MQTTClient.h File Reference

```
#include "MQTTProperties.h"
#include "MQTTReasonCodes.h"
#include "MQTTSubscribeOpts.h"
#include "MQTTClientPersistence.h"
```

Go to the source code of this file.

Data Structures

struct	**MQTTClient_init_options**
struct	**MQTTClient_message**
struct	**MQTTClient_createOptions**
struct	**MQTTClient_willOptions**
struct	**MQTTClient_SSLOptions**
struct	**MQTTClient_connectOptions**
struct	**MQTTClient_nameValue**
struct	**MQTTResponse**

Macros

#define	**MQTTCLIENT_SUCCESS**	0
#define	**MQTTCLIENT_FAILURE**	-1
#define	**MQTTCLIENT_DISCONNECTED**	-3
#define	**MQTTCLIENT_MAX_MESSAGES_INFLIGHT**	-4
#define	**MQTTCLIENT_BAD_UTF8_STRING**	-5
#define	**MQTTCLIENT_NULL_PARAMETER**	-6
#define	**MQTTCLIENT_TOPICNAME_TRUNCATED**	-7
#define	**MQTTCLIENT_BAD_STRUCTURE**	-8
#define	**MQTTCLIENT_BAD_QOS**	-9
#define	**MQTTCLIENT_SSL_NOT_SUPPORTED**	-10
#define	**MQTTCLIENT_BAD_MQTT_VERSION**	-11
#define	**MQTTCLIENT_BAD_PROTOCOL**	-14
#define	**MQTTCLIENT_BAD_MQTT_OPTION**	-15
#define	**MQTTCLIENT_WRONG_MQTT_VERSION**	-16
#define	**MQTTVERSION_DEFAULT**	0
#define	**MQTTVERSION_3_1**	3
#define	**MQTTVERSION_3_1_1**	4
#define	**MQTTVERSION_5**	5
#define	**MQTT_BAD_SUBSCRIBE**	0x80
#define	**MQTTClient_init_options_initializer**	{ {'M', 'Q', 'T', 'G'}, 0, 0 }
#define	**MQTTClient_message_initializer**	{ {'M', 'Q', 'T', 'M'}, 1, 0, NULL, 0, 0, 0, 0,

	MQTTProperties_initializer }
#define	**MQTTClient_createOptions_initializer** { {'M', 'Q', 'C', 'O'}, **MQTTVERSION_DEFAULT** }
#define	**MQTTClient_willOptions_initializer** { {'M', 'Q', 'T', 'W'}, 1, NULL, NULL, 0, 0, {0, NULL} }
#define	**MQTT_SSL_VERSION_DEFAULT** 0
#define	**MQTT_SSL_VERSION_TLS_1_0** 1
#define	**MQTT_SSL_VERSION_TLS_1_1** 2
#define	**MQTT_SSL_VERSION_TLS_1_2** 3
#define	**MQTTClient_SSLOptions_initializer** { {'M', 'Q', 'T', 'S'}, 3, NULL, NULL, NULL, NULL, NULL, 1, **MQTT_SSL_VERSION_DEFAULT**, 0, NULL, NULL, NULL }
#define	**MQTTClient_connectOptions_initializer** { {'M', 'Q', 'T', 'C'}, 6, 60, 1, 1, NULL, NULL, NULL, 30, 0, NULL, 0, NULL, **MQTTVERSION_DEFAULT**, {NULL, 0, 0}, {0, NULL}, -1, 0}
#define	**MQTTClient_connectOptions_initializer5** { {'M', 'Q', 'T', 'C'}, 6, 60, 0, 1, NULL, NULL, NULL, 30, 0, NULL, 0, NULL, **MQTTVERSION_5**, {NULL, 0, 0}, {0, NULL}, -1, 1}
#define	**MQTTResponse_initializer** {1, **MQTTREASONCODE_SUCCESS**, 0, NULL, NULL}

Typedefs

typedef void *	**MQTTClient**
typedef int	**MQTTClient_deliveryToken**
typedef int	**MQTTClient_token**
typedef int	**MQTTClient_messageArrived**(void *context, char *topicName, int topicLen, **MQTTClient_message** *message)
typedef void	**MQTTClient_deliveryComplete**(void *context, **MQTTClient_deliveryToken** dt)
typedef void	**MQTTClient_connectionLost**(void *context, char *cause)
typedef void	**MQTTClient_disconnected**(void *context, **MQTTProperties** *properties, enum **MQTTReasonCodes** reasonCode)
typedef void	**MQTTClient_published**(void *context, int dt, int packet_type, **MQTTProperties** *properties, enum **MQTTReasonCodes** reasonCode)
typedef struct **MQTTResponse**	**MQTTResponse**
typedef void	**MQTTClient_traceCallback**(enum **MQTTCLIENT_TRACE_LEVELS** level, char *message)

Enumerations

enum **MQTTCLIENT_TRACE_LEVELS** {
MQTTCLIENT_TRACE_MAXIMUM = 1, **MQTTCLIENT_TRACE_MEDIUM**,
MQTTCLIENT_TRACE_MINIMUM, **MQTTCLIENT_TRACE_PROTOCOL**,
MQTTCLIENT_TRACE_ERROR, **MQTTCLIENT_TRACE_SEVERE**, **MQTTCLIENT_TRACE_FATAL**
}

Functions

void	**MQTTClient_global_init** (**MQTTClient_init_options** *inits)
int	**MQTTClient_setCallbacks** (**MQTTClient** handle, void *context, **MQTTClient_connectionLost** *cl, **MQTTClient_messageArrived** *ma,

