Mosquitto-PHP Documentation

Release 0.4.0

Michael Maclean

Dec 21, 2017

Contents

This is an extension to allow using the Eclipse Mosquitto™ MQTT client library with PHP.

See the `examples` directory for usage.

CHAPTER 1

Contents

1.1 Overview

1.1.1 Requirements

- PHP 5.3 or newer, including PHP 7+
- libmosquitto 1.2.x or later

1.1.2 Installation

If you've used a pre-built package to install Mosquitto, you need to make sure you have the development headers installed. On Red Hat-derived systems, this is probably called `libmosquitto-devel`, and on Debian-based systems it will be `libmosquitto-dev`.

You may obtain this package using PECL:

```
pecl install Mosquitto-alpha
```

Alternatively, you can use the normal extension build process:

```
phpize
./configure --with-mosquitto=/path/to/libmosquitto
make
make install
```

Then add `extension=mosquitto.so` to your `php.ini`.

The `--with-mosquitto` argument is optional, and only required if your libmosquitto install cannot be found.

1.1.3 General Operation

The underlying library is based on callbacks and event-driven operation. As such, you have to call the `loop()` method of the `Client` frequently to permit the library to handle the messages in its queues. You can use `loopForever()` to ensure that the client handles this itself. Also, you should use the callback functions to ensure that you only attempt to publish after the client has connected, etc. For example, here is how you would correctly publish a QoS=2 message:

```
<?php

$c = new Mosquitto\Client;
$c->onConnect(function() use ($c) {
    $c->publish('mgdm/test', 'Hello', 2);
    $c->disconnect();
});

$c->connect('test.mosquitto.org');

// Loop around to permit the library to do its work
// This function will call the callback defined in `onConnect()`
// and disconnect cleanly when the message has been sent
$c->loopForever();

echo "Finished\n";
```

1.2 Mosquitto\Client

class Mosquitto**Client**

This is the main Mosquitto client.

constant LOG_DEBUG

Identifies a debug-level log message

constant LOG_INFO

Identifies an info-level log message

constant LOG_NOTICE

Identifies a notice-level log message

constant LOG_WARNING

Identifies a warning-level log message

constant LOG_ERR

Identifies an error-level log message

constant SSL_VERIFY_NONE

Used with *setTlsInsecure*. Do not verify the identity of the server, thus making the connection insecure.

constant SSL_VERIFY_PEER

Used with *setTlsInsecure*. Verify the identity of the server.

__construct ([*$id = null*, *$cleanSession = true*])

Construct a new Client instance.

Parameters

- **$id** (*string*) – The client ID. If omitted or `null`, one will be generated at random.

- **$cleanSession** (*boolean*) – Set to `true` to instruct the broker to clean all messages and subscriptions on disconnect. Must be `true` if the `$id` parameter is `null`.

setCredentials (*$username*, *$password*)
Set the username and password to use on connecting to the broker. Must be called before *connect*.

Parameters

- **$username** (*string*) – Username to supply to the broker
- **$password** (*string*) – Password to supply to the broker

setTlsCertificates (*$caPath*[, *$certFile*, *$keyFile*, *$password*])
Configure the client for certificate based SSL/TLS support. Must be called before *connect*. Cannot be used in conjunction with *setTlsPSK*.

Define the Certificate Authority certificates to be trusted (ie. the server certificate must be signed with one of these certificates) using `$caFile`. If the server you are connecting to requires clients to provide a certificate, define `$certFile` and `$keyFile` with your client certificate and private key. If your private key is encrypted, provide the password as the fourth parameter.

Parameters

- **$caPath** (*string*) – Path to the PEM encoded trusted CA certificate files, or to a directory containing them.
- **$certFile** (*string*) – Path to the PEM encoded certificate file for this client. Optional.
- **$keyFile** (*string*) – Path to a file containing the PEM encoded private key for this client. Required if certfile is set.
- **$password** (*string*) – The password for the keyfile, if it is encrypted. If null, the password will be asked for on the command line.

setTlsInsecure (*$value*)
Configure verification of the server hostname in the server certificate. If `$value` is `true`, it is impossible to guarantee that the host you are connecting to is not impersonating your server. Do not use this function in a real system. Must be called before *connect*.

Parameters

- **$value** (*boolean*) – If set to false, the default, certificate hostname checking is performed. If set to `true`, no hostname checking is performed and the connection is insecure.

setTlsOptions (*$certReqs*, *$tlsVersion*, *$ciphers*)
Set advanced SSL/TLS options. Must be called before *connect*.

Parameters

- **$certReqs** (*int*) – Whether or not to verify the server. Can be `Mosquitto\\Client::SSL_VERIFY_NONE`, to disable certificate verification, or `Mosquitto\Client::SSL_VERIFY_PEER` (the default), to verify the server certificate.
- **$tlsVersion** (*string*) – The TLS version to use. If `null`, a default is used. The default value depends on the version of OpenSSL the library was compiled against. Available options on OpenSSL >= 1.0.1 are `tlsv1.2`, `tlsv1.1` and `tlsv1`.
- **$cipers** (*string*) – A string describing the ciphers available for use. See the `openssl ciphers` tool for more information. If `null`, the default set will be used.

setTlsPSK (*$psk*, *$identity*[, *$ciphers*])
Configure the client for pre-shared-key based TLS support. Must be called before *connect*. Cannot be used in conjunction with setTlsCertificates.

Parameters

- **$psk** (*string*) – The pre-shared key in hex format with no leading "0x".
- **$identity** (*string*) – The identity of this client. May be used as the username depending on server settings.
- **$cipers** (*string*) – Optional. A string describing the ciphers available for use. See the `openssl ciphers` tool for more information. If `null`, the default set will be used.

setWill (*$topic*, *$payload*[, *$qos = 0*, *$retain = false*])
Set the client "last will and testament", which will be sent on an unclean disconnection from the broker. Must be called before *connect*.

Parameters

- **$topic** (*string*) – The topic on which to publish the will.
- **$payload** (*string*) – The data to send.
- **$qos** (*int*) – Optional. Default 0. Integer 0, 1, or 2 indicating the Quality of Service to be used.
- **$retain** (*boolean*) – Optional. Default false. If `true`, the message will be retained.

clearWill ()
Remove a previously-set will. No parameters.

setReconnectDelay (*$reconnectDelay*, *$exponentialDelay*, *$exponentialBackoff*)
Control the behaviour of the client when it has unexpectedly disconnected in *Client::loopForever*. The default behaviour if this method is not used is to repeatedly attempt to reconnect with a delay of 1 second until the connection succeeds.

Parameters

- **$reconnectDelay** (*int*) – Set delay between successive reconnection attempts.
- **$exponentialDelay** (*int*) – Set max delay between successive reconnection attempts when exponential backoff is enabled
- **$exponentialBackoff** (*bool*) – Pass `true` to enable exponential backoff

connect (*$host*[, *$port = 1883*, *$keepalive = 60*, *$interface = null*])
Connect to an MQTT broker.

Parameters

- **$host** (*string*) – Hostname to connect to
- **$port** (*int*) – Optional. Port number to connect to. Defaults to 1883.
- **$keepalive** (*int*) – Optional. Number of sections after which the broker should PING the client if no messages have been recieved.
- **$interface** (*string*) – Optional. The address or hostname of a local interface to bind to for this connection.

disconnect ()
Disconnect from the broker. No parameters.

onConnect (*$callback*)
Set the connect callback. This is called when the broker sends a CONNACK message in response to a connection.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$rc** (*int*) – Response code from the broker.
- **$message** (*string*) – String description of the response code.

Response codes are as follows:

Code	Meaning
0	Success
1	Connection refused (unacceptable protocol version)
2	Connection refused (identifier rejected)
3	Connection refused (broker unavailable)
4-255	Reserved for future use

onDisconnect (*$callback*)
Set the disconnect callback. This is called when the broker has received the DISCONNECT command and has disconnected the client.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$rc** (*int*) – Reason for the disconnection. 0 means the client requested it. Any other value indicates an unexpected disconnection.

onLog (*$callback*)
Set the logging callback.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$level** (*int*) – The log message level from the values below
- **$str** (*string*) – The message string.

The level can be one of:

- *Client::LOG_DEBUG*
- *Client::LOG_INFO*
- *Client::LOG_NOTICE*
- *Client::LOG_WARNING*
- *Client::LOG_ERR*

onSubscribe(*$callback*)
Set the subscribe callback. This is called when the broker responds to a subscription request.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$mid** (*int*) – Message ID of the subscribe message
- **$qosCount** (*int*) – Number of granted subscriptions

This function needs to return the granted QoS for each subscription, but currently cannot.

onUnsubscribe(*$callback*)
Set the unsubscribe callback. This is called when the broker responds to a unsubscribe request.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$mid** (*int*) – Message ID of the unsubscribe message

onMessage(*$callback*)
Set the message callback. This is called when a message is received from the broker.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

:param *Message* $message: A *Message* object containing the message data

onPublish(*$callback*)
Set the publish callback. This is called when a message is published by the client itself.

Warning: this may be called before the method *publish* returns the message id, so, you need to create a queue to deal with the MID list.

Parameters

- **$callback** (*callable*) – The callback

The callback should take parameters of the form:

Parameters

- **$mid** (*int*) – the message id returned by *publish*

setMaxInFlightMessages(*$maxInFlightMessages*)
Set the number of QoS 1 and 2 messages that can be "in flight" at one time. An in flight message is part way through its delivery flow. Attempts to send further messages with *publish* will result in the messages being queued until the number of in flight messages reduces.

Set to 0 for no maximum.

Parameters

- **$maxInFlightMessages** (*int*) – The maximum

setMessageRetry (*$messageRetryPeriod*)
Set the number of seconds to wait before retrying messages. This applies to publishing messages with QoS>0. May be called at any time.

Parameters

- **$messageRetryPeriod** (*int*) – The retry period

publish (*$topic*, *$payload*[, *$qos = 0*, *$retain = false*])
Publish a message on a given topic.

Parameters

- **$topic** (*string*) – The topic to publish on
- **$payload** (*string*) – The message payload
- **$qos** (*int*) – Integer value 0, 1 or 2 indicating the QoS for this message
- **$retain** (*boolean*) – If `true`, retain this message

Returns The message ID returned by the broker. **Warning**: the message ID is not unique.

Return type int

subscribe (*$topic*, *$qos*)
Subscribe to a topic.

Parameters

- **$topic** (*string*) – The topic.
- **$qos** (*int*) – The QoS to request for this subscription

Returns The message ID of the subscription message, so this can be matched up in the *onSubscribe* callback.

Return type int

unsubscribe ()
Unsubscribe from a topic.

Parameters

- **$topic** (*string*) – The topic.
- **$qos** (*int*) – The QoS to request for this subscription

Returns the message ID of the subscription message, so this can be matched up in the *onUnsubscribe* callback.

Return type int

loop ([*$timeout = 1000*])
The main network loop for the client. You must call this frequently in order to keep communications between the client and broker working. If incoming data is present it will then be processed. Outgoing commands, from e.g. *publish*, are normally sent immediately that their function is called, but this is not always possible. *loop* will also attempt to send any remaining outgoing messages, which also includes commands that are part of the flow for messages with QoS>0.

Parameters

- **$timeout** (*int*) – Optional. Number of milliseconds to wait for network activity. Pass 0 for instant timeout. Defaults to 1000.

loopForever ([*$timeout = 1000*])
Call loop() in an infinite blocking loop. Callbacks will be called as required. This will handle reconnecting if the connection is lost. Call *disconnect* in a callback to disconnect and return from the loop. Alternatively, call *exitLoop* to exit the loop without disconnecting. You will need to re-enter the loop again afterwards to maintain the connection.

Parameters

- **$timeout** (*int*) – Optional. Number of milliseconds to wait for network activity. Pass 0 for instant timeout. Defaults to 1000.

exitLoop ()
Exit the *loopForever* event loop without disconnecting. You will need to re-enter the loop afterwards in order to maintain the connection.

1.3 Mosquitto\Message

class Mosquitto**Message**
Represents a message received from a broker. All data is represented as properties.

property $topic
(*string*) The topic this message was delivered to.

property $payload
(*string*) The payload of this message.

property $mid
(*int*) The ID of this message.

property $qos
(*int*) The QoS value applied to this message.

property $retain
(*boolean*) Whether this is a retained message or not.

This class has two static methods.

static topicMatchesSub
Returns true if the supplied topic matches the supplied description, and otherwise false.

Parameters

- **$topic** (*string*) – The topic to match
- **$subscription** (*string*) – The subscription to match

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

Parameters

- **$topic** (*string*) – The topic to tokenise

1.4 Mosquitto\Exception

class Mosquitto**Exception**

This is an exception that may be thrown by many of the operations in the `Client` object. It does not add any features to the standard PHP `Exception` class.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

PHP Namespace Index

m

Index