API Documentation

Library version: 2.8

Constructor

- **PubSubClient** ()
- **PubSubClient** (client)
- **PubSubClient** (server, port, [callback], client, [stream])

Function

- boolean **connect** (clientID, [username, password], [willTopic, willQoS, willRetain, willMessage], [cleanSession])
- void **disconnect** ()
- boolean **publish** (topic, payload, [length], [retained])
- boolean **publish_P** (topic, payload, [length], [retained])
- boolean **beginPublish** (topic, length, retained)
- int **write** (byte)
- int **write** (payload, length)
- boolean **endPublish** ()
- boolean **subscribe** (topic, [qos])
- boolean **unsubscribe** (topic)
- boolean **loop** ()
- boolean **connected** ()
- int **state** ()
- PubSubClient* **setCallback** (callback)
- PubSubClient* **setClient** (client)
- PubSubClient* **setServer** (server, port)
- PubSubClient* **setStream** (stream)
- uint16_t **getBufferSize** ()
- boolean **setBufferSize** (size)
- PubSubClient* **setKeepAlive** (keepAlive)
- PubSubClient* **setSocketTimeout** (timeout)

Other

- Configuration Options
- Subscription Callback

PubSubClient ()

Creates an uninitialised client instance.

Before it can be used, it must be configured with the property setters:

```
EthernetClient ethClient;
PubSubClient client;

void setup() {
    client.setClient(ethClient);
    client.setServer("broker.example.com",1883);
    // client is now configured for use
}
```

PubSubClient (client)

Creates a partially initialised client instance.

Before it can be used, the server details must be configured:

```
EthernetClient ethClient;
PubSubClient client(ethClient);

void setup() {
    client.setServer("broker.example.com",1883);
    // client is now ready for use
}
```

Parameters

- client - the network client to use, for example `WiFiClient`

PubSubClient (server, port, [callback], client, [stream])

Creates a fully configured client instance.

Parameters

- server `IPAddress, uint8_t[] or const char[]` - the address of the server
- port `int` - the port to connect to
- callback `function*` *(optional)* - a pointer to a message callback function called when a message arrives for a subscription created by this client
- client - the network client to use, for example `WiFiClient`
- stream `Stream` *(optional)* - a stream to write received messages to

---

boolean **connect** (clientID, [username, password], [willTopic, willQoS, willRetain, willMessage], [cleanSession])

Connects the client.

Parameters

- clientID `const char[]` - the client ID to use when connecting to the server
- Credentials - *(optional)*
  - username `const char[]` - the username to use. If `NULL`, no username or password is used
  - password `const char[]` - the password to use. If `NULL`, no password is used
- Will - *(optional)*
  - willTopic `const char[]` - the topic to be used by the will message
  - willQoS `int: 0,1 or 2` - the quality of service to be used by the will message
  - willRetain `boolean` - whether the will should be published with the retain flag
  - willMessage `const char[]` - the payload of the will message
- cleanSession `boolean` *(optional)* - whether to connect clean-session or not

Returns

- `false` - connection failed
- `true` - connection succeeded

---

void **disconnect** ()

Disconnects the client.

---

boolean **publish** (topic, payload, [length], [retained])

Publishes a message to the specified topic.

Parameters

- topic `const char[]` - the topic to publish to
- payload `const char[], byte[]` - the message to publish
- length `unsigned int` *(optional)* - the length of the payload. Required if payload is a `byte[]`
- retained `boolean` *(optional)* - whether the message should be retained
  - false - not retained
  - true - retained

Returns

- `false` - publish failed, either connection lost or message too large
- `true` - publish succeeded

---

boolean **publish_P** (topic, payload, [length], [retained])

Publishes a message stored in PROGMEM to the specified topic.