	MQTTClient_deliveryComplete *dc)
int	**MQTTClient_setDisconnected** (**MQTTClient** handle, void *context, **MQTTClient_disconnected** *co)
int	**MQTTClient_setPublished** (**MQTTClient** handle, void *context, **MQTTClient_published** *co)
int	**MQTTClient_create** (**MQTTClient** *handle, const char *serverURI, const char *clientId, int persistence_type, void *persistence_context)
int	**MQTTClient_createWithOptions** (**MQTTClient** *handle, const char *serverURI, const char *clientId, int persistence_type, void *persistence_context, **MQTTClient_createOptions** *options)
MQTTClient_nameValue *	**MQTTClient_getVersionInfo** (void)
int	**MQTTClient_connect** (**MQTTClient** handle, **MQTTClient_connectOptions** *options)
void	**MQTTResponse_free** (**MQTTResponse** response)
MQTTResponse	**MQTTClient_connect5** (**MQTTClient** handle, **MQTTClient_connectOptions** *options, **MQTTProperties** *connectProperties, **MQTTProperties** *willProperties)
int	**MQTTClient_disconnect** (**MQTTClient** handle, int timeout)
int	**MQTTClient_disconnect5** (**MQTTClient** handle, int timeout, enum **MQTTReasonCodes** reason, **MQTTProperties** *props)
int	**MQTTClient_isConnected** (**MQTTClient** handle)
int	**MQTTClient_subscribe** (**MQTTClient** handle, const char *topic, int qos)
MQTTResponse	**MQTTClient_subscribe5** (**MQTTClient** handle, const char *topic, int qos, **MQTTSubscribe_options** *opts, **MQTTProperties** *props)
int	**MQTTClient_subscribeMany** (**MQTTClient** handle, int count, char *const *topic, int *qos)
MQTTResponse	**MQTTClient_subscribeMany5** (**MQTTClient** handle, int count, char *const *topic, int *qos, **MQTTSubscribe_options** *opts, **MQTTProperties** *props)
int	**MQTTClient_unsubscribe** (**MQTTClient** handle, const char *topic)
MQTTResponse	**MQTTClient_unsubscribe5** (**MQTTClient** handle, const char *topic, **MQTTProperties** *props)
int	**MQTTClient_unsubscribeMany** (**MQTTClient** handle, int count, char *const *topic)
MQTTResponse	**MQTTClient_unsubscribeMany5** (**MQTTClient** handle, int count, char *const *topic, **MQTTProperties** *props)
int	**MQTTClient_publish** (**MQTTClient** handle, const char *topicName, int payloadlen, const void *payload, int qos, int retained, **MQTTClient_deliveryToken** *dt)
MQTTResponse	**MQTTClient_publish5** (**MQTTClient** handle, const char *topicName, int payloadlen, const void *payload, int qos, int retained, **MQTTProperties** *properties, **MQTTClient_deliveryToken** *dt)
int	**MQTTClient_publishMessage** (**MQTTClient** handle, const char *topicName, **MQTTClient_message** *msg, **MQTTClient_deliveryToken** *dt)
MQTTResponse	**MQTTClient_publishMessage5** (**MQTTClient** handle, const char *topicName, **MQTTClient_message** *msg, **MQTTClient_deliveryToken** *dt)

int	**MQTTClient_waitForCompletion** (**MQTTClient** handle, **MQTTClient_deliveryToken** dt, unsigned long timeout)
int	**MQTTClient_getPendingDeliveryTokens** (**MQTTClient** handle, **MQTTClient_deliveryToken** **tokens)
void	**MQTTClient_yield** (void)
int	**MQTTClient_receive** (**MQTTClient** handle, char **topicName, int *topicLen, **MQTTClient_message** **message, unsigned long timeout)
void	**MQTTClient_freeMessage** (**MQTTClient_message** **msg)
void	**MQTTClient_free** (void *ptr)
void	**MQTTClient_destroy** (**MQTTClient** *handle)
void	**MQTTClient_setTraceLevel** (enum **MQTTCLIENT_TRACE_LEVELS** level)
void	**MQTTClient_setTraceCallback** (**MQTTClient_traceCallback** *callback)
const char *	**MQTTClient_strerror** (int code)

Macro Definition Documentation

◆ MQTTCLIENT_SUCCESS

```
#define MQTTCLIENT_SUCCESS   0
```

Return code: No error. Indicates successful completion of an MQTT client operation.

◆ MQTTCLIENT_FAILURE

```
#define MQTTCLIENT_FAILURE   -1
```

Return code: A generic error code indicating the failure of an MQTT client operation.

◆ MQTTCLIENT_DISCONNECTED

```
#define MQTTCLIENT_DISCONNECTED   -3
```

Return code: The client is disconnected.

◆ MQTTCLIENT_MAX_MESSAGES_INFLIGHT

#define MQTTCLIENT_MAX_MESSAGES_INFLIGHT -4

Return code: The maximum number of messages allowed to be simultaneously in-flight has been reached.

◆ MQTTCLIENT_BAD_UTF8_STRING

#define MQTTCLIENT_BAD_UTF8_STRING -5

Return code: An invalid UTF-8 string has been detected.

◆ MQTTCLIENT_NULL_PARAMETER

#define MQTTCLIENT_NULL_PARAMETER -6

Return code: A NULL parameter has been supplied when this is invalid.

