API Documentation

Constructors

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

Functions

- boolean **connect** (clientID)
- boolean **connect** (clientID, willTopic, willQoS, willRetain, willMessage)
- boolean **connect** (clientID, username, password)
- boolean **connect** (clientID, username, password, willTopic, willQoS, willRetain, willMessage)
- boolean **connect** (clientID, username, password, willTopic, willQoS, willRetain, willMessage, cleanSession)
- void **disconnect** ()
- int **publish** (topic, payload)
- int **publish** (topic, payload, retained)
- int **publish** (topic, payload, length)
- int **publish** (topic, payload, length, retained)
- int **publish_P** (topic, payload, length, retained)
- boolean **beginPublish** (topic, payloadLength, retained)
- size_t **write** (uint8_t)
- size_t **write** (payload, length)
- boolean **endPublish** ()
- boolean **subscribe** (topic, [qos])
- boolean **unsubscribe** (topic)
- boolean **loop** ()
- int **connected** ()
- int **state** ()
- PubSubClient **setServer** (server, port)
- PubSubClient **setCallback** (callback)
- PubSubClient **setClient** (client)
- PubSubClient **setStream** (stream)

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 : an instance of `Client`, typically `EthernetClient`.

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

Creates a fully configured client instance.

Parameters

- server : the address of the server (IPAddress, uint8_t[] or const char[])
- port : the port to connect to (int)
- callback : *optional* a pointer to a message callback function called when a message arrives for a subscription created by this client.
- client : an instance of `Client`, typically `EthernetClient`.

- stream : *optional* an instance of `Stream`, used to store received messages. See the `mqtt_stream` example for more information.

boolean **connect** (clientID)

Connects the client.

Parameters

- clientID : the client ID to use when connecting to the server.

Returns

- false - connection failed.
- true - connection succeeded.

boolean **connect** (clientID, willTopic, willQoS, willRetain, willMessage)

Connects the client with a Will message specified.

Parameters

- clientID : the client ID to use when connecting to the server.
- willTopic : the topic to be used by the will message (const char[])
- willQoS : the quality of service to be used by the will message (int : 0,1 or 2)
- willRetain : whether the will should be published with the retain flag (boolean)
- willMessage : the payload of the will message (const char[])

Returns

- false - connection failed.
- true - connection succeeded.

boolean **connect** (clientID, username, password)

Connects the client with a username and password specified.

Parameters

- clientID : the client ID to use when connecting to the server.
- username : the username to use. If NULL, no username or password is used (const char[])
- password : the password to use. If NULL, no password is used (const char[])

Returns

- false - connection failed.
- true - connection succeeded.

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

Connects the client with a Will message, username and password specified.

Parameters

- clientID : the client ID to use when connecting to the server.
- username : the username to use. If NULL, no username or password is used (const char[])
- password : the password to use. If NULL, no password is used (const char[])
- willTopic : the topic to be used by the will message (const char[])
- willQoS : the quality of service to be used by the will message (int : 0,1 or 2)
- willRetain : whether the will should be published with the retain flag (int : 0 or 1)
- willMessage : the payload of the will message (const char[])

Returns

- false - connection failed.
- true - connection succeeded.

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

Connects the client with a Will message, username, password and clean-session flag specified.

Note : even if the `cleanSession` is set to `false`/`0` the client will *not* retry failed qos 1 publishes. This flag is only of use to maintain subscriptions on the broker.

Parameters

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

Returns

- false - connection failed.
- true - connection succeeded.

void **disconnect** ()

Disconnects the client.

int **publish** (topic, payload)

Publishes a string message to the specified topic.

Parameters

- topic - the topic to publish to (const char[])
- payload - the message to publish (const char[])

Returns

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

int **publish** (topic, payload, retained)

Publishes a string message to the specified topic.

Parameters

- topic - the topic to publish to (const char[])
- payload - the message to publish (const char[])
- retained - whether the message should be retained (boolean)
  - false - not retained
  - true - retained

Returns

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

int **publish** (topic, payload, length)

Publishes a message to the specified topic.

Parameters

- topic - the topic to publish to (const char[])
- payload - the message to publish (byte[])
- length - the length of the message (byte)

Returns

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

int **publish** (topic, payload, length, retained)

Publishes a message to the specified topic, with the retained flag as specified.

Parameters

- topic - the topic to publish to (const char[])
- payload - the message to publish (byte[])
- length - the length of the message (byte)
- retained - whether the message should be retained (boolean)
  - false - not retained

    - true - retained

Returns

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

int **publish_P** (topic, payload, length, retained)

Publishes a message stored in `PROGMEN` to the specified topic, with the retained flag as specified.

Parameters

- topic - the topic to publish to (const char[])
- payload - the message to publish (PROGMEM byte[])
- length - the length of the message (byte)
- retained - whether the message should be retained (boolean)
    - false - not retained
    - true - retained

Returns

- false - publish failed
- true - publish succeeded

boolean **beginPublish** (topic, payloadLength, 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 - the topic to publish to (const char[])
- payloadLength - the length of the message to publish
- retained - whether the message should be retained (boolean)
    - false - not retained
    - true - retained

Returns

- false - publish failed
- true - publish succeeded

size_t **write** (uint8_t)

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

Parameters

- uint8_t - the byte to write

Returns

- false - publish failed
- true - publish succeeded

size_t **write** (payload, length)

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

Parameters

- payload - the bytes to write (byte[])
- length - the length of the byte array (byte)

Returns

- false - publish failed
- true - publish succeeded

boolean **endPublish** ()

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

Returns

- false - publish failed
- true - publish succeeded

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

Subscribes to messages published to the specified topic.

Parameters

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

Returns

- false - sending the subscribe failed, either connection lost, or message too large.
- true - sending the subscribe succeeded. The request completes asynchronously.

boolean **unsubscribe** (topic)

Unsubscribes from the specified topic.

Parameters

- topic - the topic to unsubscribe from (const char[])

Returns

- false - sending the unsubscribe failed, either connection lost, or message too large.
- true - sending the unsubscribe succeeded. The request completes asynchronously.

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

int **connected** ()

Checks whether the client is connected to the server.

Returns

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

Returns

- int - the client state, which can take the following values (constants defined in `PubSubClient.h`):
  - -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 **setServer** (server, port)

Sets the server details.

Parameters

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

Returns

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

PubSubClient **setCallback** (callback)

Sets the message callback function.

Parameters

- callback : 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 client.

Parameters

- client : an instance of `Client`, typically `EthernetClient`.

Returns

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

PubSubClient **setStream** (stream)

Sets the stream.

Parameters

- stream : an instance of `Stream`, used to store received messages. See the `mqtt_stream` example for more information.

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.

  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.

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`.

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

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 beyond this.