MQTT Client Library Encyclopedia

–

MQTT-Client Framework

Thanks for participating in the **MQTT Client Library Encyclopedia** Blog post series.

The following is the template you need to fill out in order to participate. A blog post for this series has essentially the following structure

* General Information
* Installation
* Usage
* A full example
* Details about the author

We included information to every passage of the template you might find helpful. The passages where you need to add your own texts are marked with yellow. (like this)

# General Information

This Section contains general information about the client library.

## Short Profile

|  |  |  |
| --- | --- | --- |
| ***Profile Information*** | ***Client library Info*** | ***Example*** |
| **Language** | Objective-C | COBOL |
| **License** | Eclipse Public License - v 1.0 | Apache License |
| **Website** | <https://github.com/ckrey/MQTT-Client-Framework> | <http://www.google.com> |
| **API Style** | Asynchronous and Blocking | Blocking, Asynchronous, Future-Based, ….. |

## Description

Please add a short description about the client library you’re writing about. This passage is intended for users who do not know about the library yet to get a quick overview if it’s worth reading more about the library. Please add any information you want to include but bear in mind, that brief passages are more likely to get read by users. This passage could include e.g. the following information:

* Motivation: Why does this library exist, which problem does it solve? What’s special about it?
* History: How long does the library exist? What’s the history of the library?
* Quality: Is this library production ready? Is it more suited for play projects than 24/7 production critical systems? Is it actively maintained?
* Community: Does it have an active community? Is the development open and can people contribute to the library?
* Any other information you want to include to give a good overview of the library

**MQTT-Client-Framework** is a native Objective-C iOS library. It uses CFNetwork for networking and CoreData for persistence. It is a complete implementation of MQTT 3.1.1 and supports TLS. Three flavors of API are available:A simple blocking API, an elaborate low level callback based API as well as a session manager which takes care of the iOS specific app life cycle.

When I started in 2013 developing the library based on the work of 2lemetry, there was no iOS MQTT client library existing which met the criteria for the OwnTracks project, especially TLS support and protocol completeness. Today there are some Objective-C and Swift wrappers for libmosquitto and Swift native implementations available as open source.

MQTT-Client-Framework is actively maintained and used in various projects. Contributions are welcome. It integrates well with Swift. The extensive test cases cover a huge percentage of MQTT 3.1.1 test cases for MQTT brokers.

## Features

This gives an overview about the supported features of the library.

Please fill out every row of the table. You only have to fill in **YES / NO** to the *Supported?* Column.

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***Supported?*** | ***What does this mean?*** |
| **MQTT 3.1** | YES | Is MQTT 3.1 supported by your library? |
| **MQTT 3.1.1** | YES | Is MQTT 3.1.1 supported by your library? |
| **QoS 0** | YES | Is Quality of Service 0 supported by your library? |
| **QoS 1** | YES | Is Quality of Service 1 supported by your library? |
| **QoS 2** | YES | Is Quality of Service 2 supported by your library? |
| **LWT** | YES | Has your library Last-Will-and-Testament support? |
| **SSL / TLS** | YES | Is SSL / TLS supported by your library? |
| **Authentication** | YES | Does your library support Authentication (Username /Password) when connecting to a MQTT broker? |
| **Automatic Reconnect** | YES | Does your library support automatic reconnects after the client lost connection to the broker? |
| **Throttling** | YES | Throttling means that the user can specify a maximum up / downstream limit and/or can limit the messages/sec |

# Usage

This section is meant to give an **overview of the basic MQTT functionality** of the library on an API level. Readers should get a grasp how the library usage “feels”. Naturally this section will contain a lot of source code. So please only add minimal examples and don’t add anything that doesn’t belong to the section, otherwise readers might get confused.

It’s also important that you add a **description** to every code snippet, which explains what is going on, especially if it’s not obvious. Remember, people may read the post who are not familiar with your library or even the programming language the library uses.

## Installation

There are 3 options to use MQTT-Client-Framework:

* Use CocoaPods  
  The simplest way is to add „MQTTClient“ to your Podfile and run „pod update“
* Include Framework  
  After cloning the repo from github <https://github.com/ckrey/MQTT-Client-Framework> you may add the iOS framework included under „…/MQTTClient/dist/MQTTClient.framework“ to your iOS project.
* Include Source  
  Include Framework  
  After cloning the repo from github <https://github.com/ckrey/MQTT-Client-Framework> you may add the sources under „…/MQTTClient/MQTTClient“ directly to your iOS project.

Doxygen generated documentation is part of the repo under …/MQTTClient/dist/documenation/html“

## Connect

This section shows the API usage how to connect with the library to a MQTT broker.