◆ MQTTCLIENT_TOPICNAME_TRUNCATED

#define MQTTCLIENT_TOPICNAME_TRUNCATED -7

Return code: The topic has been truncated (the topic string includes embedded NULL characters). String functions will not access the full topic. Use the topic length value to access the full topic.

◆ MQTTCLIENT_BAD_STRUCTURE

#define MQTTCLIENT_BAD_STRUCTURE -8

Return code: A structure parameter does not have the correct eyecatcher and version number.

◆ MQTTCLIENT_BAD_QOS

#define MQTTCLIENT_BAD_QOS -9

Return code: A QoS value that falls outside of the acceptable range (0,1,2)

◆ MQTTCLIENT_SSL_NOT_SUPPORTED

#define MQTTCLIENT_SSL_NOT_SUPPORTED -10

Return code: Attempting SSL connection using non-SSL version of library

◆ MQTTCLIENT_BAD_MQTT_VERSION

#define MQTTCLIENT_BAD_MQTT_VERSION -11

Return code: unrecognized MQTT version

◆ MQTTCLIENT_BAD_PROTOCOL

#define MQTTCLIENT_BAD_PROTOCOL -14

Return code: protocol prefix in serverURI should be tcp:// or ssl://

◆ MQTTCLIENT_BAD_MQTT_OPTION

#define MQTTCLIENT_BAD_MQTT_OPTION -15

Return code: option not applicable to the requested version of MQTT

◆ MQTTCLIENT_WRONG_MQTT_VERSION

#define MQTTCLIENT_WRONG_MQTT_VERSION -16

Return code: call not applicable to the requested version of MQTT

◆ MQTTVERSION_DEFAULT

#define MQTTVERSION_DEFAULT 0

Default MQTT version to connect with. Use 3.1.1 then fall back to 3.1

◆ MQTTVERSION_3_1

```
#define MQTTVERSION_3_1  3
```

MQTT version to connect with: 3.1

MQTTVERSION_3_1_1

```
#define MQTTVERSION_3_1_1  4
```

MQTT version to connect with: 3.1.1

MQTTVERSION_5

```
#define MQTTVERSION_5  5
```

MQTT version to connect with: 5

MQTT_BAD_SUBSCRIBE

```
#define MQTT_BAD_SUBSCRIBE  0x80
```

Bad return code from subscribe, as defined in the 3.1.1 specification

MQTTClient_init_options_initializer

```
#define MQTTClient_init_options_initializer  { {'M', 'Q', 'T', 'G'}, 0, 0 }
```

MQTTClient_message_initializer

```
#define MQTTClient_message_initializer  { {'M', 'Q', 'T', 'M'}, 1, 0, NULL, 0, 0, 0, 0,
MQTTProperties_initializer }
```

MQTTClient_createOptions_initializer

```
#define MQTTClient_createOptions_initializer  { {'M', 'Q', 'C', 'O'}, MQTTVERSION_DEFAULT }
```

MQTTClient_willOptions_initializer

#define MQTTClient_willOptions_initializer { {'M', 'Q', 'T', 'W'}, 1, NULL, NULL, 0, 0, {0, NULL} }

MQTT_SSL_VERSION_DEFAULT

#define MQTT_SSL_VERSION_DEFAULT 0

MQTT_SSL_VERSION_TLS_1_0

#define MQTT_SSL_VERSION_TLS_1_0 1

MQTT_SSL_VERSION_TLS_1_1

#define MQTT_SSL_VERSION_TLS_1_1 2

MQTT_SSL_VERSION_TLS_1_2

#define MQTT_SSL_VERSION_TLS_1_2 3

MQTTClient_SSLOptions_initializer

#define MQTTClient_SSLOptions_initializer { {'M', 'Q', 'T', 'S'}, 3, NULL, NULL, NULL, NULL, NULL, 1, **MQTT_SSL_VERSION_DEFAULT**, 0, NULL, NULL, NULL }

MQTTClient_connectOptions_initializer

#define MQTTClient_connectOptions_initializer { {'M', 'Q', 'T', 'C'}, 6, 60, 1, 1, NULL, NULL, NULL, 30, 0, NULL, 0, NULL, **MQTTVERSION_DEFAULT**, {NULL, 0, 0}, {0, NULL}, -1, 0}

MQTTClient_connectOptions_initializer5

#define MQTTClient_connectOptions_initializer5 { {'M', 'Q', 'T', 'C'}, 6, 60, 0, 1, NULL, NULL, NULL, 30, 0, NULL, 0, NULL, **MQTTVERSION_5**, {NULL, 0, 0}, {0, NULL}, -1, 1}

◆ MQTTResponse_initializer

#define MQTTResponse_initializer {1, **MQTTREASONCODE_SUCCESS**, 0, NULL, NULL}

Typedef Documentation

◆ MQTTClient

typedef void* **MQTTClient**

A handle representing an MQTT client. A valid client handle is available following a successful call to **MQTTClient_create()**.

◆ MQTTClient_deliveryToken

typedef int **MQTTClient_deliveryToken**

A value representing an MQTT message. A delivery token is returned to the client application when a message is published. The token can then be used to check that the message was successfully delivered to its destination (see **MQTTClient_publish()**, **MQTTClient_publishMessage()**, **MQTTClient_deliveryComplete()**, **MQTTClient_waitForCompletion()** and **MQTTClient_getPendingDeliveryTokens()**).

◆ MQTTClient_token

typedef int **MQTTClient_token**

◆ MQTTClient_messageArrived

typedef int MQTTClient_messageArrived(void *context, char *topicName, int topicLen, **MQTTClient_message** *message)

This is a callback function. The client application must provide an implementation of this function to enable asynchronous receipt of messages. The function is registered with the client library by passing it as an argument to **MQTTClient_setCallbacks()**. It is called by the client library when a new message that matches a client subscription has been received from the server. This function is executed on a separate thread to the one on which the client application is running.

Parameters

context	A pointer to the *context* value originally passed to **MQTTClient_setCallbacks()**, which contains any application-specific context.
topicName	The topic associated with the received message.
topicLen	The length of the topic if there are one more NULL characters embedded in *topicName*, otherwise *topicLen* is 0. If *topicLen* is 0, the value returned by *strlen(topicName)* can be trusted. If *topicLen* is greater than 0, the full topic name can be retrieved by accessing *topicName* as a byte array of length *topicLen*.
message	The **MQTTClient_message** structure for the received message. This structure contains the message payload and attributes.

Returns

This function must return a boolean value indicating whether or not the message has been safely received by the client application. Returning true indicates that the message has been successfully handled. Returning false indicates that there was a problem. In this case, the client library will reinvoke **MQTTClient_messageArrived()** to attempt to deliver the message to the application again.

◆ MQTTClient_deliveryComplete

typedef void MQTTClient_deliveryComplete(void *context, **MQTTClient_deliveryToken** dt)

This is a callback function. The client application must provide an implementation of this function to enable asynchronous notification of delivery of messages. The function is registered with the client library by passing it as an argument to **MQTTClient_setCallbacks()**. It is called by the client library after the client application has published a message to the server. It indicates that the necessary handshaking and acknowledgements for the requested quality of service (see **MQTTClient_message.qos**) have been completed. This function is executed on a separate thread to the one on which the client application is running. **Note:** **MQTTClient_deliveryComplete()** is not called when messages are published at QoS0.

Parameters

context A pointer to the *context* value originally passed to **MQTTClient_setCallbacks()**, which contains any application-specific context.

dt The **MQTTClient_deliveryToken** associated with the published message. Applications can check that all messages have been correctly published by matching the delivery tokens returned from calls to **MQTTClient_publish()** and **MQTTClient_publishMessage()** with the tokens passed to this callback.

◆ MQTTClient_connectionLost

typedef void MQTTClient_connectionLost(void *context, char *cause)

This is a callback function. The client application must provide an implementation of this function to enable asynchronous notification of the loss of connection to the server. The function is registered with the client library by passing it as an argument to **MQTTClient_setCallbacks()**. It is called by the client library if the client loses its connection to the server. The client application must take appropriate action, such as trying to reconnect or reporting the problem. This function is executed on a separate thread to the one on which the client application is running.

Parameters

context A pointer to the *context* value originally passed to **MQTTClient_setCallbacks()**, which contains any application-specific context.

cause The reason for the disconnection. Currently, *cause* is always set to NULL.

◆ MQTTClient_disconnected

typedef void MQTTClient_disconnected(void *context, **MQTTProperties** *properties, enum **MQTTReasonCodes** reasonCode)

This is a callback function, which will be called when the a disconnect packet is received from the server. This applies to MQTT V5 and above only.

Parameters

context	A pointer to the *context* value originally passed to ::MQTTAsync_setDisconnected(), which contains any application-specific context.
properties	The MQTT V5 properties received with the disconnect, if any.
reasonCode	The MQTT V5 reason code received with the disconnect. Currently, *cause* is always set to NULL.

◆ MQTTClient_published

typedef void MQTTClient_published(void *context, int dt, int packet_type, **MQTTProperties** *properties, enum **MQTTReasonCodes** reasonCode)

This is a callback function, the MQTT V5 version of **MQTTClient_deliveryComplete()**. The client application must provide an implementation of this function to enable asynchronous notification of the completed delivery of messages. It is called by the client library after the client application has published a message to the server. It indicates that the necessary handshaking and acknowledgements for the requested quality of service (see **MQTTClient_message.qos**) have been completed. This function is executed on a separate thread to the one on which the client application is running. **Note:** It is not called when messages are published at QoS0.

Parameters

context	A pointer to the *context* value originally passed to **MQTTClient_setCallbacks()**, which contains any application-specific context.
dt	The **MQTTClient_deliveryToken** associated with the published message. Applications can check that all messages have been correctly published by matching the delivery tokens returned from calls to **MQTTClient_publish()** and **MQTTClient_publishMessage()** with the tokens passed to this callback.
packet_type	the last received packet type for this completion. For QoS 1 always PUBACK. For QoS 2 could be PUBREC or PUBCOMP.
properties	the MQTT V5 properties returned with the last packet from the server
reasonCode	the reason code returned from the server

◆ MQTTResponse

typedef struct **MQTTResponse MQTTResponse**

◆ MQTTClient_traceCallback

typedef void MQTTClient_traceCallback(enum **MQTTCLIENT_TRACE_LEVELS** level, char *message)

This is a callback function prototype which must be implemented if you want to receive trace information.

Parameters

level the trace level of the message returned

meesage the trace message. This is a pointer to a static buffer which will be overwritten on each call. You must copy the data if you want to keep it for later.

