CONDITON MONITORING DOCUMENT

CONTENTS

1. MQTT DETAILS
2. VELOCITY SEVIRITY DETAILS
3. BEARING CHARACTERSTICS FREQUENCY DETAILS
4. PHASE ANALYSIS
5. MISALIGNMENT DETAILS
6. UNBALANCE DETAILS
7. SUMMARY

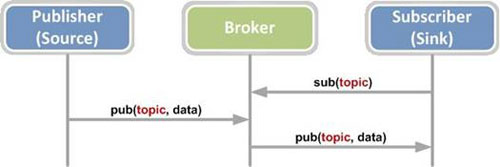
MQTT DETAILS

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport) is a lightweight, [publish-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) network [protocol](https://en.wikipedia.org/wiki/Communication_protocol) that transports messages between devices. The protocol usually runs over [TCP/IP](https://en.wikipedia.org/wiki/TCP/IP). Once the connection is established the client can send data to the broker, and the broker can send data to the client as required. You can consider a TCP/IP connection to be similar to a telephone connection. Once a telephone connection is established you can talk over it until one party hangs up.

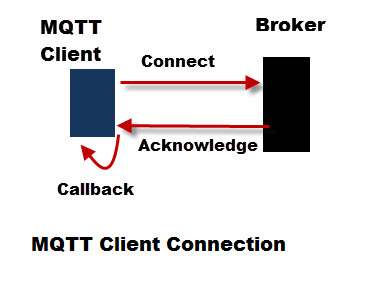
* MQTT client connects to an MQTT broker via MQTT protocol and publishes sensor readings.
* A message has a topic and a payload, like the subject and the content of an

e-mail.

* The Publisher sends a message to the network.
* The Subscriber listens for messages with a particular topic.
* The Broker is responsible for coordinating the communication between publishers and subscribers.



When a client issues a **connect request** to a broker that request should receive an acknowledgment.



The broker acknowledgement will generate a callback (**on\_connect**).

If you want to be sure that the connection attempt was successful then you will need to **setup a function** to handle this callback before you create the connection.

The function will receive 4 parameters, and it is called as **on\_connect()**.

Here is an example function definition:

on\_connect(client, userdata, flags, rc)

The**client** is a client object.

**rc** (return code) is used for checking that the connection was established.

**Connection Return Codes**

* 0: Connection successful
* 1: Connection refused – incorrect protocol version
* 2: Connection refused – invalid client identifier
* 3: Connection refused – server unavailable
* 4: Connection refused – bad username or password
* 5: Connection refused – not authorised
* 6-255: Currently unused.

Steps for establishing connection

1. Create Client object.
2. Create callback function on\_connect()
3. **Bind** callback to callback function (on\_connect())
4. Connect to Broker.
5. Start a loop.
6. Create connection
7. Verify successful connection or quit
8. Publish message and or subscribe

Sometimes **time.sleep()** is used to wait

To get better control of the connection here the flag is used in the on\_connect callback. At the start of the script we have set this flag**(connected\_flag)**  to**False**and toggle it to **True** when the Connection is successful, and back to **False** when we get a disconnect.

**Failed Connection Examples**

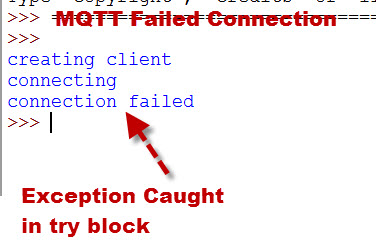
There are various conditions were the connection can fail to complete. They are:

* Incorrect client settings e.g. bad password.
* No network connection
* Bad Network Connection parameters e.g. bad port number

**Connection Failures that Create an Exception**

Trying to connect to a broker using a**bad IP address** or **port number** will generate a **socket error,** and raise an **exception**.

