code> *net)</td>
</tr>
<tr>
<td>i32</td>
<td><code>mqtt_server_lib_init</code></td>
<td>(const struct <code>mqtt_server_lib_cfg</code> *cfg, const struct <code>mqtt_server_msg_cbs</code> *cbs)</td>
</tr>
</tbody>
</table>
Detailed Description

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

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

The library is targeted to conform to MQTT 3.1.1 specification.

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

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

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

**Note:**
Any future extensions & development must follow the following guidelines. A new API or an extension to the existing API a) must be rudimentary b) must not imply a rule or policy (including a state
machine) b) must ensure simple design and implementation
Here is a list of all documented file members with links to the documentation:

- d -

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

- m -

- MAX_FH_LEN : mqtt_common.h
- MAX_REMLEN_BYTES : mqtt_common.h
- MQ_CONN_UTF8_BUF : server_pkts.h
- MQ_CONN_UTF8_LEN : server_pkts.h
- MQC_UTF8_CLIENTID : server_pkts.h
- MQC_UTF8_PASSWORD : server_pkts.h
- MQC_UTF8_USERNAME : server_pkts.h
- MQC_UTF8_WILL_MSG : server_pkts.h
- MQC_UTF8_WILL_TOP : server_pkts.h
- mqp_buf_rd_remlen() : mqtt_common.h
- mqp_buf_tail_wr_remlen() : mqtt_common.h
- mqp_buf_wr_utf8() : mqtt_common.h
- MQP_ERR_BADCALL : mqtt_common.h
- MQP_ERR_CONTENT : mqtt_common.h
- MQP_ERR_FNPARAM : mqtt_common.h
- MQP_ERR_LIBQUIT: mqtt_common.h
- MQP_ERR_NET_OPS: mqtt_common.h
- MQP_ERR_NETWORK: mqtt_common.h
- MQP_ERR_NOT_DEF: mqtt_common.h
- MQP_ERR_NOTCONN: mqtt_common.h
- MQP_ERR_PKT_AVL: mqtt_common.h
- MQP_ERR_PKT_LEN: mqtt_common.h
- MQP_ERR_TIMEOUT: mqtt_common.h
- mqp_free(): mqtt_common.h
- mqp_init(): mqtt_common.h
- mqp_prep_fh(): mqtt_common.h
- mqp_proc_msg_id_ack_rx(): mqtt_common.h
- mqp_proc_pub_rx(): mqtt_common.h
- mqp_pub_append_data(): mqtt_common.h
- mqp_pub_append_topic(): mqtt_common.h
- MQP_PUB_PAY_BUF: mqtt_common.h
- MQP_PUB_PAY_LEN: mqtt_common.h
- MQP_PUB_TOP_BUF: mqtt_common.h
- MQP_PUB_TOP_LEN: mqtt_common.h
- mqp_reset(): mqtt_common.h
- MQP_SERVER_RX_LEN: server_pkts.h
- MQTT_COMMON_VERSTR: mqtt_common.h
- MQTT_CONNECT: mqtt_common.h
- mqtt_qos: mqtt_common.h
- MQTT_QOS0: mqtt_common.h
- MQTT_QOS1: mqtt_common.h
- MQTT_QOS2: mqtt_common.h
- mqtt_server_app_pub_send(): server_core.h
- mqtt_server_app_register(): server_core.h
- mqtt_server_init(): server_core.h
- mqtt_server_lib_init(): server_pkts.h
- mqtt_server_pub_dispatch(): server_pkts.h
- mqtt_server_pub_dispatch_locked(): server_pkts.h
- mqtt_server_register_net_svc(): server_pkts.h
- mqtt_server_run(): server_pkts.h
- mqtt_server_topic_disenroll(): server_core.h
- mqtt_server_topic_enroll(): server_core.h
- mqtt_vh_msg_send(): server_pkts.h
- mqtt_vh_msg_send_locked(): server_pkts.h
- q -

- QFL_VALUE : `mqtt_common.h`

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
**client_ctx Member List**

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

<table>
<thead>
<tr>
<th>Member</th>
<th>Definition</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>ip_length</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>ka_secs</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>net</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>next</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>remote_ip</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>timeout</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
<tr>
<td>usr</td>
<td>(defined in <code>client_ctx</code>)</td>
<td><code>client_ctx</code></td>
</tr>
</tbody>
</table>
Go to the documentation of this file.

```c
00001 /************************************************************
00002 ******************************************
00003 * Copyright (C) 2014 Texas Instruments Incorporated
00004 *
00005 * All rights reserved. Property of Texas Instruments Incorporated.
00006 * Restricted rights to use, duplicate or disclose this code are
00007 * granted through contract.
00008 *
00009 * The program may not be used without the written permission of
00010 * Texas Instruments Incorporated or against the terms and conditions
00011 * stipulated in the agreement under which this program has been supplied,
00012 * and under no circumstances can it be used with non-TI connectivity device.
00013 *
00014 **************************************************************
00015 **************************************************************/
00016 */
```
This module outlines the interfaces that are common to both client and server components. The applications are not expected to utilize the services outlined in this module.

```
#ifndef __MQTT_COMMON_H__
#define __MQTT_COMMON_H__

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

#define MQTT_COMMON_VERSTR "1.0.0"

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

#define MIN(a, b) ((a > b)? b : a)

#define MQTT_CONNECT 0x01
#define MQTT_CONNACK 0x02
#define MQTT_PUBLISH 0x03
#define MQTT_PUBACK 0x04
#define MQTT_PUBREC 0x05
#define MQTT_PUBREL 0x06
#define MQTT_PUBCOMP 0x07
#define MQTT_SUBSCRIBE 0x08
#define MQTT_SUBACK 0x09
#define MQTT_UNSUBSCRIBE 0x0A
#define MQTT_UNSUBACK 0x0B
```
```c
#define MQTT_PINGREQ 0x0C
#define MQTT_PINGRSP 0x0D
#define MQTT_DISCONNECT 0x0E

#define MAX_FH_LEN 0x05
#define MAX_REMLEN_BYTES (MAX_FH_LEN - 1)

#define MAKE_FH_BYTE1(msg_type, flags) (u8)((msg_type << 4) | flags)
#define MAKE_FH_FLAGS(bool_dup, enum_qos, bool_retain) (u8)(((bool_dup << 3) | (enum_qos << 1) | bool_retain) & 0xF)
#define QOS_VALUE(enum_qos) (u8)(enum_qos & 0x3)
#define QFL_VALUE 0x80
#define DUP_FLAG_VAL(bool_val) (u8)(bool_val << 3)
#define BOOL_RETAIN(fh_byte1) ((fh_byte1 & 0x1)? true : false)
#define BOOL_DUP(fh_byte1) ((fh_byte1 & 0x8)? true : false)
#define ENUM_QOS(fh_byte1) (enum mqtt_qos)((fh_byte1 & 0x6) >> 1)
#define MSG_TYPE(fh_byte1) (u8)((fh_byte1 & 0xf0) >> 4)

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

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

    return n;
}

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

    return 2;
}

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

    return 2;
}

struct mqtt_packet {
    u8 msg_type;
    u8 fh_byte1;
    u16 msg_id;
}
00139 u8 n_refs;
00140 u8 pad[3];
00141
00142 u8 offset;
00143 u8 fh_len;
00144 u16 vh_len;
00145 u32 pl_len;
00147 u32 private;
00148
00149 u32 maxlen;
00150 u8 *buffer;
00153 void (*free)(struct mqtt_packet *mqp);
00154
00155 struct mqtt_packet *next;
00156 }
00157
00160 #define MQP_FHEADER_BUF(mqp)  (mqp->buffer + mqp->offset)
00161 #define MQP_VHEADER_BUF(mqp)  (MQP_FHEADER_BUF(mqp) + mqp->fh_len)
00162 #define MQP_PAYLOAD_BUF(mqp)  (MQP_VHEADER_BUF(mqp) + mqp->vh_len)
00163
00164 #define MQP_CONTENT_LEN(mqp)  (mqp->fh_len + mqp->vh_len + mqp->pl_len)
00165 #define MQP_FREEBUF_LEN(mqp)  (mqp->maxlen - mqp->offset - MQP_CONTENT_LEN(mqp))
00166
00167 #define MQP_FHEADER_VAL(mqp)  (mqp->fh_byte1)
00168 #define MQP_FHEADER_MSG(mqp)  (MSG_TYPE(MQP_FHEADER_VAL(mqp)))
00169 #define MQP_FHEADER_FLG(mqp)  (MSG_FLAGS(MQP_FHEADER_VAL(mqp)))
```c
#define DEFINE_MQP_VEC(num_mqp, mqp_vec) \
  static struct mqtt_packet mqp_vec[num_mqp];
#define DEFINE_MQP_BUF_VEC(num_mqp, mqp_vec, buf_len, buf_vec) DEFINE_MQP_VEC(num_mqp, mqp_vec); \
  static u8 buf_vec[num_mqp][buf_len];

/*------------------------------------------
---------------------------
* Helper MACROS for PUBLISH-RX Message Processing
---------------------------
*------------------------------------------*/
#define MQP_PUB_TOP_BUF(mqp) (MQP_VHEADER_BUF(mqp) + 2)
#define MQP_PUB_TOP_LEN(mqp) (mqp->vh_len - 2 - (mqp->msg_id ? 2 : 0))
#define MQP_PUB_PAY_BUF(mqp) (mqp->pl_len ? MQP_PAYLOAD_BUF(mqp) : NULL)
#define MQP_PUB_PAY_LEN(mqp) (mqp->pl_len)
#define WILL_RETAIN_VAL 0x20
#define WILL_CONFIG_VAL 0x04
#define CLEAN_START_VAL 0x02
#define USER_NAME_OPVAL 0x80
#define PASS_WORD_OPVAL 0x40
```
#define MQP_ERR_NETWORK  (-1)
#define MQP_ERR_TIMEOUT  (-2)
#define MQP_ERR_NET_OPS   (-3)
#define MQP_ERR_FNPARAM   (-4)
#define MQP_ERR_PKT_AVL   (-5)
#define MQP_ERR_PKT_LEN   (-6)
#define MQP_ERR_NOTCONN   (-7)
#define MQP_ERR_BADCALL   (-8)
#define MQP_ERR_CONTENT   (-9)
#define MQP_ERR_LIBQUIT   (-10)
#define MQP_ERR_NOT_DEF   (-32)

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

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

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

int mqp_buf_wr_utf8(u8 *buf, const struct utf8_string *utf8);

int mqp_buf_tail_wr_remlen(u8 *buf, u32 remlen);

int mqp_buf_rd_remlen(u8 *buf, u32 *remlen);

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

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

bool mqp_proc_msg_id_ack_rx(struct mqtt_packet *mqp_raw, bool has_payload);

bool mqp_proc_pub_rx(struct mqtt_packet *mqp_raw);

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

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

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

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

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

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

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

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

```c
enum mqtt_qos {
    MQTT_QOS0,
    MQTT_QOS1,
    MQTT_QOS2
};
```

```c
struct utf8_str_qos {
    i8 *buffer;
    u16 length;
    enum mqtt_qos qosreq;
};
```