Enumeration Type Documentation

◆ MQTTCLIENT_TRACE_LEVELS

enum **MQTTCLIENT_TRACE_LEVELS**

Enumerator	
MQTTCLIENT_TRACE_MAXIMUM	
MQTTCLIENT_TRACE_MEDIUM	
MQTTCLIENT_TRACE_MINIMUM	
MQTTCLIENT_TRACE_PROTOCOL	
MQTTCLIENT_TRACE_ERROR	
MQTTCLIENT_TRACE_SEVERE	
MQTTCLIENT_TRACE_FATAL	

Function Documentation

◆ MQTTClient_global_init()

void MQTTClient_global_init (**MQTTClient_init_options** * inits)

Global init of mqtt library. Call once on program start to set global behaviour. do_openssl_init - if mqtt library should initialize OpenSSL (1) or rely on the caller to do it before using the library (0)

◆ MQTTClient_setCallbacks()

```
int MQTTClient_setCallbacks ( MQTTClient                handle,
                              void *                    context,
                              MQTTClient_connectionLost * cl,
                              MQTTClient_messageArrived * ma,
                              MQTTClient_deliveryComplete * dc
                            )
```

This function sets the callback functions for a specific client. If your client application doesn't use a particular callback, set the relevant parameter to NULL. Calling **MQTTClient_setCallbacks()** puts the client into multi-threaded mode. Any necessary message acknowledgements and status communications are handled in the background without any intervention from the client application. See **Asynchronous vs synchronous client applications** for more information.

Note: The MQTT client must be disconnected when this function is called.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

context A pointer to any application-specific context. The the *context* pointer is passed to each of the callback functions to provide access to the context information in the callback.

cl A pointer to an **MQTTClient_connectionLost()** callback function. You can set this to NULL if your application doesn't handle disconnections.

ma A pointer to an **MQTTClient_messageArrived()** callback function. This callback function must be specified when you call **MQTTClient_setCallbacks()**.

dc A pointer to an **MQTTClient_deliveryComplete()** callback function. You can set this to NULL if your application publishes synchronously or if you do not want to check for successful delivery.

Returns

MQTTCLIENT_SUCCESS if the callbacks were correctly set, **MQTTCLIENT_FAILURE** if an error occurred.

◆ MQTTClient_setDisconnected()

```
int MQTTClient_setDisconnected ( MQTTClient                 handle,
                                 void *                     context,
                                 MQTTClient_disconnected *  co
                               )
```

Sets the **MQTTClient_disconnected()** callback function for a client. This will be called if a disconnect packet is received from the server. Only valid for MQTT V5 and above.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

context A pointer to any application-specific context. The the *context* pointer is passed to each of the callback functions to provide access to the context information in the callback.

co A pointer to an **MQTTClient_disconnected()** callback function. NULL removes the callback setting.

Returns

MQTTCLIENT_SUCCESS if the callbacks were correctly set, **MQTTCLIENT_FAILURE** if an error occurred.

◆ MQTTClient_setPublished()

```
int MQTTClient_setPublished ( MQTTClient                 handle,
                              void *                     context,
                              MQTTClient_published *     co
                            )
```

◆ MQTTClient_create()

```
int MQTTClient_create ( MQTTClient *  handle,
                        const char *  serverURI,
                        const char *  clientId,
                        int           persistence_type,
                        void *        persistence_context
                      )
```

This function creates an MQTT client ready for connection to the specified server and using the specified persistent storage (see **MQTTClient_persistence**). See also **MQTTClient_destroy()**.

Parameters

handle	A pointer to an **MQTTClient** handle. The handle is populated with a valid client reference following a successful return from this function.
serverURI	A null-terminated string specifying the server to which the client will connect. It takes the form *protocol://host:port*. Currently, *protocol* must be *tcp* or *ssl*. For *host*, you can specify either an IP address or a host name. For instance, to connect to a server running on the local machines with the default MQTT port, specify *tcp://localhost:1883*.
clientId	The client identifier passed to the server when the client connects to it. It is a null-terminated UTF-8 encoded string.
persistence_type	The type of persistence to be used by the client: **MQTTCLIENT_PERSISTENCE_NONE**: Use in-memory persistence. If the device or system on which the client is running fails or is switched off, the current state of any in-flight messages is lost and some messages may not be delivered even at QoS1 and QoS2. **MQTTCLIENT_PERSISTENCE_DEFAULT**: Use the default (file system-based) persistence mechanism. Status about in-flight messages is held in persistent storage and provides some protection against message loss in the case of unexpected failure. **MQTTCLIENT_PERSISTENCE_USER**: Use an application-specific persistence implementation. Using this type of persistence gives control of the persistence mechanism to the application. The application has to implement the **MQTTClient_persistence** interface.
persistence_context	If the application uses **MQTTCLIENT_PERSISTENCE_NONE** persistence, this argument is unused and should be set to NULL. For **MQTTCLIENT_PERSISTENCE_DEFAULT** persistence, it should be set to the location of the persistence directory (if set to NULL, the persistence directory used is the working directory). Applications that use **MQTTCLIENT_PERSISTENCE_USER** persistence set this argument to point to a valid **MQTTClient_persistence** structure.

Returns

MQTTCLIENT_SUCCESS if the client is successfully created, otherwise an error code is returned.

- MQTTClient_createWithOptions()

```
int MQTTClient_createWithOptions ( MQTTClient *                 handle,
                                   const char *                 serverURI,
                                   const char *                 clientId,
                                   int                          persistence_type,
                                   void *                       persistence_context,
                                   MQTTClient_createOptions *   options
                                 )
```

A version of :**MQTTClient_create()** with additional options. This function creates an MQTT client ready for connection to the specified server and using the specified persistent storage (see **MQTTClient_persistence**). See also **MQTTClient_destroy()**.

