wolfMQTT Documentation

2022-07-01

Contents

1 Introduction

This is an implementation of the MQTT (Message Queuing Telemetry Transport) Client written in C. This library was built from the ground up to be multi-platform, space conscience and extensible. It supports all Packet Types, all Quality of Service (QoS) levels 0-2 and supports SSL/TLS using the wolfSSL library. This implementation is based on the MQTT v3.1.1 specification.

1.1 Protocol Overview

MQTT is a lightweight open messaging protocol that was developed for constrained environments such as M2M (Machine to Machine) and IoT (Internet of Things), where a small code footprint is required. MQTT is based on the Pub/Sub messaging principle of publishing messages and subscribing to topics. The protocol efficiently packs messages to keep the overhead very low. The MQTT specification recommends TLS as a transport option to secure the protocol using port 8883 (secure-mqtt). Constrained devices can benefit from using TLS session resumption to reduce the reconnection cost.

MQTT defines QoS levels 0-2 to specify the delivery integrity required: 0 = At most once delivery: No acknowledgment. 1 = At least once delivery: Sends acknowledgment (PUBLISH_ACK). 2 = Exactly once delivery: Sends received (PUBLISH_REC), gets back released (PUBLISH_REL) and then sends complete (PUBLISH_COMP).

Highlights :

- A publish message payload can be up to 256MB (28 bits).
- Packet header remaining length is encoded using a scheme where the most significant bit (7) indicates an additional length byte.
- Packets which require a response must include a 16-bit packet Id. This needs to be unique for any outstanding transactions. Typically an incremented value.
- A client can provide a last will and testament upon connect, which will be delivered when the broker sees the client has disconnected or network keep-alive has expired.
- The packet types are: CONNECT, CONNECT_ACK, PUBLISH, PUBLISH_ACK, PUBLISH_REC, PUBLISH_REL, PUBLISH_COMP, SUBSCRIBE, SUBSCRIBE_ACK, UNSUBSCRIBE, UNSUBSCRIBE_ACK, PING_REQ, PING_RESP and DISCONNECT.
- The connect packet contains the ASCII string “MQTT” to define the protocol name. This can be useful for wireshark/sniffing.
- Multiple topics can be subscribed or unsubscribed in the same packet request.
- Each subscription topic must define a QoS level. The QoS level is confirmed in the subscription acknowledgment.
- A publish message can be sent or received by either the client or the broker.
- Publish messages can be flagged for retention on the broker.
- A QoS level 2 requires two round-trips to complete the delivery exchange confirmation.
- Strings are UTF-8 encoded.

See https://mqtt.org/documentation for additional MQTT documentation.

1.2 Broker compatibility

wolfMQTT client library has been tested with the following brokers:

- Adafruit IO by Adafruit
- AWS by Amazon
- Azure by Microsoft
- flespi by Gurtam
- HiveMQ and HiveMQ Cloud by HiveMQ GmbH
- IBM WIoTP Message Gateway by IBM
- Mosquitto by Eclipse
- Paho MQTT-SN Gateway by Eclipse
- VerneMQ by VerneMQ/Erlio

1.3 Specification Support

1.3.1 MQTT v3.1.1 Specification Support

The initially supported version with full specification support for all features and packets type such as:

- QoS 0-2
- Last Will and Testament (LWT)
- Client examples for: AWS, Azure IoT, IBM Watson, Firmware update, non-blocking and generic.

1.3.2 MQTT v5.0 Specification Support

The wolfMQTT client supports connecting to v5 enabled brokers when configured with the `--enable-mqtt5` option. Handling properties received from the server is accomplished via a callback when the `--enable-propcb` option is set. The following v5.0 specification features are supported by the wolfMQTT client:

- AUTH packet
- User properties
- Server connect ACK properties
- Format and content type for publish
- Server disconnect
- Reason codes and strings
- Maximum packet size
- Server assigned client identifier
- Subscription ID
- Topic Alias

The v5 enabled wolfMQTT client was tested with the following MQTT v5 brokers:

- Mosquitto

** Runs locally.

```
** ./examples/mqttclient/mqttclient -h localhost
```

- Flespi

** Requires an account tied token that is regenerated hourly.

```
** ./examples/mqttclient/mqttclient -h "mqtt.flespi.io" -u "<your-flespi-token>"
```

- VerneMQ MQTTv5 preview

** Runs locally.

```
** ./examples/mqttclient/mqttclient -h localhost
```

- HiveMQ 4.0.0 EAP

** Runs locally.

```
** ./examples/mqttclient/mqttclient -h localhost
```

- HiveMQ Cloud

```
** ./examples/mqttclient/mqttclient -h 833f87e253304692bd2b911f0c18dba1.s1.eu.hivemq.cloud -t -S -u
wolf1 -w NEZjcm7i8eRjFKF -p 8883
```

- Watson IoT Quickserver

** `./examples/wiot/wiot`

1.3.3 MQTT Sensor Network (MQTT-SN) Specification Support

The wolfMQTT SN Client implementation is based on the OASIS MQTT-SN v1.2 specification. The SN API is configured with the `--enable-sn` option. There is a separate API for the sensor network API, which all begin with the "SN_" prefix. The wolfMQTT SN Client operates over UDP, which is distinct from the wolfMQTT clients that use TCP. The following features are supported by the wolfMQTT SN Client:

- Register
- Will topic and message set up
- Will topic and message update
- All QoS levels
- Variable-sized packet length field

Unsupported features:

- Automatic gateway discovery is not implemented
- Multiple gateway handling

The SN client was tested using the Eclipse Paho MQTT-SN Gateway (https://github.com/eclipse/paho.mqtt-sn.embedded-c) running locally and on a separate network node. Instructions for building and running the gateway are in the project README.

2 Building wolfMQTT

wolfMQTT was written with portability in mind, and should generally be easy to build on most systems. If you have difficulty building, please don't hesitate to seek support through our **support forums** https://www.wolfssl.com/forums or contact us directly at **support@wolfssl.com** . This chapter explains how to build wolfMQTT on Unix and Windows, and provides guidance for building in a non-standard environment.

When using the autoconf / automake system to build, wolfMQTT uses a single Makefile to build all parts and examples of the library, which is both simpler and faster than using Makefiles recursively. If using the TLS features or the Firmware/Azure IoT Hub examples you'll need to have wolfSSL installed. For wolfSSL and wolfMQTT we recommend using config options below

`./configure --enable-ecc --enable-supportedcurves --enable-base64encode.`

For wolfSSL use `make && sudo make install`. If you get an error locating the libwolfssl.so, run `sudo ldconfig` from the wolfSSL directory.

2.1 Getting the Source Code

The most recent version of wolfMQTT can be downloaded from the wolfSSL downloads page here

Or from Github with the command:

```
git clone https://github.com/wolfSSL/wolfMQTT.git
```

2.2 Building on *nix/Mac

When building on Linux, *BSD, OS X, Solaris, or other* nix-like systems, use the autoconf system. If cloned from github, run the following three commands:

```
./autogen.sh
./configure
make
```

Otherwise, just run these two commands:

```
./configure
make
```

You can append any number of build options to `./configure`. For a list of available build options, run the command below:

```
./configure --help
```

To build wolfMQTT, run:

```
make
```

To install wolfMQTT run:

```
make install
```

You may need superuser privileges to install, in which case precede the command with sudo:

```
sudo make install
```

Notes: * If `wolfssl` was recently installed, run `sudo ldconfig` to update the linker cache. * Debug messages can be enabled using `--enable-debug` or `--enable-debug=verbose` (for extra logging). * For a list of build options run `./configure --help`. * The build options are generated in a file here: `wolfmqtt/options.h`.

To test the build, run the *mqttclient* program from the root wolfMQTT source directory:

```
./examples/mqttclient/mqttclient
```

If you want to build only the wolfMQTT library and not the additional items (examples), you can run the following command from the wolfMQTT root directory:

```
make src/libwolfmqtt.la
```

2.3 Building on Windows

Visual Studio :

For building wolfMQTT with TLS support in Visual Studio:

1. Open the <wolfssl-root>/wolfssl64.sln.
2. Re-target for your Visual Studio version (right-click on solution and choose Retarget solution).
3. Make sure the Debug DLL or Release DLL configuration is selected. Make note if you are building 32-bit x86 or 64-bit x64.
4. Build the wolfSSL solution.
5. Copy the wolfssl.lib and wolfssl.dll files into <wolfmqtt-root>.
    - For DLL Debug with x86 the files are in: DLL Debug.
    - For DLL Release with x86 the files are in: DLL Release.
    - For DLL Debug with x64 the files are in: x64/DLL Debug.
    - For DLL Release with x64 the files are in: x64/DLL Release.
6. Open the <wolfmqtt-root>/wolfmqtt.sln solution.
7. Make sure you have the same architecture (x86 or x64 selected) as used in wolfSSL above.
8. By default the include path for the wolfssl headers is ../../wolfssl/. If your wolfssl root location is different you can go into the project settings and adjust this in C/C++ -> General -> Additional Include Directories.
9. Configure your Visual Studio build settings using wolfmqtt/vs_settings.h.
10. Build the wolfMQTT solution.

The wolfmqtt.sln solution is included for Visual Studio 2015 in the root directory of the install. To test each build, choose “Build All” from the Visual Studio menu and then run the mqttclient program. To edit build options in the Visual Studio project, select your desired project (wolfmqtt, mqttclient) and browse to the “Properties” panel.

For instructions on building the required wolfssl.dll see here. When done copy the wolfssl.dll and wolfssl.lib into the wolfMQTT root. The project also assumes the wolfSSL headers are located ../wolfssl/.

Cygwin : If using Cygwin, or other toolsets for Windows that provides *nix-like commands and functionality, please follow the instructions in Section 2.2, above, for “Building on* nix”. If building wolfMQTT for Windows on a Windows development machine, we recommend using the included Visual Studio project files to build wolfMQTT.

2.4 Building on Arduino

2.4.1 Reformatting wolfMQTT as a compatible Arduino Library

wolfmqtt-arduino.sh is a shell script that will re-organize the wolfMQTT library to be compatible with Arduino projects. The Arduino IDE requires a library’s source files to be in the library’s root directory with a header file in the name of the library. This script copies all source files to the IDE/ARDUINO/wolfMQTT directory and creates a stub header file called wolfMQTT.h.

To configure wolfMQTT with Arduino, enter the following from within the IDE/ARDUINO directory:

```
./wolfmqtt-arduino.sh
```

2.4.2 Including wolfMQTT in Arduino Libraries (for Arduino version 1.8.2)

1. In the Arduino IDE:
    - In Sketch -> Include Library -> Add .ZIP Library... and choose the IDE/ARDUNIO/wolfMQTT folder.
    - In Sketch -> Include Library choose wolfMQTT.

To enable TLS support, add #define ENABLE_MQTT_TLS in IDE/ARDUNIO/wolfMQTT/wolfmqtt/options.h. Note: If using wolfSSL TLS then you'll need to do this for wolfSSL as well. See <wolfssl-root>/IDE/ARDUINO/README.md for instructions.

An example wolfMQTT client INO sketch exists here: wolfmqtt_client/wolfmqtt_client.ino to demonstrate using the wolfMQTT library.

2.5 Building with MinGW

After downloading both wolfSSL and wolfMQTT, run the script below to build then install both:

```
export PATH="/opt/mingw-w32-bin_i686-darwin/bin:$PATH"
export PREFIX=$PWD/build

# wolfSSL
cd wolfssl
./configure --host=i686 CC=i686-w64-mingw32-gcc LD=i686-w64-mingw32-ld CFLAGS="-DWIN32 -DMINGW
 ↪  -D_WIN32_WINNT=0x0600" LIBS="-lws2_32 -L$PREFIX/lib -lwolfssl" --prefix=$PREFIX
make
make install

# wolfMQTT
cd ../wolfmqtt
./configure --host=i686 CC=i686-w64-mingw32-gcc LD=i686-w64-mingw32-ld CFLAGS="-DWIN32 -DMINGW
 ↪  -D_WIN32_WINNT=0x0600 -DBUILDING_WOLFMQTT -I$PREFIX/include" LDFLAGS="-lws2_32 -L$PREFIX/lib
 ↪  -lwolfssl" --prefix=$PREFIX --disable-examples
make
```

2.6 Building in a non-standard environment

While not officially supported, we try to help users wishing to build wolfMQTT in a non-standard environment, particularly with embedded and cross-compilation systems. Below are some notes on getting started with this.

1. The source and header files need to remain in the same directory structure as they are in the wolfMQTT download package.
2. Some build systems will want to explicitly know where the wolfMQTT header files are located, so you may need to specify that. They are located in the <wolfmqtt_root>/wolfmqtt directory. Typically, you can add the directory to your include path to resolve header problems.
3. wolfMQTT defaults to a little endian system unless the configure process detects big endian. Since users building in a non-standard environment aren't using the configure process, BIG_ENDIAN_ORDER will need to be defined if using a big endian system.
4. Try to build the library, and let us know if you run into any problems. If you need help, contact us at *support@wolfssl.com* .

2.7 Cross Compiling

Many users on embedded platforms cross compile for their environment. The easiest way to cross compile the library is to use the ./configure system. It will generate a Makefile which can then be used to build wolfMQTT. When cross compiling, you'll need to specify the host to ./configure, such as:

```
./configure --host=arm-linux
```

You may also need to specify the compiler, linker, etc. that you want to use:

```
./configure --host=arm-linux CC=arm-linux-gcc AR=arm-linux-ar RANLIB=arm-linux
```

After correctly configuring wolfMQTT for cross-compilation, you should be able to follow standard autoconf practices for building and installing the library:

```
make
sudo make install
```

If you have any additional tips or feedback about cross compiling wolfMQTT, please let us know at *info@wolfssl.com*.

2.7.1 Install to Custom Directory

To setup a custom install directory for wolfSSL and specify the custom wolfSSL lib/include directories for wolfMQTT, use the following:

In wolfSSL:

```
./configure  --prefix=~/wolfssl
make
make install
```

This will place the libs in `~/wolfssl/lib` and includes in `~/wolfssl/include`.

In wolfMQTT:

```
./configure --prefix=~/wolfmqtt  --libdir=~/wolfssl/lib  --includedir=~/wolfssl/include
make
make install
```

Make sure the paths above match your actual location.

3 Getting Started

Here are the steps for creating your own implementation:

1. Create network callback functions for Connect, Read, Write and Disconnect. See examples/mqttnet.c and examples/mqttnet.h for reference implementation.
2. Define the network callback functions and context in a MqttNet structure.
3. Call MqttClient_Init passing in a MqttClient structure pointer, MqttNet structure pointer, MqttMsgCb function callback pointer, TX/RX buffers with maximum length and command timeout.
4. Call MqttClient_NetConnect to connect to broker over network. If use_tls is non-zero value then it will perform a TLS connection. The TLS callback MqttTlsCb should be defined for WolfSSL certificate configuration.
5. Call MqttClient_Connect passing pointer to MqttConnect structure to send MQTT connect command and wait for Connect Ack.
6. Call MqttClient_Subscribe passing pointer to MqttSubscribe structure to send MQTT Subscribe command and wait for Subscribe Ack (depending on QoS level).
7. Call MqttClient_WaitMessage passing pointer to MqttMessage to wait for incoming MQTT Publish message.

3.1 Examples

3.1.1 Client Example

The example MQTT client is located in `/examples/mqttclient/`. This example exercises many of the exposed API's and prints any incoming publish messages for subscription topic "wolfMQTT/example/testTopic". This client contains examples of many MQTTv5 features, including the property callback and server assignment of client ID. The mqqtclient example is a good starting template for your MQTT application.

3.1.2 Simple Standalone Client Example

The example MQTT client is located in `/examples/mqttsimple/`. This example demonstrates a standalone client using standard BSD sockets. This requires `HAVE_SOCKET` to be defined, which comes from the ./configure generated `wolfmqtt/config.h` file. All parameters are build-time macros defined at the top of `/examples/mqttsimple/mqttsimple.c`.

3.1.3 Non-Blocking Client Example

The example MQTT client is located in `/examples/nbclient/`. This example uses non-blocking I/O for message exchange. The wolfMQTT library must be configured with the `--enable-nonblock` option (or built with `WOLFMQTT_NONBLOCK`).

3.1.4 Firmware Example

The MQTT firmware update is located in `/examples/firmware/`. This example has two parts. The first is called "fwpush", which signs and publishes a firmware image. The second is called "fwclient", which receives the firmware image and verifies the signature. This example publishes message on the topic "wolfMQTT/example/firmware". The "fwpush" application is an example of using a publish callback to send the payload data.

3.1.5 Azure IoT Hub Example

We setup a wolfMQTT IoT Hub on the Azure server for testing. We added a device called `demoDevice`, which you can connect and publish to. The example demonstrates creation of a SasToken, which is used as the password for the MQTT connect packet. It also shows the topic names for publishing events and listening to `device`bound messages. This example only works with `ENABLE_MQTT_TLS` set and the wolfSSL library present because it requires Base64 Encode/Decode and HMAC-SHA256. Note: The wolfSSL library must be built with `./configure --enable-base64encode` or `#define WOLFSSL_BASE64_ENCODE`. The `wc_GetTime` API was added in 3.9.1 and if not present you'll need to implement your own version of this to get current UTC seconds or update your wolfSSL library. **NOTE** The Azure broker only supports MQTT v3.1.1

3.1.6 AWS IoT Example

We setup an AWS IoT endpoint and testing device certificate for testing. The AWS server uses TLS client certificate for authentication. The example is located in `/examples/aws/`. The example subscribes to `$aws/things/"AWSIOT_DEVICE_ID"/shadow/update/delta` and publishes to `$aws/things/"AWSIOT_DEVICE_ID"/shadow/update`. **NOTE** The AWS broker only supports MQTT v3.1.1

3.1.7 Watson IoT Example

This example enables the wolfMQTT client to connect to the IBM Watson Internet of Things (WIOT) Platform. The WIOT Platform has a limited test broker called "Quickstart" that allows non-secure connections to exercise the component. The example is located in `/examples/wiot/`. Works with MQTT v5 support enabled.

3.1.8 MQTT-SN Example

The Sensor Network client implements the MQTT-SN protocol for low-bandwidth networks. There are several differences from MQTT, including the ability to use a two byte Topic ID instead the full topic during subscribe and publish. The SN client requires an MQTT-SN gateway. The gateway acts as an intermediary between the SN clients and the broker. This client was tested with the Eclipse Paho MQTT-SN Gateway, which connects by default to the public Eclipse broker, much like our wolfMQTT Client example. The address of the gateway must be configured as the host. The example is located in `/examples/sn-client/`.

A special feature of MQTT-SN is the ability to use QoS level -1 (negative one) to publish to a predefined topic without first connecting to the gateway. There is no feedback in the application if there was an error, so confirmation of the test would involve running the `sn-client` first and watching for the publish from the `sn-client_qos-1`. There is an example provided in `/examples/sn-client/sn-client_qos-1`. It requires some configuration changes of the gateway.

- Enable the the QoS-1 feature, predefined topics, and change the gateway name in `gateway.conf`:

```
QoS-1=YES
PredefinedTopic=YES
PredefinedTopicList=./predefinedTopic.conf
.
.
.
#GatewayName=PahoGateway-01
GatewayName=WolfGateway
```

- Comment out all entries and add a new topic in `predefinedTopic.conf`:

```
WolfGatewayQoS-1,wolfMQTT/example/testTopic, 1
```

3.1.9 Multithread Example

This example exercises the multithreading capabilities of the client library. The client implements two tasks: one that publishes to the broker; and another that waits for messages from the broker. The publish thread is created `NUM_PUB_TASKS` times (10 by default) and sends unique messages to the broker. This feature is enabled using the `--enable-mt` configuration option. The example is located in `/examples/multithread/`.

3.2 Example Options

The command line examples can be executed with optional parameters. To see a list of the available parameters, add the `-?`

```
 ./examples/mqttclient/mqttclient -?
mqttclient:
-?          Help, print this usage
-h <host>   Host to connect to, default: test.mosquitto.org
-p <num>    Port to connect on, default: Normal 1883, TLS 8883
-t          Enable TLS
-A <file>   Load CA (validate peer)
```

```
-K <key>    Use private key (for TLS mutual auth)
-c <cert>   Use certificate (for TLS mutual auth)
-S <str>    Use Host Name Indication, blank defaults to host
-q <num>    Qos Level 0-2, default: 0
-s          Disable clean session connect flag
-k <num>    Keep alive seconds, default: 60
-i <id>     Client Id, default: WolfMQTTClient
-l          Enable LWT (Last Will and Testament)
-u <str>    Username
-w <str>    Password
-m <str>    Message, default: test
-n <str>    Topic name, default: wolfMQTT/example/testTopic
-r          Set Retain flag on publish message
-C <num>    Command Timeout, default: 30000ms
-P <num>    Max packet size the client will accept, default: 1048576
-T          Test mode
-f <file>   Use file contents for publish
```

The available options vary depending on the library configuration.

4 Library Design

Library header files are located in the `/wolfmqtt` directory. Only the `/wolfmqtt/mqtt_client.h` header is required to be included:

```
#include <wolfmqtt/mqtt_client.h>
```

The library has three components:

4.0.1 mqtt_client

This is where the top level application interfaces for the MQTT client reside.

- `int MqttClient_Init(MqttClient *client, MqttNet *net, MqttMsgCb msg_cb, byte *tx_buf, int tx_buf_len, byte *rx_buf, int rx_buf_len, int cmd_timeout_ms);`

These API's are blocking on `MqttNet.read` until error/timeout (`cmd_timeout_ms`):

- `int MqttClient_Connect(MqttClient *client, MqttConnect *connect);`
- `int MqttClient_Publish(MqttClient *client, MqttPublish *publish);`
- `int MqttClient_Subscribe(MqttClient *client, MqttSubscribe *subscribe);`
- `int MqttClient_Unsubscribe(MqttClient *client, MqttUnsubscribe *unsubscribe);`
- `int MqttClient_Ping(MqttClient *client);`
- `int MqttClient_Disconnect(MqttClient *client);`

This function blocks waiting for a new publish message to arrive for a maximum duration of `timeout_ms`.

- `int MqttClient_WaitMessage(MqttClient *client, MqttMessage *message, int timeout_ms);`

These are the network connect / disconnect interfaces that wrap the MqttNet callbacks and handle WolfSSL TLS:

- `int MqttClient_NetConnect(MqttClient *client, const char* host, word16 port, int timeout_ms, int use_tls, MqttTlsCb cb);`
- `int MqttClient_NetDisconnect(MqttClient *client);`

Helper functions:

- `const char* MqttClient_ReturnCodeToString(int return_code);`

4.0.2 mqtt_packet

This is where all the packet encoding/decoding is handled.

The header contains the MQTT Packet structures for:

- Connect: `MqttConnect`
- Publish / Message: `MqttPublish` / `MqttMessage` (they are the same)
- Subscribe: `MqttSubscribe`
- Unsubscribe: `MqttUnsubscribe`

4.0.3 mqtt_socket

This is where the transport socket optionally wraps TLS and uses the `MqttNet` callbacks for the platform specific network handling.

The header contains the MQTT Network structure `MqttNet` for network callback and context.

4.0.4 Example Design

The examples use a common examples/mqttnet.c to handle the network callbacks on the clients. This reference supports Linux (BSD sockets), FreeRTOS/LWIP, MQX RTCS, Harmony and Windows.

5 API Reference

5.1 wolfmqtt/wolfmqtt/mqtt_client.h

5.1.1 Classes

	Name
struct	**_MqttPkRead**
struct	**_MqttSk**
struct	**_MqttClient**

5.1.2 Types

	Name
enum	**MqttClientFlags** { MQTT_CLIENT_FLAG_IS_CONNECTED = 0x01, MQTT_CLIENT_FLAG_IS_TLS = 0x02}
enum	**_MqttPkStat** { MQTT_PK_BEGIN, MQTT_PK_READ_HEAD, MQTT_PK_READ}
typedef int(*)(struct _MqttClient *client, MqttMessage* message, byte msg_new, byte msg_done)	**MqttMsgCb** Mqtt Message Callback. If the message payload is larger than the maximum RX buffer then this callback is called multiple times. If msg_new = 1 its a new message. The topic_name and topic_name length are only valid when msg_new = 1. If msg_new = 0 then we are receiving additional payload. Each callback populates the payload in MqttMessage.buffer is the length of the complete payload message. If msg_done = 1 the entire publish payload has been received.
typedef int()(MqttPublish* publish)	**MqttPublishCb** Mqtt Publish Callback. If the publish payload is larger than the maximum TX buffer then this callback is called multiple times. This callback is executed from within a call to MqttPublish. It is expected to provide a buffer and it's size and return >=0 for success. Each callback populates the payload in MqttPublish.buffer is the length of the complete payload message.
typedef enum _MqttPkStat**	
typedef struct _MqttPkRead**	
typedef struct _MqttSk**	
typedef int(*)(struct _MqttClient *client, int error_code, void* ctx)	**MqttDisconnectCb**
typedef int(*)(struct _MqttClient *client, MqttProp* head, void *ctx)	**MqttPropertyCb**
typedef int()(word16 topicId, const char* topicName, void *reg_ctx)	**SN_ClientRegisterCb** Mqtt-SN Register Callback. A GW sends a REGISTER message to a client if it wants to inform that client about the topic name and the assigned topic id that it will use later on when sending PUBLISH messages of the corresponding topic name. This callback allows the client to accept and save the new ID, or reject it if the ID is unknown. If the callback is not defined, then the regack will contain the "unsupported" return code.
typedef struct _MqttClient**	

5.1.3 Functions

	Name
WOLFMQTT_API int	**MqttClient_Init * rx_buf, int rx_buf_len, int cmd_timeout_ms)Initializes the MqttClient structure.
WOLFMQTT_API void	**MqttClient_DeInit * client)Cleans up resources allocated to the MqttClient structure.
WOLFMQTT_API int	**MqttClient_SetDisconnectCallback discb, void * ctx)Sets a disconnect callback with custom context.
WOLFMQTT_API int	**MqttClient_SetPropertyCallback propCb, void * ctx)Sets a property callback with custom context.
WOLFMQTT_API int	**MqttClient_Connect * connect)Encodes and sends the MQTT Connect packet and waits for the Connect Acknowledgment packet.
WOLFMQTT_API int	**MqttClient_Publish * publish)Encodes and sends the MQTT Publish packet and waits for the Publish response (if QoS > 0). If the total size of the payload is larger than the buffer size, it can be called successively to transmit the full payload. (if QoS > 0)
WOLFMQTT_API int	**MqttClient_Publish_ex pubCb)Encodes and sends the MQTT Publish packet and waits for the Publish response (if QoS > 0). The callback function is used to copy the payload data, allowing the use of transmit buffers smaller than the total size of the payload.
WOLFMQTT_API int	**MqttClient_Publish_WriteOnly pubCb)Same as MqttClient_Publish_ex, however this API will only perform writes and requires another thread to handle the read ACK processing using MqttClient_WaitMessage_ex.
WOLFMQTT_API int	**MqttClient_Subscribe * subscribe)Encodes and sends the MQTT Subscribe packet and waits for the Subscribe Acknowledgment packet.
WOLFMQTT_API int	**MqttClient_Unsubscribe * unsubscribe)Encodes and sends the MQTT Unsubscribe packet and waits for the Unsubscribe Acknowledgment packet.
WOLFMQTT_API int	**MqttClient_Ping * client)Encodes and sends the MQTT Ping Request packet and waits for the Ping Response packet.
WOLFMQTT_API int	**MqttClient_Ping_ex * ping)Encodes and sends the MQTT Ping Request packet and waits for the Ping Response packet. This version takes a MqttPing structure and can be used with non-blocking applications.
WOLFMQTT_API int	**MqttClient_Auth * auth)Encodes and sends the MQTT Authentication Request packet and waits for the Ping Response packet.
WOLFMQTT_API MqttProp ** head)Add a new property. Allocate a property structure and add it to the head of the list pointed to by head. To be used prior to calling packet command.	
WOLFMQTT_API int	**MqttClient_PropsFree * head)Free property list. Deallocate the list pointed to by head. Must be used after the packet command that used MqttClient_Prop_Add.
WOLFMQTT_API int	**MqttClient_Disconnect * client)Encodes and sends the MQTT Disconnect packet (no response)

	Name
WOLFMQTT_API int	**MqttClient_Disconnect_ex * disconnect)Encodes and sends the MQTT Disconnect packet (no response)
WOLFMQTT_API int	**MqttClient_WaitMessage * client, int timeout_ms)Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.
WOLFMQTT_API int	**MqttClient_WaitMessage_ex * msg, int timeout_ms)Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.
WOLFMQTT_API int	**MqttClient_CancelMessage * msg)In a multi-threaded and non-blocking mode this allows you to cancel an MQTT object that was previously submitted.
WOLFMQTT_API int	**MqttClient_NetConnect cb)Performs network connect with TLS (if use_tls is non-zero value) Will perform the MqttTlsCb callback if use_tls is non-zero value.
WOLFMQTT_API int	**MqttClient_NetDisconnect * client)Performs a network disconnect.
WOLFMQTT_API int	**MqttClient_GetProtocolVersion * client)Gets number version of connected protocol version.
const WOLFMQTT_API char *	**MqttClient_GetProtocolVersionString * client)Gets string version of connected protocol version.
const WOLFMQTT_API char *	**MqttClient_ReturnCodeToString**(int return_code)Performs lookup of the WOLFMQTT_API return values.
WOLFMQTT_API int	**SN_Client_SearchGW * search)Encodes and sends the a message to search for a gateway and waits for the gateway info response message.
WOLFMQTT_API int	**SN_Client_Connect * connect)Encodes and sends the Connect packet and waits for the Connect Acknowledgment packet. If Will is enabled, the gateway prompts for LWT Topic and Message. Sending an empty will topic indicates that the client wishes to delete the Will topic and the Will message stored in the server.
WOLFMQTT_API int	**SN_Client_WillTopicUpdate * will)Encodes and sends the MQTT-SN Will Topic Update packet. Sending a NULL 'will' indicates that the client wishes to delete the Will topic and the Will message stored in the server.
WOLFMQTT_API int	**SN_Client_WillMsgUpdate * will)Encodes and sends the MQTT-SN Will Message Update packet.
WOLFMQTT_API int	**SN_Client_Register * regist)Encodes and sends the MQTT-SN Register packet and waits for the Register Acknowledge packet. The Register packet is sent by a client to a GW for requesting a topic id value for the included topic name. It is also sent by a GW to inform a client about the topic id value it has assigned to the included topic name.
WOLFMQTT_API int	**SN_Client_SetRegisterCallback regCb, void * ctx)Sets a register callback with custom context.
WOLFMQTT_API int	**SN_Client_Publish * publish)Encodes and sends the MQTT-SN Publish packet and waits for the Publish response (if QoS > 0).
WOLFMQTT_API int	**SN_Client_Subscribe * subscribe)Encodes and sends the MQTT-SN Subscribe packet and waits for the Subscribe Acknowledgment packet containing the assigned topic ID.

	Name
WOLFMQTT_API int	**SN_Client_Unsubscribe * unsubscribe)Encodes and sends the MQTT-SN Unsubscribe packet and waits for the Unsubscribe Acknowledgment packet.
WOLFMQTT_API int	**SN_Client_Disconnect * client)Encodes and sends the MQTT-SN Disconnect packet. Client may send the disconnect with a duration to indicate the client is entering the “asleep” state.
WOLFMQTT_API int	**SN_Client_Disconnect_ex * disconnect)Encodes and sends the MQTT-SN Disconnect packet. Client may send the disconnect with a duration to indicate the client is entering the “asleep” state.
WOLFMQTT_API int	**SN_Client_Ping * ping)Encodes and sends the MQTT-SN Ping Request packet and waits for the Ping Response packet. If client is in the “asleep” state and wants to notify the gateway that it is entering the “awake” state, it should add it’s client ID to the ping request.
WOLFMQTT_API int	**SN_Client_WaitMessage * client, int timeout_ms)Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.
WOLFMQTT_API int	**SN_Client_WaitMessage_ex * packet_obj, int timeout_ms)

5.1.4 Attributes

Name
C

5.1.5 Types Documentation

Enumerator	Value	Description
MQTT_CLIENT_FLAG_IS_CONNECTED	0x01	
MQTT_CLIENT_FLAG_IS_TLS	0x02	

5.1.5.1 enum MqttClientFlags

Enumerator	Value	Description
MQTT_PK_BEGIN		
MQTT_PK_READ_HEAD		
MQTT_PK_READ		

5.1.5.2 enum _MqttPkStat

5.1.5.3 typedef MqttMsgCb

```
typedef int(* MqttMsgCb) (struct _MqttClient *client, MqttMessage *message, byte msg_new, byte
↪  msg_done);
```

Mqtt Message Callback. If the message payload is larger than the maximum RX buffer then this callback is called multiple times.

If msg_new = 1 its a new message. The topic_name and topic_name length are only valid when msg_new = 1. If msg_new = 0 then we are receiving additional payload. Each callback populates the payload in MqttMessage.buffer is the length of the complete payload message. If msg_done = 1 the entire publish payload has been received.

Parameters:

- **client** Pointer to MqttClient structure
- **message** Pointer to MqttMessage structure that has been initialized with the payload properties
- **msg_new** If non-zero value then message is new and topic name / len is provided and valid.
- **msg_done** If non-zero value then we have received the entire message and payload.

Return: MQTT_CODE_SUCCESS to remain connected (other values will cause net disconnect - see enum MqttPacketResponseCodes)

5.1.5.4 typedef MqttPublishCb

```
typedef int(* MqttPublishCb) (MqttPublish *publish);
```

Mqtt Publish Callback. If the publish payload is larger than the maximum TX buffer then this callback is called multiple times. This callback is executed from within a call to MqttPublish. It is expected to provide a buffer and it's size and return >=0 for success. Each callback populates the payload in MqttPublish.buffer is the length of the complete payload message.

Parameters:

- **publish** Pointer to MqttPublish structure

Return: >= 0 Indicates success

5.1.5.5 typedef MqttPkStat

```
typedef enum _MqttPkStat MqttPkStat;
```

5.1.5.6 typedef MqttPkRead

```
typedef struct _MqttPkRead MqttPkRead;
```

5.1.5.7 typedef MqttSk

```
typedef struct _MqttSk MqttSk;
```

5.1.5.8 typedef MqttDisconnectCb

```
typedef int(* MqttDisconnectCb) (struct _MqttClient *client, int error_code, void *ctx);
```

5.1.5.9 typedef MqttPropertyCb

```
typedef int(* MqttPropertyCb) (struct _MqttClient *client, MqttProp *head, void *ctx);
```

5.1.5.10 typedef SN_ClientRegisterCb

```
typedef int(* SN_ClientRegisterCb) (word16 topicId, const char *topicName, void *reg_ctx);
```

Mqtt-SN Register Callback. A GW sends a REGISTER message to a client if it wants to inform that client about the topic name and the assigned topic id that it will use later on when sending PUBLISH messages of the corresponding topic name. This callback allows the client to accept and save the new ID, or reject it if the ID is unknown. If the callback is not defined, then the regack will contain the "unsupported" return code.

Parameters:

- **topicId** New topic ID value
- **topicName** Pointer to topic name

- **reg_ctx** Pointer to user context

Return: >= 0 Indicates acceptance

5.1.5.11 typedef MqttClient

```
typedef struct _MqttClient MqttClient;
```

5.1.6 Functions Documentation

5.1.6.1 function MqttClient_Init

```
WOLFMQTT_API int MqttClient_Init(
    MqttClient * client,
    MqttNet * net,
    MqttMsgCb msg_cb,
    byte * tx_buf,
    int tx_buf_len,
    byte * rx_buf,
    int rx_buf_len,
    int cmd_timeout_ms
)
```

Initializes the MqttClient structure.

Parameters:

- **client** Pointer to MqttClient structure (uninitialized is okay)
- **net** Pointer to MqttNet structure that has been initialized with callback pointers and context
- **msg_cb** Pointer to message callback function
- **tx_buf** Pointer to transmit buffer used during encoding
- **tx_buf_len** Maximum length of the transmit buffer
- **rx_buf** Pointer to receive buffer used during decoding
- **rx_buf_len** Maximum length of the receive buffer
- **cmd_timeout_ms** Maximum command wait timeout in milliseconds

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_BAD_ARG (see enum MqttPacketResponseCodes)

5.1.6.2 function MqttClient_DeInit

```
WOLFMQTT_API void MqttClient_DeInit(
    MqttClient * client
)
```

Cleans up resources allocated to the MqttClient structure.

Parameters:

- **client** Pointer to MqttClient structure

Return: none

5.1.6.3 function MqttClient_SetDisconnectCallback

```
WOLFMQTT_API int MqttClient_SetDisconnectCallback(
    MqttClient * client,
    MqttDisconnectCb discb,
    void * ctx
)
```

Sets a disconnect callback with custom context.

Parameters:

- **client** Pointer to MqttClient structure (uninitialized is okay)
- **discb** Pointer to disconnect callback function
- **ctx** Pointer to your own context

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_BAD_ARG (see enum MqttPacketResponseCodes)

5.1.6.4 function MqttClient_SetPropertyCallback

```
WOLFMQTT_API int MqttClient_SetPropertyCallback(
    MqttClient * client,
    MqttPropertyCb propCb,
    void * ctx
)
```

Sets a property callback with custom context.

Parameters:

- **client** Pointer to MqttClient structure (uninitialized is okay)
- **propCb** Pointer to property callback function
- **ctx** Pointer to your own context

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_BAD_ARG (see enum MqttPacketResponseCodes)

5.1.6.5 function MqttClient_Connect

```
WOLFMQTT_API int MqttClient_Connect(
    MqttClient * client,
    MqttConnect * connect
)
```

Encodes and sends the MQTT Connect packet and waits for the Connect Acknowledgment packet.

Parameters:

- **client** Pointer to MqttClient structure
- **connect** Pointer to MqttConnect structure initialized with connect parameters

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.6 function MqttClient_Publish

```
WOLFMQTT_API int MqttClient_Publish(
    MqttClient * client,
    MqttPublish * publish
)
```

Encodes and sends the MQTT Publish packet and waits for the Publish response (if QoS > 0). If the total size of the payload is larger than the buffer size, it can be called successively to transmit the full payload. (if QoS > 0)

Parameters:

- **client** Pointer to MqttClient structure
- **publish** Pointer to MqttPublish structure initialized with message data Note: MqttPublish and MqttMessage are same structure.

See:

- MqttClient_Publish_WriteOnly
- MqttClient_Publish_ex

Return: MQTT_CODE_SUCCESS, MQTT_CODE_CONTINUE (for non-blocking) or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This function that will wait for MqttNet.read to complete, timeout or MQTT_CODE_CONTINUE if non-blocking. If QoS level = 1 then will wait for PUBLISH_ACK. If QoS level = 2 then will wait for PUBLISH_REC then send PUBLISH_REL and wait for PUBLISH_COMP.

5.1.6.7 function MqttClient_Publish_ex

```
WOLFMQTT_API int MqttClient_Publish_ex(
    MqttClient * client,
    MqttPublish * publish,
    MqttPublishCb pubCb
)
```

Encodes and sends the MQTT Publish packet and waits for the Publish response (if QoS > 0). The callback function is used to copy the payload data, allowing the use of transmit buffers smaller than the total size of the payload.

Parameters:

- **client** Pointer to MqttClient structure
- **publish** Pointer to MqttPublish structure initialized with message data Note: MqttPublish and MqttMessage are same structure.
- **pubCb** Function pointer to callback routine

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This function that will wait for MqttNet.read to complete, timeout or MQTT_CODE_CONTINUE if non-blocking. If QoS level = 1 then will wait for PUBLISH_ACK. If QoS level = 2 then will wait for PUBLISH_REC then send PUBLISH_REL and wait for PUBLISH_COMP.

5.1.6.8 function MqttClient_Publish_WriteOnly

```
WOLFMQTT_API int MqttClient_Publish_WriteOnly(
    MqttClient * client,
    MqttPublish * publish,
    MqttPublishCb pubCb
)
```

Same as MqttClient_Publish_ex, however this API will only perform writes and requires another thread to handle the read ACK processing using MqttClient_WaitMessage_ex.

Parameters:

- **client** Pointer to MqttClient structure
- **publish** Pointer to MqttPublish structure initialized with message data Note: MqttPublish and MqttMessage are same structure.

See:

- MqttClient_Publish
- MqttClient_Publish_ex
- MqttClient_WaitMessage_ex

Return: MQTT_CODE_SUCCESS, MQTT_CODE_CONTINUE (for non-blocking) or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This function that will wait for MqttNet.read to complete, timeout or MQTT_CODE_CONTINUE if non-blocking. If QoS level = 1 then will wait for PUBLISH_ACK. If QoS level = 2 then will wait for PUBLISH_REC then send PUBLISH_REL and wait for PUBLISH_COMP.

5.1.6.9 function MqttClient_Subscribe

```
WOLFMQTT_API int MqttClient_Subscribe(
    MqttClient * client,
    MqttSubscribe * subscribe
)
```

Encodes and sends the MQTT Subscribe packet and waits for the Subscribe Acknowledgment packet.

Parameters:

- **client** Pointer to MqttClient structure
- **subscribe** Pointer to MqttSubscribe structure initialized with subscription topic list and desired QoS.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.10 function MqttClient_Unsubscribe

```
WOLFMQTT_API int MqttClient_Unsubscribe(
    MqttClient * client,
    MqttUnsubscribe * unsubscribe
)
```

Encodes and sends the MQTT Unsubscribe packet and waits for the Unsubscribe Acknowledgment packet.

Parameters:

- **client** Pointer to MqttClient structure
- **unsubscribe** Pointer to MqttUnsubscribe structure initialized with topic list.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.11 function MqttClient_Ping

```
WOLFMQTT_API int MqttClient_Ping(
    MqttClient * client
)
```

Encodes and sends the MQTT Ping Request packet and waits for the Ping Response packet.

Parameters:

- **client** Pointer to MqttClient structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.12 function MqttClient_Ping_ex

```
WOLFMQTT_API int MqttClient_Ping_ex(
    MqttClient * client,
    MqttPing * ping
)
```

Encodes and sends the MQTT Ping Request packet and waits for the Ping Response packet. This version takes a MqttPing structure and can be used with non-blocking applications.

Parameters:

- **client** Pointer to MqttClient structure
- **ping** Pointer to MqttPing structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.13 function MqttClient_Auth

```
WOLFMQTT_API int MqttClient_Auth(
    MqttClient * client,
    MqttAuth * auth
)
```

Encodes and sends the MQTT Authentication Request packet and waits for the Ping Response packet.

Parameters:

- **client** Pointer to MqttClient structure
- **auth** Pointer to MqttAuth structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.14 function MqttClient_PropsAdd

```
WOLFMQTT_API MqttProp * MqttClient_PropsAdd(
    MqttProp ** head
)
```

Add a new property. Allocate a property structure and add it to the head of the list pointed to by head. To be used prior to calling packet command.

Parameters:

- **head** Pointer-pointer to a property structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_BAD_ARG

5.1.6.15 function MqttClient_PropsFree

```
WOLFMQTT_API int MqttClient_PropsFree(
    MqttProp * head
)
```

Free property list. Deallocate the list pointed to by head. Must be used after the packet command that used MqttClient_Prop_Add.

Parameters:

- **head** Pointer-pointer to a property structure

Return: MQTT_CODE_SUCCESS or -1 on error (and sets errno)

5.1.6.16 function MqttClient_Disconnect

```
WOLFMQTT_API int MqttClient_Disconnect(
    MqttClient * client
)
```

Encodes and sends the MQTT Disconnect packet (no response)

Parameters:

- **client** Pointer to MqttClient structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a non-blocking function that will try and send using MqttNet.write

5.1.6.17 function MqttClient_Disconnect_ex

```
WOLFMQTT_API int MqttClient_Disconnect_ex(
    MqttClient * client,
    MqttDisconnect * disconnect
)
```

Encodes and sends the MQTT Disconnect packet (no response)

Parameters:

- **client** Pointer to MqttClient structure
- **disconnect** Pointer to MqttDisconnect structure. NULL is valid.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a non-blocking function that will try and send using MqttNet.write

5.1.6.18 function MqttClient_WaitMessage

```
WOLFMQTT_API int MqttClient_WaitMessage(
    MqttClient * client,
    int timeout_ms
)
```

Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.

Parameters:

- **client** Pointer to MqttClient structure
- **timeout_ms** Milliseconds until read timeout

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.19 function MqttClient_WaitMessage_ex

```
WOLFMQTT_API int MqttClient_WaitMessage_ex(
    MqttClient * client,
    MqttObject * msg,
    int timeout_ms
)
```

Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.

Parameters:

- **client** Pointer to MqttClient structure
- **msg** Pointer to MqttObject structure
- **timeout_ms** Milliseconds until read timeout

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.20 function MqttClient_CancelMessage

```
WOLFMQTT_API int MqttClient_CancelMessage(
    MqttClient * client,
    MqttObject * msg
)
```

In a multi-threaded and non-blocking mode this allows you to cancel an MQTT object that was previously submitted.

Parameters:

- **client** Pointer to MqttClient structure
- **msg** Pointer to MqttObject structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.21 function MqttClient_NetConnect

```
WOLFMQTT_API int MqttClient_NetConnect(
    MqttClient * client,
    const char * host,
    word16 port,
    int timeout_ms,
    int use_tls,
    MqttTlsCb cb
)
```

Performs network connect with TLS (if use_tls is non-zero value) Will perform the MqttTlsCb callback if use_tls is non-zero value.

Parameters:

- **client** Pointer to MqttClient structure
- **host** Address of the broker server
- **port** Optional custom port. If zero will use defaults
- **use_tls** If non-zero value will connect with and use TLS for encryption of data
- **cb** A function callback for configuration of the SSL context certificate checking
- **timeout_ms** Milliseconds until read timeout

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

5.1.6.22 function MqttClient_NetDisconnect

```
WOLFMQTT_API int MqttClient_NetDisconnect(
    MqttClient * client
)
```

Performs a network disconnect.

Parameters:

- **client** Pointer to MqttClient structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

5.1.6.23 function MqttClient_GetProtocolVersion

```
WOLFMQTT_API int MqttClient_GetProtocolVersion(
    MqttClient * client
)
```

Gets number version of connected protocol version.

Parameters:

- **client** Pointer to MqttClient structure

Return: 4 (v3.1.1) or 5 (v5)

5.1.6.24 function MqttClient_GetProtocolVersionString

```
const WOLFMQTT_API char * MqttClient_GetProtocolVersionString(
    MqttClient * client
)
```

Gets string version of connected protocol version.

Parameters:

- **client** Pointer to MqttClient structure

Return: String v3.1.1 or v5

5.1.6.25 function MqttClient_ReturnCodeToString

```
const WOLFMQTT_API char * MqttClient_ReturnCodeToString(
    int return_code
)
```

Performs lookup of the WOLFMQTT_API return values.

Parameters:

- **return_code** The return value from a WOLFMQTT_API function

Return: String representation of the return code

5.1.6.26 function SN_Client_SearchGW

```
WOLFMQTT_API int SN_Client_SearchGW(
    MqttClient * client,
    SN_SearchGw * search
)
```

Encodes and sends the a message to search for a gateway and waits for the gateway info response message.

Parameters:

- **client** Pointer to MqttClient structure
- **search** Pointer to SN_SearchGW structure initialized with hop radius.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.27 function SN_Client_Connect

```
WOLFMQTT_API int SN_Client_Connect(
    MqttClient * client,
    SN_Connect * connect
)
```

Encodes and sends the Connect packet and waits for the Connect Acknowledgment packet. If Will is enabled, the gateway prompts for LWT Topic and Message. Sending an empty will topic indicates that the client wishes to delete the Will topic and the Will message stored in the server.

Parameters:

- **client** Pointer to MqttClient structure
- **connect** Pointer to SN_Connect structure initialized with connect parameters

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.28 function SN_Client_WillTopicUpdate

```
WOLFMQTT_API int SN_Client_WillTopicUpdate(
    MqttClient * client,
    SN_Will * will
)
```

Encodes and sends the MQTT-SN Will Topic Update packet. Sending a NULL 'will' indicates that the client wishes to delete the Will topic and the Will message stored in the server.

Parameters:

- **client** Pointer to MqttClient structure
- **will** Pointer to SN_Will structure initialized with topic and message parameters. NULL is valid.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.29 function SN_Client_WillMsgUpdate

```
WOLFMQTT_API int SN_Client_WillMsgUpdate(
    MqttClient * client,
    SN_Will * will
)
```

Encodes and sends the MQTT-SN Will Message Update packet.

Parameters:

- **client** Pointer to MqttClient structure
- **will** Pointer to SN_Will structure initialized with topic and message parameters. NULL is valid.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.30 function SN_Client_Register

```
WOLFMQTT_API int SN_Client_Register(
    MqttClient * client,
    SN_Register * regist
)
```

Encodes and sends the MQTT-SN Register packet and waits for the Register Acknowledge packet. The Register packet is sent by a client to a GW for requesting a topic id value for the included topic name. It is also sent by a GW to inform a client about the topic id value it has assigned to the included topic name.

Parameters:

- **client** Pointer to MqttClient structure
- **regist** Pointer to SN_Register structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.31 function SN_Client_SetRegisterCallback

```
WOLFMQTT_API int SN_Client_SetRegisterCallback(
    MqttClient * client,
    SN_ClientRegisterCb regCb,
    void * ctx
)
```

Sets a register callback with custom context.

Parameters:

- **client** Pointer to MqttClient structure (uninitialized is okay)
- **regCb** Pointer to register callback function
- **ctx** Pointer to your own context

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_BAD_ARG

5.1.6.32 function SN_Client_Publish

```
WOLFMQTT_API int SN_Client_Publish(
    MqttClient * client,
    SN_Publish * publish
)
```

Encodes and sends the MQTT-SN Publish packet and waits for the Publish response (if QoS > 0).

Parameters:

- **client** Pointer to MqttClient structure
- **publish** Pointer to SN_Publish structure initialized with message data Note: SN_Publish and MqttMessage are same structure.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read If QoS level = 1 then will wait for PUBLISH_ACK. If QoS level = 2 then will wait for PUBLISH_REC then send PUBLISH_REL and wait for PUBLISH_COMP.

5.1.6.33 function SN_Client_Subscribe

```
WOLFMQTT_API int SN_Client_Subscribe(
    MqttClient * client,
    SN_Subscribe * subscribe
)
```

Encodes and sends the MQTT-SN Subscribe packet and waits for the Subscribe Acknowledgment packet containing the assigned topic ID.

Parameters:

- **client** Pointer to MqttClient structure
- **subscribe** Pointer to SN_Subscribe structure initialized with subscription topic list and desired QoS.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.34 function SN_Client_Unsubscribe

```
WOLFMQTT_API int SN_Client_Unsubscribe(
    MqttClient * client,
    SN_Unsubscribe * unsubscribe
)
```

Encodes and sends the MQTT-SN Unsubscribe packet and waits for the Unsubscribe Acknowledgment packet.

Parameters:

- **client** Pointer to MqttClient structure
- **unsubscribe** Pointer to SN_Unsubscribe structure initialized with topic ID.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.35 function SN_Client_Disconnect

```
WOLFMQTT_API int SN_Client_Disconnect(
    MqttClient * client
)
```

Encodes and sends the MQTT-SN Disconnect packet. Client may send the disconnect with a duration to indicate the client is entering the “asleep” state.

Parameters:

- **client** Pointer to MqttClient structure

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a non-blocking function that will try and send using MqttNet.write

5.1.6.36 function SN_Client_Disconnect_ex

```
WOLFMQTT_API int SN_Client_Disconnect_ex(
    MqttClient * client,
    SN_Disconnect * disconnect
)
```

Encodes and sends the MQTT-SN Disconnect packet. Client may send the disconnect with a duration to indicate the client is entering the “asleep” state.

Parameters:

- **client** Pointer to MqttClient structure
- **disconnect** Pointer to SN_Disconnect structure. NULL is valid.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a non-blocking function that will try and send using MqttNet.write

5.1.6.37 function SN_Client_Ping

```
WOLFMQTT_API int SN_Client_Ping(
    MqttClient * client,
    SN_PingReq * ping
)
```

Encodes and sends the MQTT-SN Ping Request packet and waits for the Ping Response packet. If client is in the “asleep” state and wants to notify the gateway that it is entering the “awake” state, it should add it’s client ID to the ping request.

Parameters:

- **client** Pointer to MqttClient structure
- **ping** Pointer to SN_PingReq structure. NULL is valid.

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.38 function SN_Client_WaitMessage

```
WOLFMQTT_API int SN_Client_WaitMessage(
    MqttClient * client,
    int timeout_ms
)
```

Waits for packets to arrive. Incoming publish messages will arrive via callback provided in MqttClient_Init.

Parameters:

- **client** Pointer to MqttClient structure
- **timeout_ms** Milliseconds until read timeout

Return: MQTT_CODE_SUCCESS or MQTT_CODE_ERROR_* (see enum MqttPacketResponseCodes)

Note: This is a blocking function that will wait for MqttNet.read

5.1.6.39 function SN_Client_WaitMessage_ex

```
WOLFMQTT_API int SN_Client_WaitMessage_ex(
    MqttClient * client,
    SN_Object * packet_obj,
    int timeout_ms
)
```

5.1.7 Attributes Documentation

5.1.7.1 variable C

```
C {
#endif


#if !defined(WOLFMQTT_USER_SETTINGS) && \
    !defined(_WIN32) && !defined(USE_WINDOWS_API)

    #include <wolfmqtt/options.h>
#endif
#include "wolfmqtt/mqtt_types.h"
#include "wolfmqtt/mqtt_packet.h"
#include "wolfmqtt/mqtt_socket.h"

#if defined(WOLFMQTT_PROPERTY_CB) && !defined(WOLFMQTT_V5)
    #error "WOLFMQTT_V5 must be defined to use WOLFMQTT_PROPERTY_CB"
#endif

struct _MqttClient;
```

5.1.8 Source code

```
/* mqtt_client.h
 *
 * Copyright (C) 2006-2022 wolfSSL Inc.
 *
 * This file is part of wolfMQTT.
 *
 * wolfMQTT is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * wolfMQTT is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

/* Implementation by: David Garske
 * Based on specification for MQTT v3.1.1
 * See http://mqtt.org/documentation for additional MQTT documentation.
 */

#ifndef WOLFMQTT_CLIENT_H
#define WOLFMQTT_CLIENT_H

#ifdef __cplusplus
    extern "C" {
#endif

/* Windows uses the vs_settings.h file included vis mqtt_types.h */
#if !defined(WOLFMQTT_USER_SETTINGS) && \
    !defined(_WIN32) && !defined(USE_WINDOWS_API)
    /* If options.h is missing use the "./configure" script. Otherwise, copy
     * the template "wolfmqtt/options.h.in" into "wolfmqtt/options.h" */
    #include <wolfmqtt/options.h>
#endif
#include "wolfmqtt/mqtt_types.h"
#include "wolfmqtt/mqtt_packet.h"
#include "wolfmqtt/mqtt_socket.h"

#if defined(WOLFMQTT_PROPERTY_CB) && !defined(WOLFMQTT_V5)
    #error "WOLFMQTT_V5 must be defined to use WOLFMQTT_PROPERTY_CB"
#endif

struct _MqttClient;

typedef int (*MqttMsgCb)(struct _MqttClient *client, MqttMessage *message,
    byte msg_new, byte msg_done);
```

```
typedef int (*MqttPublishCb)(MqttPublish* publish);

/* Client flags */
enum MqttClientFlags {
    MQTT_CLIENT_FLAG_IS_CONNECTED = 0x01,
    MQTT_CLIENT_FLAG_IS_TLS = 0x02,
};

typedef enum _MqttPkStat {
    MQTT_PK_BEGIN,
    MQTT_PK_READ_HEAD,
    MQTT_PK_READ,
} MqttPkStat;

typedef struct _MqttPkRead {
    MqttPkStat stat;
    int header_len;
    int remain_len;
    int buf_len;
} MqttPkRead;

typedef struct _MqttSk {
    int pos;
    int len;
} MqttSk;

#ifdef WOLFMQTT_DISCONNECT_CB
    typedef int (*MqttDisconnectCb)(struct _MqttClient* client, int error_code, void* ctx);
#endif
#ifdef WOLFMQTT_PROPERTY_CB
    typedef int (*MqttPropertyCb)(struct _MqttClient* client, MqttProp* head, void* ctx);
#endif
#ifdef WOLFMQTT_SN

    typedef int (*SN_ClientRegisterCb)(word16 topicId, const char* topicName, void *reg_ctx);
#endif

/* Client structure */
typedef struct _MqttClient {
    word32       flags; /* MqttClientFlags */
    int          cmd_timeout_ms;

    byte        *tx_buf;
    int          tx_buf_len;
    byte        *rx_buf;
    int          rx_buf_len;

    MqttNet     *net;   /* Pointer to network callbacks and context */
#ifdef ENABLE_MQTT_TLS
    MqttTls      tls;   /* WolfSSL context for TLS */
#endif

    MqttPkRead   packet; /* publish packet state - protected by read lock */
    MqttPublishResp packetAck; /* publish ACK - protected by write lock */
```

```
    MqttSk       read;   /* read socket state - protected by read lock */
    MqttSk       write;  /* write socket state - protected by write lock */

    MqttMsgCb    msg_cb;
    MqttObject   msg;   /* generic incoming message used by MqttClient_WaitType */
#ifdef WOLFMQTT_SN
    SN_Object    msgSN;
    SN_ClientRegisterCb reg_cb;
    void               *reg_ctx;
#endif
    void*        ctx;   /* user supplied context for publish callbacks */

#ifdef WOLFMQTT_V5
    word32 packet_sz_max; /* Server property */
    byte   max_qos;       /* Server property */
    byte   retain_avail;  /* Server property */
    byte   enable_eauth;  /* Enhanced authentication */
    byte   protocol_level;
#endif

#ifdef WOLFMQTT_DISCONNECT_CB
    MqttDisconnectCb disconnect_cb;
    void            *disconnect_ctx;
#endif
#ifdef WOLFMQTT_PROPERTY_CB
    MqttPropertyCb property_cb;
    void          *property_ctx;
#endif
#ifdef WOLFMQTT_MULTITHREAD
    wm_Sem lockSend;
    wm_Sem lockRecv;
    wm_Sem lockClient;
    struct _MqttPendResp* firstPendResp; /* protected with client lock */
    struct _MqttPendResp* lastPendResp;  /* protected with client lock */
#endif
#if defined(WOLFMQTT_NONBLOCK) && defined(WOLFMQTT_DEBUG_CLIENT)
    int lastRc;
#endif
} MqttClient;


/* Application Interfaces */

WOLFMQTT_API int MqttClient_Init(
    MqttClient *client,
    MqttNet *net,
    MqttMsgCb msg_cb,
    byte *tx_buf, int tx_buf_len,
    byte *rx_buf, int rx_buf_len,
    int cmd_timeout_ms);

WOLFMQTT_API void MqttClient_DeInit(MqttClient *client);

#ifdef WOLFMQTT_DISCONNECT_CB
```

```
WOLFMQTT_API int MqttClient_SetDisconnectCallback(
    MqttClient *client,
    MqttDisconnectCb discb,
    void* ctx);
#endif

#ifdef WOLFMQTT_PROPERTY_CB

WOLFMQTT_API int MqttClient_SetPropertyCallback(
    MqttClient *client,
    MqttPropertyCb propCb,
    void* ctx);
#endif

WOLFMQTT_API int MqttClient_Connect(
    MqttClient *client,
    MqttConnect *connect);

WOLFMQTT_API int MqttClient_Publish(
    MqttClient *client,
    MqttPublish *publish);

WOLFMQTT_API int MqttClient_Publish_ex(
    MqttClient *client,
    MqttPublish *publish,
    MqttPublishCb pubCb);


#ifdef WOLFMQTT_MULTITHREAD

WOLFMQTT_API int MqttClient_Publish_WriteOnly(
    MqttClient *client,
    MqttPublish *publish,
    MqttPublishCb pubCb);
#endif

WOLFMQTT_API int MqttClient_Subscribe(
    MqttClient *client,
    MqttSubscribe *subscribe);

WOLFMQTT_API int MqttClient_Unsubscribe(
    MqttClient *client,
    MqttUnsubscribe *unsubscribe);

WOLFMQTT_API int MqttClient_Ping(
    MqttClient *client);

WOLFMQTT_API int MqttClient_Ping_ex(MqttClient *client, MqttPing* ping);

#ifdef WOLFMQTT_V5

WOLFMQTT_API int MqttClient_Auth(
    MqttClient *client,
```

```
    MqttAuth *auth);


WOLFMQTT_API MqttProp* MqttClient_PropsAdd(
    MqttProp **head);

WOLFMQTT_API int MqttClient_PropsFree(
    MqttProp *head);
#endif


WOLFMQTT_API int MqttClient_Disconnect(
    MqttClient *client);


WOLFMQTT_API int MqttClient_Disconnect_ex(
    MqttClient *client,
    MqttDisconnect *disconnect);


WOLFMQTT_API int MqttClient_WaitMessage(
    MqttClient *client,
    int timeout_ms);

WOLFMQTT_API int MqttClient_WaitMessage_ex(
    MqttClient *client,
    MqttObject* msg,
    int timeout_ms);

WOLFMQTT_API int MqttClient_CancelMessage(
    MqttClient *client,
    MqttObject* msg);

WOLFMQTT_API int MqttClient_NetConnect(
    MqttClient *client,
    const char *host,
    word16 port,
    int timeout_ms,
    int use_tls,
    MqttTlsCb cb);

WOLFMQTT_API int MqttClient_NetDisconnect(
    MqttClient *client);

WOLFMQTT_API int MqttClient_GetProtocolVersion(MqttClient *client);

WOLFMQTT_API const char* MqttClient_GetProtocolVersionString(MqttClient *client);

#ifndef WOLFMQTT_NO_ERROR_STRINGS

WOLFMQTT_API const char* MqttClient_ReturnCodeToString(
    int return_code);
#else
    #define MqttClient_ReturnCodeToString(x) \
```

```
                                        "not compiled in"
#endif /* WOLFMQTT_NO_ERROR_STRINGS */

#ifdef WOLFMQTT_SN

WOLFMQTT_API int SN_Client_SearchGW(
        MqttClient *client,
        SN_SearchGw *search);

WOLFMQTT_API int SN_Client_Connect(
    MqttClient *client,
    SN_Connect *connect);

WOLFMQTT_API int SN_Client_WillTopicUpdate(MqttClient *client, SN_Will *will);

WOLFMQTT_API int SN_Client_WillMsgUpdate(MqttClient *client, SN_Will *will);

WOLFMQTT_API int SN_Client_Register(
    MqttClient *client,
    SN_Register *regist);


WOLFMQTT_API int SN_Client_SetRegisterCallback(
    MqttClient *client,
    SN_ClientRegisterCb regCb,
    void* ctx);


WOLFMQTT_API int SN_Client_Publish(
    MqttClient *client,
    SN_Publish *publish);

WOLFMQTT_API int SN_Client_Subscribe(
    MqttClient *client,
    SN_Subscribe *subscribe);

WOLFMQTT_API int SN_Client_Unsubscribe(
    MqttClient *client,
    SN_Unsubscribe *unsubscribe);

WOLFMQTT_API int SN_Client_Disconnect(
    MqttClient *client);

WOLFMQTT_API int SN_Client_Disconnect_ex(
    MqttClient *client,
    SN_Disconnect *disconnect);


WOLFMQTT_API int SN_Client_Ping(
    MqttClient *client,
    SN_PingReq *ping);

WOLFMQTT_API int SN_Client_WaitMessage(
    MqttClient *client,
```

```
    int timeout_ms);

WOLFMQTT_API int SN_Client_WaitMessage_ex(MqttClient *client, SN_Object* packet_obj,
    int timeout_ms);

#endif /* WOLFMQTT_SN */

#ifdef __cplusplus
    } /* extern "C" */
#endif

#endif /* WOLFMQTT_CLIENT_H */
```

5.2 wolfmqtt/wolfmqtt/mqtt_socket.h

5.2.1 Classes

	Name
struct	**_MqttTls**
struct	**_MqttNet**

5.2.2 Types

	Name
typedef int(*)(struct _MqttClient *client)	**MqttTlsCb**
typedef int()(*void* context, const char *host, word16 port, int timeout_ms)	**MqttNetConnectCb**
typedef int()(*void* context, const byte *buf, int buf_len, int timeout_ms)	**MqttNetWriteCb**
typedef int()(*void* context, byte *buf, int buf_len, int timeout_ms)	**MqttNetReadCb**
typedef int()(*void* context, byte *buf, int buf_len, int timeout_ms)	**MqttNetPeekCb**
typedef int()(*void* context)	**MqttNetDisconnectCb**
typedef struct _MqttTls**	
typedef struct _MqttNet**	

5.2.3 Functions

	Name
WOLFMQTT_LOCAL int	**MqttSocket_Init(MqttNet * net)
WOLFMQTT_LOCAL int	**MqttSocket_Write(MqttNet * net, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Read(MqttNet * net, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Peek(MqttNet * net, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Connect(MqttNet * net, MqttTlsCb cb)
WOLFMQTT_LOCAL int	**MqttSocket_Disconnect**(struct _MqttClient * client)
WOLFMQTT_API int	**MqttSocket_TlsSocketReceive**(WOLFSSL * ssl, char * buf, int sz, void * ptr)
WOLFMQTT_API int	**MqttSocket_TlsSocketSend**(WOLFSSL * ssl, char * buf, int sz, void * ptr)

5.2.4 Attributes

Name
C

5.2.5 Types Documentation

5.2.5.1 typedef MqttTlsCb

```
typedef int(* MqttTlsCb) (struct _MqttClient *client);
```

5.2.5.2 typedef MqttNetConnectCb

```
typedef int(* MqttNetConnectCb) (void *context, const char *host, word16 port, int timeout_ms);
```

5.2.5.3 typedef MqttNetWriteCb

```
typedef int(* MqttNetWriteCb) (void *context, const byte *buf, int buf_len, int timeout_ms);
```

5.2.5.4 typedef MqttNetReadCb

```
typedef int(* MqttNetReadCb) (void *context, byte *buf, int buf_len, int timeout_ms);
```

5.2.5.5 typedef MqttNetPeekCb

```
typedef int(* MqttNetPeekCb) (void *context, byte *buf, int buf_len, int timeout_ms);
```

5.2.5.6 typedef MqttNetDisconnectCb

```
typedef int(* MqttNetDisconnectCb) (void *context);
```

5.2.5.7 typedef MqttTls

```
typedef struct _MqttTls MqttTls;
```

5.2.5.8 typedef MqttNet

```
typedef struct _MqttNet MqttNet;
```

5.2.6 Functions Documentation

5.2.6.1 function MqttSocket_Init

```
WOLFMQTT_LOCAL int MqttSocket_Init(
    struct _MqttClient * client,
    MqttNet * net
)
```

5.2.6.2 function MqttSocket_Write

```
WOLFMQTT_LOCAL int MqttSocket_Write(
    struct _MqttClient * client,
    const byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.2.6.3 function MqttSocket_Read

```
WOLFMQTT_LOCAL int MqttSocket_Read(
    struct _MqttClient * client,
    byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.2.6.4 function MqttSocket_Peek

```
WOLFMQTT_LOCAL int MqttSocket_Peek(
    struct _MqttClient * client,
    byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.2.6.5 function MqttSocket_Connect

```
WOLFMQTT_LOCAL int MqttSocket_Connect(
    struct _MqttClient * client,
    const char * host,
    word16 port,
    int timeout_ms,
    int use_tls,
    MqttTlsCb cb
)
```

5.2.6.6 function MqttSocket_Disconnect

```
WOLFMQTT_LOCAL int MqttSocket_Disconnect(
    struct _MqttClient * client
)
```

5.2.6.7 function MqttSocket_TlsSocketReceive

```
WOLFMQTT_API int MqttSocket_TlsSocketReceive(
    WOLFSSL * ssl,
    char * buf,
    int sz,
    void * ptr
)
```

5.2.6.8 function MqttSocket_TlsSocketSend

```
WOLFMQTT_API int MqttSocket_TlsSocketSend(
    WOLFSSL * ssl,
    char * buf,
    int sz,
    void * ptr
)
```

5.2.7 Attributes Documentation

5.2.7.1 variable C

```
C {
#endif

#include "wolfmqtt/mqtt_types.h"
#ifdef ENABLE_MQTT_TLS
    #ifndef WOLF_TLS_DHKEY_BITS_MIN
        #ifdef WOLFSSL_MAX_STRENGTH
            #define WOLF_TLS_DHKEY_BITS_MIN 2048
        #else
            #define WOLF_TLS_DHKEY_BITS_MIN 1024
        #endif
    #endif
#endif


#define MQTT_DEFAULT_PORT   1883
#define MQTT_SECURE_PORT    8883


struct _MqttClient;
```

5.2.8 Source code

```
/* mqtt_socket.h
 *
 * Copyright (C) 2006-2022 wolfSSL Inc.
 *
 * This file is part of wolfMQTT.
 *
 * wolfMQTT is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * wolfMQTT is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

/* Implementation by: David Garske
 * Based on specification for MQTT v3.1.1
 * See http://mqtt.org/documentation for additional MQTT documentation.
 */

#ifndef WOLFMQTT_SOCKET_H
#define WOLFMQTT_SOCKET_H

#ifdef __cplusplus
    extern "C" {
```

```
#endif

#include "wolfmqtt/mqtt_types.h"
#ifdef ENABLE_MQTT_TLS
    #ifndef WOLF_TLS_DHKEY_BITS_MIN /* allow define to be overridden */
        #ifdef WOLFSSL_MAX_STRENGTH
            #define WOLF_TLS_DHKEY_BITS_MIN 2048
        #else
            #define WOLF_TLS_DHKEY_BITS_MIN 1024
        #endif
    #endif
#endif

/* Default Port Numbers */
#define MQTT_DEFAULT_PORT   1883
#define MQTT_SECURE_PORT    8883


struct _MqttClient;

/* Function callbacks */
typedef int (*MqttTlsCb)(struct _MqttClient* client);

typedef int (*MqttNetConnectCb)(void *context,
    const char* host, word16 port, int timeout_ms);
typedef int (*MqttNetWriteCb)(void *context,
    const byte* buf, int buf_len, int timeout_ms);
typedef int (*MqttNetReadCb)(void *context,
    byte* buf, int buf_len, int timeout_ms);
#ifdef WOLFMQTT_SN
typedef int (*MqttNetPeekCb)(void *context,
    byte* buf, int buf_len, int timeout_ms);
#endif
typedef int (*MqttNetDisconnectCb)(void *context);

/* Structure for Network Security */
#ifdef ENABLE_MQTT_TLS
typedef struct _MqttTls {
    WOLFSSL_CTX         *ctx;
    WOLFSSL             *ssl;
    int                 sockRc;
    int                 timeout_ms;
} MqttTls;
#endif

/* Structure for Network callbacks */
typedef struct _MqttNet {
    void                *context;
    MqttNetConnectCb    connect;
    MqttNetReadCb       read;
    MqttNetWriteCb      write;
    MqttNetDisconnectCb disconnect;
#ifdef WOLFMQTT_SN
    MqttNetPeekCb       peek;
```

```
    void                *multi_ctx;
#endif
} MqttNet;


/* MQTT SOCKET APPLICATION INTERFACE */
WOLFMQTT_LOCAL int MqttSocket_Init(struct _MqttClient *client, MqttNet* net);
WOLFMQTT_LOCAL int MqttSocket_Write(struct _MqttClient *client, const byte* buf,
        int buf_len, int timeout_ms);
WOLFMQTT_LOCAL int MqttSocket_Read(struct _MqttClient *client, byte* buf,
        int buf_len, int timeout_ms);
#ifdef WOLFMQTT_SN
WOLFMQTT_LOCAL int MqttSocket_Peek(struct _MqttClient *client, byte* buf,
        int buf_len, int timeout_ms);
#endif
WOLFMQTT_LOCAL int MqttSocket_Connect(struct _MqttClient *client,
        const char* host, word16 port, int timeout_ms, int use_tls,
        MqttTlsCb cb);
WOLFMQTT_LOCAL int MqttSocket_Disconnect(struct _MqttClient *client);

#ifdef ENABLE_MQTT_TLS
/* make these public for cases where user needs to create
 * WOLFSSL_CTX context and WOLFSSL object in the TLS callback */
WOLFMQTT_API int MqttSocket_TlsSocketReceive(WOLFSSL* ssl, char *buf, int sz, void *ptr);
WOLFMQTT_API int MqttSocket_TlsSocketSend(WOLFSSL* ssl, char *buf, int sz, void *ptr);
#endif

#ifdef __cplusplus
    } /* extern "C" */
#endif

#endif /* WOLFMQTT_SOCKET_H */
```

5.3 wolfmqtt/wolfmqtt/mqtt_socket.h

5.3.1 Classes

	Name
struct	**_MqttTls**
struct	**_MqttNet**

5.3.2 Types

	Name
typedef int(*)(struct _MqttClient *client)	**MqttTlsCb**
typedef int()(*void* context, const char *host, word16 port, int timeout_ms)	**MqttNetConnectCb**
typedef int()(*void* context, const byte *buf, int buf_len, int timeout_ms)	**MqttNetWriteCb**
typedef int()(*void* context, byte *buf, int buf_len, int timeout_ms)	**MqttNetReadCb**

	Name
typedef int()(*void* context, byte *buf, int buf_len, int timeout_ms)	**MqttNetPeekCb**
typedef int()(*void* context)	**MqttNetDisconnectCb**
typedef struct _MqttTls**	
typedef struct _MqttNet**	

5.3.3 Functions

	Name
WOLFMQTT_LOCAL int	**MqttSocket_Init(struct _MqttNet * net)
WOLFMQTT_LOCAL int	**MqttSocket_Write(struct _MqttClient * client, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Read(struct _MqttClient * client, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Peek(struct _MqttClient * client, byte * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Connect(struct _MqttNet * net, const char * host, word16 port, int timeout_ms, int flags, struct _MqttTls * tls)
WOLFMQTT_LOCAL int	**MqttSocket_Disconnect**(struct _MqttClient * client)
WOLFMQTT_API int	**MqttSocket_TlsSocketReceive**(WOLFSSL * ssl, char * buf, int sz, void * ptr)
WOLFMQTT_API int	**MqttSocket_TlsSocketSend**(WOLFSSL * ssl, char * buf, int sz, void * ptr)

5.3.4 Attributes

Name
C

5.3.5 Types Documentation

5.3.5.1 typedef MqttTlsCb

```
typedef int(* MqttTlsCb) (struct _MqttClient *client);
```

5.3.5.2 typedef MqttNetConnectCb

```
typedef int(* MqttNetConnectCb) (void *context, const char *host, word16 port, int timeout_ms);
```

5.3.5.3 typedef MqttNetWriteCb

```
typedef int(* MqttNetWriteCb) (void *context, const byte *buf, int buf_len, int timeout_ms);
```

5.3.5.4 typedef MqttNetReadCb

```
typedef int(* MqttNetReadCb) (void *context, byte *buf, int buf_len, int timeout_ms);
```

5.3.5.5 typedef MqttNetPeekCb

```
typedef int(* MqttNetPeekCb) (void *context, byte *buf, int buf_len, int timeout_ms);
```

5.3.5.6 typedef MqttNetDisconnectCb

```
typedef int(* MqttNetDisconnectCb) (void *context);
```

5.3.5.7 typedef MqttTls

```
typedef struct _MqttTls MqttTls;
```

5.3.5.8 typedef MqttNet

```
typedef struct _MqttNet MqttNet;
```

5.3.6 Functions Documentation

5.3.6.1 function MqttSocket_Init

```
WOLFMQTT_LOCAL int MqttSocket_Init(
    struct _MqttClient * client,
    MqttNet * net
)
```

5.3.6.2 function MqttSocket_Write

```
WOLFMQTT_LOCAL int MqttSocket_Write(
    struct _MqttClient * client,
    const byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.3.6.3 function MqttSocket_Read

```
WOLFMQTT_LOCAL int MqttSocket_Read(
    struct _MqttClient * client,
    byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.3.6.4 function MqttSocket_Peek

```
WOLFMQTT_LOCAL int MqttSocket_Peek(
    struct _MqttClient * client,
    byte * buf,
    int buf_len,
    int timeout_ms
)
```

5.3.6.5 function MqttSocket_Connect

```
WOLFMQTT_LOCAL int MqttSocket_Connect(
    struct _MqttClient * client,
    const char * host,
    word16 port,
    int timeout_ms,
    int use_tls,
    MqttTlsCb cb
)
```

5.3.6.6 function MqttSocket_Disconnect

```
WOLFMQTT_LOCAL int MqttSocket_Disconnect(
    struct _MqttClient * client
)
```

5.3.6.7 function MqttSocket_TlsSocketReceive

```
WOLFMQTT_API int MqttSocket_TlsSocketReceive(
    WOLFSSL * ssl,
    char * buf,
    int sz,
    void * ptr
)
```

5.3.6.8 function MqttSocket_TlsSocketSend

```
WOLFMQTT_API int MqttSocket_TlsSocketSend(
    WOLFSSL * ssl,
    char * buf,
    int sz,
    void * ptr
)
```

5.3.7 Attributes Documentation

5.3.7.1 variable C

```
C {
#endif

#include "wolfmqtt/mqtt_types.h"
#ifdef ENABLE_MQTT_TLS
    #ifndef WOLF_TLS_DHKEY_BITS_MIN
        #ifdef WOLFSSL_MAX_STRENGTH
            #define WOLF_TLS_DHKEY_BITS_MIN 2048
        #else
            #define WOLF_TLS_DHKEY_BITS_MIN 1024
        #endif
    #endif
#endif


#define MQTT_DEFAULT_PORT   1883
#define MQTT_SECURE_PORT    8883


struct _MqttClient;
```

5.3.8 Source code

```
/* mqtt_socket.h
 *
 * Copyright (C) 2006-2022 wolfSSL Inc.
 *
 * This file is part of wolfMQTT.
```

```
 *
 * wolfMQTT is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * wolfMQTT is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

/* Implementation by: David Garske
 * Based on specification for MQTT v3.1.1
 * See http://mqtt.org/documentation for additional MQTT documentation.
 */

#ifndef WOLFMQTT_SOCKET_H
#define WOLFMQTT_SOCKET_H

#ifdef __cplusplus
    extern "C" {
#endif

#include "wolfmqtt/mqtt_types.h"
#ifdef ENABLE_MQTT_TLS
    #ifndef WOLF_TLS_DHKEY_BITS_MIN /* allow define to be overridden */
        #ifdef WOLFSSL_MAX_STRENGTH
            #define WOLF_TLS_DHKEY_BITS_MIN 2048
        #else
            #define WOLF_TLS_DHKEY_BITS_MIN 1024
        #endif
    #endif
#endif

/* Default Port Numbers */
#define MQTT_DEFAULT_PORT   1883
#define MQTT_SECURE_PORT    8883


struct _MqttClient;

/* Function callbacks */
typedef int (*MqttTlsCb)(struct _MqttClient* client);

typedef int (*MqttNetConnectCb)(void *context,
    const char* host, word16 port, int timeout_ms);
typedef int (*MqttNetWriteCb)(void *context,
    const byte* buf, int buf_len, int timeout_ms);
typedef int (*MqttNetReadCb)(void *context,
```

```
    byte* buf, int buf_len, int timeout_ms);
#ifdef WOLFMQTT_SN
typedef int (*MqttNetPeekCb)(void *context,
    byte* buf, int buf_len, int timeout_ms);
#endif
typedef int (*MqttNetDisconnectCb)(void *context);

/* Structure for Network Security */
#ifdef ENABLE_MQTT_TLS
typedef struct _MqttTls {
    WOLFSSL_CTX         *ctx;
    WOLFSSL             *ssl;
    int                 sockRc;
    int                 timeout_ms;
} MqttTls;
#endif

/* Structure for Network callbacks */
typedef struct _MqttNet {
    void                *context;
    MqttNetConnectCb    connect;
    MqttNetReadCb       read;
    MqttNetWriteCb      write;
    MqttNetDisconnectCb disconnect;
#ifdef WOLFMQTT_SN
    MqttNetPeekCb       peek;
    void                *multi_ctx;
#endif
} MqttNet;


/* MQTT SOCKET APPLICATION INTERFACE */
WOLFMQTT_LOCAL int MqttSocket_Init(struct _MqttClient *client, MqttNet* net);
WOLFMQTT_LOCAL int MqttSocket_Write(struct _MqttClient *client, const byte* buf,
        int buf_len, int timeout_ms);
WOLFMQTT_LOCAL int MqttSocket_Read(struct _MqttClient *client, byte* buf,
        int buf_len, int timeout_ms);
#ifdef WOLFMQTT_SN
WOLFMQTT_LOCAL int MqttSocket_Peek(struct _MqttClient *client, byte* buf,
        int buf_len, int timeout_ms);
#endif
WOLFMQTT_LOCAL int MqttSocket_Connect(struct _MqttClient *client,
        const char* host, word16 port, int timeout_ms, int use_tls,
        MqttTlsCb cb);
WOLFMQTT_LOCAL int MqttSocket_Disconnect(struct _MqttClient *client);

#ifdef ENABLE_MQTT_TLS
/* make these public for cases where user needs to create
 * WOLFSSL_CTX context and WOLFSSL object in the TLS callback */
WOLFMQTT_API int MqttSocket_TlsSocketReceive(WOLFSSL* ssl, char *buf, int sz, void *ptr);
WOLFMQTT_API int MqttSocket_TlsSocketSend(WOLFSSL* ssl, char *buf, int sz, void *ptr);
#endif

#ifdef __cplusplus
```

```
    } /* extern "C" */
#endif

#endif /* WOLFMQTT_SOCKET_H */
```

5.4 wolfmqtt/wolfmqtt/mqtt_types.h

5.4.1 Classes

	Name
struct	**wm_Sem**

5.4.2 Types

	Name
enum	**MqttPacketResponseCodes** { MQTT_CODE_SUCCESS = 0, MQTT_CODE_ERROR_BAD_ARG = -1, MQTT_CODE_ERROR_OUT_OF_BUFFER = -2, MQTT_CODE_ERROR_MALFORMED_DATA = -3, MQTT_CODE_ERROR_PACKET_TYPE = -4, MQTT_CODE_ERROR_PACKET_ID = -5, MQTT_CODE_ERROR_TLS_CONNECT = -6, MQTT_CODE_ERROR_TIMEOUT = -7, MQTT_CODE_ERROR_NETWORK = -8, MQTT_CODE_ERROR_MEMORY = -9, MQTT_CODE_ERROR_STAT = -10, MQTT_CODE_ERROR_PROPERTY = -11, MQTT_CODE_ERROR_SERVER_PROP = -12, MQTT_CODE_ERROR_CALLBACK = -13, MQTT_CODE_ERROR_SYSTEM = -14, MQTT_CODE_ERROR_NOT_FOUND = -15, MQTT_CODE_CONTINUE = -101, MQTT_CODE_STDIN_WAKE = -102, MQTT_CODE_PUB_CONTINUE = -103}
typedef SemaphoreHandle_t	**wm_Sem**
typedef unsigned char	**byte**
typedef unsigned short	**word16**
typedef unsigned int	**word32**

5.4.3 Functions

	Name
WOLFMQTT_API int	**wm_SemInit**(wm_Sem * s)
WOLFMQTT_API int	**wm_SemFree**(wm_Sem * s)
WOLFMQTT_API int	**wm_SemLock**(wm_Sem * s)
WOLFMQTT_API int	**wm_SemUnlock**(wm_Sem * s)
void	**SYS_CMD_PRINT**(const char * format, ...)

5.4.4 Attributes

Name
C

5.4.5 Types Documentation

Enumerator	Value	Description
MQTT_CODE_SUCCESS	0	
MQTT_CODE_ERROR_BAD_ARG	-1	
MQTT_CODE_ERROR_OUT_OF_BUFFER	-2	
MQTT_CODE_ERROR_MALFORMED_DATA	-3	
MQTT_CODE_ERROR_PACKET_TYPE	-4	
MQTT_CODE_ERROR_PACKET_ID	-5	
MQTT_CODE_ERROR_TLS_CONNECT	-6	
MQTT_CODE_ERROR_TIMEOUT	-7	
MQTT_CODE_ERROR_NETWORK	-8	
MQTT_CODE_ERROR_MEMORY	-9	
MQTT_CODE_ERROR_STAT	-10	
MQTT_CODE_ERROR_PROPERTY	-11	
MQTT_CODE_ERROR_SERVER_PROP	-12	
MQTT_CODE_ERROR_CALLBACK	-13	
MQTT_CODE_ERROR_SYSTEM	-14	
MQTT_CODE_ERROR_NOT_FOUND	-15	
MQTT_CODE_CONTINUE	-101	
MQTT_CODE_STDIN_WAKE	-102	
MQTT_CODE_PUB_CONTINUE	-103	

5.4.5.1 enum MqttPacketResponseCodes

5.4.5.2 typedef wm_Sem

```
typedef HANDLE wm_Sem;
```

5.4.5.3 typedef byte

```
typedef unsigned char byte;
```

5.4.5.4 typedef word16

```
typedef unsigned short word16;
```

5.4.5.5 typedef word32

```
typedef unsigned int word32;
```

5.4.6 Functions Documentation

5.4.6.1 function wm_SemInit

```
WOLFMQTT_API int wm_SemInit(
    wm_Sem * s
)
```

5.4.6.2 function wm_SemFree

```
WOLFMQTT_API int wm_SemFree(
    wm_Sem * s
)
```

5.4.6.3 function wm_SemLock

```
WOLFMQTT_API int wm_SemLock(
    wm_Sem * s
)
```

5.4.6.4 function wm_SemUnlock

```
WOLFMQTT_API int wm_SemUnlock(
    wm_Sem * s
)
```

5.4.6.5 function SYS_CMD_PRINT

```
void SYS_CMD_PRINT(
    const char * format,
    ...
)
```

5.4.7 Attributes Documentation

5.4.7.1 variable C

```
C;
```

5.4.8 Source code

```
/* mqtt_types.h
 *
 * Copyright (C) 2006-2022 wolfSSL Inc.
 *
 * This file is part of wolfMQTT.
 *
 * wolfMQTT is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * wolfMQTT is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

/* Implementation by: David Garske
 * Based on specification for MQTT v3.1.1
 * See http://mqtt.org/documentation for additional MQTT documentation.
```

```
 */

#ifndef WOLFMQTT_TYPES_H
#define WOLFMQTT_TYPES_H

/* configuration for Arduino */
#ifdef ARDUINO
    #include "wolfmqtt/options.h"

    /* make sure arduino can see the wolfssl library directory */
    #ifdef ENABLE_MQTT_TLS
        #include <wolfssl.h>
    #endif
#endif

#ifdef __cplusplus
    extern "C" {
#endif

#include "wolfmqtt/visibility.h"

#ifdef _WIN32
    #define USE_WINDOWS_API

    /* Make sure a level of Win compatibility is defined */
    #ifndef _WIN32_WINNT
    #define _WIN32_WINNT 0x0501
    #endif

    /* Allow "unsafe" strncpy */
    #ifndef _CRT_SECURE_NO_WARNINGS
    #define _CRT_SECURE_NO_WARNINGS
    #endif

    /* Visual Studio build settings from wolfmqtt/vs_settings.h */
    #include "wolfmqtt/vs_settings.h"
#endif

#ifdef WOLFMQTT_USER_SETTINGS
#include "user_settings.h"
#endif

#ifdef ENABLE_MQTT_TLS
    #if !defined(WOLFSSL_USER_SETTINGS) && !defined(USE_WINDOWS_API)
        #include <wolfssl/options.h>
    #endif
    #include <wolfssl/wolfcrypt/settings.h>
    #include <wolfssl/ssl.h>
    #include <wolfssl/wolfcrypt/types.h>
    #include <wolfssl/wolfcrypt/error-crypt.h>

    #ifndef WOLF_TLS_DHKEY_BITS_MIN /* allow define to be overridden */
        #ifdef WOLFSSL_MAX_STRENGTH
            #define WOLF_TLS_DHKEY_BITS_MIN 2048
```

```
        #else
            #define WOLF_TLS_DHKEY_BITS_MIN 1024
        #endif
    #endif
#endif

#ifdef WOLFMQTT_MULTITHREAD
    /* Multi-threading uses binary semaphores */
    #if defined(WOLFMQTT_USER_THREADING)
        /* User provides API's and wm_Sem type.
         * Add your wc_Sem into user_settings.h */

    #elif defined(__MACH__)
        /* Apple Style Dispatch Semaphore */
        #include <dispatch/dispatch.h>
        typedef struct {
            dispatch_semaphore_t sem;
        } wm_Sem;

    #elif defined(__FreeBSD__) || defined(__linux__) || defined(__QNX__)
        /* Posix Style Pthread Mutex and Conditional */
        #define WOLFMQTT_POSIX_SEMAPHORES
        #include <pthread.h>
        typedef struct {
            volatile int lockCount;
            pthread_mutex_t mutex;
            pthread_cond_t cond;
        } wm_Sem;

    #elif defined(FREERTOS)
        /* FreeRTOS binary semaphore */
        #include <FreeRTOS.h>
        #include <semphr.h>
        typedef SemaphoreHandle_t wm_Sem;

    #elif defined(USE_WINDOWS_API)
        /* Windows semaphore object */
        #include <winsock2.h> /* winsock2.h needs included before windows.h */
        #include <ws2tcpip.h>
        #include <windows.h>
        typedef HANDLE wm_Sem;

    #else
        #error "Multithreading requires binary semaphore implementation!"
    #endif

    WOLFMQTT_API int wm_SemInit(wm_Sem* s);
    WOLFMQTT_API int wm_SemFree(wm_Sem* s);
    WOLFMQTT_API int wm_SemLock(wm_Sem* s);
    WOLFMQTT_API int wm_SemUnlock(wm_Sem* s);
#endif

/* configuration for Harmony */
#ifdef MICROCHIP_MPLAB_HARMONY
```

```
    #define NO_EXIT

    /* make sure we are using non-blocking for Harmony */
    #ifndef WOLFMQTT_NONBLOCK
        #define WOLFMQTT_NONBLOCK
    #endif

    #include "system_config.h"
    #ifdef SYS_CMD_ENABLE
        extern void SYS_CMD_PRINT(const char *format, ...);

        /* use SYS_PRINT for printf */
        #define WOLFMQTT_CUSTOM_PRINTF
        #define PRINTF(_f_, ...)  SYS_CMD_PRINT( (_f_ "\n"), ##__VA_ARGS__)
    #endif

#endif

#ifndef WOLFMQTT_NO_STDIO
    #include <stdio.h>
#endif

/* Allow custom override of data types */
#if !defined(WOLFMQTT_CUSTOM_TYPES) && !defined(WOLF_CRYPT_TYPES_H)
    /* Basic Types */
    #ifndef byte
        typedef unsigned char  byte;
    #endif
    #ifndef word16
        typedef unsigned short word16;
    #endif
    #ifndef word32
        typedef unsigned int   word32;
    #endif
    #define WOLFSSL_TYPES /* make sure wolfSSL knows we defined these types */
#endif

/* Response Codes */
enum MqttPacketResponseCodes {
    MQTT_CODE_SUCCESS = 0,
    MQTT_CODE_ERROR_BAD_ARG = -1,
    MQTT_CODE_ERROR_OUT_OF_BUFFER = -2,
    MQTT_CODE_ERROR_MALFORMED_DATA = -3, /* Error (Malformed Remaining Len) */
    MQTT_CODE_ERROR_PACKET_TYPE = -4,
    MQTT_CODE_ERROR_PACKET_ID = -5,
    MQTT_CODE_ERROR_TLS_CONNECT = -6,
    MQTT_CODE_ERROR_TIMEOUT = -7,
    MQTT_CODE_ERROR_NETWORK = -8,
    MQTT_CODE_ERROR_MEMORY = -9,
    MQTT_CODE_ERROR_STAT = -10,
    MQTT_CODE_ERROR_PROPERTY = -11,
    MQTT_CODE_ERROR_SERVER_PROP = -12,
    MQTT_CODE_ERROR_CALLBACK = -13,
    MQTT_CODE_ERROR_SYSTEM = -14,
```

```
    MQTT_CODE_ERROR_NOT_FOUND = -15,

    MQTT_CODE_CONTINUE = -101,
    MQTT_CODE_STDIN_WAKE = -102,
    MQTT_CODE_PUB_CONTINUE = -103,
};


/* Standard wrappers */
#ifndef WOLFMQTT_CUSTOM_STRING
    #include <string.h>

    #ifndef XSTRLEN
        #define XSTRLEN(s1)         strlen((s1))
    #endif
    #ifndef XSTRCHR
        #define XSTRCHR(s,c)        strchr((s),(c))
    #endif
    #ifndef XSTRNCMP
        #define XSTRNCMP(s1,s2,n)   strncmp((s1),(s2),(n))
    #endif
    #ifndef XSTRNCPY
        #define XSTRNCPY(s1,s2,n)   strncpy((s1),(s2),(n))
    #endif
    #ifndef XMEMCPY
        #define XMEMCPY(d,s,l)      memcpy((d),(s),(l))
    #endif
    #ifndef XMEMSET
        #define XMEMSET(b,c,l)      memset((b),(c),(l))
    #endif
    #ifndef XMEMCMP
        #define XMEMCMP(s1,s2,n)    memcmp((s1),(s2),(n))
    #endif
    #ifndef XATOI
        #define XATOI(s)            atoi((s))
    #endif
    #ifndef XISALNUM
        #define XISALNUM(c)         isalnum((c))
    #endif
    #ifndef XSNPRINTF
        #ifndef USE_WINDOWS_API
            #define XSNPRINTF        snprintf
        #else
            #define XSNPRINTF        _snprintf
        #endif
    #endif
#endif

#ifndef WOLFMQTT_CUSTOM_MALLOC
    #ifndef WOLFMQTT_MALLOC
        #define WOLFMQTT_MALLOC(s)  malloc((s))
    #endif
    #ifndef WOLFMQTT_FREE
        #define WOLFMQTT_FREE(p)    {void* xp = (p); if((xp)) free((xp));}
```

```
    #endif
#endif

#ifndef WOLFMQTT_PACK
    #if defined(__GNUC__)
        #define WOLFMQTT_PACK __attribute__ ((packed))
    #else
        #define WOLFMQTT_PACK
    #endif
#endif

/* use inlining if compiler allows */
#ifndef INLINE
#ifndef NO_INLINE
    #if defined(__GNUC__) || defined(__MINGW32__) || defined(__IAR_SYSTEMS_ICC__)
           #define INLINE inline
    #elif defined(_MSC_VER)
        #define INLINE __inline
    #elif defined(THREADX)
        #define INLINE _Inline
    #else
        #define INLINE
    #endif
#else
    #define INLINE
#endif /* !NO_INLINE */
#endif /* !INLINE */

#ifndef OFFSETOF
    #if defined(__clang__) || defined(__GNUC__)
        #define OFFSETOF(type, field) __builtin_offsetof(type, field)
    #else
        #define OFFSETOF(type, field) ((size_t)&(((type *)0)->field))
    #endif
#endif

/* printf */
#ifndef WOLFMQTT_CUSTOM_PRINTF
    #ifndef LINE_END
        #define LINE_END    "\n"
    #endif
    #ifndef PRINTF
        #if defined(WOLFMQTT_MULTITHREAD) && defined(WOLFMQTT_DEBUG_THREAD)
            #ifdef USE_WINDOWS_API
                #define PRINTF(_f_, ...)  printf( ("%lx: "_f_ LINE_END), GetCurrentThreadId(),
                 ↪  ##__VA_ARGS__)
            #elif defined(__MACH__)
                #include <pthread.h>
                #define PRINTF(_f_, ...)  printf( ("%p: "_f_ LINE_END), (void*)pthread_self(),
                 ↪  ##__VA_ARGS__)
            #else
                #include <pthread.h>
                #define PRINTF(_f_, ...)  printf( ("%lx: "_f_ LINE_END), pthread_self(),
                 ↪  ##__VA_ARGS__)
```

```
            #endif
        #else
            #define PRINTF(_f_, ...)  printf( (_f_ LINE_END), ##__VA_ARGS__)
        #endif
    #endif

    #ifndef WOLFMQTT_NO_STDIO
        #include <stdlib.h>
        #include <string.h>
        #include <stdio.h>
    #else
        #undef PRINTF
        #define PRINTF
    #endif
#endif

#ifndef FALL_THROUGH
    /* GCC 7 has new switch() fall-through detection */
    #if defined(__GNUC__)
        #if ((__GNUC__ > 7) || ((__GNUC__ == 7) && (__GNUC_MINOR__ >= 1)))
            #undef  FALL_THROUGH
            #if defined(WOLFSSL_LINUXKM) && defined(fallthrough)
                #define FALL_THROUGH fallthrough
            #else
                #define FALL_THROUGH __attribute__ ((fallthrough));
            #endif
        #endif
    #endif
#endif /* FALL_THROUGH */
#if !defined(FALL_THROUGH) || defined(__XC32)
    /* use stub for fall through by default or for Microchip compiler */
    #undef  FALL_THROUGH
    #define FALL_THROUGH
#endif

/* No return macro */
#if defined(__IAR_SYSTEMS_ICC__) || defined(__GNUC__)
    #define WOLFMQTT_NORETURN __attribute__((noreturn))
#else
    #define WOLFMQTT_NORETURN
#endif

/* Logging / Tracing */
#ifdef WOLFMQTT_NO_STDIO
    #undef WOLFMQTT_DEBUG_CLIENT
    #undef WOLFMQTT_DEBUG_SOCKET
#endif

#ifdef WOLFMQTT_DEBUG_TRACE
#define MQTT_TRACE_ERROR(err) ({ PRINTF("ERROR: %d (%s:%d)", err, __FUNCTION__, __LINE__); err; })
#define MQTT_TRACE_MSG(msg)      PRINTF("%s: (%s:%d)", msg, __FUNCTION__, __LINE__);
#else
#define MQTT_TRACE_ERROR(err) err
#define MQTT_TRACE_MSG(msg)
```

```
#endif /* WOLFMQTT_DEBUG_TRACE */

#ifdef __cplusplus
    } /* extern "C" */
#endif

#endif /* WOLFMQTT_TYPES_H */
```