```c
struct secure_conn {
    void *method;
    void *cipher;
    u32 n_file;
    char **files;
};
```
struct device_net_services {
    #define DEV_NETCONN_OPT_TCP 0x01
    #define DEV_NETCONN_OPT_UDP 0x02
    #define DEV_NETCONN_OPT_IP6 0x04
    #define DEV_NETCONN_OPT_URL 0x08
    #define DEV_NETCONN_OPT_SEC 0x10

    i32 (*open)(u32 nwconn_opts, const i8 *server_addr, u16 port_number,
                const struct secure_conn *nw_security);

    i32 (*send)(i32 comm, const u8 *buf, u32 len, void *ctx);

    i32 (*recv)(i32 comm, u8 *buf, u32 len, u32 wait_secs, bool *err_timeo, void *ctx);

    i32 (*send_dest)(i32 comm, const u8 *buf, u32 len, u16 dest_port,
                     const i8 *dest_ip, u32 ip_len);

    i32 (*recv_from)(i32 comm, u8 *buf, u32 len, u16 *from_port,
                     i8 *from_ip, u32 *ip_len);

    i32 (*close)(i32 comm);

    i32 (*listen)(u32 nwconn_opts, u16 port_number,
                  const struct secure_conn *nw_security);

}
i32 (*accept)(i32 listen, i8 *client_ip, u32 *ip_length);
i32 (*io_mon)(i32 *recv_cvec, i32 *send_cvec, i32 *rsvd_cvec, u32 wait_secs);
u32 (*time)(void);

/* device_net_services */
/* Receive data from the specified network and read into the 'mqp' */

i32 mqp_recv(i32 net, const struct device_net_services *net_ops, struct mqtt_packet *mqp, u32 wait_secs, bool *timed_out, void *ctx);

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

struct pub_qos2_cq { /* Circular Queue CQ to track QoS2 PUB RX messages */
    u16 id_vec[MAX_PUBREL_INFLT]; /* Vector to store RX Message-IDs */
    u8 n_free; /* Num of free elements in vector */
/* Index to Read next Message-ID */

u8 rd_idx;

/* Index to Write next Message-ID */

u8 wr_idx;

};

/* Reset the specified Circular Queue (CQ) */

void qos2_pub_cq_reset(struct pub_qos2_cq *cq);

/* Append the message-id into the CQ tail. Return true on success, else false */

bool qos2_pub_cq_logup(struct pub_qos2_cq *cq, u16 msg_id);

/* Remove the message-id from the CQ head. Return true on success, else false */

bool qos2_pub_cq_unlog(struct pub_qos2_cq *cq, u16 msg_id);

/* Is the message-id available in the CQ ? Return true on success, else false */

bool qos2_pub_cq_check(struct pub_qos2_cq *cq, u16 msg_id);

/* Get the count of message-ID(s) available in the CQ */

static inline i32 qos2_pub_cq_count(struct pub_qos2_cq *cq)

{ 
  return MAX_PUBREL_INFLT - cq->n_free;
}

struct client_ctx {

}
void *usr; /* Client Usr */
i32 net; /* Socket HND */
i8 remote_ip[16];
u32 ip_length;
u32 timeout;
u16 ka_secs;
u32 flags;
struct client_ctx *next;
}

void cl_ctx_reset(struct client_ctx *cl_ctx);
void cl_ctx_timeout_insert(struct client_ctx **head, struct client_ctx *elem);
void cl_ctx_remove(struct client_ctx **head, struct client_ctx *elem);
#define KA_TIMEOUT_NONE 0xffffffff /* Different than KA SECS = 0 */
void cl_ctx_timeout_update(struct client_ctx *cl_ctx, u32 now_secs);

#endif

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
device_net_services Member List

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

- `accept` `device_net_services`
- `close` `device_net_services`
- `io_mon` `device_net_services`
- `listen` `device_net_services`
- `open` `device_net_services`
- `recv` `device_net_services`
- `recv_from` `device_net_services`
- `send` `device_net_services`
- `send_dest` `device_net_services`
- `time` `device_net_services`
mqtt_ack_wlist Member List

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

- **head** (defined in `mqtt_ack_wlist`) `mqtt_ack_wlist`
- **tail** (defined in `mqtt_ack_wlist`) `mqtt_ack_wlist`
### mqtt_packet Member List

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

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

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

- `connect mqtt_server_app_cbs`
- `disconn mqtt_server_app_cbs`
- `publish mqtt_server_app_cbs`

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
Go to the documentation of this file.

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

#include "server_pkts.h"

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

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

i32 mqtt_server_topic_disenroll(const void *app_hnd, const struct utf8_string *topic);

i32 mqtt_server_app_pub_send(const struct utf8_string *topic, const struct utf8_string *payload);
```

typedef

const u8 *data_
buf, u32 data_len,
enum mqtt_qos q
os, bool retain);

void *mqtt_server_app_register(const struct mqtt_server_app_cbs *cbs,
const i8 *name);

struct mqtt_server_app_cfg {
    void *place_holder;
};

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

/* End of server_daemon */

#endif

 Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
mqtt_server_app_cfg Member List

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

place_holder mqtt_server_app_cfg
### mqtt_server_lib_cfg Member List

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

<table>
<thead>
<tr>
<th>Member</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>aux_debug_en</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>debug_printf</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>listener_port</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>loopback_port</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>mutex</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>mutex_lockin</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
<tr>
<td>mutex_unlock</td>
<td><code>mqtt_server_lib_cfg</code></td>
</tr>
</tbody>
</table>

`mutex_lockin` (defined in `mqtt_server_lib_cfg`)

`mutex_unlock` (defined in `mqtt_server_lib_cfg`)

---

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
Go to the documentation of this file.

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

#include "mqtt_common.h"

/ *------------------------------------------
*  Note: Do not create additional dependency of this file on any header other
*  than mqtt_common.h. Specifically, server_pkts.[hc] in conjunction with the
*  mqtt_common.[hc] files must be facilitate d to create a stand-alone library.
*------------------------------------------*/

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

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

#define MQC_UTF8_CLIENTID(utf8_vec) (utf8_vec[0])
#define MQC_UTF8_WILL_TOP(utf8_vec) (utf8_vec[1])
#define MQC_UTF8_WILL_MSG(utf8_vec) (utf8_vec[2])
#define MQC_UTF8_USERNAME(utf8_vec) (utf8_vec[3])
#define MQC_UTF8_PASSWORD(utf8_vec) (utf8_vec[4])
#define MQC_CLIENTID_BUF(utf8_vec) MQ_CONN_UTF8_BUF(MQC_UTF8_CLIENTID(utf8_vec))
#define MQC_CLIENTID_LEN(utf8_vec) MQ_CONN_UTF8_LEN(MQC_UTF8_CLIENTID(utf8_vec))
```c
#define MQC_WILL_TOP_BUF(utf8_vec) MQ_CONN_UTF8_BUF(MQC_UTF8_WILL_TOP(utf8_vec))
#define MQC_WILL_TOP_LEN(utf8_vec) MQ_CONN_UTF8_LEN(MQC_UTF8_WILL_TOP(utf8_vec))
#define MQC_WILL_MSG_BUF(utf8_vec) MQ_CONN_UTF8_BUF(MQC_UTF8_WILL_MSG(utf8_vec))
#define MQC_WILL_MSG_LEN(utf8_vec) MQ_CONN_UTF8_LEN(MQC_UTF8_WILL_MSG(utf8_vec))
#define MQC_USERNAME_BUF(utf8_vec) MQ_CONN_UTF8_BUF(MQC_UTF8_USERNAME(utf8_vec))
#define MQC_USERNAME_LEN(utf8_vec) MQ_CONN_UTF8_LEN(MQC_UTF8_USERNAME(utf8_vec))
#define MQC_PASSWORD_BUF(utf8_vec) MQ_CONN_UTF8_BUF(MQC_UTF8_PASSWORD(utf8_vec))
#define MQC_PASSWORD_LEN(utf8_vec) MQ_CONN_UTF8_LEN(MQC_UTF8_PASSWORD(utf8_vec))

#ifndef CFG_SR_MAX_MQP_RX_LEN
#define MQP_SERVER_RX_LEN 1024
#else
#define MQP_SERVER_RX_LEN CFG_SR_MAX_MQP_RX_LEN
#endif

i32 mqtt_vh_msg_send(void *ctx_cl, u8 msg_type, enum mqtt_qos qos, bool has_vh, u16 vh_data);
i32 mqtt_vh_msg_send_locked(void *ctx_cl, u8 msg_type, enum mqtt_qos qos, bool has_vh, u16 vh_data);
```
i32 mqtt_server_pub_dispatch(void *ctx_cl, struct mqtt_packet *mqp, bool dup);

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

i32 mqtt_server_run(u32 wait_secs);

i32 mqtt_server_register_net_svc(const struct device_net_services *net);

struct mqtt_server_msg_cbs {
    u16 (*connect_rx)(void *ctx_cl, u8 conn_flags,
    struct utf8_string * const *utf8_vec, void **usr);

    bool (*sub_msg_rx)(void *usr_cl, const struct utf8_strqos *qos_topics,
    u32 n_topics, u16 msg_id, u8 *acks);

    bool (*un_sub_msg)(void *usr_cl, const struct utf8_string *topics,
    u32 n_topics, u16 msg_id);

    bool (*pub_msg_rx)(void *usr_cl, const struct utf8_string *topic,
    const u8 *data_buf, u32 data_len, u16 msg_id, bool dup, enum mqtt_qos qos, bool retain);
bool (*ack_notify)(void *usr_cl, u8 msg_type, u16 msg_id);
void (*on_cl_net_close)(void *usr_cl, bool due2err);
void (*on_connack_send)(void *usr_cl, bool clean_session);
### mqtt_server_msg_cbs Member List

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

<table>
<thead>
<tr>
<th>Member</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack_notify</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>connect_rx</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>on_cl_net_close</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>on_connack_send</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>pub_msg_rx</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>sub_msg_rx</td>
<td>mqtt_server_msg_cbs</td>
</tr>
<tr>
<td>un_sub_msg</td>
<td>mqtt_server_msg_cbs</td>
</tr>
</tbody>
</table>
pub_qos2_cq Member List

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

- `id_vec` (defined in `pub_qos2_cq`) `pub_qos2_cq`
- `n_free` (defined in `pub_qos2_cq`) `pub_qos2_cq`
- `rd_idx` (defined in `pub_qos2_cq`) `pub_qos2_cq`
- `wr_idx` (defined in `pub_qos2_cq`) `pub_qos2_cq`
secure_conn Member List

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

| cipher | secure_conn |
| files  | secure_conn |
| method | secure_conn |
| n_file | secure_conn |

Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4
# utf8_string Member List

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

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buffer utf8_string</code></td>
<td></td>
</tr>
<tr>
<td><code>length utf8_string</code></td>
<td></td>
</tr>
</tbody>
</table>
utf8_strqos Member List

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

- buffer utf8_strqos
- length utf8_strqos
- qosreq utf8_strqos
- a -
  - accept: device_net_services
  - ack_notify: mqtt_server_msg_cbs
  - aux_debug_en: mqtt_server_lib_cfg

- b -
  - buffer: mqtt_packet, utf8_strqos, utf8_string

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

- d -
  - debug_printf: mqtt_server_lib_cfg
  - disconn: mqtt_server_app_cbs

- f -
  - fh_byte1: mqtt_packet
  - fh_len: mqtt_packet
  - files: secure_conn
- i -
  - io_mon : device_net_services