Parameters

handle	A pointer to an **MQTTClient** handle. The handle is populated with a valid client reference following a successful return from this function.
serverURI	A null-terminated string specifying the server to which the client will connect. It takes the form *protocol://host:port*. Currently, *protocol* must be *tcp* or *ssl*. For *host*, you can specify either an IP address or a host name. For instance, to connect to a server running on the local machines with the default MQTT port, specify *tcp://localhost:1883*.
clientId	The client identifier passed to the server when the client connects to it. It is a null-terminated UTF-8 encoded string.
persistence_type	The type of persistence to be used by the client: **MQTTCLIENT_PERSISTENCE_NONE**: Use in-memory persistence. If the device or system on which the client is running fails or is switched off, the current state of any in-flight messages is lost and some messages may not be delivered even at QoS1 and QoS2. **MQTTCLIENT_PERSISTENCE_DEFAULT**: Use the default (file system-based) persistence mechanism. Status about in-flight messages is held in persistent storage and provides some protection against message loss in the case of unexpected failure. **MQTTCLIENT_PERSISTENCE_USER**: Use an application-specific persistence implementation. Using this type of persistence gives control of the persistence mechanism to the application. The application has to implement the **MQTTClient_persistence** interface.
persistence_context	If the application uses **MQTTCLIENT_PERSISTENCE_NONE** persistence, this argument is unused and should be set to NULL. For **MQTTCLIENT_PERSISTENCE_DEFAULT** persistence, it should be set to the location of the persistence directory (if set to NULL, the persistence directory used is the working directory). Applications that use **MQTTCLIENT_PERSISTENCE_USER** persistence set this argument to point to a valid **MQTTClient_persistence** structure.
options	additional options for the create.

Returns

MQTTCLIENT_SUCCESS if the client is successfully created, otherwise an error code is returned.

◆ MQTTClient_getVersionInfo()

MQTTClient_nameValue* MQTTClient_getVersionInfo (void)

This function returns version information about the library. no trace information will be returned.

Returns

an array of strings describing the library. The last entry is a NULL pointer.

◆ MQTTClient_connect()

int MQTTClient_connect (**MQTTClient** handle,
MQTTClient_connectOptions * options
)

This function attempts to connect a previously-created client (see **MQTTClient_create()**) to an MQTT server using the specified options. If you want to enable asynchronous message and status notifications, you must call **MQTTClient_setCallbacks()** prior to **MQTTClient_connect()**.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

options A pointer to a valid **MQTTClient_connectOptions** structure.

Returns

MQTTCLIENT_SUCCESS if the client successfully connects to the server. An error code is returned if the client was unable to connect to the server. Error codes greater than 0 are returned by the MQTT protocol:

1: Connection refused: Unacceptable protocol version
2: Connection refused: Identifier rejected
3: Connection refused: Server unavailable
4: Connection refused: Bad user name or password
5: Connection refused: Not authorized
6-255: Reserved for future use

◆ MQTTResponse_free()

void MQTTResponse_free (**MQTTResponse** response)

◆ MQTTClient_connect5()

```
MQTTResponse MQTTClient_connect5 ( MQTTClient                    handle,
                                   MQTTClient_connectOptions *   options,
                                   MQTTProperties *              connectProperties,
                                   MQTTProperties *              willProperties
                                 )
```

◆ MQTTClient_disconnect()

```
int MQTTClient_disconnect ( MQTTClient  handle,
                            int         timeout
                          )
```

This function attempts to disconnect the client from the MQTT server. In order to allow the client time to complete handling of messages that are in-flight when this function is called, a timeout period is specified. When the timeout period has expired, the client disconnects even if there are still outstanding message acknowledgements. The next time the client connects to the same server, any QoS 1 or 2 messages which have not completed will be retried depending on the cleansession settings for both the previous and the new connection (see **MQTTClient_connectOptions.cleansession** and **MQTTClient_connect()**).

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

timeout The client delays disconnection for up to this time (in milliseconds) in order to allow in-flight message transfers to complete.

Returns

MQTTCLIENT_SUCCESS if the client successfully disconnects from the server. An error code is returned if the client was unable to disconnect from the server

◆ MQTTClient_disconnect5()

```
int MQTTClient_disconnect5 ( MQTTClient                 handle,
                             int                        timeout,
                             enum MQTTReasonCodes       reason,
                             MQTTProperties *           props
                           )
```

◆ MQTTClient_isConnected()

int MQTTClient_isConnected (**MQTTClient** handle)

This function allows the client application to test whether or not a client is currently connected to the MQTT server.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

Returns

Boolean true if the client is connected, otherwise false.

◆ MQTTClient_subscribe()

```
int MQTTClient_subscribe ( MQTTClient  handle,
                           const char *  topic,
                           int           qos
                         )
```

This function attempts to subscribe a client to a single topic, which may contain wildcards (see **Subscription wildcards**). This call also specifies the **Quality of service** requested for the subscription (see also **MQTTClient_subscribeMany()**).

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

topic The subscription topic, which may include wildcards.

qos The requested quality of service for the subscription.

Returns

MQTTCLIENT_SUCCESS if the subscription request is successful. An error code is returned if there was a problem registering the subscription.

◆ MQTTClient_subscribe5()

```
MQTTResponse MQTTClient_subscribe5 ( MQTTClient                handle,
                                     const char *              topic,
                                     int                       qos,
                                     MQTTSubscribe_options *   opts,
                                     MQTTProperties *          props
                                   )
```

◆ MQTTClient_subscribeMany()

```
int MQTTClient_subscribeMany ( MQTTClient handle,
                               int count,
                               char *const * topic,
                               int * qos
                             )
```

This function attempts to subscribe a client to a list of topics, which may contain wildcards (see **Subscription wildcards**). This call also specifies the **Quality of service** requested for each topic (see also **MQTTClient_subscribe()**).