When the connection attempt failed, we would see:

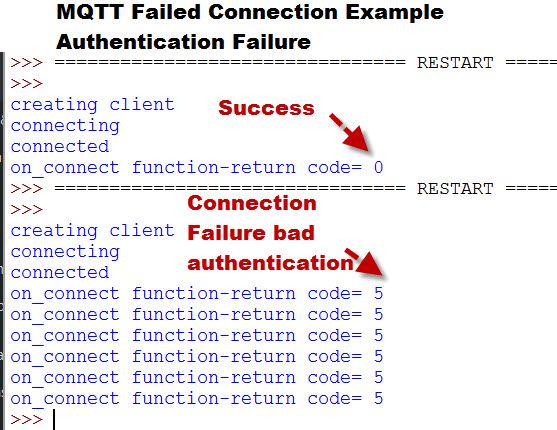
****

**Connection Failures Detected Through Return Code**

To determine if the connection was successful we need to examine the **return code** of the **on\_connect** callback.

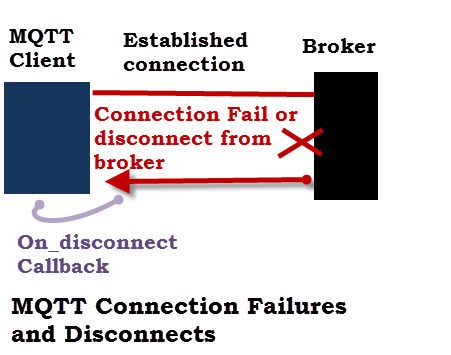
A **return code of 0** is successful, whereas other values indicate a failure.

In the example below we will try to connect to a broker without providing the required authentication.



We can stop the loop in the **on\_connect** callback. However to stop the main script we need to set a flag that we can use to exit.

If the connection fails for some reason then you will need to decide whether or not you should**try to reconnect.**



A disconnect triggers the **on\_disconnect** callback which you will need to examine.

This callback takes 3 parameters:

**Client-** Client object that disconnected

**Userdata-** user defined data not often used

**Return Code (rc)-** Indication of disconnect reason.**0 is normal** all other values indicate abnormal disconnection

**Reconnecting**

Generally we will need to **reconnect** as soon as possible.

If you run a network loop using **loop\_start()** or **loop\_forever()** then **re-connections** are automatically handled for you.

A new connection attempt is made automatically in the background every 3 to 6 seconds.

If you call the **loop() function** manually then you will need to handle the re-connection attempts yourself. See [understanding the loop](http://www.steves-internet-guide.com/loop-python-mqtt-client/).

You can do this by using a connection flag that is toggled by the**on\_connect** and **on\_disconnect** callbacks.

**Client Connection Summary**

Taking into account the above our client connection code should.

* Connect to broker
* Examine connection status and proceed if good
* If connection status is bad attempt retry and or quit.
* Handle disconnects and reconnects

### on\_message

Getting back to the problem of messages not being displayed despite subscribing to the topic: The on\_message() callback is used to process messages that are published to a subscribed topic.

In our program, we need to do 3 things:  
1. Subscribe to the topic that we are publishing to.  
2. Process the published message using the callback. In our program, we will simply print the message.  
3. Assign the callback function to the on\_message attribute of the client object.

If the message doesn’t get printed, check if:

* You have added the function on\_message() to the client object’s attribute.
* If you have subscribed to the topic in the on\_connect() callback in our program.
* If you have entered the topic names correctly in the subscribe() and publish() functions in our program respectively.
* If you are printing the message in the on\_message() function.

**How to organize & process the messages?**

If you have multiple messages being received from different topics and need to process each topic’s message in a different way then we have to sort the messages.

One way to do this is by using the message.topic attribute to check which topic is the message published to. Then you can create if conditions to process the messages accordingly.

The better way to do this is to use the message\_callback\_add(sub, callback) function of the client object. This creates multiple callbacks to process messages from different topics.  
The sub parameter takes the topic name and the callback parameter takes the name of the callback that will process messages of that topic.

on\_publish()

The on\_publish() function gets called when the publish() function is executed. This will return a tuple (result, mid).

result is the error code. An error code of 0 means the message is published successfully.

mid stands for message id. It is an integer that is a unique message identifier assigned by the client. If you use QoS levels 1 or 2 then the client loop will use the mid to identify messages that have not been sent.

on\_subscribe ()

The on\_subscribe() callback is called when a client subscribes to a topic. Ex: on\_subscribe (client, userdata, mid, granted\_qos)