Menu

Overview

Paho MQTT C Client - MQTT Client Library Encyclopedia

Related Links

HiveMQ Product Info

Get HiveMQ

Client Library

Written by Ian Craggs

Category: MQTT • MQTT Client • MQTT Client Library

Published: September 21, 2015

Short info

Paho MQTT C Client	
Language	C
License	EPL and EDL
Website	**eclipse.org/paho/clients/c**
API Style	Blocking and non-blocking

Description

The Paho C client libraries started life back in 2007, when I first started writing a small **MQTT** server, RSMB (Really Small Message Broker). I thought I would reuse as much of the internal code of RSMB as I could, to save myself some time. As it turned out, I probably didn't save as much time as I expected, because in some ways writing a client library for **MQTT** is more complex than writing a server. RSMB is single-threaded, which I mainly retained in the design of these client libraries, for better or worse.

I started writing RSMB in C++, but the template support in gcc at the time did not work well, so I switched to standard ANSI C. That's why the client libraries are C too. It seemed to me at the time that Linux was likely to take over the world for embedded systems, with Windows CE maybe getting a look in. That's why I wrote these libraries with Linux (and some other forms of Unix) and Windows in mind, rather than being any more portable. And that's why the Paho embedded client libraries now exist - for all those other embedded operating systems that are now popular!

In 2008, there weren't too many MQTT client libraries to learn from, so for the first steps I followed the IBM Java client of the time. I thought it was better to copy an existing model to make it easier for the MQTT application programmer to transfer knowledge from one to the other. As a result the synchronous client was born, just called MQTTClient. This has the following design points:

1. threading: no background thread, or just one, regardless of how many connections are created
2. blocking MQTT calls, to simplify application programming
3. an inflight window for QoS 1 and 2 messages of 1 or 10 only, to limit the amount of damage that can be inflicted on a server
4. internal tracing and memory tracking, for serviceability and elimination of memory leaks

Features

Feature	
MQTT 3.1	✓
MQTT 3.1.1	✓
LWT	✓
SSL/TLS	✓
Automatic Reconnect	✗

Feature	
QoS 0	
QoS 1	
QoS 2	
Authentication	
Throttling	
Offline Message	

Feature	
Disk Persistence	✓

Message	
Feature	
Buffering	

Usage

Installation

On Linux (or Unix) the simplest method is to clone the repo and install:

```
git clone https://git.eclipse.org/r/paho/or
make
sudo make install
```

Pre-built libraries for MacOS and Windows are available on the **Paho downloads page**.

Unzip to a location of your choice.

First we have to include the header file.

```
#include "MQTTClient.h"
```

Now we can create a client object.

```
MQTTClient client;
rc = MQTTClient_create(&client, url, client
```

Where the url is of the form **`host:port`**, and **`clientid`** is a **`string`**. The fourth parameter specifies the disk persistence required. **`MQTTCLIENT_PERSISTENCE_NONE`** says that no persistence is required. **`MQTTCLIENT_PERSISTENCE_DEFAULT`** asks for the supplied, default disk persistence to be used. This stores all inflight message data to disk, and means that the application can end, restart, and recreate the client object with the same clientid and url. Any inflight message data will be read from disk and restored so that the application can continue where its previous incarnation left off.

Next we can optionally set some callback functions. At a minimum, we must have a **`messageArrived callback`**, which will be called whenever an MQTT publish message arrives.

```
rc = MQTTClient_setCallbacks(client, contex
```

There is also a **`connectionLost callback`** which will be called whenever the connection to the server has been broken unexpectedly. That is, we have called connect previously, it has succeeded, we have not called disconnect, and now the connection has been broken. Often we simply want to reconnect in the **`connectionLost callback`**. The final callback function is **`deliveryComplete`**, which is called when an MQTT publish exchange finishes. The MQTTClient_publish call blocks, but only until the publish packet is written to the socket. For QoS 0, this is the end of the story. For QoS 1 and

For QoS 0, this is the end of the story. For QoS 1 and 2, the MQTT packet exchange continues. When the final acknowledgement is received from the server, **`deliveryComplete`** is called. **`context`** is a pointer which is passed to the callbacks and can contain any useful information, such as the client object for which this callback is being made.

For this synchronous client, you do have the option of not setting any callbacks. In that case, no background thread will be started. Under these circumstances, the application, on a regular basis, must call

```
rc = MQTTClient_receive(client, topicName,
```

to receive messages, or

```
MQTTClient_yield();
```

to allow any necessary MQTT background processing to take place. This mode was intended specifically when the application wanted no threads to be created.