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

count The number of topics for which the client is requesting subscriptions.

topic An array (of length *count*) of pointers to topics, each of which may include wildcards.

qos An array (of length *count*) of **Quality of service** values. qos[n] is the requested QoS for topic[n].

Returns

MQTTCLIENT_SUCCESS if the subscription request is successful. An error code is returned if there was a problem registering the subscriptions.

◆ MQTTClient_subscribeMany5()

```
MQTTResponse MQTTClient_subscribeMany5 ( MQTTClient handle,
                                         int count,
                                         char *const * topic,
                                         int * qos,
                                         MQTTSubscribe_options * opts,
                                         MQTTProperties * props
                                       )
```

◆ MQTTClient_unsubscribe()

```
int MQTTClient_unsubscribe ( MQTTClient  handle,
                             const char *  topic
                           )
```

This function attempts to remove an existing subscription made by the specified client.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

topic The topic for the subscription to be removed, which may include wildcards (see **Subscription wildcards**).

Returns

MQTTCLIENT_SUCCESS if the subscription is removed. An error code is returned if there was a problem removing the subscription.

◆ MQTTClient_unsubscribe5()

```
MQTTResponse MQTTClient_unsubscribe5 ( MQTTClient          handle,
                                       const char *        topic,
                                       MQTTProperties *    props
                                     )
```

◆ MQTTClient_unsubscribeMany()

```
int MQTTClient_unsubscribeMany ( MQTTClient  handle,
                                 int         count,
                                 char *const *  topic
                               )
```

This function attempts to remove existing subscriptions to a list of topics made by the specified client.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

count The number subscriptions to be removed.

topic An array (of length *count*) of pointers to the topics of the subscriptions to be removed, each of which may include wildcards.

Returns

MQTTCLIENT_SUCCESS if the subscriptions are removed. An error code is returned if there was a problem removing the subscriptions.

◆ MQTTClient_unsubscribeMany5()

```
MQTTResponse MQTTClient_unsubscribeMany5 ( MQTTClient          handle,
                                           int                 count,
                                           char *const *       topic,
                                           MQTTProperties *    props
                                         )
```

◆ MQTTClient_publish()

```
int MQTTClient_publish ( MQTTClient                 handle,
                         const char *               topicName,
                         int                        payloadlen,
                         const void *               payload,
                         int                        qos,
                         int                        retained,
                         MQTTClient_deliveryToken * dt
                       )
```

This function attempts to publish a message to a given topic (see also **MQTTClient_publishMessage()**). An **MQTTClient_deliveryToken** is issued when this function returns successfully. If the client application needs to test for succesful delivery of QoS1 and QoS2 messages, this can be done either asynchronously or synchronously (see **Asynchronous vs synchronous client applications**, **MQTTClient_waitForCompletion** and **MQTTClient_deliveryComplete()**).

Parameters

handle	A valid client handle from a successful call to **MQTTClient_create()**.
topicName	The topic associated with this message.
payloadlen	The length of the payload in bytes.
payload	A pointer to the byte array payload of the message.
qos	The **Quality of service** of the message.
retained	The retained flag for the message.
dt	A pointer to an **MQTTClient_deliveryToken**. This is populated with a token representing the message when the function returns successfully. If your application does not use delivery tokens, set this argument to NULL.

Returns

MQTTCLIENT_SUCCESS if the message is accepted for publication. An error code is returned if there was a problem accepting the message.

◆ MQTTClient_publish5()

```
MQTTResponse MQTTClient_publish5 ( MQTTClient                handle,
                                   const char *              topicName,
                                   int                       payloadlen,
                                   const void *              payload,
                                   int                       qos,
                                   int                       retained,
                                   MQTTProperties *          properties,
                                   MQTTClient_deliveryToken * dt
                                 )
```

◆ MQTTClient_publishMessage()

```
int MQTTClient_publishMessage ( MQTTClient                handle,
                                const char *              topicName,
                                MQTTClient_message *      msg,
                                MQTTClient_deliveryToken * dt
                              )
```

This function attempts to publish a message to a given topic (see also **MQTTClient_publish()**). An **MQTTClient_deliveryToken** is issued when this function returns successfully. If the client application needs to test for succesful delivery of QoS1 and QoS2 messages, this can be done either asynchronously or synchronously (see **Asynchronous vs synchronous client applications**, **MQTTClient_waitForCompletion** and **MQTTClient_deliveryComplete()**).

Parameters

handle	A valid client handle from a successful call to **MQTTClient_create()**.
topicName	The topic associated with this message.
msg	A pointer to a valid **MQTTClient_message** structure containing the payload and attributes of the message to be published.
dt	A pointer to an **MQTTClient_deliveryToken**. This is populated with a token representing the message when the function returns successfully. If your application does not use delivery tokens, set this argument to NULL.

Returns

MQTTCLIENT_SUCCESS if the message is accepted for publication. An error code is returned if there was a problem accepting the message.

