wolfMQTT Documentation

2025-04-22

Contents

1 イントロダクション

このライブラリは MQTT (Message Queuing Telemetry Transport) クライアントの C 言語実装です。マルチプラットフォーム対応、省コード量、拡張性を重視してゼロから記述しました。すべてのパケットタイプ、すべての QoS レベル（0～2）に対応しさらに wolfSSL ライブラリを使用して SSL/TLS をサポートしています。また、本実装は MQTT v3.1.1 仕様に基づいています。

1.1 プロトコル概要

MQTT は軽量なオープンメッセージ送信プロトコルで M2M（機器対機器）、IoT（モノのインターネット）などのコードサイズに制限のある環境向けに開発されました。MQTT はトピックの購読と発行という Publish/Subscribe メッセージ配信モデルに基づいています。このプロトコルは効率的にメッセージをパッケージングしオーバーヘッドを非常に少なく保つことができます。MQTT 仕様ではトランスポートの選択子として TLS を使用し、8883 番（セキュア MQTT) ポートを使ってこのプロトコルをセキュアにすることを推奨しています。制限の多い機器での TLS 使用にはセッション再開を活用して再接続のコストを低減できるかもしれません。

MQTT では以下に要求される配信整合性を QoS レベル 0-2 として定義しています： 0 = たかだか一回の配信を保障：受け取り確認はしない 1 = 少なくとも一回の配信を保障：受け取り確認 (PUBLISH_ACK) を送信する 2 = 正確に一回の配信を保証：受信済み (PUBLISH_REC) を送信、解放 (PUBLISH_REL) を受信、完了 (PUBLISH_COMP) を送信

特筆すべき特徴:

- パブリッシュメッセージは 256MB (28 bits) までのペイロードを含むことが可能
- パケットヘッダーの remaining length は最上位ビット (7) が追加長バイトの存在を意味するようにエンコードされます。
- レスポンスを必要とするパケットは 16bit の packet Id を含めなければなりません。この Id は未応答のトランザクションの識別に必要なのでユニークでなければなりません。通常、連番が使われます
- クライアントは last will(遺言) を接続時に提示することができます。last will はブローカーがこのクライアントは切断してしまったかネットワークの keep-alive 時間を超過したと判断した際に配信されます。
- パケットタイプには、CONNECT, CONNECT_ACK, PUBLISH, PUBLISH_ACK, PUBLISH_REC, PUBLISH_REL, PUBLISH_COMP, SUBSCRIBE, SUBSCRIBE_ACK, UNSUBSCRIBE, UNSUBSCRIBE_ACK, PING_REQ, PING_RESP と DISCONNECT　があります。
- CONNECT パケットは ASCII 文字列の “MQTT” を含んでいて、プロトコル名として定義するために使っています。Wireshark 等を使ったパケット解析時に役に立ちます。
- 複数のトピックを単一のパケットで購読依頼、購読解除することができます。
- 購読されている各トピックは QoS レベルを指定しなければなりません。確定したレベルは購読確認応答パケットで確認できます。
- PUBLISH メッセージはクライアントかブローカーのいずれかによって送信/受信できます。
- QoS レベル 2 は配信確認を完了するために 2 往復の通信を必要とします。
- 文字列は UTF-8 でエンコードされます。

さらなるドキュメントが必要な場合には https://mqtt.org/documentation を参照してください。

1.2 ブローカー互換性

wolfMQTT ライブラリは下記のブローカーとの動作確認を行っています:

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

1.3 サポート仕様

1.3.1 MQTT v3.1.1

以下の様なフル機能と全パケットタイプをサポートした最初にサポートしたバージョンです：

- QoS 0-2
- Last Will and Testament (LWT)
- 次のブローカ対応したクライアントサンプル（AWS, Azure IoT, IBM Watson）
- ファームウエア更新、ノンブロッキング とジェネリックに対応

1.3.2 MQTT v5.0

wolfMQTT クライアントは v5.0 仕様が有効になっているブローカーへの接続をサポートしています（`--enable-mqtt5` オプションを指定してビルドした場合)。ブローカーからのプロパティはコールバックを使ってのハンドリングされます（`--enable-propcb` オプションを指定してビルドした場合)。以下の v5.0 仕様の機能が wolfMQTT クライアントによってサポートされています：

- AUTH パケット
- User プロパティ
- Server connect ACK プロパティ
- Format タイプ と　content タイプ（publish 時）
- Server disconnect
- Reason コードと文字列
- 最大パケットサイズ
- サーバー側付与のクライアント ID
- 購読 ID
- トピックエイリアス

v5.0 仕様が有効化されている wolfMQTT クライアントは次の v5.0 ブローカーとの接続がテストされています：

- Mosquitto

ローカル PC で実行:

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

- Flespi

毎時生成されるトークンと結びつけられたアカウントが必要です:

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

- VerneMQ MQTTv5 preview

ローカル PC で実行:

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

- HiveMQ 4.0.0 EAP

ローカル PC で実行:

```
./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 センサーネットワーク (MQTT-SN)

wolfMQTT クライアント実装は OASIS MQTT-SN v1.2 仕様に基づいています。この SN API は`--enable-sn`オプションと共にビルドすると実行可能に構成されます。センサーネットワーク用 API は”SN_” がプレフィクスとして付加されている別の API です。wolfMQTT SN クライアントは UDP 上で動作しますが、TCP 上で動作する wolfMQTT とは区別されています。以下の機能が wolfMQTT SN クライアントとしてサポートされています：

- Register
- Will topic とメッセージセットアップ
- Will topic と メッセージ更新
- 全 QoS レベル
- 可変サイズのパケット長フィールド

サポートしていない機能：

- 自動ゲートウェイ検索
- 複数ゲートウェイハンドリング

SN クライアントは Eclipse Paho MQTT-SN Gateway (https://github.com/eclipse/paho.mqtt-sn.embedded-c) を分離したネットワークノード上でローカルに動作させた環境でテストしました。ゲートウェイのビルドと実行に関してはプロジェクトに含まれている README に記載しています。

2 wolfMQTT のビルド

wolfMQTT は移植性を念頭に置いて作成されており、通常、ほとんどのシステムで簡単に構築できるはずです。ビルドに問題がある場合は、**サポート フォーラム** https://www.wolfssl.com/forums からサポートを求めるか、**support@wolfssl.com** まで直接お問い合わせください。この章では、Unix および Windows で wolfMQTT を構築する方法を説明し、非標準環境で構築するためのガイダンスを提供します。

autoconf / automake システムを使用してビルドする場合、wolfMQTT は単一の Makefile を使用してライブラリのすべてのパーツとサンプルをビルドします。これは、Makefile を再帰的に使用するよりも簡単で高速です。TLS 機能またはファームウェア/Azure IoT Hub の例を使用する場合は、wolfSSL をインストールする必要があります。wolfSSL と wolfMQTT については、以下の設定オプションを使用することをお勧めします

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

wolfSSL の場合は、`make && sudo make install` を使用します。libwolfssl.so の検索でエラーが発生した場合は、wolfSSL ディレクトリから `sudo ldconfig` を実行します。

2.1 ソースコードの取得

wolfMQTT の最新バージョンは、wolfSSL ダウンロード ページからダウンロードできます。あるいは Github から次のコマンドで取得できます:

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

2.2 *nix/Mac 上でのビルド

Linux, *BSD, OS X, Solaris,* あるいは他の nix-ライクなシステムでのビルドは autoconf システムを使ってください。github からクローンした場合には, 次のコマンドを実行してください:

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

そのほかの場合には次の 2 行のコマンドを実行してください:

```
./configure
make
```

`./configure` にはビルド オプションをいくつでも追加できます。利用可能なビルド オプションのリストについては、以下のコマンドを実行してください：

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

wolfMQTT のビルドには次の make コマンドを実行:

```
make
```

wolfMQTT をインストールする為に:

```
make install
```

インストールの実行の為にはスーパーユーザー権限が必要かもしれません。その場合にはコマンドの前に sudo をつけてください：

```
sudo make install
```

ノート：* wolfssl が最近インストールされた場合は、sudo ldconfig を実行してリンカ キャッシュを更新します。* デバッグ メッセージは、--enable-debug または --enable-debug=verbose を使用して有効にできます (追加のロギング用)。* ビルド オプションのリストについては、./configure --help を実行してください。* ビルド オプションは、wolfmqtt/options.h のファイルに生成されます。

ビルドをテストするには、ルート wolfMQTT ソース ディレクトリから *mqttclient* プログラムを実行します:

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

wolfMQTT ライブラリのみをビルドし、追加のアイテム (サンプルプログラム等) はビルドしないようにする場合は、wolfMQTT ルート ディレクトリから次のコマンドを実行できます:

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

2.3 Windows 上でのビルド

Visual Studio :

Visual Studio で TLS をサポートする wolfMQTT を構築する場合:

1. <wolfssl-root>/wolfssl64.sln を開きます。
2. Visual Studio のバージョンを再ターゲットします (ソリューションを右クリックし、ソリューションの再ターゲット を選択します)。
3. DLL のデバッグまたは DLL のリリース構成が選択されていることを確認します。32 ビットの x86 または 64 ビットの x64 をビルドしている場合は注意してください。
4. wolfSSL ソリューションを構築します。
5. wolfssl.lib および wolfssl.dll ファイルを <wolfmqtt-root> にコピーします。
   - x86 を使用した DLL Debug の場合、ファイルは DLL Debug にあります。
   - x86 の DLL Release の場合、ファイルは DLL Release にあります。
   - x64 を使用した DLL Debug の場合、ファイルは x64/DLL Debug にあります。
   - x64 の DLL Release の場合、ファイルは x64/DLL Release にあります。
6. <wolfmqtt-root>/wolfmqtt.sln ソリューションを開きます。
7. 上記の wolfSSL で使用されているのと同じアーキテクチャ (x86 または x64 が選択されている) であることを確認してください。
8. デフォルトでは、wolfSSL ヘッダーのインクルード パスは ./../wolfssl/ です。wolfSSL ルートの場所が異なる場合は、プロジェクト設定に移動し、C/C++ -> 一般 -> 追加のインクルード ディレクトリ でこれを調整できます。
9. wolfmqtt/vs_settings.h を使用して Visual Studio のビルド設定を構成します。
10. wolfMQTT ソリューションをビルドします。

Visual Studio 2015 の wolfmqtt.sln ソリューションは、インストールのルート ディレクトリに含まれています。各ビルドをテストするには、Visual Studio メニューから “Build All” を選択し、mqttclient プログラムを実行します。Visual Studio プロジェクトでビルド オプションを編集するには、目的のプロジェクト (wolfmqtt、mqttclient) を選択し、“プロパティ” パネルを参照します。

必要な wolfssl.dll のビルド手順については、こちら を参照してください。完了したら、wolfssl.dll と wolfssl.lib を wolfMQTT ルートにコピーします。プロジェクトは、wolfSSL ヘッダーが ../wolfssl/ にあることも前提としています。

Cygwin : Cygwin、または *nix* のようなコマンドと機能を提供する *Windows* 用のその他のツールセットを使用している場合は、上記のセクション *2.2* の「nix でのビルド」の手順に従ってください。Windows 開発マシンで Windows 用の wolfMQTT をビルドする場合、付属の Visual Studio プロジェクト ファイルを使用して wolfMQTT をビルドすることをお勧めします。

2.4 Arduino 上でのビルド

2.4.1 wolfMQTT を Arduino ライブラリとコンパチブルに再構成

wolfmqtt-arduino.sh は、Arduino プロジェクトと互換性があるように wolfMQTT ライブラリを再編成するシェル スクリプトです。Arduino IDE では、ライブラリのソース ファイルがライブラリのルート ディレクトリにあり、ライブラリの名前にヘッダーファイルが含まれている必要があります。このスクリプトは、すべてのソース ファイルを IDE/ARDUINO/wolfMQTT ディレクトリにコピーし、wolfMQTT.h という名前のスタブヘッダー ファイルを作成します。

Arduino で wolfMQTT を設定するには、IDE/ARDUINO ディレクトリ内から次のように入力します:

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

2.4.2 wolfMQTT を Arduino ライブラリ群に含める (for Arduino version 1.8.2)

1. Arduino IDE で:
   - Sketch -> Include Library -> Add .ZIP Library... で、「IDE/ARDUNIO/wolfMQTT」フォルダー。
   - Sketch -> Include Library で wolfMQTT を選択します。

TLS サポートを有効にするには、#define ENABLE_MQTT_TLS を追加します IDE/ARDUNIO/wolfMQTT/wolfmqtt/options.h. 注: wolfSSL TLS を使用している場合は、wolfSSL に対してもこれを行う必要があります。手順については、<wolfssl-root>/IDE/ARDUINO/README.md を参照してください。

サンプルの wolfMQTT クライアント INO スケッチは次の場所にあります。wolfmqtt_client/wolfmqtt_client.ino を参照して、wolfMQTT ライブラリの使用方法を示します。

2.5 MinGW を使ってビルド

wolfSSL と wolfMQTT の両方をダウンロードしたら、以下のスクリプトを実行してビルドし、両方をインストールします:

```
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 非標準環境でのビルド

公式にはサポートされていませんが、特に組み込みシステムやクロスコンパイルシステムを使用して、非標準環境で wolfMQTT を構築したいユーザーを支援しようとしています。以下は、これを開始する際の注意

事項です。

1. ソース ファイルとヘッダー ファイルは、wolfMQTT ダウンロード パッケージと同じディレクトリ構造にある必要があります。
2. 一部のビルド システムでは、wolfMQTT ヘッダー ファイルの場所を明示的に知りたい場合があるため、それを指定する必要がある場合があります。これらは、<wolfmqtt_root>/wolfmqtt ディレクトリにあります。通常、ディレクトリをインクルード パスに追加して、ヘッダーの問題を解決できます。
3. 構成プロセスがビッグ エンディアンを検出しない限り、wolfMQTT はリトルエンディアン システムにデフォルト設定されます。非標準環境でビルドするユーザーは構成プロセスを使用しないため、ビッグ エンディアン システムを使用する場合は BIG_ENDIAN_ORDER を定義する必要があります。
4. ライブラリの構築を試み、問題が発生した場合はお知らせください。サポートが必要な場合は、*support@wolfssl.com* までご連絡ください。

2.7 クロスコンパイル

組み込みプラットフォームの多くのユーザーは、環境に合わせてクロス コンパイルします。ライブラリをクロスコンパイルする最も簡単な方法は、./configure システムを使用することです。これにより、wolfMQTT のビルドに使用できる Makefile が生成されます。クロス コンパイルするときは、次のように ./configure にホストを指定する必要があります:

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

使用するコンパイラ、リンカーなどを指定する必要がある場合もあります:

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

クロスコンパイル用に wolfMQTT を正しく設定した後は、ライブラリのビルドとインストールのための標準的な autoconf プラクティスに従うことができるはずです:

```
make
sudo make install
```

wolfMQTT のクロス コンパイルに関する追加のヒントやフィードバックがある場合は、*info@wolfssl.com* までお知らせください。

2.7.1 特定のディレクトリへのインストール

wolfSSL のカスタムインストール ディレクトリをセットアップし、wolfMQTT のカスタム wolfSSL lib/include ディレクトリを指定するには、以下を使用します:

wolfSSL ディレクトリで:

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

これにより、ライブラリが ~/wolfssl/lib に配置され、~/wolfssl/include にインクルードされます。

wolfMQTT ディレクトリで:

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

上記のパスが実際の場所と一致していることを確認してください。# 始めましょう

皆さんが実装を開始するためのステップを説明します：

1. Connect、Read、Write、および Disconnect のネットワーク コールバック関数を作成します。参考にする実装は、examples/mqttnet.c および examples/mqttnet.h です。
2. MqttNet 構造でネットワーク コールバック関数とコンテキストを定義します。
3. MqttClient_Init を呼び出して、MqttClient 構造ポインター、MqttNet 構造ポインター、MqttMsgCb 関数コールバック ポインター、最大長の TX/RX バッファー、およびコマンド タイムアウトを渡します。
4. MqttClient_NetConnect を呼び出して、ネットワーク経由でブローカーに接続します。use_tls がゼロ以外の値の場合、TLS 接続を実行します。wolfSSL 証明書の構成には、TLS コールバック MqttTlsCb を定義する必要があります。
5. MqttConnect 構造体へのポインターを渡す MqttClient_Connect を呼び出して、MQTT 接続コマンドを送信し、Connect Ack を待ちます。
6. MqttSubscribe 構造体へのポインターを渡す MqttClient_Subscribe を呼び出して、MQTT Subscribe コマンドを送信し、Subscribe Ack を待機します (QoS レベルによって異なります)。
7. MqttClient_WaitMessage を呼び出して、MqttMessage へのポインターを渡して、着信 MQTT Publish メッセージを待機します。

2.8 サンプルプログラム

2.8.1 クライアント サンプルプログラム

MQTT クライアントのサンプルプログラムは、`/examples/mqttclient/` にあります。このサンプルプログラムでは、公開 API の多くを実行し、サブスクライブしたトピック「wolfMQTT/example/testTopic」に対してパブリッシュされたメッセージを受信して内容を出力します。このクライアントには、プロパティ コールバックやクライアント ID のサーバー割り当てなど、多くの MQTTv5 機能の例が含まれています。mqqtclient の例は、MQTT アプリケーションの足がかりのテンプレートとして適しています。

2.8.2 シンプルスタンドアロンクライアント サンプルプログラム

このサンプルプログラムは、`/examples/mqttsimple/` にあります。このサンプルプログラムは、標準の BSD ソケットを使用するスタンドアロン クライアントを示しています。これには、./configure によって生成された `wolfmqtt/config.h` ファイルに `HAVE_SOCKET` の定義を追加する必要があります。すべてのパラメーターは、`/examples/mqttsimple/mqttsimple.c` の上部で定義されたビルド時マクロです。

2.8.3 ノンブロッキングクライアント サンプルプログラム

このサンプルプログラムは、`/examples/nbclient/` にあります。このサンプルプログラムは、メッセージ交換にノンブロッキング I/O を使用しています。wolfMQTT ライブラリは、`--enable-nonblock` オプションを指定する必要があります (または `WOLFMQTT_NONBLOCK` を定義してビルドします)。

2.8.4 ファームウエア サンプルプログラム

このサンプルプログラムは `/examples/firmware/`にあります。このサンプルプログラムは 2 つの実行プログラムから成っています。1 つ目は”fwpush” と呼ばれ、ファームウェア イメージに署名してパブリッシュします。2 つ目は”fwclient” と呼ばれ、ファームウェア イメージを受信して署名を検証します。“fwpush” はトピック「wolfMQTT/example/firmware」にメッセージをパブリッシュし、パブリッシュコールバックを使用してペイロード データを送信します。

2.8.5 Azure IoT Hub サンプルプログラム

テスト用に Azure サーバーに wolfMQTT IoT Hub をセットアップします。弊社では既に皆さんが接続してパブリッシュすることができる「demoDevice」というデバイスを追加してあります。このサンプルプログラムは、MQTT 接続パケットのパスワードとして使用される SasToken の作成を示しています。また、イベントを発行し、「devicebound」メッセージをリッスンするためのトピック名も示しています。このサンプルプログラムは、Base64 エンコード/デコードと HMAC-SHA256 を必要とするため、「ENABLE_MQTT_TLS」が設

定され、wolfSSL ライブラリが存在する場合にのみ機能します。(注：wolfSSL ライブラリは、./configure --enable-base64encode または #define WOLFSSL_BASE64_ENCODE でビルドする必要があります)。wc_GetTime API は 3.9.1 で追加されました。存在しない場合は、独自のバージョンを実装して現在の UTC 秒を取得するか、wolfSSL ライブラリを更新する必要があります。**注** Azure ブローカーは MQTT v3.1.1 のみをサポートします

2.8.6 AWS IoT サンプルプログラム

テスト用に AWS IoT エンドポイントとテスト デバイス証明書をセットアップします。AWS サーバーは、認証に TLS クライアント証明書を使用します。サンプルプログラムは「/examples/aws/」にあります。このサンプルプログラムでは、$aws/things/"AWSIOT_DEVICE_ID"/shadow/update/delta にサブスクライブし、$aws/things/"AWSIOT_DEVICE_ID"/shadow/update に発行します。**注** AWS ブローカーは MQTT v3.1.1 のみをサポートします

2.8.7 Watson IoT サンプルプログラム

このサンプルプログラムでは、wolfMQTT クライアントが IBM Watson Internet of Things (WIOT) Platform に接続できるようにします。WIOT プラットフォームには、「クイックスタート」と呼ばれる限定的なテスト ブローカーがあり、安全でない接続でコンポーネントを実行できます。サンプルプログラムは /examples/wiot/ にあります。MQTT v5 サポートを有効にして動作します。

2.8.8 MQTT-SN サンプルプログラム

Sensor Network クライアントは、低帯域幅ネットワーク用に MQTT-SN プロトコルを実装します。サブスクライブおよびパブリッシュ時に完全なトピックの代わりに 2 バイトのトピック ID を使用できるなど、MQTT とはいくつかの違いがあります。SN クライアントには MQTT-SN ゲートウェイが必要です。ゲートウェイは、SN クライアントとブローカーの間の仲介者として機能します。このクライアントは、Eclipse Paho MQTT-SN ゲートウェイでテストしました。これは、wolfMQTT クライアントの例と同様に、デフォルトでパブリック Eclipse ブローカーに接続します。ゲートウェイのアドレスは、ホストとして構成する必要があります。サンプルは /examples/sn-client/ にあります。

MQTT-SN の特別な機能は、最初にゲートウェイに接続することなく、QoS レベル -1 (負のレベル) を使用して定義済みのトピックに発行できることです。エラーが発生した場合、アプリケーションにはフィードバックがないため、テストの確認には、最初に「sn-client」を実行し、「sn-client_qos-1」からの発行を監視する必要があります。/examples/sn-client/sn-client_qos-1 で提供されるサンプルプログラムがありますが、利用にはゲートウェイの構成をいくつか変更する必要があります。

- QoS-1 機能と定義済みのトピックを有効にし、gateway.conf でゲートウェイ名を変更します:

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

- すべてのエントリをコメントアウトし、predefinedTopic.conf に新しいトピックを追加します:

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

2.8.9 マルチスレッド サンプルプログラム

このサンプルプログラムでは、クライアントライブラリのマルチスレッド機能を実行します。クライアントは 2 つのタスクを実装します。1 つはブローカーにパブリッシュするタスクです。もう 1 つはブローカーか

らのメッセージを待ちます。パブリッシュ スレッドは「NUM_PUB_TASKS」回 (デフォルトでは 10 回) 作成され、一意のメッセージをブローカーに送信します。この機能は、--enable-mt 構成オプションを使用して有効にします。サンプルは /examples/multithread/ にあります。

2.9 サンプルプログラムのオプション

コマンド ラインの例は、オプションのパラメーターを使用して実行できます。使用可能なパラメーターのリストを表示するには、-? を追加します。

```
 ./examples/mqttclient/mqttclient -?
mqttclient:
-?          ヘルプ, この使用法ページをプリント
-h <host>   接続先ホスト, デフォルトでは: test.mosquitto.org
-p <num>    接続先ポート番号, デフォルトでは: Normal 1883, TLS 8883
-t          TLSを有効にする
-A <file>   CA 証明書をロード(相手の認証に必要)
-K <key>    秘密鍵(TLS の相互認証に必要)
-c <cert>   自分の証明書(TLS の相互認証に必要)
-S <str>    ホスト名インジケーションを使用、空白のデフォルトはhost
-q <num>    Qos Level 0-2, default: 0
-s          クリーンセッションコネクトフラグをDisableにする
-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          メッセージのパブリッシュ時に保持フラグをセット
-C <num>    Command Timeout, default: 30000ms
-P <num>    受け付ける最大パケットサイズ, default: 1048576
-T          テストモード
-f <file>   パブリッシュにファイルの内容を使う
```

使用できるオプションはライブラリのコンフィグレーションによって変わります。

3 ライブラリデザイン

ライブラリのヘッダーファイル群は/wolfmqtt ディレクトリに格納してあります。/wolfmqtt/mqtt_client.h ファイルだけがプロジェクトでインクルードすべきヘッダーファイルです:

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

ライブラリは３つのコンポーネントから構成されています：

3.0.1 mqtt_client

MQTT クライアント向けの最上位アプリケーションインターフェースが位置する場所です。

- `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);`

次の API はエラーかタイムアウト (cmd_timeout_ms) が発生するまで、MqttNet.read でブロックする API 群です：

- `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);`

この関数は新しいパブリッシュメッセージの到着を待って最大 timeout_ms 時間だけブロックします：

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

ネットワークの接続/切断　用インターフェースで MqttNet コールバック関数をラップし wolfSSL TLS をハンドリングする API：

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

ヘルパー関数:

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

3.0.2 mqtt_packet

ここでは全パケットのエンコード/デコードをハンドリングするコードが含まれています。ヘッダーは MQTT 構造体を含んでいます：

- 接続: MqttConnect
- パブリッシュ / メッセージ: MqttPublish / MqttMessage (これらは同一)
- サブスクライブ: MqttSubscribe
- サブスクライブ解除: MqttUnsubscribe

3.0.3 mqtt_socket

これこには、トランスポート ソケットがオプションで TLS をラップし、プラットフォーム固有のネットワーク処理のために MqttNet コールバックを使用するコードが含まれています。

ヘッダーは MQTT ネットワーク構造体 MqttNet、コールバック関数、コンテキスト定義を含んでいます。

3.0.4 サンプルプログラムのデザイン

サンプルプログラムでは、共通の examples/mqttnet.c を使用して、クライアントでネットワーク コールバックを処理します。このプログラムは、Linux (BSD ソケット)、FreeRTOS/LWIP、MQX RTCS、Harmony、および Windows をサポートしています。# API Reference

3.1 header-ja/mqtt_client.h

3.1.1 Classes

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

3.1.2 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 メッセージ受信コールバック関数。メッセージ本文が最大受信バッファ（RX）サイズ以上の場合には、このコールバック関数が複数回呼び出されます。もし、msg_new が 1 の場合には新しいメッセージを意味します。 topic_name と topic_name length は msg_new が 1 の場合にのみ意味があります。 msg_new が 0 の場合には追加のペイロードを受信中であることを意味します。コールバックの都度、MqttMessage.buffer にペイロードが格納されます。MqttMessage.buffer_len はペイロード用のバッファーのサイズです。 MqttMessage.buffer_pos は受信したのペイロードの位置を表します。MqttMessage.total_len は受信が完了したペイロードの長さを表します。 msg_done が 1 の場合には、パブリッシュされたペイロード全体が受信済みであることを表します。

	Name
typedef int*)(MqttPublish* publish)	**MqttPublishCb** Mqtt パブリッシュコールバック関数。パブリッシュするペイロードが最大送信バッファ（TX) サイズ以上の場合にはこのコールバック関数が複数回呼び出されます。このコールバック関数では MqttPublish を呼び出した際にその内部から呼び出されます。このコールバック関数はバッファーを提供し、そのサイズを成功時の戻り値（0 以上）として返却することが求められます。コールバック関数の呼び出しの都度、ペーロードが MqttPublish.buffer にコピーされます。MqttPublish.buffer_len はバッファー内のペイロードのサイズを表します。MqttPublish.total_len は完全なペイロードメッセージの長さを表します。
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 レジスターコールバック関数。ゲートウェイはクライアントに対してトピック名と PUBLISH メッセージ送信時に使用するために割り当てられたトピック id をクライアントに通知する目的で REGISTER メッセージを送信します。 このコールバック関数はクライアントにトピック ID を受け入れて保存するか、その ID が未知の場合には拒否できます。拒否する場合には、regack は”unsupported” 戻り値を伴います。
typedef struct _MqttClient**	

3.1.3 Functions

	Name
WOLFMQTT_API int	**MqttClient_Init * rx_buf, int rx_buf_len, int cmd_timeout_ms)MqttClient 構造体を初期化します。
WOLFMQTT_API void	**MqttClient_DeInit * client)MqttClient 構造体にアロケートされたリソースをクリーンアップします。
WOLFMQTT_API int	**MqttClient_SetDisconnectCallback discb, void * ctx)disconnect コールバック関数とユーザーコンテキストをセットします。
WOLFMQTT_API int	**MqttClient_SetPropertyCallback propCb, void * ctx)property コールバック関数とユーザーコンテキストをセットします。

	Name
WOLFMQTT_API int	**MqttClient_Connect * connect)MQTT Connect パケットをエンコードして送信し、Connect Acknowledgment パケットを待ちます。
WOLFMQTT_API int	**MqttClient_Publish * publish)MQTT Publish パケットをエンコードして送信し、Publish response を待ちます (QoS > 0 の場合)。ペイロードのサイズがバッファーサイズより大きい場合には、ペイロード全体を送信するまでこの関数を連続して複数回呼び出すことができます (QoS > 0 の場合)。
WOLFMQTT_API int	**MqttClient_Publish_ex pubCb)MQTT Publish パケットをエンコードして送信し、Publish response を待ちます (QoS > 0 の場合)。コールバック関数は送信バッファーより大きなサイズのペイロードデータを送信バッファーにコピーすることに使用されます。
WOLFMQTT_API int	**MqttClient_Publish_WriteOnly pubCb)MqttClient_Publish_ex と同様の機能ですが, 本関数はデータの書き込みだけを行い、他のスレッドで MqttClient_WaitMessage_ex を呼び出して read ACK のハンドリングを行うことを必要としてます。
WOLFMQTT_API int	**MqttClient_Subscribe * subscribe)MQTT Subscribe パケットをエンコードして送信し、Subscribe Acknowledgment パケットを待ちます。
WOLFMQTT_API int	**MqttClient_Unsubscribe * unsubscribe)MQTT Unsubscribe パケットをエンコードして送信し、Unsubscribe Acknowledgment パケットを待ちます。
WOLFMQTT_API int	**MqttClient_Ping * client)MQTT Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。
WOLFMQTT_API int	**MqttClient_Ping_ex * ping)MQTT Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。この ex バージョンでは、MqttPing 構造体を引数にとり、ノンブロッキングアプリケーションで使用できます。
WOLFMQTT_API int	**MqttClient_Auth * auth)MQTT Authentication Request パケットをエンコードして送信し、Ping Response パケットを待ちます。
WOLFMQTT_API MqttProp ** head) 新しいプロパティを追加します。プロパティ構造体をアロケートし、head で指すリストヘッドに追加します。パケットコマンドに先だって呼び出す必要があります。	
WOLFMQTT_API int	**MqttClient_PropsFree * head) プロパティリストを解放します。head で指すリストを解放します。MqttClient_Prop_Add を使ったパケットコマンドの使用後に呼び出さなければなりません。

	Name
WOLFMQTT_API int	**MqttClient_Disconnect * client)MQTT Disconnect パケットをエンコードして送信します (response はありません)。
WOLFMQTT_API int	**MqttClient_Disconnect_ex * disconnect)MQTT Disconnect パケットをエンコードして送信します (response はありません)。
WOLFMQTT_API int	**MqttClient_WaitMessage * client, int timeout_ms) パケット受信を待ちます。受信する publish メッセージは MqttClient_Init に渡したコールバック関数を介して受信します。
WOLFMQTT_API int	**MqttClient_WaitMessage_ex * msg, int timeout_ms) パケット受信を待ちます。受信する publish メッセージは MqttClient_Init に渡したコールバック関数を介して受信します。
WOLFMQTT_API int	**MqttClient_CancelMessage * msg) マルチスレッド環境でノンブロッキングモード時に、msg で指定した MQTT オブジェクトをキャンセルします。
WOLFMQTT_API int	**MqttClient_NetConnect cb)TLS 接続を実行し、MqttTlsCb コールバック関数を呼び出します (use_tls が非ゼロの場合)。
WOLFMQTT_API int	**MqttClient_NetDisconnect * client)TLS 接続を切断します。
WOLFMQTT_API int	**MqttClient_GetProtocolVersion * client) プロトコルバージョン番号を取得します。
WOLFMQTT_API const char *	**MqttClient_GetProtocolVersionString * client) プロトコルバージョン文字列を取得します。
WOLFMQTT_API const char *	**MqttClient_ReturnCodeToString**(int return_code)WOLFMQTT_API からの戻り値を文字列に変換します。
WOLFMQTT_API int	**SN_Client_SearchGW * search) ゲートウェイを検索するメッセージをエンコードして送信し、ゲートウェイ情報応答メッセージを待ちます。
WOLFMQTT_API int	**SN_Client_Connect * connect)Connect パケットをエンコードして送信し、Connect Acknowledge パケットを待ちます。もし、Will（遺言）がイネーブルとなっている場合には、ゲートウェイは LWT トピックとメッセージを送信するように別パケットで促してきます。空の will トピックを指定すると will トピックと will メッセージをサーバーから削除するものとみなします。the Will message stored in the server.
WOLFMQTT_API int	**SN_Client_WillTopicUpdate * will)MQTT-SN Will Topic Update パケットをエンコードして送信します。will 引数として NULL を渡すと、Will トピックとサーバーに保持されている Will メッセージを削除することを意味します。
WOLFMQTT_API int	**SN_Client_WillMsgUpdate * will)MQTT-SN Will Message Update パケットをエンコードして送信します。

	Name
WOLFMQTT_API int	**SN_Client_Register * regist)MQTT-SN Register パケットをエンコードして送信し、Register Acknowledge パケットを待ちます。クライアントがゲートウェイに送信する Register パケットは、含まれているトピック名に対してトピック ID を割り当てるように要求する目的で送信されます。また、ゲートウェイからクライアントに送信される Register パケットはトピック名に対してトピック ID が割り当てられたことを通知する目的で送信されます。
WOLFMQTT_API int	**SN_Client_SetRegisterCallback regCb, void * ctx) カスタム（ユーザー）コンテキストと共にレジスターコールバック関数をセットします。
WOLFMQTT_API int	**SN_Client_Publish * publish)MQTT-SN Publish パケットをエンコードして送信し Publish response を待ちます (QoS > 0 の場合)。
WOLFMQTT_API int	**SN_Client_Subscribe * subscribe)MQTT-SN Subscribe パケットをエンコードして送信し、割り当てられたトピック ID を含む Subscribe Acknowledgment パケットを待ちます。
WOLFMQTT_API int	**SN_Client_Unsubscribe * unsubscribe)MQTT-SN Unsubscribe パケットをエンコードして送信し、Unsubscribe Acknowledgment パケットを待ちます。
WOLFMQTT_API int	**SN_Client_Disconnect * client)MQTT-SN Disconnect パケットをエンコードして送信します。その際に、クライアントはスリープ状態に入るまでの時間を送信できます。
WOLFMQTT_API int	**SN_Client_Disconnect_ex * disconnect)MQTT-SN Disconnect パケットをエンコードして送信します。その際に、クライアントはスリープ状態に入るまでの時間を送信できます。
WOLFMQTT_API int	**SN_Client_Ping * ping)MQTT-SN Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。クライアントがスリープ状態の場合でゲートウェイに起床状態に遷移したことを通知したい場合にはクライアント ID を ping request に追加する必要があります。
WOLFMQTT_API int	**SN_Client_WaitMessage * client, int timeout_ms) パケット受信を待ちます。受信した Publish メッセーは MqttClient_Init で与えられたコールバック関数を介して通知されます。
WOLFMQTT_API int	**SN_Client_WaitMessage_ex * packet_obj, int timeout_ms)

3.1.4 Attributes

Name
C

3.1.5 Types Documentation

3.1.5.1 enum MqttClientFlags

Enumerator	Value	Description
MQTT_CLIENT_FLAG_IS_CONNECTED	0x01	
MQTT_CLIENT_FLAG_IS_TLS	0x02	

3.1.5.2 enum _MqttPkStat

Enumerator	Value	Description
MQTT_PK_BEGIN		
MQTT_PK_READ_HEAD		
MQTT_PK_READ		

3.1.5.3 typedef MqttMsgCb

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

Mqtt メッセージ受信コールバック関数。メッセージ本文が最大受信バッファ（RX）サイズ以上の場合には、このコールバック関数が複数回呼び出されます。もし、msg_new が 1 の場合には新しいメッセージを意味します。topic_name と topic_name length は msg_new が 1 の場合にのみ意味があります。msg_new が 0 の場合には追加のペイロードを受信中であることを意味します。コールバックの都度、MqttMessage.buffer にペイロードが格納されます。MqttMessage.buffer_len はペイロード用のバッファーのサイズです。MqttMessage.buffer_pos は受信したのペイロードの位置を表します。MqttMessage.total_len は受信が完了したペイロードの長さを表します。msg_done が 1 の場合には、パブリッシュされたペイロード全体が受信済みであることを表します。

Parameters:

- **client** MqttClient 構造体へのポインター
- **message** ペイロードの属性が設定済みの MqttMessage 構造体へのポインター
- **msg_new** 非ゼロ値の場合、メッセージが新規でトピック名とトピック長は有効な値を保持しています。
- **msg_done** 非ゼロ値の場合、メッセージ全体とトピック全体を受信済です。

Return: MQTT_CODE_SUCCESS：接続を継続します。他の値の場合には接続は切られます。MqttPacketResponseCodes を参照のこと）

3.1.5.4 typedef MqttPublishCb

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

Mqtt パブリッシュコールバック関数。パブリッシュするペイロードが最大送信バッファ（TX) サイズ以上の場合にはこのコールバック関数が複数回呼び出されます。このコールバック関数では MqttPublish を呼び出した際にその内部から呼び出されます。このコールバック関数はバッファーを提供し、そのサイズを成功時の戻り値（0 以上）として返却することが求められます。コールバック関数の呼び出しの都度、ペーロ

ードが MqttPublish.buffer にコピーされます。MqttPublish.buffer_len はバッファー内のペイロードのサイズを表します。MqttPublish.total_len は完全なペイロードメッセージの長さを表します。

Parameters:

- **publish** MqttPublish 構造体へのポインター

Return: >= 0 成功

3.1.5.5 typedef MqttPkStat

```
typedef enum _MqttPkStat MqttPkStat;
```

3.1.5.6 typedef MqttPkRead

```
typedef struct _MqttPkRead MqttPkRead;
```

3.1.5.7 typedef MqttSk

```
typedef struct _MqttSk MqttSk;
```

3.1.5.8 typedef MqttDisconnectCb

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

3.1.5.9 typedef MqttPropertyCb

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

3.1.5.10 typedef SN_ClientRegisterCb

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

Mqtt-SN レジスターコールバック関数。ゲートウェイはクライアントに対してトピック名と PUBLISH メッセージ送信時に使用するために割り当てられたトピック id をクライアントに通知する目的で REGISTER メッセージを送信します。 このコールバック関数はクライアントにトピック ID を受け入れて保存するか、その ID が未知の場合には拒否できます。拒否する場合には、regack は”unsupported” 戻り値を伴います。

Parameters:

- **topicId** 新トピック ID
- **topicName** トピック名へのポインター
- **reg_ctx** ユーザーコンテキストへのポインター

Return: >= 0 受け入れることを表します

3.1.5.11 typedef MqttClient

```
typedef struct _MqttClient MqttClient;
```

3.1.6 Functions Documentation

3.1.6.1 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
)
```

MqttClient 構造体を初期化します。

Parameters:

- **client** MqttClient 構造体へのポインター（未初期化で可）
- **net** コールバック関数とコンテキストへのポインターが格納された MqttNet 構造体へのポインター
- **msg_cb** メッセージ受信コールバック関数へのポインター
- **tx_buf** エンコード用送信バッファーへのポインター
- **tx_buf_len** エンコード用送信バッファーの最大サイズ
- **rx_buf** デコード用受信バッファーへのポインター
- **rx_buf_len** デコード用受信バッファーの最大サイズ
- **cmd_timeout_ms** 最大コマンド待ち時間（ミリ秒）

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_BAD_ARG

(enum MqttPacketResponseCodes を参照)

3.1.6.2 function MqttClient_DeInit

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

MqttClient 構造体にアロケートされたリソースをクリーンアップします。

Parameters:

- **client** MqttClient 構造体へのポインター

3.1.6.3 function MqttClient_SetDisconnectCallback

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

disconnect コールバック関数とユーザーコンテキストをセットします。

Parameters:

- **client** MqttClient 構造体へのポインター（未初期化で可）
- **discb** disconnect コールバック関数へのポインター
- **ctx** ユーザーコンテキストへのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_BAD_ARG

(enum MqttPacketResponseCodes を参照)

3.1.6.4 function MqttClient_SetPropertyCallback

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

property コールバック関数とユーザーコンテキストをセットします。

Parameters:

- **client** MqttClient 構造体へのポインター（未初期化で可）
- **propCb** property コールバック関数へのポインター
- **ctx** ユーザーコンテキストへのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_BAD_ARG

(enum MqttPacketResponseCodes を参照)

3.1.6.5 function MqttClient_Connect

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

MQTT Connect パケットをエンコードして送信し、Connect Acknowledgment パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **connect** connect パラメータを与えられた MqttConnect 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.6 function MqttClient_Publish

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

MQTT Publish パケットをエンコードして送信し、Publish response を待ちます (QoS > 0 の場合)。ペイロードのサイズがバッファーサイズより大きい場合には、ペイロード全体を送信するまでこの関数を連続して複数回呼び出すことができます (QoS > 0 の場合)。

Parameters:

- **client** MqttClient 構造体へのポインター
- **publish** メッセージデータがセットされた MqttPublish 構造体へのポインター

注: MqttPublish と MqttMessage は同じ構造体です。

See:

- MqttClient_Publish_WriteOnly
- MqttClient_Publish_ex

Return: MQTT_CODE_SUCCESS, MQTT_CODE_CONTINUE (for non-blocking) または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待ってブロッキングしますが、ノンブロッキングの場合にはタイムアウトするか MQTT_CODE_CONTINUE を返します。QoS level が 1 の場合には PUBLISH_ACK を待ちます。QoS level が 2 の場合には PUBLISH_REC を待ちその後、PUBLISH_REL とを送信し、PUBLISH_COMP を待ちます。

3.1.6.7 function MqttClient_Publish_ex

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

MQTT Publish パケットをエンコードして送信し、Publish response を待ちます (QoS > 0 の場合)。コールバック関数は送信バッファーより大きなサイズのペイロードデータを送信バッファーにコピーすることに使用されます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **publish** メッセージデータがセットされた MqttPublish 構造体へのポインター。注: MqttPublish と MqttMessage は同じ構造体です。
- **pubCb** コールバック関数へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待ってブロッキングしますが、ノンブロッキングの場合にはタイムアウトするか MQTT_CODE_CONTINUE を返します。QoS level が 1 の場合には PUBLISH_ACK を待ちます。QoS level が 2 の場合には PUBLISH_REC を待ちその後、PUBLISH_REL とを送信し、PUBLISH_COMP を待ちます。

3.1.6.8 function MqttClient_Publish_WriteOnly

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

MqttClient_Publish_ex と同様の機能ですが, 本関数はデータの書き込みだけを行い、他のスレッドで MqttClient_WaitMessage_ex を呼び出して read ACK のハンドリングを行うことを必要としてます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **publish** メッセージデータがセットされた MqttPublish 構造体へのポインター

注: MqttPublish と MqttMessage は同じ構造体です。* **pubCb** コールバック関数へのポインター

See:

- MqttClient_Publish
- MqttClient_Publish_ex

- MqttClient_WaitMessage_ex

Return: MQTT_CODE_SUCCESS, MQTT_CODE_CONTINUE (for non-blocking) または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待ってブロッキングしますが、ノンブロッキングの場合にはタイムアウトするか MQTT_CODE_CONTINUE を返します。QoS level が 1 の場合には PUBLISH_ACK を待ちます。QoS level が 2 の場合には PUBLISH_REC を待ちその後、PUBLISH_REL とを送信し、PUBLISH_COMP を待ちます。

3.1.6.9 function MqttClient_Subscribe

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

MQTT Subscribe パケットをエンコードして送信し、Subscribe Acknowledgment パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **subscribe** トピックリストと QoS を与えられた MqttSubscribe 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.10 function MqttClient_Unsubscribe

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

MQTT Unsubscribe パケットをエンコードして送信し、Unsubscribe Acknowledgment パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **unsubscribe** トピックリストを与えられた MqttUnsubscribe 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.11 function MqttClient_Ping

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

MQTT Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.12 function MqttClient_Ping_ex

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

MQTT Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。この ex バージョンでは、MqttPing 構造体を引数にとり、ノンブロッキングアプリケーションで使用できます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **ping** MqttPing 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.13 function MqttClient_Auth

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

MQTT Authentication Request パケットをエンコードして送信し、Ping Response パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **auth** MqttAuth 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.14 function MqttClient_PropsAdd

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

新しいプロパティを追加します。プロパティ構造体をアロケートし、head で指すリストヘッドに追加します。パケットコマンドに先だって呼び出す必要があります。

Parameters:

- **head** プロパティ構造体へのポインターのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_BAD_ARG

3.1.6.15 function MqttClient_PropsFree

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

プロパティリストを解放します。head で指すリストを解放します。MqttClient_Prop_Add を使ったパケットコマンドの使用後に呼び出さなければなりません。

Parameters:

- **head** プロパティ構造体へのポインターのポインター

Return: MQTT_CODE_SUCCESS または -1 （エラー発生時。errno にもセットされます）

3.1.6.16 function MqttClient_Disconnect

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

MQTT Disconnect パケットをエンコードして送信します (response はありません)。

Parameters:

- **client** MqttClient 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.write を使って送信を試みるノンブロッキング関数です。

3.1.6.17 function MqttClient_Disconnect_ex

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

MQTT Disconnect パケットをエンコードして送信します (response はありません)。

Parameters:

- **client** MqttClient 構造体へのポインター
- **disconnect** MqttDisconnect 構造体へのポインター。NULL を指定しても可。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.write を使って送信を試みるノンブロッキング関数です。

3.1.6.18 function MqttClient_WaitMessage

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

パケット受信を待ちます。受信する publish メッセージは MqttClient_Init に渡したコールバック関数を介して受信します。

Parameters:

- **client** MqttClient 構造体へのポインター
- **timeout_ms** 受信タイムアウト時間（ミリ秒）

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.19 function MqttClient_WaitMessage_ex

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

パケット受信を待ちます。受信する publish メッセージは MqttClient_Init に渡したコールバック関数を介して受信します。

Parameters:

- **client** MqttClient 構造体へのポインター
- **msg** MqttObject 構造体へのポインター
- **timeout_ms** 受信タイムアウト時間（ミリ秒）

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.20 function MqttClient_CancelMessage

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

マルチスレッド環境でノンブロッキングモード時に、msg で指定した MQTT オブジェクトをキャンセルします。

Parameters:

- **client** MqttClient 構造体へのポインター
- **msg** MqttObject 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です

3.1.6.21 function MqttClient_NetConnect

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