Parameters

- topic `const char[]` - the topic to publish to
- payload `const char[], byte[]` - the message to publish
- length `unsigned int` *(optional)* - the length of the payload. Required if payload is a `byte[]`
- retained `boolean` *(optional)* - whether the message should be retained
  - false - not retained
  - true - retained

Returns

- `false` - publish failed, either connection lost or message too large
- `true` - publish succeeded

---

boolean **beginPublish** (topic, length, retained)

Begins sending a publish message. The payload of the message is provided by one or more calls to `write` followed by a call to `endPublish`.

Parameters

- topic `const char[]` - the topic to publish to
- length `unsigned int` - the length of the payload to be sent
- retained `boolean` - whether the message should be retained
  - false - not retained
  - true - retained

Returns

- `false` - publish failed, either connection lost or message too large
- `true` - publish succeeded

---

int **write** (byte)

Writes a byte as a component of a publish started with a call to `beginPublish`.

Parameters

- byte `uint8_t` - a byte to write to the publish payload

Returns

- `int` - the number of bytes written

---

int **write** (payload, length)

Writes an array of bytes as a component of a publish started with a call to `beginPublish`.

Parameters

- payload `byte[]` - the bytes to write
- length `unsigned int` - the length of the payload to be sent

Returns

- `int` - the number of bytes written

---

boolean **endPublish** ()

Finishing sending a message that was started with a call to `beginPublish`.

Returns

- `false` - publish failed, either connection lost or message too large
- `true` - publish succeeded

---

boolean **subscribe** (topic, [qos])

Subscribes to messages published to the specified topic.

Parameters

- topic `const char[]` - the topic to subscribe to
- qos `int: 0 or 1 only` *(optional)* - the qos to subscribe at

Returns

- `false` - sending the subscribe failed, either connection lost or message too large
- `true` - sending the subscribe succeeded

---

boolean **unsubscribe** (topic)

Unsubscribes from the specified topic.

Parameters

- topic `const char[]` - the topic to unsubscribe from

Returns

- `false` - sending the unsubscribe failed, either connection lost or message too large
- `true` - sending the unsubscribe succeeded

---

boolean **loop** ()

This should be called regularly to allow the client to process incoming messages and maintain its connection to the server.

Returns

- `false` - the client is no longer connected
- `true` - the client is still connected

---

boolean **connected** ()

Checks whether the client is connected to the server.

Returns

- `false` - the client is not connected
- `true` - the client is connected

---

int **state** ()

Returns the current state of the client. If a connection attempt fails, this can be used to get more information about the failure.

All of the values have corresponding constants defined in `PubSubClient.h`.

Returns

- `-4 : MQTT_CONNECTION_TIMEOUT` - the server didn't respond within the keepalive time
- `-3 : MQTT_CONNECTION_LOST` - the network connection was broken
- `-2 : MQTT_CONNECT_FAILED` - the network connection failed
- `-1 : MQTT_DISCONNECTED` - the client is disconnected cleanly
- `0 : MQTT_CONNECTED` - the client is connected
- `1 : MQTT_CONNECT_BAD_PROTOCOL` - the server doesn't support the requested version of MQTT
- `2 : MQTT_CONNECT_BAD_CLIENT_ID` - the server rejected the client identifier
- `3 : MQTT_CONNECT_UNAVAILABLE` - the server was unable to accept the connection

- `4 : MQTT_CONNECT_BAD_CREDENTIALS` - the username/password were rejected
- `5 : MQTT_CONNECT_UNAUTHORIZED` - the client was not authorized to connect

PubSubClient* **setCallback** (callback)

Sets the message callback function.

Parameters

- callback `function*` - a pointer to a message callback function called when a message arrives for a subscription created by this client.

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

PubSubClient* **setClient** (client)

Sets the network client instance to use.

Parameters

- client - the network client to use, for example `WiFiClient`

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

PubSubClient* **setServer** (server, port)

Sets the server details.

Parameters

- server `IPAddress, uint8_t[] or const char[]` - the address of the server
- port `int` - the port to connect to

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

PubSubClient* **setStream** (stream)

Sets the stream to write received messages to.