◆ MQTTClient_publishMessage5()

```
MQTTResponse MQTTClient_publishMessage5 ( MQTTClient                handle,
                                          const char *              topicName,
                                          MQTTClient_message *      msg,
                                          MQTTClient_deliveryToken * dt
                                        )
```

◆ MQTTClient_waitForCompletion()

```
int MQTTClient_waitForCompletion ( MQTTClient                handle,
                                   MQTTClient_deliveryToken  dt,
                                   unsigned long             timeout
                                 )
```

This function is called by the client application to synchronize execution of the main thread with completed publication of a message. When called, **MQTTClient_waitForCompletion()** blocks execution until the message has been successful delivered or the specified timeout has expired. See **Asynchronous vs synchronous client applications**.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

dt The **MQTTClient_deliveryToken** that represents the message being tested for successful delivery. Delivery tokens are issued by the publishing functions **MQTTClient_publish()** and **MQTTClient_publishMessage()**.

timeout The maximum time to wait in milliseconds.

Returns

MQTTCLIENT_SUCCESS if the message was successfully delivered. An error code is returned if the timeout expires or there was a problem checking the token.

◆ MQTTClient_getPendingDeliveryTokens()

int MQTTClient_getPendingDeliveryTokens (**MQTTClient** handle,
MQTTClient_deliveryToken ** tokens
)

This function sets a pointer to an array of delivery tokens for messages that are currently in-flight (pending completion).

Important note: The memory used to hold the array of tokens is malloc()'d in this function. The client application is responsible for freeing this memory when it is no longer required.

Parameters

handle A valid client handle from a successful call to **MQTTClient_create()**.

tokens The address of a pointer to an **MQTTClient_deliveryToken**. When the function returns successfully, the pointer is set to point to an array of tokens representing messages pending completion. The last member of the array is set to -1 to indicate there are no more tokens. If no tokens are pending, the pointer is set to NULL.

Returns

MQTTCLIENT_SUCCESS if the function returns successfully. An error code is returned if there was a problem obtaining the list of pending tokens.

◆ MQTTClient_yield()

void MQTTClient_yield (void)

When implementing a single-threaded client, call this function periodically to allow processing of message retries and to send MQTT keepalive pings. If the application is calling **MQTTClient_receive()** regularly, then it is not necessary to call this function.

◆ MQTTClient_receive()

```
int MQTTClient_receive ( MQTTClient             handle,
                         char **                topicName,
                         int *                  topicLen,
                         MQTTClient_message **  message,
                         unsigned long          timeout
                       )
```

This function performs a synchronous receive of incoming messages. It should be used only when the client application has not set callback methods to support asynchronous receipt of messages (see **Asynchronous vs synchronous client applications** and **MQTTClient_setCallbacks()**). Using this function allows a single-threaded client subscriber application to be written. When called, this function blocks until the next message arrives or the specified timeout expires (see also **MQTTClient_yield()**).

Important note: The application must free() the memory allocated to the topic and the message when processing is complete (see **MQTTClient_freeMessage()**).

Parameters

handle	A valid client handle from a successful call to **MQTTClient_create()**.
topicName	The address of a pointer to a topic. This function allocates the memory for the topic and returns it to the application by setting *topicName* to point to the topic.
topicLen	The length of the topic. If the return code from this function is **MQTTCLIENT_TOPICNAME_TRUNCATED**, the topic contains embedded NULL characters and the full topic should be retrieved by using *topicLen*.
message	The address of a pointer to the received message. This function allocates the memory for the message and returns it to the application by setting *message* to point to the received message. The pointer is set to NULL if the timeout expires.
timeout	The length of time to wait for a message in milliseconds.

Returns

MQTTCLIENT_SUCCESS or **MQTTCLIENT_TOPICNAME_TRUNCATED** if a message is received. **MQTTCLIENT_SUCCESS** can also indicate that the timeout expired, in which case *message* is NULL. An error code is returned if there was a problem trying to receive a message.

◆ MQTTClient_freeMessage()

void MQTTClient_freeMessage (**MQTTClient_message** ** msg)

This function frees memory allocated to an MQTT message, including the additional memory allocated to the message payload. The client application calls this function when the message has been fully processed. **Important note:** This function does not free the memory allocated to a message topic string. It is the responsibility of the client application to free this memory using the **MQTTClient_free()** library function.

Parameters

msg The address of a pointer to the **MQTTClient_message** structure to be freed.

◆ MQTTClient_free()

void MQTTClient_free (void * ptr)

This function frees memory allocated by the MQTT C client library, especially the topic name. This is needed on Windows when the client libary and application program have been compiled with different versions of the C compiler. It is thus good policy to always use this function when freeing any MQTT C client- allocated memory.

Parameters

ptr The pointer to the client library storage to be freed.

◆ MQTTClient_destroy()

void MQTTClient_destroy (**MQTTClient** * handle)

This function frees the memory allocated to an MQTT client (see **MQTTClient_create()**). It should be called when the client is no longer required.

Parameters

handle A pointer to the handle referring to the **MQTTClient** structure to be freed.

◆ MQTTClient_setTraceLevel()

void MQTTClient_setTraceLevel (enum **MQTTCLIENT_TRACE_LEVELS** level)

This function sets the level of trace information which will be returned in the trace callback.

Parameters

level the trace level required

◆ MQTTClient_setTraceCallback()

void MQTTClient_setTraceCallback (**MQTTClient_traceCallback** * callback)

This function sets the trace callback if needed. If set to NULL, no trace information will be returned. The default trace level is MQTTASYNC_TRACE_MINIMUM.

Parameters

callback a pointer to the function which will handle the trace information

◆ MQTTClient_strerror()

const char* MQTTClient_strerror (int code)

Returns a pointer to the string representation of the error or NULL.

Do not free after use. Returns NULL if the error code is unknown.