#import "MQTTClient.h"

MQTTSession \*session = [[MQTTSession alloc] init];

[session connectToHost:@"192.168.0.1" port:1883 usingSSL:NO];

## Connect with MQTT 3.1 or MQTT 3.1.1 (optional)

This section shows the API usage how to connect with the library to a MQTT broker with either MQTT 3.1 or MQTT 3.1.1. If your library only supports one MQTT version, you can omit this section.

session.protocolLevel = 4;

[session connectToHost:@"192.168.0.1" port:1883 usingSSL:NO];

**protocolLevel**

specifies the protocol to be used. The value of the Protocol Level field for the version 3.1.1 of the protocol is 4. The value for the version 3.1 is 3.

## Connect with LWT (optional)

This section shows the API usage how to connect with the library to a MQTT broker with the Last-Will-and-Testament feature enabled. If your library does not support that feature, you can omit this section.

session.willFlag = TRUE;

session.willTopic = @"example/status";

session.willMsg = [@"offline" dataUsingEncoding:NSUTF8StringEncoding];

session.willQoS = MQTTQosLevelExactlyOnce;

[session connectToHost:@"192.168.0.1" port:1883 usingSSL:NO];

**willFlag**

If the Will Flag is set to YES this indicates that a Will Message MUST be published by the Server when the Server detects that the Client is disconnected for any reason other than the Client flowing a DISCONNECT Packet.

**willTopic**

If the Will Flag is set to YES, the Will Topic is a string, nil otherwise.

**willMsg**

If the Will Flag is set to YES the Will Message must be specified, nil otherwise.

**willQoS**

specifies the QoS level to be used when publishing the Will Message. If the Will Flag is set to NO, then the Will QoS MUST be set to 0. If the Will Flag is set to YES, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02).

## Connect with Username / Password (optional)

This section shows the API usage how to connect with the library to a MQTT broker with username / password. If your library does not support that feature, you can omit this section.

session.userName = @"myname";

session.password = @"secret";

[session connectToHost:@"192.168.0.1" port:1883 usingSSL:NO];

**userName**

an NSString object containing the user's name (or ID) for authentication. May be nil.

**password**

an NSString object containing the user's password. If userName is nil, password must be nil as well.

## Publish

This section shows the API usage how to publish a MQTT message to a MQTT broker. Feel free to show how to publish QoS 0, 1 and 2.

[session publishData:[@"Sample Data" dataUsingEncoding:NSUTF8StringEncoding]

topic:@"example/data"

retain:NO

qos:MQTTQosLevelAtMostOnce];

## Publish a retained message

This section shows the API usage how to publish a retained MQTT message to a MQTT broker.

[session publishData:[@"Sample Data" dataUsingEncoding:NSUTF8StringEncoding]

topic:@"example/data"

retain:YES

qos:MQTTQosLevelExactlyOnce];

## Subscribe

This section shows the API usage how to subscribe to MQTT messages. Depending on the API style, you may want to also show the callback code.

[session subscribeToTopic:@"example/#" atLevel:MQTTQosLevelExactlyOnce];

## Unsubscribe

This section shows the API usage how to unsubscribe to MQTT topics. Depending on the API style, you may want to also show the callback code.

[session unsubscribeTopic:@"example/#"];

## Disconnect

This section shows the API usage how to disconnect from a MQTT broker.

[session close];

## Using SSL / TLS (optional)

This section shows how to enable SSL / TLS for a secure connection to the MQTT broker.

* You may use iOS’ keyChain to validate server certificates:

[session connectToHost:@"192.168.0.1" port:8883 usingSSL:YES];

* Or you may use a custom security policy:

NSString\* certificate = [[NSBundle bundleForClass:[MQTTSession class]] pathForResource:@"certificate" ofType:@"cer"];

session.securityPolicy = [MQTTSSLSecurityPolicy policyWithPinningMode:MQTTSSLPinningModeCertificate];

session.securityPolicy.pinnedCertificates = @[ [NSData dataWithContentsOfFile:certificate] ];

session.securityPolicy.allowInvalidCertificates = YES;

[session connectToHost:@"192.168.0.1" port:8883 usingSSL:YES];

* And you may use client certificates:

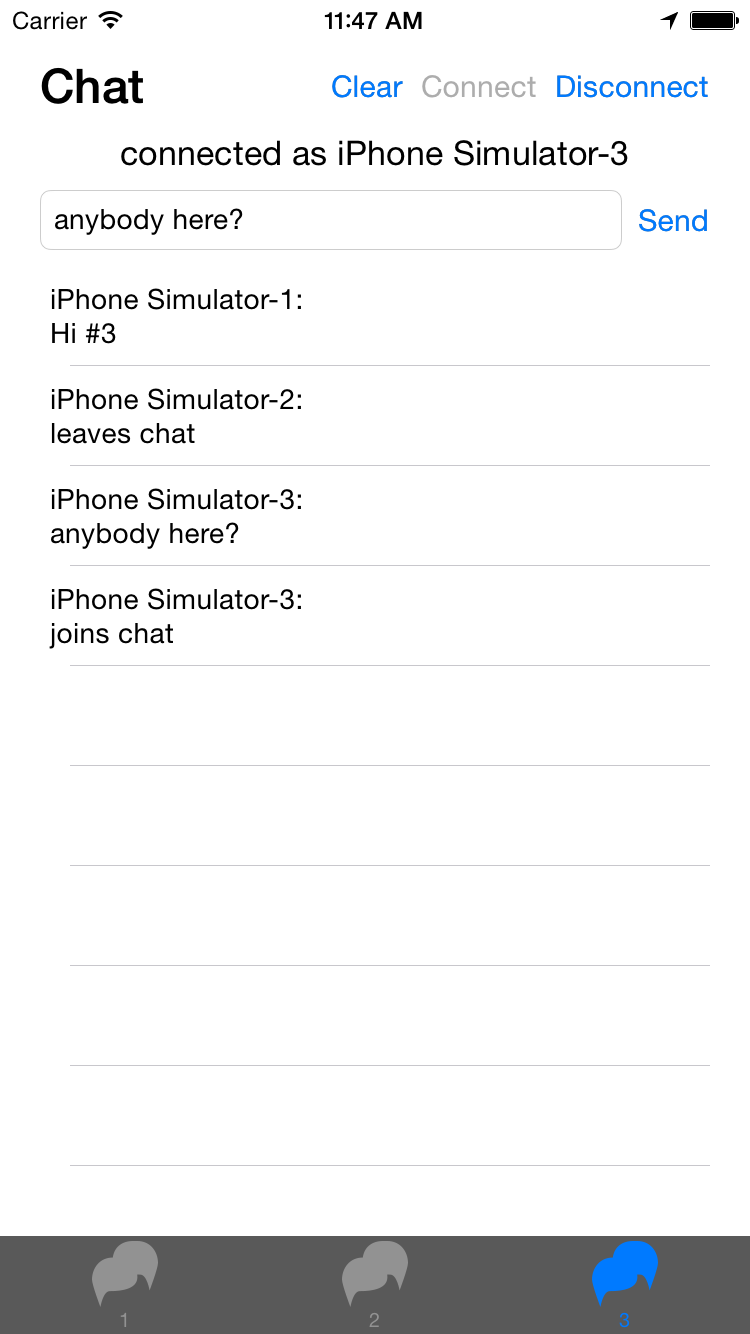
NSString \*path = [[NSBundle bundleForClass:[MQTTClientTests class]] pathForResource:parameters[@„client"] ofType:@"p12"];

session.certificates = [MQTTSession clientCertsFromP12:path passphrase:@„secret“];

[session connectToHost:@"192.168.0.1" port:8883 usingSSL:YES];

# Full Example application MQTTChat

You find a full iPhone application MQTTChat in github at <https://github.com/ckrey/MQTTChat>



MQTTChat implements a chat protocol over MQTT on an MQTT broker of your choice.

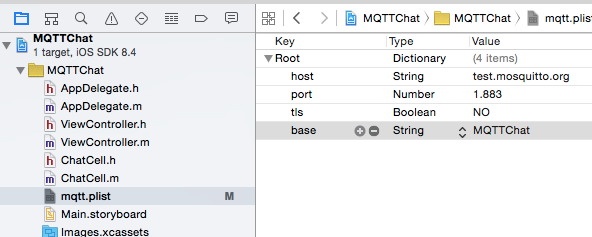
The application presents a tab view with 3 tabs to chat within the app or you may use the app on different devices. A separate MQTT session is used in each tab.

The app CONNECTs to the MQTT broker and sends a „join“ message when the tab is first opened. The MQTT connection can be gracefully DISCONNECTed (after sending a „leave“ message) and reCONNECTed on user request.

Simple text messages can be entered and send and will be displayed on all other connected clients and tabs).

The app publishes the text messages as UTF8 encoded strings to the MQTT broker to the topic „<base>/<devicename>-<tabnumber>“, e.g. „MQTTChat/iPhone Simulator-3“. Messages are not retained. Each session subscribes to the topic „<base>/#“, e.g. „MQTTChat/#“. Both PUBLISH and SUBSCRIBE use QoS 2.

To run the app, you need Xcode 6 and CocoaPods (<https://cocoapods.org>). You may change the connect parameters for your MQTT broker by updating the „mqtt.plist“ file.



# Author Information

Please add the following:

* Christoph Krey
* krey.christoph@gmail.com
* OwnTracks UG (haftungsbeschränkt)
* https://github.com/ckrey
* Christoph Krey is a freelance network application specialist who designs and implements client/server solutions in corporate environments since 1983. After a career in IT Management, he began his focus on protocols, mobile devices, and distributed applications. Christoph Krey maintains the MQTT client framework and has created MQTTInspector and OwnTracks for iOS.
* Picture: