1. MQTT API

Energydata.dk provides a real-time publishing and subscription API. This API serves two purposes:

- It allows data providers to send data to be stored in Energydata.dk.
- It allows subscribers to receive real-time messages with the data published to Energydata.dk.

The API is built on top of the MQTT protocol, version 3.1.1.

This document describes how to connect to the MQTT broker, how MQTT topics relate to datasets defined in the energydata.dk and specifies the format and content of messages exchanged on these topics.

1.1 Connecting

The MQTT broker is available at mqtts.energydata.dk, and is open for MQTT connections on port 8883.

1.2 Authentication

Before you can connect to the broker, you must create a token in Energydata.dk, with the mqtt pubsub permission as well permission to use the licenses that gives you access to the datasets you want.

When connecting you must use the token as the username.

An example in Python (requires paho-mqtt <https://pypi.python.org/pypi/paho-mqtt/>_):

```
#! /usr/bin/env python

import paho.mqtt.client as mqtt

broker_host = 'mqtts.energydata.dk'
broker_port = 8883
heartbeat = 60

def connect_logger(client, userdata, flags, rc):
    print('Connected with rc {}'.format(rc))
```

DEN EUROPÆISKE UNION
Den Europæiske Socialfond
DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Finansieret som et led i EU's reaktion på COVID-19-pandemien
Vi investerer i din fremtid

```
client = mqtt.Client(client_id='<your client_ID name>')
client.username_pw_set('my-token')
client.on_connect = connect_logger
client.tls_set()
client.connect(broker_host, broker_port, heartbeat)
print('Connecting')
client.loop_forever()
```

An example using the mosquitto_sub command-line client (bundled with Mosquitto <https://mosquitto.org/>_):

```
#! /bin/bash
mosquitto_sub -t '#' \
  -h mqtts.energydata.dk \
  -p 8883 \
  -i '<your client_ID name>' \
  -u 'your-token'
```

1.3 Topics and Datasets

In Energydata.dk time series are defined as datastreams and grouped into a dataset.

To send data to Energydata.dk using MQTT, you must create a dataset with a topic prefix and a datastream within that dataset with a topic suffix. The topic prefix is one string of alphanumerics, the suffix can be several alphanumeric strings separated by `/`. So if the prefix is `my-topic-prefix` and the suffix is `my/topic/suffix`, the complete MQTT topic of that datastream is `my-topic-prefix/my/topic/suffix`.

This topic will be used on the MQTT broker, to identify the datastream.

Examples with Mosquitto:

```
#! /bin/bash

mosquitto_sub -t 'my-topic-prefix/my/topic/suffix' \
  -h mqtts.energydata.dk \
  -p 8883 \
  -i '<your client_ID name>' \
  -u 'your-token'
#! /bin/bash
```

DEN EUROPÆISKE UNION
Den Europæiske Socialfond
DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Finansieret som et led i EU's reaktion på COVID-19-pandemien
Vi investerer i din fremtid

```
mosquitto_pub -t 'my-topic-prefix/my/topic/suffix' \
  -h mqtts.energydata.dk \
  -p 8883 \
  -i '<your client_ID name>' \
  -u 'your-token'
  -m '{"value":"0.16300000250339508","timestamp":1521797973052}'
```

1.4 Message format

MQTT messages can be formatted a one of two types:

1. Single value message
2. Multi value message

1.4.1 Single value messages

Single value messages contain a timestamp and value, as described in the table below. The datastream associated with the message is inferred from the MQTT topic.

Key	Type	Required	Description	Example
timestamp	Integer	Yes	Number of milliseconds since 1970-01-01 (Unix epoch)	1521797973469
value	String, Double or Integer	Yes	Value, encoded either as String, Double or Integer, depending on property definition in DMS	14.47

Example message:

```
{
   "timestamp": 1521797973469,
   "value": 14.47
}
```

1.5 Multi value messages

Multi value messages are used in situations where you need to publish several different but related values. They will be persisted with the same timestamp in the same dataset, on topics specified that follows.

Multi value messages are published with the topic prefix as the topic of the mqtt message, for example `my-topic-prefix`. The payload has the following form:

DEN EUROPÆISKE UNION
Den Europæiske Socialfond
DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Finansieret som et led i EU's reaktion på COVID-19-pandemien
Vi investerer i din fremtid

```
{
   "timestamp": 1521797973469,
   "value": {
     "my/topic/suffix1": 14.47
     "my/topic/suffix2": 34
     "my/topic/suffix3": 4.87
     "my/topic/suffix4": 1
     ....
     'my/topic/suffixn': 300
   }
}
```

All message payloads must be serialized as JSON.

1.6 Best Practices - Ensuring Message Delivery on Publish

When publishing to Energydata.dk using MQTT you should use Quality of Service (QoS) 1 to ensure that messages are delivered. This is especially important if you publish with a high throughput, as overload protection mechanisms can discard your messages. Why and how is described in detail below.

When the Energydata.dk MQTT broker receives a publish message from your client, the message is added to a queue of incoming messages. This queue is per-client, and can contain up to 1000 messages. Messages in the queue are dequeued serially in a FIFO manner. When dequeued, the MQTT broker checks that the client is authorized for publishing messages on the given topic.

If authorization succeeds, the message is forwarded for storage in Energydata.dk, and to any clients subscribed to the given topic. If the message was published with QoS 1 or 2, the MQTT broker sends the acknowledgement after authorization succeeds.

If authorization does not succeed, the client is immediately disconnected and any messages in the incoming message queue are discarded.

If the MQTT broker is not able to keep up with the client, the incoming message queue will eventually be full. When the queue is full, the MQTT broker discards any messages it receives from the client.

DEN EUROPÆISKE UNION
Den Europæiske Socialfond
DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Finansieret som et led i EU's reaktion på COVID-19-pandemien
Vi investerer i din fremtid

This means your client must throttle itself to avoid losing messages. You do this by publishing your messages with QoS 1. Your client must then wait to receive an acknowledgement from the broker before publishing more messages.
This can be done in Python with the paho-mqtt <https://pypi.python.org/pypi/paho-mqtt/>_ library with the function wait_for_publish:

```
#! /usr/bin/env python

import time
import json
import paho.mqtt.client as mqtt

broker_host = 'mqtts.energydata.dk'
broker_port = 8883
heartbeat = 60

client = mqtt.Client(client_id='<your client_ID name>')
client.username_pw_set('my-token')
client.connect(broker_host, broker_port, heartbeat)

# Remember to start the loop in order to receive QoS acknowledgements
client.loop_start()

for i in range(0,10000):
   timestamp = int(time.time() * 1000)
   message = json.dumps({'timestamp': timestamp, 'value': i})
   # Publish with QoS1 (`qos=1`) and wait to receive QoS acknowledgements from
broker
   message_info = client.publish('my-topic', message, qos=1)
   message_info.wait_for_publish()
```


DEN EUROPÆISKE UNION
Den Europæiske Socialfond
DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Finansieret som et led i EU's reaktion på COVID-19-pandemien
Vi investerer i din fremtid