Application Setup - MQTTAsync

The application setup for the asynchronous client library is very similar to the synchronous library. Header file:

```
#include "MQTTAsync.h"
```

client object creation:

```
MQTTAsync client;
rc = MQTTAsync_create(&client, url, clienti
```

and setting the callbacks:

```
rc = MQTTAsync_setCallbacks(client, context
```

There is no non-threaded mode for the async client, background threads will always be created and the arrival of MQTT publications will be notified by the **`messageArrived callback`**.

Connect

To connect to a server, we create a connect options structure and call connect:

```
MQTTClient_connectOptions conn_opts = MQTTC
conn_opts.keepAliveInterval = 10;
conn_opts.cleansession = 1;
rc = MQTTClient_connect(client, conn_opts);
```

which will block until the MQTT connect is complete, or has failed. Similarly, for the async client:

```
MQTTAsync_connectOptions conn_opts = MQTTAsy
conn_opts.keepAliveInterval = 10;
conn_opts.cleansession = 1;
conn_opts.onSuccess = onConnect;
conn_opts.onFailure = onConnectFailure;
conn_opts.context = client;
rc = MQTTAsync_connect(client, &conn_opts);
```

except that we add two pointers to callback functions. The connect call will not block, success or failure will be notified by the invocation of one or other of the callback functions. For the other connect examples, I’ll just show the synchronous client, but the pattern is the same for the async client. A typical pattern for the async client is to make all MQTT calls in callback functions: a subscribe or publish call can be made in the success callback for connect, for instance.

Connect with LWT

```
MQTTClient_willOptions will_opts = MQTTClie
MQTTClient_connectOptions conn_opts = MQTTC
conn_opts.keepAliveInterval = 10;
conn_opts.cleansession = 1;
conn_opts.will = will_opts;
will_opts.topicName = "will topic";
will_opts.message = "will message";
rc = MQTTClient_connect(client, conn_opts);
```

Connect with Username and Password

```
MQTTClient_connectOptions conn_opts = MQTTC
conn_opts.keepAliveInterval = 10;
conn_opts.cleansession = 1;
conn_opts.username = "username";
conn_opts.password = "password";
rc = MQTTClient_connect(client, conn_opts);
```

Publish

Publish looks like the following, for the synchronous client:

```
char* payload = "a payload";
int payloadlen = strlen(payload);
int qos = 1;
int retained = 0;
MQTTClient_deliveryToken dt;
rc = MQTTClient_publish(client, topicName, 
```

There is also another version of the call which takes a message structure:

```
MQTTClient_message msg = MQTTClient_message_
msg.payload = "a payload";
msg.payloadlen = strlen(payload);
msg.qos = 1;
msg.retained = 0;
MQTTClient_deliveryToken dt;
rc = MQTTClient_publishMessage(client, topi
```

The delivery token can be used in a waitForCompletion call, to synchronize on the completion of the MQTT packet exchange. The async calls for publish look very similar, except I renamed publish to send, and topic to destination.

This was to be compatible with the JavaScript Paho client, because the asynchronous programming models are very similar. It seemed like a good idea at the time!

```
MQTTAsync_responseOptions response;
response.onSuccess = onPublish;
response.onFailure = onPublishFailure;
response.context = client;
rc = MQTTAsync_sendMessage(client, destinat
```

Again, the biggest change is the addition of the success and failure callbacks.

Subscribe

The subscribe call is straightforward:

```
const char* topic = "mytopic";
int qos = 2;
rc = MQTTClient_subscribe(client, topic, qo
```

and for the async client:

```
MQTTAsync_responseOptions opts = MQTTAsync_
opts.onSuccess = onSubscribe;
opts.onFailure = onSubscribeFailure;
opts.context = client;
rc = MQTTAsync_subscribe(client, topic, qos
```

Unsubscribe

The unsubscribe call is also straightforward:

```
const char* topic = "mytopic";
rc = MQTTClient_unsubscribe(client, topic);
```

and for the async client:

```
MQTTAsync_responseOptions opts = MQTTAsync_
opts.onSuccess = onUnsubscribe;
opts.onFailure = onUnsubscribeFailure;
opts.context = client;
rc = MQTTAsync_unsubscribe(client, topic, &
```

Disconnect

There is a timeout parameter in milliseconds on the disconnect call, which allows outstanding MQTT packet exchanges to complete.

```
int timeout = 100;
MQTTClient_disconnect(client, timeout);
```

or:

```
MQTTAsync_disconnectOptions opts = MQTTAsyn
opts.onSuccess = onDisconnect;
opts.context = client;
rc = MQTTAsync_disconnect(client, &opts);
```

When you’ve finished with a client object, call destroy to free up the memory:

```
MQTTClient_destroy(&client);
```

or:

```
MQTTAsync_destroy(&client);
```

Receiving Messages

A typical message arrived callback function might look like this:

```
int messageArrived(void *context, char *to
{
    printf("Message arrived\n");
    printf("   topic: %s\n", topicName);
    printf("   message: .*s\n", message.pa

    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
```

Note the calls to free memory, and the return value: 1 means that the message was received properly, 0 means it wasn't. The **`messageArrived callback`** will be invoked again for the same message if you return 0.

