mosquitto.h
mosquitto_broker.h
mosquitto_plugin.h
mqtt_protocol.h

INDEX
Constants
Everything
Files
Functions
Types

Search

mosquitto.h

This header contains functions and definitions for use with libmosquitto, the Mosquitto client library.
The definitions are also used in Mosquitto broker plugins, and some functions are available to plugins.

Summary

mosquitto.h This header contains functions and definitions for use with libmosquitto, the Mosquitto client library.

Threads libmosquitto provides thread safe operation, with the exception of mosquitto_lib_init which is not thread safe.

Library version, init, and cleanup

FUNCTIONS

mosquitto_lib_version	Can be used to obtain version information for the mosquitto library.
mosquitto_lib_init	Must be called before any other mosquitto functions.
mosquitto_lib_cleanup	Call to free resources associated with the library.

Client creation, destruction, and reinitialisation

FUNCTIONS

mosquitto_new	Create a new mosquitto client instance.
mosquitto_destroy	Use to free memory associated with a mosquitto client instance.
mosquitto_reinitialise	This function allows an existing mosquitto client to be reused.

Will

FUNCTIONS

mosquitto_will_set	Configure will information for a mosquitto instance.
mosquitto_will_set_v5	Configure will information for a mosquitto instance, with attached properties.
mosquitto_will_clear	Remove a previously configured will.

Username and password

FUNCTIONS

mosquitto_username_pw_set	Configure username and password for a mosquitto instance.

Connecting, reconnecting, disconnecting

FUNCTIONS

mosquitto_connect	Connect to an MQTT broker.
mosquitto_connect_bind	Connect to an MQTT broker.
mosquitto_connect_bind_v5	Connect to an MQTT broker.
mosquitto_connect_async	Connect to an MQTT broker.
mosquitto_connect_bind_async	Connect to an MQTT broker.
mosquitto_connect_srv	Connect to an MQTT broker.
mosquitto_reconnect	Reconnect to a broker.
mosquitto_reconnect_async	Reconnect to a broker.
mosquitto_disconnect	Disconnect from the broker.
mosquitto_disconnect_v5	Disconnect from the broker, with attached MQTT properties.

Publishing, subscribing, unsubscribing

FUNCTIONS

mosquitto_publish	Publish a message on a given topic.
mosquitto_publish_v5	Publish a message on a given topic, with attached MQTT properties.
mosquitto_subscribe	Subscribe to a topic.
mosquitto_subscribe_v5	Subscribe to a topic, with attached MQTT properties.
mosquitto_subscribe_multiple	Subscribe to multiple topics.
mosquitto_unsubscribe	Unsubscribe from a topic.
mosquitto_unsubscribe_v5	Unsubscribe from a topic, with attached MQTT properties.
mosquitto_unsubscribe_multiple	Unsubscribe from multiple topics.

Struct mosquitto_message helper functions

FUNCTIONS

mosquitto_message_copy	Copy the contents of a mosquitto message to another message.
mosquitto_message_free	Completely free a mosquitto_message struct.
mosquitto_message_free_contents	Free a mosquitto_message struct contents, leaving the struct unaffected.

Network loop (managed by libmosquitto)

The internal network loop must be called at a regular interval.

FUNCTIONS

mosquitto_loop_forever	This function call loop() for you in an infinite blocking loop.
mosquitto_loop_start	This is part of the threaded client interface.
mosquitto_loop_stop	This is part of the threaded client interface.
mosquitto_loop	The main network loop for the client.

Network loop (for use in other event loops)

FUNCTIONS

mosquitto_loop_read	Carry out network read operations.
mosquitto_loop_write	Carry out network write operations.
mosquitto_loop_misc	Carry out miscellaneous operations required as part of the network loop.

Network loop (helper functions)

FUNCTIONS

mosquitto_socket	Return the socket handle for a mosquitto instance.
mosquitto_want_write	Returns true if there is data ready to be written on the socket.
mosquitto_threaded_set	Used to tell the library that your application is using threads, but not using mosquitto_loop_start.

Client options

FUNCTIONS

mosquitto_opts_set	Used to set options for the client.
mosquitto_int_option	Used to set integer options for the client.
mosquitto_string_option	Used to set const char* options for the client.
mosquitto_void_option	Used to set void* options for the client.
mosquitto_reconnect_delay_set	Control the behaviour of the client when it has unexpectedly disconnected in mosquitto_loop_forever or after mosquitto_loop_start.
mosquitto_max_inflight_messages_set	This function is deprecated.
mosquitto_message_retry_set	This function now has no effect.
mosquitto_user_data_set	When mosquitto_new is called, the pointer given as the “obj” parameter will be passed to the callbacks as user data.
mosquitto_userdata	Retrieve the “userdata” variable for a mosquitto client.

TLS support

FUNCTIONS

mosquitto_tls_set	Configure the client for certificate based SSL/TLS support.
mosquitto_tls_insecure_set	Configure verification of the server hostname in the server certificate.
mosquitto_tls_opts_set	Set advanced SSL/TLS options.
mosquitto_tls_psk_set	Configure the client for pre-shared-key based TLS support.
mosquitto_ssl_get	Retrieve a pointer to the SSL structure used for TLS connections in this client.

Callbacks

FUNCTIONS

mosquitto_connect_callback_set	Set the connect callback.
mosquitto_connect_with_flags_callback_set	Set the connect callback.
mosquitto_connect_v5_callback_set	Set the connect callback.
mosquitto_disconnect_callback_set	Set the disconnect callback.
mosquitto_disconnect_v5_callback_set	Set the disconnect callback.
mosquitto_publish_callback_set	Set the publish callback.
mosquitto_publish_v5_callback_set	Set the publish callback.
mosquitto_message_callback_set	Set the message callback.
mosquitto_message_v5_callback_set	Set the message callback.
mosquitto_subscribe_callback_set	Set the subscribe callback.
mosquitto_subscribe_v5_callback_set	Set the subscribe callback.
mosquitto_unsubscribe_callback_set	Set the unsubscribe callback.
mosquitto_unsubscribe_v5_callback_set	Set the unsubscribe callback.
mosquitto_log_callback_set	Set the logging callback.

SOCKS5 proxy functions

FUNCTIONS

mosquitto_socks5_set	Configure the client to use a SOCKS5 proxy when connecting.

Utility functions

FUNCTIONS

mosquitto_strerror	Call to obtain a const string description of a mosquitto error number.
mosquitto_connack_string	Call to obtain a const string description of an MQTT connection result.
mosquitto_reason_string	Call to obtain a const string description of an MQTT reason code.
mosquitto_string_to_command	Take a string input representing an MQTT command and convert it to the libmosquitto integer representation.
mosquitto_sub_topic_tokenise	Tokenise a topic or subscription string into an array of strings representing the topic hierarchy.
mosquitto_sub_topic_tokens_free	Free memory that was allocated in mosquitto_sub_topic_tokenise.
mosquitto_topic_matches_sub	Check whether a topic matches a subscription.
mosquitto_topic_matches_sub2	Check whether a topic matches a subscription.
mosquitto_pub_topic_check	Check whether a topic to be used for publishing is valid.
mosquitto_pub_topic_check2	Check whether a topic to be used for publishing is valid.
mosquitto_sub_topic_check	Check whether a topic to be used for subscribing is valid.
mosquitto_sub_topic_check2	Check whether a topic to be used for subscribing is valid.
mosquitto_validate_utf8	Helper function to validate whether a UTF-8 string is valid, according to the UTF-8 spec and the MQTT additions.

One line client helper functions

FUNCTIONS

mosquitto_subscribe_simple	Helper function to make subscribing to a topic and retrieving some messages very straightforward.
mosquitto_subscribe_callback	Helper function to make subscribing to a topic and processing some messages very straightforward.

Properties

FUNCTIONS

mosquitto_property_add_byte	Add a new byte property to a property list.
mosquitto_property_add_int16	Add a new int16 property to a property list.
mosquitto_property_add_int32	Add a new int32 property to a property list.
mosquitto_property_add_varint	Add a new varint property to a property list.
mosquitto_property_add_binary	Add a new binary property to a property list.
mosquitto_property_add_string	Add a new string property to a property list.
mosquitto_property_add_string_pair	Add a new string pair property to a property list.
mosquitto_property_identifier	Return the property identifier for a single property.
mosquitto_property_next	Return the next property in a property list.
mosquitto_property_read_byte	Attempt to read a byte property matching an identifier, from a property list or single property.
mosquitto_property_read_int16	Read an int16 property value from a property.
mosquitto_property_read_int32	Read an int32 property value from a property.
mosquitto_property_read_varint	Read a varint property value from a property.
mosquitto_property_read_binary	Read a binary property value from a property.
mosquitto_property_read_string	Read a string property value from a property.
mosquitto_property_read_string_pair	Read a string pair property value pair from a property.
mosquitto_property_free_all	Free all properties from a list of properties.
mosquitto_property_copy_all	
mosquitto_property_check_command	Check whether a property identifier is valid for the given command.
mosquitto_property_check_all	Check whether a list of properties are valid for a particular command, whether there are duplicates, and whether the values are valid where possible.
mosquitto_property_identifier_to_string	Return the property name as a string for a property identifier.
mosquitto_string_to_property_info	Parse a property name string and convert to a property identifier and data type.

Threads

libmosquitto provides thread safe operation, with the exception of mosquitto_lib_init which is not thread safe.
If your application uses threads you must use mosquitto_threaded_set to tell the library this is the case, otherwise it makes some optimisations for the single threaded case that may result in unexpected behaviour for the multi threaded case.

Library version, init, and cleanup

Summary

FUNCTIONS

mosquitto_lib_version	Can be used to obtain version information for the mosquitto library.
mosquitto_lib_init	Must be called before any other mosquitto functions.
mosquitto_lib_cleanup	Call to free resources associated with the library.

FUNCTIONS

mosquitto_lib_version

```
libmosq_EXPORT int mosquitto_lib_version(int *major,
                                         int *minor,
                                         int *revision)
```

Can be used to obtain version information for the mosquitto library. This allows the application to compare the library version against the version it was compiled against by using the LIBMOSQUITTO_MAJOR, LIBMOSQUITTO_MINOR and LIBMOSQUITTO_REVISION defines.

Parameters

`major`	an integer pointer. If not NULL, the major version of the library will be returned in this variable.
`minor`	an integer pointer. If not NULL, the minor version of the library will be returned in this variable.
`revision`	an integer pointer. If not NULL, the revision of the library will be returned in this variable.

Returns

`LIBMOSQUITTO_VERSION_NUMBER` which is a unique number based on the major, minor and revision values. See Also: mosquitto_lib_cleanup, mosquitto_lib_init

mosquitto_lib_init

```
libmosq_EXPORT int mosquitto_lib_init(void)
```

Must be called before any other mosquitto functions.
This function is **not** thread safe.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_UNKNOWN`	on Windows, if sockets couldn't be initialized.

See Also

mosquitto_lib_cleanup, mosquitto_lib_version

mosquitto_lib_cleanup

```
libmosq_EXPORT int mosquitto_lib_cleanup(void)
```

Call to free resources associated with the library.

Returns

`MOSQ_ERR_SUCCESS`	always

See Also

mosquitto_lib_init, mosquitto_lib_version

Client creation, destruction, and reinitialisation

Summary

FUNCTIONS

mosquitto_new	Create a new mosquitto client instance.
mosquitto_destroy	Use to free memory associated with a mosquitto client instance.
mosquitto_reinitialise	This function allows an existing mosquitto client to be reused.

FUNCTIONS

mosquitto_new

```
libmosq_EXPORT struct mosquitto *mosquitto_new(const char *id,
                                               bool  clean_session,
                                               void *obj           )
```

Create a new mosquitto client instance.

Parameters

`id`	String to use as the client id. If NULL, a random client id will be generated. If id is NULL, clean_session must be true.
`clean_session`	set to true to instruct the broker to clean all messages and subscriptions on disconnect, false to instruct it to keep them. See the man page mqtt(7) for more details. Note that a client will never discard its own outgoing messages on disconnect. Calling mosquitto_connect or mosquitto_reconnect will cause the messages to be resent. Use mosquitto_reinitialise to reset a client to its original state. Must be set to true if the id parameter is NULL.
`obj`	A user pointer that will be passed as an argument to any callbacks that are specified.

Returns

Pointer to a struct mosquitto on success. NULL on failure. Interrogate errno to determine the cause for the failure:

- ENOMEM on out of memory.
- EINVAL on invalid input parameters.

See Also

mosquitto_reinitialise, mosquitto_destroy, mosquitto_user_data_set

mosquitto_destroy

```
libmosq_EXPORT void mosquitto_destroy(struct mosquitto *mosq)
```

Use to free memory associated with a mosquitto client instance.

Parameters

`mosq`	a struct mosquitto pointer to free.

See Also

mosquitto_new, mosquitto_reinitialise

mosquitto_reinitialise

```
libmosq_EXPORT int mosquitto_reinitialise(struct mosquitto *mosq,
                                          const char      *id,
                                                bool       clean_session,
                                                void      *obj           )
```

This function allows an existing mosquitto client to be reused. Call on a mosquitto instance to close any open network connections, free memory and reinitialise the client with the new parameters. The end result is the same as the output of mosquitto_new.

Parameters

`mosq`	a valid mosquitto instance.
`id`	string to use as the client id. If NULL, a random client id will be generated. If id is NULL, clean_session must be true.
`clean_session`	set to true to instruct the broker to clean all messages and subscriptions on disconnect, false to instruct it to keep them. See the man page mqtt(7) for more details. Must be set to true if the id parameter is NULL.
`obj`	A user pointer that will be passed as an argument to any callbacks that are specified.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.

See Also

mosquitto_new, mosquitto_destroy

Will

Summary

FUNCTIONS

mosquitto_will_set	Configure will information for a mosquitto instance.
mosquitto_will_set_v5	Configure will information for a mosquitto instance, with attached properties.
mosquitto_will_clear	Remove a previously configured will.

FUNCTIONS

mosquitto_will_set

```
libmosq_EXPORT int mosquitto_will_set(struct mosquitto *mosq,
                                      const char      *topic,
                                            int        payloadlen,
                                      const void      *payload,
                                            int        qos,
                                            bool       retain     )
```

Configure will information for a mosquitto instance. By default, clients do not have a will. This must be called before calling mosquitto_connect.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 Will properties, use mosquitto_will_set_v5 instead.

Parameters

`mosq`	a valid mosquitto instance.
`topic`	the topic on which to publish the will.
`payloadlen`	the size of the payload (bytes). Valid values are between 0 and 268,435,455.
`payload`	pointer to the data to send. If payloadlen > 0 this must be a valid memory location.
`qos`	integer value 0, 1 or 2 indicating the Quality of Service to be used for the will.
`retain`	set to true to make the will a retained message.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_PAYLOAD_SIZE`	if payloadlen is too large.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8.

mosquitto_will_set_v5

```
libmosq_EXPORT int mosquitto_will_set_v5(struct mosquitto          *mosq,
                                         const char                *topic,
                                               int                  payloadlen,
                                         const void                *payload,
                                               int                  qos,
                                               bool                 retain,
                                               mosquitto_property *properties )
```

Configure will information for a mosquitto instance, with attached properties. By default, clients do not have a will. This must be called before calling mosquitto_connect.
If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the Will. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

`mosq`	a valid mosquitto instance.
`topic`	the topic on which to publish the will.
`payloadlen`	the size of the payload (bytes). Valid values are between 0 and 268,435,455.
`payload`	pointer to the data to send. If payloadlen > 0 this must be a valid memory location.
`qos`	integer value 0, 1 or 2 indicating the Quality of Service to be used for the will.
`retain`	set to true to make the will a retained message.
`properties`	list of MQTT 5 properties. Can be NULL. On success only, the property list becomes the property of libmosquitto once this function is called and will be freed by the library. The property list must be freed by the application on error.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_PAYLOAD_SIZE`	if payloadlen is too large.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8.
`MOSQ_ERR_NOT_SUPPORTED`	if properties is not NULL and the client is not using MQTT v5
`MOSQ_ERR_PROTOCOL`	if a property is invalid for use with wills.
`MOSQ_ERR_DUPLICATE_PROPERTY`	if a property is duplicated where it is forbidden.

mosquitto_will_clear

```
libmosq_EXPORT int mosquitto_will_clear(struct mosquitto *mosq)
```

Remove a previously configured will. This must be called before calling mosquitto_connect.

Parameters

`mosq`	a valid mosquitto instance.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.

Username and password

Summary

FUNCTIONS

mosquitto_username_pw_set	Configure username and password for a mosquitto instance.

FUNCTIONS

mosquitto_username_pw_set

```
libmosq_EXPORT int mosquitto_username_pw_set(struct mosquitto *mosq,
                                             const char      *username,
                                             const char      *password )
```

Configure username and password for a mosquitto instance. By default, no username or password will be sent. For v3.1 and v3.1.1 clients, if username is NULL, the password argument is ignored.
This is must be called before calling mosquitto_connect.

Parameters

`mosq`	a valid mosquitto instance.
`username`	the username to send as a string, or NULL to disable authentication.
`password`	the password to send as a string. Set to NULL when username is valid in order to send just a username.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.

Connecting, reconnecting, disconnecting

Summary

FUNCTIONS

mosquitto_connect	Connect to an MQTT broker.
mosquitto_connect_bind	Connect to an MQTT broker.
mosquitto_connect_bind_v5	Connect to an MQTT broker.
mosquitto_connect_async	Connect to an MQTT broker.
mosquitto_connect_bind_async	Connect to an MQTT broker.
mosquitto_connect_srv	Connect to an MQTT broker.
mosquitto_reconnect	Reconnect to a broker.
mosquitto_reconnect_async	Reconnect to a broker.
mosquitto_disconnect	Disconnect from the broker.
mosquitto_disconnect_v5	Disconnect from the broker, with attached MQTT properties.

FUNCTIONS

mosquitto_connect

```
libmosq_EXPORT int mosquitto_connect(struct mosquitto *mosq,
                                     const char      *host,
                                           int        port,
                                           int        keepalive)
```

Connect to an MQTT broker.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 CONNECT properties, use mosquitto_connect_bind_v5 instead.

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname or ip address of the broker to connect to.
`port`	the network port to connect to. Usually 1883.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid, which could be any of: • mosq == NULL • host == NULL • port < 0 • keepalive < 5
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect_bind, mosquitto_connect_async, mosquitto_reconnect, mosquitto_disconnect, mosquitto_tls_set

mosquitto_connect_bind

```
libmosq_EXPORT int mosquitto_connect_bind(struct mosquitto *mosq,
                                          const char      *host,
                                                int        port,
                                                int        keepalive,
                                          const char      *bind_address)
```

Connect to an MQTT broker. This extends the functionality of mosquitto_connect by adding the bind_address parameter. Use this function if you need to restrict network communication over a particular interface.

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname or ip address of the broker to connect to.
`port`	the network port to connect to. Usually 1883.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.
`bind_address`	the hostname or ip address of the local network interface to bind to.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect, mosquitto_connect_async, mosquitto_connect_bind_async

mosquitto_connect_bind_v5

```
libmosq_EXPORT int mosquitto_connect_bind_v5(
   struct mosquitto          *mosq,
    const char               *host,
          int                 port,
          int                 keepalive,
```

```
    const char              *bind_address,
    const mosquitto_property *properties
)
```

Connect to an MQTT broker. This extends the functionality of mosquitto_connect by adding the bind_address parameter and MQTT v5 properties. Use this function if you need to restrict network communication over a particular interface.

Use e.g. mosquitto_property_add_string and similar to create a list of properties, then attach them to this publish. Properties need freeing with mosquitto_property_free_all.

If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the CONNECT message. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname or ip address of the broker to connect to.
`port`	the network port to connect to. Usually 1883.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.
`bind_address`	the hostname or ip address of the local network interface to bind to.
`properties`	the MQTT 5 properties for the connect (not for the Will).

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid, which could be any of: • mosq == NULL • host == NULL • port < 0 • keepalive < 5
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.
`MOSQ_ERR_DUPLICATE_PROPERTY`	if a property is duplicated where it is forbidden.
`MOSQ_ERR_PROTOCOL`	if any property is invalid for use with CONNECT.

See Also

mosquitto_connect, mosquitto_connect_async, mosquitto_connect_bind_async

mosquitto_connect_async

```
libmosq_EXPORT int mosquitto_connect_async(struct mosquitto *mosq,
                                           const char      *host,
                                                 int        port,
                                                 int        keepalive)
```

Connect to an MQTT broker. This is a non-blocking call. If you use mosquitto_connect_async your client must use the threaded interface mosquitto_loop_start. If you need to use mosquitto_loop, you must use mosquitto_connect to connect the client.

May be called before or after mosquitto_loop_start.

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname or ip address of the broker to connect to.
`port`	the network port to connect to. Usually 1883.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect_bind_async, mosquitto_connect, mosquitto_reconnect, mosquitto_disconnect, mosquitto_tls_set

mosquitto_connect_bind_async

```
libmosq_EXPORT int mosquitto_connect_bind_async(struct mosquitto *mosq,
                                                const char      *host,
                                                      int        port,
                                                      int        keepalive,
                                                const char      *bind_address)
```

Connect to an MQTT broker. This is a non-blocking call. If you use mosquitto_connect_bind_async your client must use the threaded interface mosquitto_loop_start. If you need to use mosquitto_loop, you must use mosquitto_connect to connect the client.

This extends the functionality of mosquitto_connect_async by adding the bind_address parameter. Use this function if you need to restrict network communication over a particular interface.

May be called before or after mosquitto_loop_start.

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname or ip address of the broker to connect to.
`port`	the network port to connect to. Usually 1883.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.
`bind_address`	the hostname or ip address of the local network interface to bind to.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid, which could be any of: • mosq == NULL • host == NULL • port < 0 • keepalive < 5
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect_async, mosquitto_connect, mosquitto_connect_bind

mosquitto_connect_srv

```
libmosq_EXPORT int mosquitto_connect_srv(struct mosquitto *mosq,
                                         const char       *host,
                                               int         keepalive,
                                         const char       *bind_address)
```

Connect to an MQTT broker.
If you set `host` to `example.com`, then this call will attempt to retrieve the DNS SRV record for `_secure-mqtt._tcp.example.com` or `_mqtt._tcp.example.com` to discover which actual host to connect to.
DNS SRV support is not usually compiled in to libmosquitto, use of this call is not recommended.

Parameters

`mosq`	a valid mosquitto instance.
`host`	the hostname to search for an SRV record.
`keepalive`	the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time.
`bind_address`	the hostname or ip address of the local network interface to bind to.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid, which could be any of: • mosq == NULL • host == NULL • port < 0 • keepalive < 5
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect_async, mosquitto_connect, mosquitto_connect_bind

mosquitto_reconnect

```
libmosq_EXPORT int mosquitto_reconnect(struct mosquitto *mosq)
```

Reconnect to a broker.
This function provides an easy way of reconnecting to a broker after a connection has been lost. It uses the values that were provided in the mosquitto_connect call. It must not be called before mosquitto_connect.

Parameters

`mosq`	a valid mosquitto instance.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect, mosquitto_disconnect, mosquitto_reconnect_async

mosquitto_reconnect_async

```
libmosq_EXPORT int mosquitto_reconnect_async(struct mosquitto *mosq)
```

Reconnect to a broker. Non blocking version of mosquitto_reconnect.
This function provides an easy way of reconnecting to a broker after a connection has been lost. It uses the values that were provided in the mosquitto_connect or mosquitto_connect_async calls. It must not be called before mosquitto_connect.

Parameters

`mosq`	a valid mosquitto instance.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_connect, mosquitto_disconnect

mosquitto_disconnect

```
libmosq_EXPORT int mosquitto_disconnect(struct mosquitto *mosq)
```

Disconnect from the broker.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 DISCONNECT properties, use mosquitto_disconnect_v5 instead.

Parameters

`mosq`	a valid mosquitto instance.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.

mosquitto_disconnect_v5

```
libmosq_EXPORT int mosquitto_disconnect_v5(
   struct mosquitto          *mosq,
          int                 reason_code,
    const mosquitto_property *properties
)
```

Disconnect from the broker, with attached MQTT properties.
Use e.g. mosquitto_property_add_string and similar to create a list of properties, then attach them to this publish. Properties need freeing with mosquitto_property_free_all.
If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the DISCONNECT message. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

`mosq`	a valid mosquitto instance.
`reason_code`	the disconnect reason code.
`properties`	a valid mosquitto_property list, or NULL.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_DUPLICATE_PROPERTY`	if a property is duplicated where it is forbidden.
`MOSQ_ERR_PROTOCOL`	if any property is invalid for use with DISCONNECT.

Publishing, subscribing, unsubscribing

Summary

FUNCTIONS

mosquitto_publish	Publish a message on a given topic.
mosquitto_publish_v5	Publish a message on a given topic, with attached MQTT properties.
mosquitto_subscribe	Subscribe to a topic.
mosquitto_subscribe_v5	Subscribe to a topic, with attached MQTT properties.
mosquitto_subscribe_multiple	Subscribe to multiple topics.
mosquitto_unsubscribe	Unsubscribe from a topic.
mosquitto_unsubscribe_v5	Unsubscribe from a topic, with attached MQTT properties.
mosquitto_unsubscribe_multiple	Unsubscribe from multiple topics.

FUNCTIONS

mosquitto_publish

```
libmosq_EXPORT int mosquitto_publish(struct mosquitto *mosq,
                                     int       *mid,
                               const char      *topic,
                                     int        payloadlen,
                               const void      *payload,
                                     int        qos,
                                     bool       retain     )
```

Publish a message on a given topic.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 PUBLISH properties, use mosquitto_publish_v5 instead.

Parameters

`mosq`	a valid mosquitto instance.
`mid`	pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the publish callback to determine when the message has been sent. Note that although the MQTT protocol doesn't use message ids for messages with QoS=0, libmosquitto assigns them message ids so they can be tracked with this parameter.
`topic`	null terminated string of the topic to publish to.
`payloadlen`	the size of the payload (bytes). Valid values are between 0 and 268,435,455.
`payload`	pointer to the data to send. If payloadlen > 0 this must be a valid memory location.
`qos`	integer value 0, 1 or 2 indicating the Quality of Service to be used for the message.
`retain`	set to true to make the message retained.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_PROTOCOL`	if there is a protocol error communicating with the broker.
`MOSQ_ERR_PAYLOAD_SIZE`	if payloadlen is too large.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8
`MOSQ_ERR_QOS_NOT_SUPPORTED`	if the QoS is greater than that supported by the broker.
`MOSQ_ERR_OVERSIZE_PACKET`	if the resulting packet would be larger than supported by the broker.

See Also

mosquitto_max_inflight_messages_set

mosquitto_publish_v5

```
libmosq_EXPORT int mosquitto_publish_v5(struct mosquitto          *mosq,
                                        int                       *mid,
                                  const char                      *topic,
                                        int                        payloadlen,
                                  const void                      *payload,
                                        int                        qos,
                                        bool                       retain,
                                  const mosquitto_property *properties )
```

Publish a message on a given topic, with attached MQTT properties.
Use e.g. mosquitto_property_add_string and similar to create a list of properties, then attach them to this publish. Properties need freeing with mosquitto_property_free_all.
If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the PUBLISH message. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

`mosq`	a valid mosquitto instance.

mid	pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the publish callback to determine when the message has been sent. Note that although the MQTT protocol doesn't use message ids for messages with QoS=0, libmosquitto assigns them message ids so they can be tracked with this parameter.
topic	null terminated string of the topic to publish to.
payloadlen	the size of the payload (bytes). Valid values are between 0 and 268,435,455.
payload	pointer to the data to send. If payloadlen > 0 this must be a valid memory location.
qos	integer value 0, 1 or 2 indicating the Quality of Service to be used for the message.
retain	set to true to make the message retained.
properties	a valid mosquitto_property list, or NULL.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.
MOSQ_ERR_PROTOCOL	if there is a protocol error communicating with the broker.
MOSQ_ERR_PAYLOAD_SIZE	if payloadlen is too large.
MOSQ_ERR_MALFORMED_UTF8	if the topic is not valid UTF-8
MOSQ_ERR_DUPLICATE_PROPERTY	if a property is duplicated where it is forbidden.
MOSQ_ERR_PROTOCOL	if any property is invalid for use with PUBLISH.
MOSQ_ERR_QOS_NOT_SUPPORTED	if the QoS is greater than that supported by the broker.
MOSQ_ERR_OVERSIZE_PACKET	if the resulting packet would be larger than supported by the broker.

mosquitto_subscribe

```
libmosq_EXPORT int mosquitto_subscribe(struct mosquitto *mosq,
                                       int       *mid,
                                 const char      *sub,
                                       int        qos  )
```

Subscribe to a topic.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 SUBSCRIBE properties, use mosquitto_subscribe_v5 instead.

Parameters

mosq	a valid mosquitto instance.
mid	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the subscribe callback to determine when the message has been sent.
sub	the subscription pattern.
qos	the requested Quality of Service for this subscription.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.
MOSQ_ERR_MALFORMED_UTF8	if the topic is not valid UTF-8
MOSQ_ERR_OVERSIZE_PACKET	if the resulting packet would be larger than supported by the broker.

mosquitto_subscribe_v5

```
libmosq_EXPORT int mosquitto_subscribe_v5(struct mosquitto          *mosq,
                                          int               *mid,
                                    const char              *sub,
                                          int                qos,
                                          int                options,
                                    const mosquitto_property *properties)
```

Subscribe to a topic, with attached MQTT properties.
Use e.g. mosquitto_property_add_string and similar to create a list of properties, then attach them to this publish. Properties need freeing with mosquitto_property_free_all.
If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the PUBLISH message. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

mosq	a valid mosquitto instance.
mid	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the subscribe callback to determine when the message has been sent.
sub	the subscription pattern.
qos	the requested Quality of Service for this subscription.
options	options to apply to this subscription, OR'd together. Set to 0 to use the default options, otherwise choose from list of mqtt5_sub_options
properties	a valid mosquitto_property list, or NULL.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.
MOSQ_ERR_MALFORMED_UTF8	if the topic is not valid UTF-8
MOSQ_ERR_DUPLICATE_PROPERTY	if a property is duplicated where it is forbidden.
MOSQ_ERR_PROTOCOL	if any property is invalid for use with SUBSCRIBE.
MOSQ_ERR_OVERSIZE_PACKET	if the resulting packet would be larger than supported by the broker.

mosquitto_subscribe_multiple

```
libmosq_EXPORT int mosquitto_subscribe_multiple(
        struct mosquitto           *mosq,
               int                 *mid,
               int                  sub_count,
   char *const *const               sub,
               int                  qos,
               int                  options,
        const mosquitto_property *properties
)
```

Subscribe to multiple topics.

Parameters

`mosq`	a valid mosquitto instance.
`mid`	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the subscribe callback to determine when the message has been sent.
`sub_count`	the count of subscriptions to be made
`sub`	array of sub_count pointers, each pointing to a subscription string. The "char *const **const" datatype ensures that neither the array of pointers nor the strings that they point to are mutable. If you aren't familiar with this, just think of it as a safer "char *"**, equivalent to "const char *" for a simple string pointer.
`qos`	the requested Quality of Service for each subscription.
`options`	options to apply to this subscription, OR'd together. This argument is not used for MQTT v3 susbcriptions. Set to 0 to use the default options, otherwise choose from list of mqtt5_sub_options
`properties`	a valid mosquitto_property list, or NULL. Only used with MQTT v5 clients.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_MALFORMED_UTF8`	if a topic is not valid UTF-8
`MOSQ_ERR_OVERSIZE_PACKET`	if the resulting packet would be larger than supported by the broker.

mosquitto_unsubscribe

```
libmosq_EXPORT int mosquitto_unsubscribe(struct mosquitto *mosq,
                                         int       *mid,
                                   const char      *sub  )
```

Unsubscribe from a topic.

Parameters

`mosq`	a valid mosquitto instance.
`mid`	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the unsubscribe callback to determine when the message has been sent.
`sub`	the unsubscription pattern.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8
`MOSQ_ERR_OVERSIZE_PACKET`	if the resulting packet would be larger than supported by the broker.

mosquitto_unsubscribe_v5

```
libmosq_EXPORT int mosquitto_unsubscribe_v5(
   struct mosquitto          *mosq,
          int                *mid,
    const char                *sub,
    const mosquitto_property *properties
)
```

Unsubscribe from a topic, with attached MQTT properties.
It is valid to use this function for clients using all MQTT protocol versions. If you need to set MQTT v5 UNSUBSCRIBE properties, use mosquitto_unsubscribe_v5 instead.
Use e.g. mosquitto_property_add_string and similar to create a list of properties, then attach them to this publish. Properties need freeing with mosquitto_property_free_all.
If the mosquitto instance `mosq` is using MQTT v5, the `properties` argument will be applied to the PUBLISH message. For MQTT v3.1.1 and below, the `properties` argument will be ignored.

Set your client to use MQTT v5 immediately after it is created

mosquitto_int_option(mosq, MOSQ_OPT_PROTOCOL_VERSION, MQTT_PROTOCOL_V5);

Parameters

`mosq`	a valid mosquitto instance.
`mid`	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the unsubscribe callback to determine when the message has been sent.
`sub`	the unsubscription pattern.
`properties`	a valid mosquitto_property list, or NULL. Only used with MQTT v5 clients.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8
`MOSQ_ERR_DUPLICATE_PROPERTY`	if a property is duplicated where it is forbidden.
`MOSQ_ERR_PROTOCOL`	if any property is invalid for use with UNSUBSCRIBE.
`MOSQ_ERR_OVERSIZE_PACKET`	if the resulting packet would be larger than supported by the broker.

mosquitto_unsubscribe_multiple

```
libmosq_EXPORT int mosquitto_unsubscribe_multiple(
        struct mosquitto          *mosq,
               int                *mid,
               int                 sub_count,
   char *const *const              sub,
         const mosquitto_property *properties
)
```

Unsubscribe from multiple topics.

Parameters

`mosq`	a valid mosquitto instance.
`mid`	a pointer to an int. If not NULL, the function will set this to the message id of this particular message. This can be then used with the subscribe callback to determine when the message has been sent.

`sub_count`	the count of unsubscriptions to be made
`sub`	array of sub_count pointers, each pointing to an unsubscription string. The "char *const **const" datatype ensures that neither the array of pointers nor the strings that they point to are mutable. If you aren't familiar with this, just think of it as a safer "char *"**, equivalent to "const char *" for a simple string pointer.
`properties`	a valid mosquitto_property list, or NULL. Only used with MQTT v5 clients.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_MALFORMED_UTF8`	if a topic is not valid UTF-8
`MOSQ_ERR_OVERSIZE_PACKET`	if the resulting packet would be larger than supported by the broker.

Struct mosquitto_message helper functions

Summary

FUNCTIONS

mosquitto_message_copy	Copy the contents of a mosquitto message to another message.
mosquitto_message_free	Completely free a mosquitto_message struct.
mosquitto_message_free_contents	Free a mosquitto_message struct contents, leaving the struct unaffected.

FUNCTIONS

mosquitto_message_copy

```
libmosq_EXPORT int mosquitto_message_copy(      struct mosquitto_message *dst,
                                          const struct mosquitto_message *src )
```

Copy the contents of a mosquitto message to another message. Useful for preserving a message received in the on_message() callback.

Parameters

`dst`	a pointer to a valid mosquitto_message struct to copy to.
`src`	a pointer to a valid mosquitto_message struct to copy from.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.

See Also

mosquitto_message_free

mosquitto_message_free

```
libmosq_EXPORT void mosquitto_message_free(struct mosquitto_message **message)
```

Completely free a mosquitto_message struct.

Parameters

`message`	pointer to a mosquitto_message pointer to free.

See Also

mosquitto_message_copy, mosquitto_message_free_contents

mosquitto_message_free_contents

```
libmosq_EXPORT void mosquitto_message_free_contents(
   struct mosquitto_message *message
)
```

Free a mosquitto_message struct contents, leaving the struct unaffected.

Parameters

`message`	pointer to a mosquitto_message struct to free its contents.

See Also

mosquitto_message_copy, mosquitto_message_free

Network loop (managed by libmosquitto)

The internal network loop must be called at a regular interval. The two recommended approaches are to use either mosquitto_loop_forever or mosquitto_loop_start. mosquitto_loop_forever is a blocking call and is suitable for the situation where you only want to handle incoming messages in callbacks. mosquitto_loop_start is a non-blocking call, it creates a separate thread to run the loop for you. Use this function when you have other tasks you need to run at the same time as the MQTT client, e.g. reading data from a sensor.

Summary

FUNCTIONS

mosquitto_loop_forever	This function call loop() for you in an infinite blocking loop.
mosquitto_loop_start	This is part of the threaded client interface.
mosquitto_loop_stop	This is part of the threaded client interface.
mosquitto_loop	The main network loop for the client.

FUNCTIONS

mosquitto_loop_forever

```
libmosq_EXPORT int mosquitto_loop_forever(struct mosquitto *mosq,
                                          int        timeout,
                                          int        max_packets)
```

This function call loop() for you in an infinite blocking loop. It is useful for the case where you only want to run the MQTT client loop in your program.
It handles reconnecting in case server connection is lost. If you call mosquitto_disconnect() in a callback it will return.

Parameters

`mosq`	a valid mosquitto instance.
`timeout`	Maximum number of milliseconds to wait for network activity in the select() call before timing out. Set to 0 for instant return. Set negative to use the default of 1000ms.
`max_packets`	this parameter is currently unused and should be set to 1 for future compatibility.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_CONN_LOST`	if the connection to the broker was lost.
`MOSQ_ERR_PROTOCOL`	if there is a protocol error communicating with the broker.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_loop, mosquitto_loop_start

mosquitto_loop_start

```
libmosq_EXPORT int mosquitto_loop_start(struct mosquitto *mosq)
```

This is part of the threaded client interface. Call this once to start a new thread to process network traffic. This provides an alternative to repeatedly calling mosquitto_loop yourself.

Parameters

`mosq`	a valid mosquitto instance.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOT_SUPPORTED`	if thread support is not available.

See Also

mosquitto_connect_async, mosquitto_loop, mosquitto_loop_forever, mosquitto_loop_stop

mosquitto_loop_stop

```
libmosq_EXPORT int mosquitto_loop_stop(struct mosquitto *mosq,
                                       bool       force)
```

This is part of the threaded client interface. Call this once to stop the network thread previously created with mosquitto_loop_start. This call will block until the network thread finishes. For the network thread to end, you must have previously called mosquitto_disconnect or have set the force parameter to true.

Parameters

`mosq`	a valid mosquitto instance.
`force`	set to true to force thread cancellation. If false, mosquitto_disconnect must have already been called.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOT_SUPPORTED`	if thread support is not available.

See Also

mosquitto_loop, mosquitto_loop_start

mosquitto_loop

```
libmosq_EXPORT int mosquitto_loop(struct mosquitto *mosq,
                                  int        timeout,
                                  int        max_packets)
```

The main network loop for the client. This must be called frequently to keep communications between the client and broker working. This is carried out by mosquitto_loop_forever and mosquitto_loop_start, which are the recommended ways of handling the network loop. You may also use this function if you wish. It must not be called inside a callback.
If incoming data is present it will then be processed. Outgoing commands, from e.g. mosquitto_publish, are normally sent immediately that their function is called, but this is not always possible. mosquitto_loop will also attempt to send any remaining outgoing messages, which also includes commands that are part of the flow for messages with QoS>0.
This calls select() to monitor the client network socket. If you want to integrate mosquitto client operation with your own select() call, use mosquitto_socket, mosquitto_loop_read, mosquitto_loop_write and mosquitto_loop_misc.

Threads

Parameters

`mosq`	a valid mosquitto instance.
`timeout`	Maximum number of milliseconds to wait for network activity in the select() call before timing out. Set to 0 for instant return. Set negative to use the default of 1000ms.
`max_packets`	this parameter is currently unused and should be set to 1 for future compatibility.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_NO_CONN`	if the client isn't connected to a broker.
`MOSQ_ERR_CONN_LOST`	if the connection to the broker was lost.
`MOSQ_ERR_PROTOCOL`	if there is a protocol error communicating with the broker.
`MOSQ_ERR_ERRNO`	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows. See Also: mosquitto_loop_forever, mosquitto_loop_start, mosquitto_loop_stop

Network loop (for use in other event loops)

Summary

FUNCTIONS

mosquitto_loop_read	Carry out network read operations.
mosquitto_loop_write	Carry out network write operations.
mosquitto_loop_misc	Carry out miscellaneous operations required as part of the network loop.

FUNCTIONS

mosquitto_loop_read

```
libmosq_EXPORT int mosquitto_loop_read(struct mosquitto *mosq,
                                       int        max_packets)
```

Carry out network read operations. This should only be used if you are not using mosquitto_loop() and are monitoring the client network socket for activity yourself.

Parameters

mosq	a valid mosquitto instance.
max_packets	this parameter is currently unused and should be set to 1 for future compatibility.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.
MOSQ_ERR_CONN_LOST	if the connection to the broker was lost.
MOSQ_ERR_PROTOCOL	if there is a protocol error communicating with the broker.
MOSQ_ERR_ERRNO	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_socket, mosquitto_loop_write, mosquitto_loop_misc

mosquitto_loop_write

```
libmosq_EXPORT int mosquitto_loop_write(struct mosquitto *mosq,
                                        int        max_packets)
```

Carry out network write operations. This should only be used if you are not using mosquitto_loop() and are monitoring the client network socket for activity yourself.

Parameters

mosq	a valid mosquitto instance.
max_packets	this parameter is currently unused and should be set to 1 for future compatibility.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.
MOSQ_ERR_CONN_LOST	if the connection to the broker was lost.
MOSQ_ERR_PROTOCOL	if there is a protocol error communicating with the broker.
MOSQ_ERR_ERRNO	if a system call returned an error. The variable errno contains the error code, even on Windows. Use strerror_r() where available or FormatMessage() on Windows.

See Also

mosquitto_socket, mosquitto_loop_read, mosquitto_loop_misc, mosquitto_want_write

mosquitto_loop_misc

```
libmosq_EXPORT int mosquitto_loop_misc(struct mosquitto *mosq)
```

Carry out miscellaneous operations required as part of the network loop. This should only be used if you are not using mosquitto_loop() and are monitoring the client network socket for activity yourself.

This function deals with handling PINGs and checking whether messages need to be retried, so should be called fairly frequently, around once per second is sufficient.

Parameters

mosq	a valid mosquitto instance.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NO_CONN	if the client isn't connected to a broker.

See Also

mosquitto_socket, mosquitto_loop_read, mosquitto_loop_write

Network loop (helper functions)

Summary

FUNCTIONS

mosquitto_socket	Return the socket handle for a mosquitto instance.
mosquitto_want_write	Returns true if there is data ready to be written on the socket.
mosquitto_threaded_set	Used to tell the library that your application is using threads, but not using mosquitto_loop_start.

FUNCTIONS

mosquitto_socket

```
libmosq_EXPORT int mosquitto_socket(struct mosquitto *mosq)
```

Return the socket handle for a mosquitto instance. Useful if you want to include a mosquitto client in your own select() calls.

Parameters

`mosq` a valid mosquitto instance.

Returns

The socket for the mosquitto client or -1 on failure.

mosquitto_want_write

```
libmosq_EXPORT bool mosquitto_want_write(struct mosquitto *mosq)
```

Returns true if there is data ready to be written on the socket.

Parameters

`mosq` a valid mosquitto instance.

See Also

mosquitto_socket, mosquitto_loop_read, mosquitto_loop_write

mosquitto_threaded_set

```
libmosq_EXPORT int mosquitto_threaded_set(struct mosquitto *mosq,
                                          bool       threaded)
```

Used to tell the library that your application is using threads, but not using mosquitto_loop_start. The library operates slightly differently when not in threaded mode in order to simplify its operation. If you are managing your own threads and do not use this function you will experience crashes due to race conditions.
When using mosquitto_loop_start, this is set automatically.

Parameters

`mosq` a valid mosquitto instance.

`threaded` true if your application is using threads, false otherwise.

Client options

Summary

FUNCTIONS

mosquitto_opts_set	Used to set options for the client.
mosquitto_int_option	Used to set integer options for the client.
mosquitto_string_option	Used to set const char* options for the client.
mosquitto_void_option	Used to set void* options for the client.
mosquitto_reconnect_delay_set	Control the behaviour of the client when it has unexpectedly disconnected in mosquitto_loop_forever or after mosquitto_loop_start.
mosquitto_max_inflight_messages_set	This function is deprected.
mosquitto_message_retry_set	This function now has no effect.
mosquitto_user_data_set	When mosquitto_new is called, the pointer given as the "obj" parameter will be passed to the callbacks as user data.
mosquitto_userdata	Retrieve the "userdata" variable for a mosquitto client.

FUNCTIONS

mosquitto_opts_set

```
libmosq_EXPORT int mosquitto_opts_set(struct mosquitto  *mosq,
                                      enum mosq_opt_t  option,
                                           void       *value  )
```

Used to set options for the client.
This function is deprecated, the replacement mosquitto_int_option, mosquitto_string_option and mosquitto_void_option functions should be used instead.

Parameters

`mosq` a valid mosquitto instance.

`option` the option to set.

`value` the option specific value.

Options

`MOSQ_OPT_PROTOCOL_VERSION`	Value must be an int, set to either MQTT_PROTOCOL_V31 or MQTT_PROTOCOL_V311. Must be set before the client connects. Defaults to MQTT_PROTOCOL_V31.
`MOSQ_OPT_SSL_CTX`	Pass an openssl SSL_CTX to be used when creating TLS connections rather than libmosquitto creating its own. This must be called before connecting to have any effect. If you use this option, the onus is on you to ensure that you are using secure settings. Setting to NULL means that libmosquitto will use its own SSL_CTX if TLS is to be used. This option is only available for openssl 1.1.0 and higher.
`MOSQ_OPT_SSL_CTX_WITH_DEFAULTS`	Value must be an int set to 1 or 0. If set to 1, then the user specified SSL_CTX passed in using MOSQ_OPT_SSL_CTX will have the default options applied to it. This means that you only need to change the values that are relevant to you. If you use this option then you must configure the TLS options as normal, i.e. you should use mosquitto_tls_set to configure the cafile/capath as a minimum. This option is only available for openssl 1.1.0 and higher.

mosquitto_int_option

```
libmosq_EXPORT int mosquitto_int_option(struct mosquitto  *mosq,
                                        enum mosq_opt_t  option,
                                           int          value  )
```

Used to set integer options for the client.

Parameters

`mosq` a valid mosquitto instance.

`option` the option to set.

`value` the option specific value.

Options

`MOSQ_OPT_TCP_NODELAY`	Set to 1 to disable Nagle's algorithm on client sockets. This has the effect of reducing latency of individual messages at the potential cost of increasing the number of packets being sent. Defaults to 0, which means Nagle remains enabled.
`MOSQ_OPT_PROTOCOL_VERSION`	Value must be set to either MQTT_PROTOCOL_V31, MQTT_PROTOCOL_V311, or MQTT_PROTOCOL_V5. Must be set before the client connects. Defaults to MQTT_PROTOCOL_V311.
`MOSQ_OPT_RECEIVE_MAXIMUM`	Value can be set between 1 and 65535 inclusive, and represents the maximum number of incoming QoS 1 and QoS 2 messages that this client wants to process at once. Defaults to 20. This option is not valid for MQTT v3.1 or v3.1.1 clients. Note that if the MQTT_PROP_RECEIVE_MAXIMUM property is in the proplist passed to mosquitto_connect_v5(), then that property will override this option. Using this option is the recommended method however.
`MOSQ_OPT_SEND_MAXIMUM`	Value can be set between 1 and 65535 inclusive, and represents the maximum number of outgoing QoS 1 and QoS 2 messages that this client will attempt to have "in flight" at once. Defaults to 20. This option is not valid for MQTT v3.1 or v3.1.1 clients. Note that if the broker being connected to sends a MQTT_PROP_RECEIVE_MAXIMUM property that has a lower value than this option, then the broker provided value will be used.
`MOSQ_OPT_SSL_CTX_WITH_DEFAULTS`	If value is set to a non zero value, then the user specified SSL_CTX passed in using MOSQ_OPT_SSL_CTX will have the default options applied to it. This means that you only need to change the values that are relevant to you. If you use this option then you must configure the TLS options as normal, i.e. you should use mosquitto_tls_set to configure the cafile/capath as a minimum. This option is only available for openssl 1.1.0 and higher.
`MOSQ_OPT_TLS_OCSP_REQUIRED`	Set whether OCSP checking on TLS connections is required. Set to 1 to enable checking, or 0 (the default) for no checking.
`MOSQ_OPT_TLS_USE_OS_CERTS`	Set to 1 to instruct the client to load and trust OS provided CA certificates for use with TLS connections. Set to 0 (the default) to only use manually specified CA certs.

mosquitto_string_option

```
libmosq_EXPORT int mosquitto_string_option(struct mosquitto  *mosq,
                                           enum mosq_opt_t  option,
                                          const char       *value  )
```

Used to set const char* options for the client.

Parameters

`mosq` a valid mosquitto instance.

`option` the option to set.

`value` the option specific value.

Options

`MOSQ_OPT_TLS_ENGINE`	Configure the client for TLS Engine support. Pass a TLS Engine ID to be used when creating TLS connections. Must be set before mosquitto_connect.
`MOSQ_OPT_TLS_KEYFORM`	Configure the client to treat the keyfile differently depending on its type. Must be set before mosquitto_connect. Set as either "pem" or "engine", to determine from where the private key for a TLS connection will be obtained. Defaults to "pem", a normal private key file.
`MOSQ_OPT_TLS_KPASS_SHA1`	Where the TLS Engine requires the use of a password to be accessed, this option allows a hex encoded SHA1 hash of the private key password to be passed to the engine directly. Must be set before mosquitto_connect.
`MOSQ_OPT_TLS_ALPN`	If the broker being connected to has multiple services available on a single TLS port, such as both MQTT and WebSockets, use this option to configure the ALPN option for the connection.
`MOSQ_OPT_BIND_ADDRESS`	Set the hostname or ip address of the local network interface to bind to when connecting.

mosquitto_void_option

```
libmosq_EXPORT int mosquitto_void_option(struct mosquitto  *mosq,
                                         enum mosq_opt_t  option,
                                              void       *value  )
```

Used to set void* options for the client.

Parameters

`mosq` a valid mosquitto instance.

`option` the option to set.

`value` the option specific value.

Options

`MOSQ_OPT_SSL_CTX`	Pass an openssl SSL_CTX to be used when creating TLS connections rather than libmosquitto creating its own. This must be called before connecting to have any effect. If you use this option, the onus is on you to ensure that you are using secure settings. Setting to NULL means that libmosquitto will use its own SSL_CTX if TLS is to be used. This option is only available for openssl 1.1.0 and higher.

mosquitto_reconnect_delay_set

```
libmosq_EXPORT int mosquitto_reconnect_delay_set(
     struct mosquitto *mosq,
   unsigned int        reconnect_delay,
   unsigned int        reconnect_delay_max,
            bool       reconnect_exponential_backoff
)
```

Control the behaviour of the client when it has unexpectedly disconnected in mosquitto_loop_forever or after mosquitto_loop_start. The default behaviour if this function is not used is to repeatedly attempt to reconnect with a delay of 1 second until the connection succeeds.

Use reconnect_delay parameter to change the delay between successive reconnection attempts. You may also enable exponential backoff of the time between reconnections by setting reconnect_exponential_backoff to true and set an upper bound on the delay with reconnect_delay_max.

Example 1

delay=2, delay_max=10, exponential_backoff=False Delays would be: 2, 4, 6, 8, 10, 10, ...

Example 2

delay=3, delay_max=30, exponential_backoff=True Delays would be: 3, 6, 12, 24, 30, 30, ...

Parameters

`mosq`	a valid mosquitto instance.
`reconnect_delay`	the number of seconds to wait between reconnects.
`reconnect_delay_max`	the maximum number of seconds to wait between reconnects.
`reconnect_exponential_backoff`	use exponential backoff between reconnect attempts. Set to true to enable exponential backoff.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.

mosquitto_max_inflight_messages_set

```
libmosq_EXPORT int mosquitto_max_inflight_messages_set(
    struct mosquitto *mosq,
   unsigned int        max_inflight_messages
)
```

This function is deprected. Use the mosquitto_int_option function with the MOSQ_OPT_SEND_MAXIMUM option instead.
Set the number of QoS 1 and 2 messages that can be "in flight" at one time. An in flight message is part way through its delivery flow. Attempts to send further messages with mosquitto_publish will result in the messages being queued until the number of in flight messages reduces.
A higher number here results in greater message throughput, but if set higher than the maximum in flight messages on the broker may lead to delays in the messages being acknowledged.
Set to 0 for no maximum.

Parameters

`mosq`	a valid mosquitto instance.
`max_inflight_messages`	the maximum number of inflight messages. Defaults to 20.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.

mosquitto_message_retry_set

```
libmosq_EXPORT void mosquitto_message_retry_set(  struct mosquitto *mosq,
                                                  unsigned int        message_retry)
```

This function now has no effect.

mosquitto_user_data_set

```
libmosq_EXPORT void mosquitto_user_data_set(struct mosquitto *mosq,
                                            void      *obj  )
```

When mosquitto_new is called, the pointer given as the "obj" parameter will be passed to the callbacks as user data. The mosquitto_user_data_set function allows this obj parameter to be updated at any time. This function will not modify the memory pointed to by the current user data pointer. If it is dynamically allocated memory you must free it yourself.

Parameters

`mosq`	a valid mosquitto instance.
`obj`	A user pointer that will be passed as an argument to any callbacks that are specified.

mosquitto_userdata

```
libmosq_EXPORT void *mosquitto_userdata(struct mosquitto *mosq)
```

Retrieve the "userdata" variable for a mosquitto client.

Parameters

`mosq`	a valid mosquitto instance.

Returns

A pointer to the userdata member variable.

TLS support

Summary

FUNCTIONS

mosquitto_tls_set	Configure the client for certificate based SSL/TLS support.
mosquitto_tls_insecure_set	Configure verification of the server hostname in the server certificate.
mosquitto_tls_opts_set	Set advanced SSL/TLS options.
mosquitto_tls_psk_set	Configure the client for pre-shared-key based TLS support.
mosquitto_ssl_get	Retrieve a pointer to the SSL structure used for TLS connections in this client.

FUNCTIONS

mosquitto_tls_set

```
libmosq_EXPORT int mosquitto_tls_set(
   struct mosquitto *mosq,
    const char      *cafile,
    const char      *capath,
    const char      *certfile,
    const char      *keyfile,
          int        (*pw_callback)(char *buf, int size, int rwflag, void *userdata)
)
```

Configure the client for certificate based SSL/TLS support. Must be called before mosquitto_connect.
Cannot be used in conjunction with mosquitto_tls_psk_set.
Define the Certificate Authority certificates to be trusted (ie. the server certificate must be signed with one of these certificates) using cafile.
If the server you are connecting to requires clients to provide a certificate, define certfile and keyfile with your client certificate and private key. If your private key is encrypted, provide a password callback function or you will have to enter the password at the command line.

Parameters

`mosq`	a valid mosquitto instance.
`cafile`	path to a file containing the PEM encoded trusted CA certificate files. Either cafile or capath must not be NULL.
`capath`	path to a directory containing the PEM encoded trusted CA certificate files. See mosquitto.conf for more details on configuring this directory. Either cafile or capath must not be NULL.
`certfile`	path to a file containing the PEM encoded certificate file for this client. If NULL, keyfile must also be NULL and no client certificate will be used.
`keyfile`	path to a file containing the PEM encoded private key for this client. If NULL, certfile must also be NULL and no client certificate will be used.
`pw_callback`	if keyfile is encrypted, set pw_callback to allow your client to pass the correct password for decryption. If set to NULL, the password must be entered on the command line. Your callback must write the password into "buf", which is "size" bytes long. The return value must be the length of the password. "userdata" will be set to the calling mosquitto instance. The mosquitto userdata member variable can be retrieved using mosquitto_userdata.

Returns

`MOSQ_ERR_SUCCESS`	on success.
`MOSQ_ERR_INVAL`	if the input parameters were invalid.

MOSQ_ERR_NOMEM	if an out of memory condition occurred.

See Also

mosquitto_tls_opts_set, mosquitto_tls_psk_set, mosquitto_tls_insecure_set, mosquitto_userdata

mosquitto_tls_insecure_set

```
libmosq_EXPORT int mosquitto_tls_insecure_set(struct mosquitto *mosq,
                                              bool       value)
```

Configure verification of the server hostname in the server certificate. If value is set to true, it is impossible to guarantee that the host you are connecting to is not impersonating your server. This can be useful in initial server testing, but makes it possible for a malicious third party to impersonate your server through DNS spoofing, for example. Do not use this function in a real system. Setting value to true makes the connection encryption pointless. Must be called before mosquitto_connect.

Parameters

mosq	a valid mosquitto instance.
value	if set to false, the default, certificate hostname checking is performed. If set to true, no hostname checking is performed and the connection is insecure.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.

See Also

mosquitto_tls_set

mosquitto_tls_opts_set

```
libmosq_EXPORT int mosquitto_tls_opts_set(struct mosquitto *mosq,
                                          int        cert_reqs,
                                          const char      *tls_version,
                                          const char      *ciphers     )
```

Set advanced SSL/TLS options. Must be called before mosquitto_connect.

Parameters

mosq	a valid mosquitto instance.
cert_reqs	an integer defining the verification requirements the client will impose on the server. This can be one of:

- SSL_VERIFY_NONE (0): the server will not be verified in any way.
- SSL_VERIFY_PEER (1): the server certificate will be verified and the connection aborted if the verification fails. The default and recommended value is SSL_VERIFY_PEER. Using SSL_VERIFY_NONE provides no security.

tls_version	the version of the SSL/TLS protocol to use as a string. If NULL, the default value is used. The default value and the available values depend on the version of openssl that the library was compiled against. For openssl >= 1.0.1, the available options are tlsv1.2, tlsv1.1 and tlsv1, with tlv1.2 as the default. For openssl < 1.0.1, only tlsv1 is available.
ciphers	a string describing the ciphers available for use. See the "openssl ciphers" tool for more information. If NULL, the default ciphers will be used.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.

See Also

mosquitto_tls_set

mosquitto_tls_psk_set

```
libmosq_EXPORT int mosquitto_tls_psk_set(struct mosquitto *mosq,
                                         const char      *psk,
                                         const char      *identity,
                                         const char      *ciphers  )
```

Configure the client for pre-shared-key based TLS support. Must be called before mosquitto_connect.
Cannot be used in conjunction with mosquitto_tls_set.

Parameters

mosq	a valid mosquitto instance.
psk	the pre-shared-key in hex format with no leading "0x".
identity	the identity of this client. May be used as the username depending on the server settings.
ciphers	a string describing the PSK ciphers available for use. See the "openssl ciphers" tool for more information. If NULL, the default ciphers will be used.

Returns

MOSQ_ERR_SUCCESS	on success.
MOSQ_ERR_INVAL	if the input parameters were invalid.
MOSQ_ERR_NOMEM	if an out of memory condition occurred.

See Also

mosquitto_tls_set

mosquitto_ssl_get

```
libmosq_EXPORT void *mosquitto_ssl_get(struct mosquitto *mosq)
```

Retrieve a pointer to the SSL structure used for TLS connections in this client. This can be used in e.g. the connect callback to carry out additional verification steps.

Parameters

mosq	a valid mosquitto instance

Returns

A valid pointer to an openssl SSL structure	if the client is using TLS.
NULL	if the client is not using TLS, or TLS support is not compiled in.

Callbacks

Summary

FUNCTIONS

mosquitto_connect_callback_set	Set the connect callback.
mosquitto_connect_with_flags_callback_set	Set the connect callback.
mosquitto_connect_v5_callback_set	Set the connect callback.
mosquitto_disconnect_callback_set	Set the disconnect callback.
mosquitto_disconnect_v5_callback_set	Set the disconnect callback.
mosquitto_publish_callback_set	Set the publish callback.
mosquitto_publish_v5_callback_set	Set the publish callback.
mosquitto_message_callback_set	Set the message callback.
mosquitto_message_v5_callback_set	Set the message callback.
mosquitto_subscribe_callback_set	Set the subscribe callback.
mosquitto_subscribe_v5_callback_set	Set the subscribe callback.
mosquitto_unsubscribe_callback_set	Set the unsubscribe callback.
mosquitto_unsubscribe_v5_callback_set	Set the unsubscribe callback.
mosquitto_log_callback_set	Set the logging callback.

FUNCTIONS

mosquitto_connect_callback_set

```
libmosq_EXPORT void mosquitto_connect_callback_set(
   struct mosquitto *mosq,
          void       (*on_connect)(struct mosquitto *, void *, int)
)
```

Set the connect callback. This is called when the broker sends a CONNACK message in response to a connection.

Parameters

`mosq`	a valid mosquitto instance.
`on_connect`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int rc)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`rc`	the return code of the connection response. The values are defined by the MQTT protocol version in use. For MQTT v5.0, look at section 3.2.2.2 Connect Reason code: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html For MQTT v3.1.1, look at section 3.2.2.3 Connect Return code: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

mosquitto_connect_with_flags_callback_set

```
libmosq_EXPORT void mosquitto_connect_with_flags_callback_set(
   struct mosquitto *mosq,
          void       (*on_connect)(struct mosquitto *, void *, int, int)
)
```

Set the connect callback. This is called when the broker sends a CONNACK message in response to a connection.

Parameters

`mosq`	a valid mosquitto instance.
`on_connect`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int rc)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`rc`	the return code of the connection response. The values are defined by the MQTT protocol version in use. For MQTT v5.0, look at section 3.2.2.2 Connect Reason code: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html For MQTT v3.1.1, look at section 3.2.2.3 Connect Return code: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
`flags`	the connect flags.

mosquitto_connect_v5_callback_set

```
libmosq_EXPORT void mosquitto_connect_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_connect)(struct mosquitto *, void *, int, int, const mosquitto_property *props)
)
```

Set the connect callback. This is called when the broker sends a CONNACK message in response to a connection.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_connect`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int rc)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`rc`	the return code of the connection response. The values are defined by the MQTT protocol version in use. For MQTT v5.0, look at section 3.2.2.2 Connect Reason code: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html For MQTT v3.1.1, look at section 3.2.2.3 Connect Return code: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
`flags`	the connect flags.
`props`	list of MQTT 5 properties, or NULL

mosquitto_disconnect_callback_set

```
libmosq_EXPORT void mosquitto_disconnect_callback_set(
   struct mosquitto *mosq,
          void       (*on_disconnect)(struct mosquitto *, void *, int)
)
```

Set the disconnect callback. This is called when the broker has received the DISCONNECT command and has disconnected the client.

Parameters

`mosq`	a valid mosquitto instance.
`on_disconnect`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`rc`	integer value indicating the reason for the disconnect. A value of 0 means the client has called mosquitto_disconnect. Any other value indicates that the disconnect is unexpected.

mosquitto_disconnect_v5_callback_set

```
libmosq_EXPORT void mosquitto_disconnect_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_disconnect)(struct mosquitto *, void *, int, const mosquitto_property *props)
)
```

Set the disconnect callback. This is called when the broker has received the DISCONNECT command and has disconnected the client.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_disconnect`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`rc`	integer value indicating the reason for the disconnect. A value of 0 means the client has called mosquitto_disconnect. Any other value indicates that the disconnect is unexpected.
`props`	list of MQTT 5 properties, or NULL

mosquitto_publish_callback_set

```
libmosq_EXPORT void mosquitto_publish_callback_set(
   struct mosquitto *mosq,
          void       (*on_publish)(struct mosquitto *, void *, int)
)
```

Set the publish callback. This is called when a message initiated with mosquitto_publish has been sent to the broker successfully.

Parameters

`mosq`	a valid mosquitto instance.
`on_publish`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`mid`	the message id of the sent message.

mosquitto_publish_v5_callback_set

```
libmosq_EXPORT void mosquitto_publish_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_publish)(struct mosquitto *, void *, int, int, const mosquitto_property *props)
)
```

Set the publish callback. This is called when a message initiated with mosquitto_publish has been sent to the broker. This callback will be called both if the message is sent successfully, or if the broker responded with an error, which will be reflected in the reason_code parameter.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_publish`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`mid`	the message id of the sent message.
`reason_code`	the MQTT 5 reason code
`props`	list of MQTT 5 properties, or NULL

mosquitto_message_callback_set

```
libmosq_EXPORT void mosquitto_message_callback_set(
   struct mosquitto *mosq,
          void       (*on_message)(struct mosquitto *, void *, const struct mosquitto_message *)
)
```

Set the message callback. This is called when a message is received from the broker.

Parameters

`mosq`	a valid mosquitto instance.
`on_message`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, const struct mosquitto_message *message)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`message`	the message data. This variable and associated memory will be freed by the library after the callback completes. The client should make copies of any of the data it requires.

See Also

mosquitto_message_copy

mosquitto_message_v5_callback_set

```
libmosq_EXPORT void mosquitto_message_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_message)(struct mosquitto *, void *, const struct mosquitto_message *, const mosquitto_property *props)
)
```

Set the message callback. This is called when a message is received from the broker.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_message`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, const struct mosquitto_message *message)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`message`	the message data. This variable and associated memory will be freed by the library after the callback completes. The client should make copies of any of the data it requires.
`props`	list of MQTT 5 properties, or NULL

See Also

mosquitto_message_copy

mosquitto_subscribe_callback_set

```
libmosq_EXPORT void mosquitto_subscribe_callback_set(
   struct mosquitto *mosq,
          void       (*on_subscribe)(struct mosquitto *, void *, int, int, const int *)
)
```

Set the subscribe callback. This is called when the broker responds to a subscription request.

Parameters

`mosq`	a valid mosquitto instance.
`on_subscribe`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid, int qos_count, const int *granted_qos)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`mid`	the message id of the subscribe message.
`qos_count`	the number of granted subscriptions (size of granted_qos).
`granted_qos`	an array of integers indicating the granted QoS for each of the subscriptions.

mosquitto_subscribe_v5_callback_set

```
libmosq_EXPORT void mosquitto_subscribe_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_subscribe)(struct mosquitto *, void *, int, int, const int *, const mosquitto_property *props)
)
```

Set the subscribe callback. This is called when the broker responds to a subscription request.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_subscribe`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid, int qos_count, const int *granted_qos)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`mid`	the message id of the subscribe message.
`qos_count`	the number of granted subscriptions (size of granted_qos).
`granted_qos`	an array of integers indicating the granted QoS for each of the subscriptions.
`props`	list of MQTT 5 properties, or NULL

mosquitto_unsubscribe_callback_set

```
libmosq_EXPORT void mosquitto_unsubscribe_callback_set(
   struct mosquitto *mosq,
          void       (*on_unsubscribe)(struct mosquitto *, void *, int)
)
```

Set the unsubscribe callback. This is called when the broker responds to a unsubscription request.

Parameters

`mosq`	a valid mosquitto instance.
`on_unsubscribe`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new
`mid`	the message id of the unsubscribe message.

mosquitto_unsubscribe_v5_callback_set

```
libmosq_EXPORT void mosquitto_unsubscribe_v5_callback_set(
   struct mosquitto *mosq,
          void       (*on_unsubscribe)(struct mosquitto *, void *, int, const mosquitto_property *props)
)
```

Set the unsubscribe callback. This is called when the broker responds to a unsubscription request.
It is valid to set this callback for all MQTT protocol versions. If it is used with MQTT clients that use MQTT v3.1.1 or earlier, then the `props` argument will always be NULL.

Parameters

`mosq`	a valid mosquitto instance.
`on_unsubscribe`	a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int mid)

Callback Parameters

`mosq`	the mosquitto instance making the callback.
`obj`	the user data provided in mosquitto_new

`mid` the message id of the unsubscribe message.

`props` list of MQTT 5 properties, or NULL

mosquitto_log_callback_set

```
libmosq_EXPORT void mosquitto_log_callback_set(
   struct mosquitto *mosq,
          void       (*on_log)(struct mosquitto *, void *, int, const char *)
)
```

Set the logging callback. This should be used if you want event logging information from the client library.

`mosq` a valid mosquitto instance.

`on_log` a callback function in the following form: void callback(struct mosquitto *mosq, void *obj, int level, const char *str)

Callback Parameters

`mosq` the mosquitto instance making the callback.

`obj` the user data provided in mosquitto_new

`level` the log message level from the values: MOSQ_LOG_INFO MOSQ_LOG_NOTICE MOSQ_LOG_WARNING MOSQ_LOG_ERR MOSQ_LOG_DEBUG

`str` the message string.

SOCKS5 proxy functions

Summary

FUNCTIONS

mosquitto_socks5_set	Configure the client to use a SOCKS5 proxy when connecting.

FUNCTIONS

mosquitto_socks5_set

```
libmosq_EXPORT int mosquitto_socks5_set(struct mosquitto *mosq,
                                        const char      *host,
                                              int        port,
                                        const char      *username,
                                        const char      *password )
```

Configure the client to use a SOCKS5 proxy when connecting. Must be called before connecting. “None” and “username/password” authentication is supported.

Parameters

`mosq` a valid mosquitto instance.

`host` the SOCKS5 proxy host to connect to.

`port` the SOCKS5 proxy port to use.

`username` if not NULL, use this username when authenticating with the proxy.

`password` if not NULL and username is not NULL, use this password when authenticating with the proxy.

Utility functions

Summary

FUNCTIONS

mosquitto_strerror	Call to obtain a const string description of a mosquitto error number.
mosquitto_connack_string	Call to obtain a const string description of an MQTT connection result.
mosquitto_reason_string	Call to obtain a const string description of an MQTT reason code.
mosquitto_string_to_command	Take a string input representing an MQTT command and convert it to the libmosquitto integer representation.
mosquitto_sub_topic_tokenise	Tokenise a topic or subscription string into an array of strings representing the topic hierarchy.
mosquitto_sub_topic_tokens_free	Free memory that was allocated in mosquitto_sub_topic_tokenise.
mosquitto_topic_matches_sub	Check whether a topic matches a subscription.
mosquitto_topic_matches_sub2	Check whether a topic matches a subscription.
mosquitto_pub_topic_check	Check whether a topic to be used for publishing is valid.
mosquitto_pub_topic_check2	Check whether a topic to be used for publishing is valid.
mosquitto_sub_topic_check	Check whether a topic to be used for subscribing is valid.
mosquitto_sub_topic_check2	Check whether a topic to be used for subscribing is valid.
mosquitto_validate_utf8	Helper function to validate whether a UTF-8 string is valid, according to the UTF-8 spec and the MQTT additions.

FUNCTIONS

mosquitto_strerror

```
libmosq_EXPORT const char *mosquitto_strerror(int mosq_errno)
```

Call to obtain a const string description of a mosquitto error number.

Parameters

`mosq_errno` a mosquitto error number.

Returns

A constant string describing the error.

mosquitto_connack_string

```
libmosq_EXPORT const char *mosquitto_connack_string(int connack_code)
```

Call to obtain a const string description of an MQTT connection result.

Parameters

`connack_code` an MQTT connection result.

Returns

A constant string describing the result.

mosquitto_reason_string

```
libmosq_EXPORT const char *mosquitto_reason_string(int reason_code)
```

Call to obtain a const string description of an MQTT reason code.

Parameters

`reason_code`	an MQTT reason code.

Returns

A constant string describing the reason.

mosquitto_string_to_command

```
libmosq_EXPORT int mosquitto_string_to_command(const char *str,
                                               int  *cmd )
```

Take a string input representing an MQTT command and convert it to the libmosquitto integer representation.

Parameters

`str`	the string to parse.
`cmd`	pointer to an int, for the result.

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	on an invalid input.

Example

```
mosquitto_string_to_command("CONNECT", &cmd);
// cmd == CMD_CONNECT
```

mosquitto_sub_topic_tokenise

```
libmosq_EXPORT int mosquitto_sub_topic_tokenise(const char *  subtopic,
                                                char ***topics,
                                                int  *  count    )
```

Tokenise a topic or subscription string into an array of strings representing the topic hierarchy.

For example

subtopic: “a/deep/topic/hierarchy”

Would result in

topics[0] = “a” topics[1] = “deep” topics[2] = “topic” topics[3] = “hierarchy”

and

subtopic: “/a/deep/topic/hierarchy/”

Would result in

topics[0] = NULL topics[1] = “a” topics[2] = “deep” topics[3] = “topic” topics[4] = “hierarchy”

Parameters

`subtopic`	the subscription/topic to tokenise
`topics`	a pointer to store the array of strings
`count`	an int pointer to store the number of items in the topics array.

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.
`MOSQ_ERR_MALFORMED_UTF8`	if the topic is not valid UTF-8

Example

```
char **topics;
int topic_count;
int i;

mosquitto_sub_topic_tokenise("$SYS/broker/uptime", &topics, &topic_count);

for(i=0; i<token_count; i++){
    printf("%d: %s\n", i, topics[i]);
}
```

See Also

mosquitto_sub_topic_tokens_free

mosquitto_sub_topic_tokens_free

```
libmosq_EXPORT int mosquitto_sub_topic_tokens_free(char ***topics,
                                                   int     count  )
```

Free memory that was allocated in mosquitto_sub_topic_tokenise.

Parameters

`topics`	pointer to string array.
`count`	count of items in string array.

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if the input parameters were invalid.

See Also

mosquitto_sub_topic_tokenise

mosquitto_topic_matches_sub

```
libmosq_EXPORT int mosquitto_topic_matches_sub(const char *sub,
                                               const char *topic,
                                                     bool *result)
```

Check whether a topic matches a subscription.

For example

foo/bar would match the subscription foo/# or +/bar non/matching would not match the subscription non/+/+

Parameters

`sub`	subscription string to check topic against.
`topic`	topic to check.
`result`	bool pointer to hold result. Will be set to true if the topic matches the subscription.

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.

mosquitto_topic_matches_sub2

```
libmosq_EXPORT int mosquitto_topic_matches_sub2(const char   *sub,
                                                      size_t  sublen,
                                                const char   *topic,
                                                      size_t  topiclen,
                                                      bool   *result   )
```

Check whether a topic matches a subscription.

For example

foo/bar would match the subscription foo/# or +/bar non/matching would not match the subscription non/+/+

Parameters

`sub`	subscription string to check topic against.
`sublen`	length in bytes of sub string
`topic`	topic to check.
`topiclen`	length in bytes of topic string
`result`	bool pointer to hold result. Will be set to true if the topic matches the subscription.

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if the input parameters were invalid.
`MOSQ_ERR_NOMEM`	if an out of memory condition occurred.

mosquitto_pub_topic_check

```
libmosq_EXPORT int mosquitto_pub_topic_check(const char *topic)
```

Check whether a topic to be used for publishing is valid.
This searches for + or # in a topic and checks its length.
This check is already carried out in mosquitto_publish and mosquitto_will_set, there is no need to call it directly before them. It may be useful if you wish to check the validity of a topic in advance of making a connection for example.

Parameters

`topic`	the topic to check

Returns

`MOSQ_ERR_SUCCESS`	for a valid topic
`MOSQ_ERR_INVAL`	if the topic contains a + or a #, or if it is too long.
`MOSQ_ERR_MALFORMED_UTF8`	if topic is not valid UTF-8

See Also

mosquitto_sub_topic_check

mosquitto_pub_topic_check2

```
libmosq_EXPORT int mosquitto_pub_topic_check2(const char   *topic,
                                                    size_t  topiclen)
```

Check whether a topic to be used for publishing is valid.
This searches for + or # in a topic and checks its length.
This check is already carried out in mosquitto_publish and mosquitto_will_set, there is no need to call it directly before them. It may be useful if you wish to check the validity of a topic in advance of making a connection for example.

Parameters

`topic`	the topic to check
`topiclen`	length of the topic in bytes

Returns

`MOSQ_ERR_SUCCESS`	for a valid topic
`MOSQ_ERR_INVAL`	if the topic contains a + or a #, or if it is too long.
`MOSQ_ERR_MALFORMED_UTF8`	if topic is not valid UTF-8

See Also

mosquitto_sub_topic_check

mosquitto_sub_topic_check

```
libmosq_EXPORT int mosquitto_sub_topic_check(const char *topic)
```

Check whether a topic to be used for subscribing is valid.
This searches for + or # in a topic and checks that they aren't in invalid positions, such as with foo/#/bar, foo/+bar or foo/bar#, and checks its length.
This check is already carried out in mosquitto_subscribe and mosquitto_unsubscribe, there is no need to call it directly before them. It may be useful if you wish to check the validity of a topic in advance of making a connection for example.

Parameters

`topic`	the topic to check

Returns

`MOSQ_ERR_SUCCESS`	for a valid topic
`MOSQ_ERR_INVAL`	if the topic contains a + or a # that is in an invalid position, or if it is too long.
`MOSQ_ERR_MALFORMED_UTF8`	if topic is not valid UTF-8

See Also

mosquitto_sub_topic_check

mosquitto_sub_topic_check2

```
libmosq_EXPORT int mosquitto_sub_topic_check2(const char   *topic,
                                              size_t  topiclen)
```

Check whether a topic to be used for subscribing is valid.
This searches for + or # in a topic and checks that they aren't in invalid positions, such as with foo/#/bar, foo/+bar or foo/bar#, and checks its length.
This check is already carried out in mosquitto_subscribe and mosquitto_unsubscribe, there is no need to call it directly before them. It may be useful if you wish to check the validity of a topic in advance of making a connection for example.

Parameters

`topic`	the topic to check
`topiclen`	the length in bytes of the topic

Returns

`MOSQ_ERR_SUCCESS`	for a valid topic
`MOSQ_ERR_INVAL`	if the topic contains a + or a # that is in an invalid position, or if it is too long.
`MOSQ_ERR_MALFORMED_UTF8`	if topic is not valid UTF-8

See Also

mosquitto_sub_topic_check

mosquitto_validate_utf8

```
libmosq_EXPORT int mosquitto_validate_utf8(const char *str,
                                           int   len )
```

Helper function to validate whether a UTF-8 string is valid, according to the UTF-8 spec and the MQTT additions.

Parameters

`str`	a string to check
`len`	the length of the string in bytes

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if str is NULL or len<0 or len>65536
`MOSQ_ERR_MALFORMED_UTF8`	if str is not valid UTF-8

One line client helper functions

Summary

FUNCTIONS

mosquitto_subscribe_simple	Helper function to make subscribing to a topic and retrieving some messages very straightforward.
mosquitto_subscribe_callback	Helper function to make subscribing to a topic and processing some messages very straightforward.

FUNCTIONS

mosquitto_subscribe_simple

```
libmosq_EXPORT int mosquitto_subscribe_simple(
         struct mosquitto_message **messages,
                int                 msg_count,
                bool                want_retained,
          const char              * topic,
                int                 qos,
          const char              * host,
                int                 port,
          const char              * client_id,
                int                 keepalive,
                bool                clean_session,
          const char              * username,
          const char              * password,
   const struct libmosquitto_will * will,
   const struct libmosquitto_tls  * tls
)
```

Helper function to make subscribing to a topic and retrieving some messages very straightforward.
This connects to a broker, subscribes to a topic, waits for msg_count messages to be received, then returns after disconnecting cleanly.

Parameters

`messages`	pointer to a "struct mosquitto_message *". The received messages will be returned here. On error, this will be set to NULL.
`msg_count`	the number of messages to retrieve.
`want_retained`	if set to true, stale retained messages will be treated as normal messages with regards to msg_count. If set to false, they will be ignored.
`topic`	the subscription topic to use (wildcards are allowed).
`qos`	the qos to use for the subscription.
`host`	the broker to connect to.
`port`	the network port the broker is listening on.
`client_id`	the client id to use, or NULL if a random client id should be generated.
`keepalive`	the MQTT keepalive value.
`clean_session`	the MQTT clean session flag.

username	the username string, or NULL for no username authentication.
password	the password string, or NULL for an empty password.
will	a libmosquitto_will struct containing will information, or NULL for no will.
tls	a libmosquitto_tls struct containing TLS related parameters, or NULL for no use of TLS.

Returns

MOSQ_ERR_SUCCESS	on success
Greater than 0	on error.

mosquitto_subscribe_callback

```
libmosq_EXPORT int mosquitto_subscribe_callback(
                int               (*callback)(struct mosquitto *, void *, const struct mosquitto_message *),
                int               *userdata,
          const char              *topic,
                int               qos,
          const char              *host,
                int               port,
          const char              *client_id,
                int               keepalive,
                bool              clean_session,
          const char              *username,
          const char              *password,
   const struct libmosquitto_will *will,
   const struct libmosquitto_tls  *tls
)
```

Helper function to make subscribing to a topic and processing some messages very straightforward.
This connects to a broker, subscribes to a topic, then passes received messages to a user provided callback. If the callback returns a 1, it then disconnects cleanly and returns.

Parameters

callback	a callback function in the following form: int callback(struct mosquitto *mosq, void *obj, const struct mosquitto_message *message) Note that this is the same as the normal on_message callback, except that it returns an int.
userdata	user provided pointer that will be passed to the callback.
topic	the subscription topic to use (wildcards are allowed).
qos	the qos to use for the subscription.
host	the broker to connect to.
port	the network port the broker is listening on.
client_id	the client id to use, or NULL if a random client id should be generated.
keepalive	the MQTT keepalive value.
clean_session	the MQTT clean session flag.
username	the username string, or NULL for no username authentication.
password	the password string, or NULL for an empty password.
will	a libmosquitto_will struct containing will information, or NULL for no will.
tls	a libmosquitto_tls struct containing TLS related parameters, or NULL for no use of TLS.

Returns

MOSQ_ERR_SUCCESS	on success
Greater than 0	on error.

Properties

Summary

FUNCTIONS

mosquitto_property_add_byte	Add a new byte property to a property list.
mosquitto_property_add_int16	Add a new int16 property to a property list.
mosquitto_property_add_int32	Add a new int32 property to a property list.
mosquitto_property_add_varint	Add a new varint property to a property list.
mosquitto_property_add_binary	Add a new binary property to a property list.
mosquitto_property_add_string	Add a new string property to a property list.
mosquitto_property_add_string_pair	Add a new string pair property to a property list.
mosquitto_property_identifier	Return the property identifier for a single property.
mosquitto_property_next	Return the next property in a property list.
mosquitto_property_read_byte	Attempt to read a byte property matching an identifier, from a property list or single property.
mosquitto_property_read_int16	Read an int16 property value from a property.
mosquitto_property_read_int32	Read an int32 property value from a property.
mosquitto_property_read_varint	Read a varint property value from a property.
mosquitto_property_read_binary	Read a binary property value from a property.
mosquitto_property_read_string	Read a string property value from a property.
mosquitto_property_read_string_pair	Read a string pair property value pair from a property.
mosquitto_property_free_all	Free all properties from a list of properties.
mosquitto_property_copy_all	
mosquitto_property_check_command	Check whether a property identifier is valid for the given command.
mosquitto_property_check_all	Check whether a list of properties are valid for a particular command, whether there are duplicates, and whether the values are valid where possible.
mosquitto_property_identifier_to_string	Return the property name as a string for a property identifier.
mosquitto_string_to_property_info	Parse a property name string and convert to a property identifier and data type.

FUNCTIONS

mosquitto_property_add_byte

```
libmosq_EXPORT int mosquitto_property_add_byte(mosquitto_property **proplist,
                                               int                 identifier,
                                               uint8_t             value      )
```

Add a new byte property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

proplist	pointer to mosquitto_property pointer, the list of properties
identifier	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
value	integer value for the new property

Returns

MOSQ_ERR_SUCCESS	on success
MOSQ_ERR_INVAL	if identifier is invalid, or if proplist is NULL
MOSQ_ERR_NOMEM	on out of memory

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_byte(&proplist, MQTT_PROP_PAYLOAD_FORMAT_IDENTIFIER, 1);
```

mosquitto_property_add_int16

```
libmosq_EXPORT int mosquitto_property_add_int16(mosquitto_property **proplist,
                                                int                 identifier,
                                                uint16_t            value      )
```

Add a new int16 property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

`proplist`	pointer to mosquitto_property pointer, the list of properties
`identifier`	property identifier (e.g. MQTT_PROP_RECEIVE_MAXIMUM)
`value`	integer value for the new property

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if identifier is invalid, or if proplist is NULL
`MOSQ_ERR_NOMEM`	on out of memory

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_int16(&proplist, MQTT_PROP_RECEIVE_MAXIMUM, 1000);
```

mosquitto_property_add_int32

```
libmosq_EXPORT int mosquitto_property_add_int32(mosquitto_property **proplist,
                                                int                 identifier,
                                                uint32_t            value      )
```

Add a new int32 property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

`proplist`	pointer to mosquitto_property pointer, the list of properties
`identifier`	property identifier (e.g. MQTT_PROP_MESSAGE_EXPIRY_INTERVAL)
`value`	integer value for the new property

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if identifier is invalid, or if proplist is NULL
`MOSQ_ERR_NOMEM`	on out of memory

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_int32(&proplist, MQTT_PROP_MESSAGE_EXPIRY_INTERVAL, 86400);
```

mosquitto_property_add_varint

```
libmosq_EXPORT int mosquitto_property_add_varint(mosquitto_property **proplist,
                                                 int                 identifier,
                                                 uint32_t            value      )
```

Add a new varint property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

`proplist`	pointer to mosquitto_property pointer, the list of properties
`identifier`	property identifier (e.g. MQTT_PROP_SUBSCRIPTION_IDENTIFIER)
`value`	integer value for the new property

Returns

`MOSQ_ERR_SUCCESS`	on success
`MOSQ_ERR_INVAL`	if identifier is invalid, or if proplist is NULL
`MOSQ_ERR_NOMEM`	on out of memory

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_varint(&proplist, MQTT_PROP_SUBSCRIPTION_IDENTIFIER, 1);
```

mosquitto_property_add_binary

```
libmosq_EXPORT int mosquitto_property_add_binary(      mosquitto_property **proplist,
                                                       int                 identifier,
                                                 const void              * value,
                                                       uint16_t            len        )
```

Add a new binary property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

`proplist`	pointer to mosquitto_property pointer, the list of properties
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to the property data
`len`	length of property data in bytes

Returns

MOSQ_ERR_SUCCESS	on success
MOSQ_ERR_INVAL	if identifier is invalid, or if proplist is NULL
MOSQ_ERR_NOMEM	on out of memory

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_binary(&proplist, MQTT_PROP_AUTHENTICATION_DATA, auth_data, auth_data_len);
```

mosquitto_property_add_string

```
libmosq_EXPORT int mosquitto_property_add_string(      mosquitto_property **proplist,
                                                       int                  identifier,
                                                 const char               * value      )
```

Add a new string property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

proplist	pointer to mosquitto_property pointer, the list of properties
identifier	property identifier (e.g. MQTT_PROP_CONTENT_TYPE)
value	string value for the new property, must be UTF-8 and zero terminated

Returns

MOSQ_ERR_SUCCESS	on success
MOSQ_ERR_INVAL	if identifier is invalid, if value is NULL, or if proplist is NULL
MOSQ_ERR_NOMEM	on out of memory
MOSQ_ERR_MALFORMED_UTF8	value is not valid UTF-8.

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_string(&proplist, MQTT_PROP_CONTENT_TYPE, "application/json");
```

mosquitto_property_add_string_pair

```
libmosq_EXPORT int mosquitto_property_add_string_pair(
         mosquitto_property **proplist,
         int                  identifier,
   const char               * name,
   const char               * value
)
```

Add a new string pair property to a property list.
If *proplist == NULL, a new list will be created, otherwise the new property will be appended to the list.

Parameters

proplist	pointer to mosquitto_property pointer, the list of properties
identifier	property identifier (e.g. MQTT_PROP_USER_PROPERTY)
name	string name for the new property, must be UTF-8 and zero terminated
value	string value for the new property, must be UTF-8 and zero terminated

Returns

MOSQ_ERR_SUCCESS	on success
MOSQ_ERR_INVAL	if identifier is invalid, if name or value is NULL, or if proplist is NULL
MOSQ_ERR_NOMEM	on out of memory
MOSQ_ERR_MALFORMED_UTF8	if name or value are not valid UTF-8.

Example

```
mosquitto_property *proplist = NULL;
mosquitto_property_add_string_pair(&proplist, MQTT_PROP_USER_PROPERTY, "client", "mosquitto_pub");
```

mosquitto_property_identifier

```
libmosq_EXPORT int mosquitto_property_identifier(
   const mosquitto_property *property
)
```

Return the property identifier for a single property.

Parameters

property	pointer to a valid mosquitto_property pointer.

Returns

A valid property identifier on success 0 - on error

mosquitto_property_next

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_next(
   const mosquitto_property *proplist
)
```

Return the next property in a property list. Use to iterate over a property list, e.g.:

```
for(prop = proplist; prop != NULL; prop = mosquitto_property_next(prop)){
 if(mosquitto_property_identifier(prop) == MQTT_PROP_CONTENT_TYPE){
     ...
 }
}
```

Parameters

proplist	pointer to mosquitto_property pointer, the list of properties

Returns

Pointer to the next item in the list NULL, if proplist is NULL, or if there are no more items in the list.

mosquitto_property_read_byte

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_byte(
   const mosquitto_property *proplist,
         int                 identifier,
         uint8_t            *value,
         bool                skip_first
)
```

Attempt to read a byte property matching an identifier, from a property list or single property. This function can search for multiple entries of the same identifier by using the returned value and skip_first. Note however that it is forbidden for most properties to be duplicated.
If the property is not found, *value will not be modified, so it is safe to pass a variable with a default value to be potentially overwritten:

```
uint16_t keepalive = 60; // default value
// Get value from property list, or keep default if not found.
mosquitto_property_read_int16(proplist, MQTT_PROP_SERVER_KEEP_ALIVE, &keepalive, false);
```

Parameters

`proplist`	mosquitto_property pointer, the list of properties or single property
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to store the value, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL.

Example

```
// proplist is obtained from a callback
mosquitto_property *prop;
prop = mosquitto_property_read_byte(proplist, identifier, &value, false);
while(prop){
    printf("value: %s\n", value);
    prop = mosquitto_property_read_byte(prop, identifier, &value);
}
```

mosquitto_property_read_int16

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_int16(
   const mosquitto_property *proplist,
         int                 identifier,
         uint16_t           *value,
         bool                skip_first
)
```

Read an int16 property value from a property.

Parameters

`property`	property to read
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to store the value, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL.

Example

See mosquitto_property_read_byte

mosquitto_property_read_int32

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_int32(
   const mosquitto_property *proplist,
         int                 identifier,
         uint32_t           *value,
         bool                skip_first
)
```

Read an int32 property value from a property.

Parameters

`property`	pointer to mosquitto_property pointer, the list of properties
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to store the value, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL.

Example

See mosquitto_property_read_byte

mosquitto_property_read_varint

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_varint(
   const mosquitto_property *proplist,
         int                 identifier,
         uint32_t           *value,
         bool                skip_first
)
```

Read a varint property value from a property.

Parameters

`property`	property to read
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to store the value, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL.

Example

See mosquitto_property_read_byte

mosquitto_property_read_binary

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_binary(
   const mosquitto_property * proplist,
         int                  identifier,
         void               **value,
         uint16_t           * len,
         bool                 skip_first
)
```

Read a binary property value from a property.
On success, value must be free()'d by the application.

Parameters

`property`	property to read
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to store the value, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL, or if an out of memory condition occurred.

Example

See mosquitto_property_read_byte

mosquitto_property_read_string

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_string(
   const mosquitto_property * proplist,
         int                  identifier,
         char               **value,
         bool                 skip_first
)
```

Read a string property value from a property.
On success, value must be free()'d by the application.

Parameters

`property`	property to read
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`value`	pointer to char*, for the property data to be stored in, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL, or if an out of memory condition occurred.

Example

See mosquitto_property_read_byte

mosquitto_property_read_string_pair

```
libmosq_EXPORT const mosquitto_property *mosquitto_property_read_string_pair(
   const mosquitto_property * proplist,
         int                  identifier,
         char               **name,
         char               **value,
         bool                 skip_first
)
```

Read a string pair property value pair from a property.
On success, name and value must be free()'d by the application.

Parameters

`property`	property to read
`identifier`	property identifier (e.g. MQTT_PROP_PAYLOAD_FORMAT_INDICATOR)
`name`	pointer to char* for the name property data to be stored in, or NULL if the name is not required.
`value`	pointer to char*, for the property data to be stored in, or NULL if the value is not required.
`skip_first`	boolean that indicates whether the first item in the list should be ignored or not. Should usually be set to false.

Returns

A valid property pointer if the property is found NULL, if the property is not found, or proplist is NULL, or if an out of memory condition occurred.

Example

See mosquitto_property_read_byte

mosquitto_property_free_all

```
libmosq_EXPORT void mosquitto_property_free_all(mosquitto_property **properties)
```

Free all properties from a list of properties. Frees the list and sets *properties to NULL.

Parameters

`properties`	list of properties to free

Example

```
mosquitto_properties *properties = NULL;
// Add properties
mosquitto_property_free_all(&properties);
```

mosquitto_property_copy_all

```
libmosq_EXPORT int mosquitto_property_copy_all(      mosquitto_property **dest,
                                                const mosquitto_property * src  )
```

Parameters

dest	pointer for new property list
src	property list

Returns

MOSQ_ERR_SUCCESS	on successful copy
MOSQ_ERR_INVAL	if dest is NULL
MOSQ_ERR_NOMEM	on out of memory (dest will be set to NULL)

mosquitto_property_check_command

```
libmosq_EXPORT int mosquitto_property_check_command(int command,
                                                    int identifier)
```

Check whether a property identifier is valid for the given command.

Parameters

command	MQTT command (e.g. CMD_CONNECT)
identifier	MQTT property (e.g. MQTT_PROP_USER_PROPERTY)

Returns

MOSQ_ERR_SUCCESS	if the identifier is valid for command
MOSQ_ERR_PROTOCOL	if the identifier is not valid for use with command.

mosquitto_property_check_all

```
libmosq_EXPORT int mosquitto_property_check_all(
        int                 command,
    const mosquitto_property *properties
)
```

Check whether a list of properties are valid for a particular command, whether there are duplicates, and whether the values are valid where possible.
Note that this function is used internally in the library whenever properties are passed to it, so in basic use this is not needed, but should be helpful to check property lists **before** the point of using them.

Parameters

command	MQTT command (e.g. CMD_CONNECT)
properties	list of MQTT properties to check.

Returns

MOSQ_ERR_SUCCESS	if all properties are valid
MOSQ_ERR_DUPLICATE_PROPERTY	if a property is duplicated where it is forbidden.
MOSQ_ERR_PROTOCOL	if any property is invalid

mosquitto_property_identifier_to_string

```
libmosq_EXPORT const char *mosquitto_property_identifier_to_string(
   int identifier
)
```

Return the property name as a string for a property identifier. The property name is as defined in the MQTT specification, with - as a separator, for example: payload-format-indicator.

Parameters

identifier	valid MQTT property identifier integer

Returns

A const string to the property name on success NULL on failure

mosquitto_string_to_property_info

```
libmosq_EXPORT int mosquitto_string_to_property_info(const char *propname,
                                                     int  *identifier,
                                                     int  *type       )
```

Parse a property name string and convert to a property identifier and data type. The property name is as defined in the MQTT specification, with - as a separator, for example: payload-format-indicator.

Parameters

propname	the string to parse
identifier	pointer to an int to receive the property identifier
type	pointer to an int to receive the property type

Returns

MOSQ_ERR_SUCCESS	on success
MOSQ_ERR_INVAL	if the string does not match a property

Example

```
mosquitto_string_to_property_info("response-topic", &id, &type);
// id == MQTT_PROP_RESPONSE_TOPIC
// type == MQTT_PROP_TYPE_STRING
```