```
    MqttTlsCb cb
)
```

TLS 接続を実行し、MqttTlsCb コールバック関数を呼び出します (use_tls が非ゼロの場合)。

Parameters:

- **client** MqttClient 構造体へのポインター
- **host** ブローカーアドレス
- **port** ポート番号（オプション)。ゼロが指定された場合はデフォルトのポート番号（8883）を使用。
- **use_tls** 非ゼロの値が指定された場合には TLS 接続を行う。
- **cb** TLS 接続コールバック関数。コールバック関数は SSL コンテキストこコンフィグレーションや証明書チェックの為に呼び出します。
- **timeout_ms** 受信タイムアウト時間（ミリ秒）

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

3.1.6.22 function MqttClient_NetDisconnect

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

TLS 接続を切断します。

Parameters:

- **client** MqttClient 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_* (enum MqttPacketResponseCodes を参照)

3.1.6.23 function MqttClient_GetProtocolVersion

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

プロトコルバージョン番号を取得します。

Parameters:

- **client** MqttClient 構造体へのポインター

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

3.1.6.24 function MqttClient_GetProtocolVersionString

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

プロトコルバージョン文字列を取得します。

Parameters:

- **client** MqttClient 構造体へのポインター

Return: String v3.1.1 or v5

3.1.6.25 function MqttClient_ReturnCodeToString

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

WOLFMQTT_API からの戻り値を文字列に変換します。

Parameters:

- **return_code** WOLFMQTT_API からの戻り値

Return: String 戻り値の文字列表現

3.1.6.26 function SN_Client_SearchGW

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

ゲートウェイを検索するメッセージをエンコードして送信し、ゲートウェイ情報応答メッセージを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **search** hop radius が与えられた SN_SearchGW 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.27 function SN_Client_Connect

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

Connect パケットをエンコードして送信し、Connect Acknowledge パケットを待ちます。もし、Will（遺言）がイネーブルとなっている場合には、ゲートウェイは LWT トピックとメッセージを送信するように別パケットで促してきます。空の will トピックを指定すると will トピックと will メッセージをサーバーから削除するものとみなします。the Will message stored in the server.

Parameters:

- **client** MqttClient 構造体へのポインター
- **connect** 接続パラメータが与えられた SN_Connect 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.28 function SN_Client_WillTopicUpdate

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

MQTT-SN Will Topic Update パケットをエンコードして送信します。will 引数として NULL を渡すと、Will トピックとサーバーに保持されている Will メッセージを削除することを意味します。

Parameters:

- **client** MqttClient 構造体へのポインター
- **will** トピックとメッセージパラメータが与えられた SN_Will 構造体へのポインター。NULL 指定も可。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.29 function SN_Client_WillMsgUpdate

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

MQTT-SN Will Message Update パケットをエンコードして送信します。

Parameters:

- **client** MqttClient 構造体へのポインター
- **will** トピックとメッセージパラメータが与えられた SN_Will 構造体へのポインター。NULL 指定でも可。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.30 function SN_Client_Register

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

MQTT-SN Register パケットをエンコードして送信し、Register Acknowledge パケットを待ちます。クライアントがゲートウェイに送信する Register パケットは、含まれているトピック名に対してトピック ID を割り当てるように要求する目的で送信されます。また、ゲートウェイからクライアントに送信される Register パケットはトピック名に対してトピック ID が割り当てられたことを通知する目的で送信されます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **regist** SN_Register 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.31 function SN_Client_SetRegisterCallback

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

カスタム（ユーザー）コンテキストと共にレジスターコールバック関数をセットします。

Parameters:

- **client** MqttClient 構造体へのポインター（未初期化で可）
- **regCb** レジスターコールバック関数へのポインター
- **ctx** ユーザーコンテキストへのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_BAD_ARG

3.1.6.32 function SN_Client_Publish

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

MQTT-SN Publish パケットをエンコードして送信し Publish response を待ちます (QoS > 0 の場合)。

Parameters:

- **client** MqttClient 構造体へのポインター
- **publish** メッセージデータを与えられた SN_Publish 構造体へのポインター。注: SN_Publish と MqttMessage は同じ構造体です。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。 QoS level が 1 の場合には PUBLISH_ACK を待ちます。 QoS level が 2 の場合には PUBLISH_REC を待ちその後、PUBLISH_REL を送信し、PUBLISH_COMP を待ちます。

3.1.6.33 function SN_Client_Subscribe

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

MQTT-SN Subscribe パケットをエンコードして送信し、割り当てられたトピック ID を含む Subscribe Acknowledgment パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **subscribe** トピックリストと QoS レベルを与えられた SN_Subscribe 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.34 function SN_Client_Unsubscribe

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

MQTT-SN Unsubscribe パケットをエンコードして送信し、Unsubscribe Acknowledgment パケットを待ちます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **unsubscribe** トピック ID が与えられた SN_Unsubscribe 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.35 function SN_Client_Disconnect

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

MQTT-SN Disconnect パケットをエンコードして送信します。その際に、クライアントはスリープ状態に入るまでの時間を送信できます。

Parameters:

- **client** MqttClient 構造体へのポインター

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.write を呼び出して待つブロッキング関数です。

3.1.6.36 function SN_Client_Disconnect_ex

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

MQTT-SN Disconnect パケットをエンコードして送信します。その際に、クライアントはスリープ状態に入るまでの時間を送信できます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **disconnect** SN_Disconnect 構造体へのポインター。NULL 指定も可。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.write を呼び出して待つブロッキング関数です。

3.1.6.37 function SN_Client_Ping

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

MQTT-SN Ping Request パケットをエンコードして送信し、Ping Response パケットを待ちます。クライアントがスリープ状態の場合でゲートウェイに起床状態に遷移したことを通知したい場合にはクライアント ID を ping request に追加する必要があります。

Parameters:

- **client** MqttClient 構造体へのポインター
- **ping** SN_PingReq 構造体へのポインター。NULL 指定も可。

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.38 function SN_Client_WaitMessage

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

パケット受信を待ちます。受信した Publish メッセーは MqttClient_Init で与えられたコールバック関数を介して通知されます。

Parameters:

- **client** MqttClient 構造体へのポインター
- **timeout_ms** ミリ秒の受信タイムアウト時間

Return: MQTT_CODE_SUCCESS または MQTT_CODE_ERROR_*

(enum MqttPacketResponseCodes を参照)

Note: この関数は MqttNet.read で待つブロッキング関数です。

3.1.6.39 function SN_Client_WaitMessage_ex

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

3.1.7 Attributes Documentation

3.1.7.1 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"


#ifndef WOLFMQTT_USE_CB_ON_DISCONNECT
    #undef WOLFMQTT_USE_CB_ON_DISCONNECT

#endif

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

struct _MqttClient;
```

3.1.8 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"

/* This macro allows the disconnect callback to be triggered when
 * MqttClient_Disconnect_ex is called. Normally the CB is only used to handle
 * errors from MqttPacket_HandleNetError.
 */
#ifndef WOLFMQTT_USE_CB_ON_DISCONNECT
    #undef WOLFMQTT_USE_CB_ON_DISCONNECT
    /* #define WOLFMQTT_USE_CB_ON_DISCONNECT */
#endif

#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 */
```

3.2 header-ja/mqtt_socket.h

3.2.1 Classes

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

3.2.2 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**	

3.2.3 Functions

	Name
WOLFMQTT_LOCAL int	**MqttSocket_Init(struct _MqttNet * net)
WOLFMQTT_LOCAL int	**MqttSocket_Write(struct _MqttNet * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Read(struct _MqttNet * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Peek(struct _MqttNet * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Connect(struct _MqttNet * 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)

3.2.4 Attributes

Name
C

3.2.5 Types Documentation

3.2.5.1 typedef MqttTlsCb

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

3.2.5.2 typedef MqttNetConnectCb

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

3.2.5.3 typedef MqttNetWriteCb

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

3.2.5.4 typedef MqttNetReadCb

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

3.2.5.5 typedef MqttNetPeekCb

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

3.2.5.6 typedef MqttNetDisconnectCb

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

3.2.5.7 typedef MqttTls

```
typedef struct _MqttTls MqttTls;
```

3.2.5.8 typedef MqttNet

```
typedef struct _MqttNet MqttNet;
```

3.2.6 Functions Documentation

3.2.6.1 function MqttSocket_Init

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

3.2.6.2 function MqttSocket_Write

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

3.2.6.3 function MqttSocket_Read

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

3.2.6.4 function MqttSocket_Peek

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

3.2.6.5 function MqttSocket_Connect

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

3.2.6.6 function MqttSocket_Disconnect

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

3.2.6.7 function MqttSocket_TlsSocketReceive

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

3.2.6.8 function MqttSocket_TlsSocketSend

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

3.2.7 Attributes Documentation

3.2.7.1 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;
```

3.2.8 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 */
```

3.3 header-ja/mqtt_socket.h

3.3.1 Classes

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

3.3.2 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**
typedef int()(*void* context, byte *buf, int buf_len, int timeout_ms)	**MqttNetPeekCb**
typedef int()(*void* context)	**MqttNetDisconnectCb**
typedef struct _MqttTls**	

	Name
typedef struct _MqttNet**	

3.3.3 Functions

	Name
WOLFMQTT_LOCAL int	**MqttSocket_Init * net)
WOLFMQTT_LOCAL int	**MqttSocket_Write * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Read * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Peek * buf, int buf_len, int timeout_ms)
WOLFMQTT_LOCAL int	**MqttSocket_Connect 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)

3.3.4 Attributes

Name
C

3.3.5 Types Documentation

3.3.5.1 typedef MqttTlsCb

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

3.3.5.2 typedef MqttNetConnectCb

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

3.3.5.3 typedef MqttNetWriteCb

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

3.3.5.4 typedef MqttNetReadCb

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

3.3.5.5 typedef MqttNetPeekCb

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

3.3.5.6 typedef MqttNetDisconnectCb

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

3.3.5.7 typedef MqttTls

```
typedef struct _MqttTls MqttTls;
```

3.3.5.8 typedef MqttNet

```
typedef struct _MqttNet MqttNet;
```

3.3.6 Functions Documentation

3.3.6.1 function MqttSocket_Init

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

3.3.6.2 function MqttSocket_Write

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

3.3.6.3 function MqttSocket_Read

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

3.3.6.4 function MqttSocket_Peek

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

3.3.6.5 function MqttSocket_Connect

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

3.3.6.6 function MqttSocket_Disconnect

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

3.3.6.7 function MqttSocket_TlsSocketReceive

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

3.3.6.8 function MqttSocket_TlsSocketSend

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

3.3.7 Attributes Documentation

3.3.7.1 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;
```

3.3.8 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 */
```

3.4 header-ja/mqtt_types.h

3.4.1 Classes

	Name
struct	**wm_Sem**

3.4.2 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**

	Name
typedef unsigned char	**byte**
typedef unsigned short	**word16**
typedef unsigned int	**word32**

3.4.3 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, ...)

3.4.4 Attributes

Name
C

3.4.5 Types Documentation

3.4.5.1 enum MqttPacketResponseCodes

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	

3.4.5.2 typedef wm_Sem

```
typedef HANDLE wm_Sem;
```

3.4.5.3 typedef byte

```
typedef unsigned char byte;
```

3.4.5.4 typedef word16

```
typedef unsigned short word16;
```

3.4.5.5 typedef word32

```
typedef unsigned int word32;
```

3.4.6 Functions Documentation

3.4.6.1 function wm_SemInit

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

3.4.6.2 function wm_SemFree

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

3.4.6.3 function wm_SemLock

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

3.4.6.4 function wm_SemUnlock

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

3.4.6.5 function SYS_CMD_PRINT

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

3.4.7 Attributes Documentation

3.4.7.1 variable C

```
C;
```

3.4.8 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) || defined(BUILDING_CMAKE))
        #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 */
```