Using TLS / SSL

To use TLS, you need to add the SSL options data to the connect options:

```
MQTTClient_SSLOptions ssl_opts = MQTTClient_
ssl_opts.enableServerCertAuth = 0;
conn_opts.ssl = &ssl_opts;
```

The TLS implementation uses OpenSSL, so the configuration parameters are the same:

```
/** The file in PEM format containing the
const char* trustStore;

/** The file in PEM format containing the
* the client's private key.
*/
const char* keyStore;

/** If not included in the sslKeyStore, th
* the client's private key.
*/
const char* privateKey;
/** The password to load the client's priv
const char* privateKeyPassword;

/**
* The list of cipher suites that the clien
* full explanation of the cipher list form
* http://www.openssl.org/docs/apps/ciphers
* If this setting is ommitted, its default
* those offering no encryption- will be co
* This setting can be used to set an SSL a
*/
const char* enabledCipherSuites;

/** True/False option to enable verificati
int enableServerCertAuth;
```

Example application

Sample application for the Paho C clients are in the samples directory of the **github repository** and in the download packages.

About Ian Craggs

Thanks for this guest blog post by Ian Craggs | IBM

Ian Craggs works for IBM, and has been involved with MQTT for more than 10 years. He wrote the IBM MQTT server Really Small Message Broker which became the inspiration for the Eclipse Mosquitto project. He contributed C client libraries to the Eclipse Paho project at its onset and is now the project leader.

← **Whats New in HiveMQ 3?**

Arduino PubSubClient - MQTT Client Library Encyclopedia →

Keep up to date on HiveMQ

Subscribe to our newsletter for updates on HiveMQ, MQTT, and IoT.

Enter Your Email-Adress*

Submit

By clicking the *subscribe* button, you give your consent to the use of your data according to our **Privacy Policy**.You can withdraw your consent at any time with future effect.

6 Comments

1 Login ▾

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Share Best Newest Oldest

HA

Hardware and networking

4 years ago

Thank you for this blog..This is really very helpful ..I used to prefer your blog since long time for keeping me updates..looking forward for new blogs..
ALL THE BEST

0 0 • Reply • Share ›

SG

Shakti Gupta

6 years ago

Hi,
I am beginner in MQTT. i am using eclipse paho C. i am facing some problem in a sample program. Please have a look into code. i have pasted code and output of publisher and subscriber below. In subscriber, it show some corrupted string and then it lost connection. Please help me to find mistake.

sample_publisher.c

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "MQTTClient.h"

#define ADDRESS "tcp://m2m.eclipse.org:1883"
#define CLIENTID "ExampleClientPub"
#define TOPIC "MQTT Examples"
#define PAYLOAD "Hellooooooo"
```

```
#define QOS 1
#define TIMEOUT 10000L

int main(int argc, char* argv[])
{
MQTTClient client;
MQTTClient_connectOptions conn_opts =
MQTTClient_connectOptions_initializer;
MQTTClient_message pubmsg =
MQTTClient_message_initializer;
MQTTClient_deliveryToken token;
int rc;
int status;
char* str="HELLO";

MQTTClient_create(&client, ADDRESS, CLIENTID,
MQTTCLIENT_PERSISTENCE_NONE, NULL);
conn_opts.keepAliveInterval = 20;
conn_opts.cleansession = 1;

if ((rc = MQTTClient_connect(client, &conn_opts))
!= MQTTCLIENT_SUCCESS)
{
printf("Failed to connect, return code %d\n", rc);
exit(-1);
}
pubmsg.payload = (void*)str;
pubmsg.payloadlen = strlen(str);
pubmsg.qos = QOS;
pubmsg.retained = 0;
printf("lenght is %d", pubmsg.payloadlen);
status=MQTTClient_publishMessage(client, TOPIC,
&pubmsg, &token);
printf("status is %d \n",status);
printf("Waiting for up to %d seconds for publication
of %s\n"
"on topic %s for client with ClientID: %s\n",
(int)(TIMEOUT/1000), PAYLOAD, TOPIC, CLIENTID);
rc = MQTTClient_waitForCompletion(client, token,
TIMEOUT);
printf("Message with delivery token %d delivered\n",
token);
MQTTClient_disconnect(client, 10000);
MQTTClient_destroy(&client);
return rc;
}
```

Output of Sample_publish

Output of Sample_publish