Parameters

- stream `Stream` - a stream to write received messages to

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

uint16_t **getBufferSize** ()

Gets the current size of the internal buffer.

By default, it is set to `256` bytes - as defined by the `MQTT_MAX_MESSAGE_SIZE` constant in `PubSubClient.h`.

Returns

- `uint16_t` - the size of the internal buffer

---

boolean **setBufferSize** (size)

Sets the size, in bytes, of the internal send/receive buffer. This must be large enough to contain the full MQTT packet. When sending or receiving messages, the packet will contain the full topic string, the payload data and a small number of header bytes.

By default, it is set to `256` bytes - as defined by the `MQTT_MAX_MESSAGE_SIZE` constant in `PubSubClient.h`.

Note : `setBufferSize` returns a boolean flag to indicate whether it was able to reallocate the memory to change the buffer size. This means, unlike the other `setXYZ` functions that return a reference to the client, this function cannot be chained with those functions.

Parameters

- size `uint16_t` - the size, in bytes, for the internal buffer

Returns

- `false` - the buffer could not be resized
- `true` - the buffer was resized

PubSubClient* **setKeepAlive** (keepAlive)

Sets the keep alive interval used by the client. This value should only be changed when the client is not connected.

By default, it is set to `15` seconds - as defined by the `MQTT_KEEPALIVE` constant in `PubSubClient.h`.

Parameters

- keepAlive `uint16_t` - the keep alive interval, in seconds

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

PubSubClient* **setSocketTimeout** (timeout)

Sets the socket timeout used by the client. This determines how long the client will wait for incoming data when it expects data to arrive - for example, whilst it is in the middle of reading an MQTT packet.

By default, it is set to `15` seconds - as defined by the `MQTT_SOCKET_TIMEOUT` constant in `PubSubClient.h`.

Parameters

- timeout `uint16_t` - the socket timeout, in seconds

Returns

- `PubSubClient*` - the client instance, allowing the function to be chained

---

Configuration Options

The following configuration options can be used to configure the library. They are contained in `PubSubClient.h`.

`MQTT_MAX_PACKET_SIZE`
: Sets the largest packet size, in bytes, the client will handle. Any packet received that exceeds this size will be ignored.

  This value can be overridden by calling `setBufferSize(size)`.

  Default: 128 bytes

`MQTT_KEEPALIVE`
: Sets the keepalive interval, in seconds, the client will use. This is used to maintain the connection when no other packets are being sent or received.

  This value can be overridden by calling `setKeepAlive(keepAlive)`.

  Default: 15 seconds

`MQTT_VERSION`
: Sets the version of the MQTT protocol to use.

  Default: MQTT 3.1.1

`MQTT_MAX_TRANSFER_SIZE`

Sets the maximum number of bytes passed to the network client in each write call. Some hardware has a limit to how much data can be passed to them in one go, such as the Arduino Wifi Shield.

Default: undefined (complete packet passed in each write call)

`MQTT_SOCKET_TIMEOUT`

Sets the timeout when reading from the network. This also applies as the timeout for calls to `connect`.

This value can be overridden by calling `setSocketTimeout(timeout)`.

Default: 15 seconds

Subscription Callback

If the client is used to subscribe to topics, a callback function must be provided in the constructor. This function is called when new messages arrive at the client.

The callback function has the following signature:

```
void callback(const char[] topic, byte* payload, unsigned int length)
```

Parameters

- topic `const char[]` - the topic the message arrived on
- payload `byte[]` - the message payload
- length `unsigned int` - the length of the message payload

Internally, the client uses the same buffer for both inbound and outbound messages. After the callback function returns, or if a call to either `publish` or subscribe is made from within the callback function, the `topic` and `payload` values passed to the function will be overwritten. The application should create its own copy of the values if they are required after the callback returns.