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

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

- n -
  - n_file : secure_conn
  - n_refs : mqtt_packet

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

- p -
  - pl_len : mqtt_packet
  - place_holder : mqtt_server_app_cfg
  - pub_msg_rx : mqtt_server_msg_cbs
  - publish : mqtt_server_app_cbs
qosreq : utf8_strqos

recv : device_net_services
recv_from : device_net_services

send : device_net_services
send_dest : device_net_services
sub_msg_rx : mqtt_server_msg_cbs

time : device_net_services

time : device_net_services

un_sub_msg : mqtt_server_msg_cbs

vh_len : mqtt_packet
### File List

#### MainPage.h

00001

---

*Generated on Mon Nov 17 2014 12:12:08 for MQTT Server by doxygen 1.7.4*
### Files

- `mqp_buf_rd_remlen()`: `mqtt_common.h`
- `mqp_buf_tail_wr_remlen()`: `mqtt_common.h`
- `mqp_buf_wr_utf8()`: `mqtt_common.h`
- `mqp_free()`: `mqtt_common.h`
- `mqp_init()`: `mqtt_common.h`
- `mqp_prep_fh()`: `mqtt_common.h`
- `mqp_proc_msg_id_ack_rx()`: `mqtt_common.h`
- `mqp_proc_pub_rx()`: `mqtt_common.h`
- `mqp_pub_append_data()`: `mqtt_common.h`
- `mqp_pub_append_topic()`: `mqtt_common.h`
- `mqp_reset()`: `mqtt_common.h`
- `mqtt_server_app_pub_send()`: `server_core.h`
- `mqtt_server_app_register()`: `server_core.h`
- `mqtt_server_init()`: `server_core.h`
- `mqtt_server_lib_init()`: `server_pkts.h`
- `mqtt_server_pub_dispatch()`: `server_pkts.h`
- `mqtt_server_pub_dispatch_locked()`: `server_pkts.h`
- `mqtt_server_register_net_svc()`: `server_pkts.h`
- `mqtt_server_run()`: `server_pkts.h`
- `mqtt_server_topic_disenroll()`: `server_core.h`
- `mqtt_server_topic_enroll()`: `server_core.h`
- `mqtt_vh_msg_send()`: `server_pkts.h`
- `mqtt_vh_msg_send_locked()`: `server_pkts.h`
MQTT Server 1.0.0

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

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

- m -

- MAX_FH_LEN : mqtt_common.h
- MAX_REMLEN_BYTES : mqtt_common.h
- MQ_CONN_UTF8_BUF : server_pkts.h
- MQ_CONN_UTF8_LEN : server_pkts.h
- MQC_UTF8_CLIENTID : server_pkts.h
- MQC_UTF8_PASSWORD : server_pkts.h
- MQC_UTF8_USERNAME : server_pkts.h
- MQC_UTF8_WILL_MSG : server_pkts.h
- MQC_UTF8_WILL_TOP : server_pkts.h
- MQP_ERR_BADCALL : mqtt_common.h
- MQP_ERR_CONTENT : mqtt_common.h
- MQP_ERR_FNPARAM : mqtt_common.h
- MQP_ERR_LIBQUIT : mqtt_common.h
- MQP_ERR_NET_OPS : mqtt_common.h
- MQP_ERR_NETWORK : mqtt_common.h
- MQP_ERR_NOT_DEF : mqtt_common.h
• MQP_ERR_NOTCONN : mqtt_common.h
• MQP_ERR_PKT_AVL : mqtt_common.h
• MQP_ERR_PKT_LEN : mqtt_common.h
• MQP_ERR_TIMEOUT : mqtt_common.h
• MQP_PUB_PAY_BUF : mqtt_common.h
• MQP_PUB_PAY_LEN : mqtt_common.h
• MQP_PUB_TOP_BUF : mqtt_common.h
• MQP_PUB_TOP_LEN : mqtt_common.h
• MQP_SERVER_RX_LEN : server_pkts.h
• MQTT_COMMON_VERSTR : mqtt_common.h
• MQTT_CONNECT : mqtt_common.h

- q -

• QFL_VALUE : mqtt_common.h