```
./sample_publish
lenght is 5status is 0
Waiting for up to 10 seconds for publication of
Hellooooooo
on topic MQTT Examples for client with ClientID:
ExampleClientPub
Message with delivery token 1 delivered
```

Sample_subscribe.c

```
/*
* sample_subscribe.c
*
* Created on: 22-Sep-2016
* Author: shakti
*/

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "MQTTClient.h"
#include "sample_subscribe.h"

#define ADDRESS "tcp://m2m.eclipse.org:1883"
#define CLIENTID "ExampleClientPub"
#define TOPIC "MQTT Examples"
#define PAYLOAD "Hellooooooo"
#define QOS 1
#define TIMEOUT 10000L

int main(int argc, char* argv[])
{
MQTTClient client;
MQTTClient_connectOptions conn_opts =
MQTTClient_connectOptions_initializer;
int rc;
int ch;
//a. Create an instance of MQTT client
MQTTClient_create(&client, ADDRESS,
CLIENTID,MQTTCLIENT_PERSISTENCE_NONE,
NULL);
//b. Prepare connection options
conn_opts.keepAliveInterval = 20;
conn_opts.cleansession = 1;
MQTTClient_setCallbacks(client, NULL, connlost,
msgarrvd, delivered);
//c. Connect to broker with the connection options
if ((rc = MQTTClient_connect(client, &conn_opts))
!= MQTTCLIENT_SUCCESS)
{
printf("Failed to connect, return code %d\n", rc);
exit(-1);
}
//d. Subscribe interested topics.
printf("Subscribing to topic %s\nfor client %s using
QoS%d\n\n"
"Press Q to quit\n\n", TOPIC, CLIENTID, QOS);
MQTTClient_subscribe(client, TOPIC, QOS);
do
{
ch = getchar();
} while(ch!='Q' && ch != 'q');
//MQTTClient_setCallbacks(client, NULL, connlost,
msgarrvd, delivered);
//e. Disconnect to broker
MQTTClient_disconnect(client, 10000);
//f. Release resources
MQTTClient_destroy(&client);
return rc;
}
```

Sample_subscribe.h

```
#ifndef SAMPLES_SAMPLE_SUBSCRIBE_H_
#define SAMPLES_SAMPLE_SUBSCRIBE_H_

volatile MQTTClient_deliveryToken deliveredtoken;
void delivered(void *context,
MQTTClient_deliveryToken dt)
{
printf("Message with token value %d delivery
confirmed\n", dt);
deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int
topicLen, MQTTClient_message* message)
{
int i;
char* payloadptr;
printf("Message arrived\n");
printf(" topic: %s\n", topicName);
printf(" message: ");
printf("address of message is %u \n",message);
payloadptr = (char*)message->payload;
printf("11111 \n");
printf("message is.... %s",payloadptr);
for(i=0; ipayloadlen; i++)
{
printf("2222 \n");
putchar(*payloadptr++);
printf("33333 \n");
}
putchar('\n');
if(message!=NULL)
{
printf("555 \n");
MQTTClient_freeMessage(&message);
printf("666 \n");
}
//free(topicName);
printf("4444 \n");
return 1;
}

void connlost(void *context, char *cause)
{
printf("\nConnection lost\n");
printf(" cause: %s\n", cause);
}

#endif /* SAMPLES_SAMPLE_SUBSCRIBE_H_ */
```

Output of sample_subscribe

```
./sample_subscribe
Subscribing to topic MQTT Examples
for client ExampleClientPub using QoS1

Press Q to quit

Message arrived
topic: MQTT Examples
message: address of message is 2281704372
11111
message is.... crashedket.c2222
c33333
2222
r33333
2222
a33333
2222
s33333
```

2222
h33333
2222
e33333
2222
d33333

555
666
4444

Connection lost

cause (null)

Thank You.
Shakti Gupta

0 0 • Reply • Share ›

H

Himanshu

🕒 7 years ago

Hi,
Do we have a MQTT Client Library in c programming for Windows CE 6.0

Regards,
Himanshu

0 0 • Reply • Share ›

TH

The HiveMQ Team ➜ **Himanshu**

🕒 7 years ago

I believe the Paho embedded C library (http://www.hivemq.com/blog/... can be integrated with Windows CE. The best way to check this would to reach out to the Paho folks via the mailing list at https://dev.eclipse.org/mai..., they're always very helpful.

Hope this helps,
Dominik from the HiveMQ Team

0 0 • Reply • Share ›

W

Wouter

🕒 7 years ago

thanks for the info

thanks for the info