AdafruitMiniMQTT Library Documentation

Release 1.0

Brent Rubell

Jun 07, 2021

Contents

MQTT Client library for CircuitPython.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-minimqtt
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-minimqtt
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-minimqtt
```

CHAPTER 3

Usage Example

Please check the examples folder for usage examples for this library.

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/minimqtt_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import ssl
import socketpool
import wifi
import adafruit_minimqtt.adafruit_minimqtt as MQTT

# Add a secrets.py to your filesystem that has a dictionary called secrets with "ssid
↪" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it␣
↪into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# Set your Adafruit IO Username and Key in secrets.py
# (visit io.adafruit.com if you need to create an account,
# or if you need your Adafruit IO key.)
aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]

print("Connecting to %s" % secrets["ssid"])
```

(continues on next page)

(continued from previous page)

```
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!" % secrets["ssid"])

### Topic Setup ###

# MQTT Topic
# Use this topic if you'd like to connect to a standard MQTT broker
mqtt_topic = "test/topic"

# Adafruit IO-style Topic
# Use this topic if you'd like to connect to io.adafruit.com
# mqtt_topic = secrets["aio_username"] + '/feeds/temperature'

### Code ###
# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connect(mqtt_client, userdata, flags, rc):
    # This function will be called when the mqtt_client is connected
    # successfully to the broker.
    print("Connected to MQTT Broker!")
    print("Flags: {0}\n RC: {1}".format(flags, rc))


def disconnect(mqtt_client, userdata, rc):
    # This method is called when the mqtt_client disconnects
    # from the broker.
    print("Disconnected from MQTT Broker!")


def subscribe(mqtt_client, userdata, topic, granted_qos):
    # This method is called when the mqtt_client subscribes to a new feed.
    print("Subscribed to {0} with QOS level {1}".format(topic, granted_qos))


def unsubscribe(mqtt_client, userdata, topic, pid):
    # This method is called when the mqtt_client unsubscribes from a feed.
    print("Unsubscribed from {0} with PID {1}".format(topic, pid))


def publish(mqtt_client, userdata, topic, pid):
    # This method is called when the mqtt_client publishes data to a feed.
    print("Published to {0} with PID {1}".format(topic, pid))


def message(client, topic, message):
    # Method callled when a client's subscribed feed has a new value.
    print("New message on topic {0}: {1}".format(topic, message))


# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Set up a MiniMQTT Client
mqtt_client = MQTT.MQTT(
    broker=secrets["broker"],
    port=secrets["port"],
    username=secrets["aio_username"],
```

(continues on next page)

(continued from previous page)

```
    password=secrets["aio_key"],
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)

# Connect callback handlers to mqtt_client
mqtt_client.on_connect = connect
mqtt_client.on_disconnect = disconnect
mqtt_client.on_subscribe = subscribe
mqtt_client.on_unsubscribe = unsubscribe
mqtt_client.on_publish = publish
mqtt_client.on_message = message

print("Attempting to connect to %s" % mqtt_client.broker)
mqtt_client.connect()

print("Subscribing to %s" % mqtt_topic)
mqtt_client.subscribe(mqtt_topic)

print("Publishing to %s" % mqtt_topic)
mqtt_client.publish(mqtt_topic, "Hello Broker!")

print("Unsubscribing from %s" % mqtt_topic)
mqtt_client.unsubscribe(mqtt_topic)

print("Disconnecting from %s" % mqtt_client.broker)
mqtt_client.disconnect()
```

6.2 adafruit_minimqtt

A minimal MQTT Library for CircuitPython.

- Author(s): Brent Rubell

6.2.1 Implementation Notes

Adapted from https://github.com/micropython/micropython-lib/tree/master/umqtt.simple/umqtt

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

exception adafruit_minimqtt.adafruit_minimqtt.**MMQTTException**
MiniMQTT Exception class.

class adafruit_minimqtt.adafruit_minimqtt.**MQTT**(*broker*, *port=None*, *username=None*, *password=None*, *client_id=None*, *is_ssl=True*, *keep_alive=60*, *socket_pool=None*, *ssl_context=None*)
MQTT Client for CircuitPython. :param str broker: MQTT Broker URL or IP Address. :param int port: Optional port definition, defaults to 8883. :param str username: Username for broker authentication. :param str password: Password for broker authentication. :param network_manager: NetworkManager object, such as WiFiManager from ESPSPI_WiFiManager. :param str client_id: Optional client identifier, defaults to a unique, generated string. :param bool is_ssl: Sets a secure or insecure connection with the broker. :param int keep_alive:

KeepAlive interval between the broker and the MiniMQTT client. :param socket socket_pool: A pool of socket resources available for the given radio. :param ssl_context: SSL context for long-lived SSL connections.

add_topic_callback(*mqtt_topic*, *callback_method*)
Registers a callback_method for a specific MQTT topic.

Parameters

- **mqtt_topic** (*str*) – MQTT topic identifier.
- **callback_method** (*str*) – Name of callback method.

connect(*clean_session=True*, *host=None*, *port=None*, *keep_alive=None*)
Initiates connection with the MQTT Broker. :param bool clean_session: Establishes a persistent session. :param str host: Hostname or IP address of the remote broker. :param int port: Network port of the remote broker. :param int keep_alive: Maximum period allowed for communication, in seconds.

deinit()
De-initializes the MQTT client and disconnects from the mqtt broker.

disable_logger()
Disables logging.

disconnect()
Disconnects the MiniMQTT client from the MQTT broker.

enable_logger(*logger*, *log_level=20*)
Enables library logging provided a logger object. :param logger: A python logger pacakge. :param log_level: Numeric value of a logging level, defaults to INFO.

is_connected()
Returns MQTT client session status as True if connected, raises a *MMQTTException* if `False`.

loop(*timeout=1*)
Non-blocking message loop. Use this method to check incoming subscription messages. Returns response codes of any messages received. :param int timeout: Socket timeout, in seconds.

mqtt_msg
Returns maximum MQTT payload and topic size.

on_message
Called when a new message has been received on a subscribed topic.

Expected method signature is `on_message(client, topic, message)`

ping()
Pings the MQTT Broker to confirm if the broker is alive or if there is an active network connection. Returns response codes of any messages received while waiting for PINGRESP.

publish(*topic*, *msg*, *retain=False*, *qos=0*)
Publishes a message to a topic provided. :param str topic: Unique topic identifier. :param str,int,float msg: Data to send to the broker. :param bool retain: Whether the message is saved by the broker. :param int qos: Quality of Service level for the message, defaults to zero.

reconnect(*resub_topics=True*)
Attempts to reconnect to the MQTT broker. :param bool resub_topics: Resubscribe to previously subscribed topics.

remove_topic_callback(*mqtt_topic*)
Removes a registered callback method.

Parameters **mqtt_topic** (*str*) – MQTT topic identifier string.

subscribe(*topic*, *qos=0*)
Subscribes to a topic on the MQTT Broker. This method can subscribe to one topics or multiple topics.

Parameters

- **topic** (*str*, *tuple*, *list*) – Unique MQTT topic identifier string. If this is a `tuple`, then the tuple should contain topic identifier string and qos level integer. If this is a `list`, then each list element should be a tuple containing a topic identifier string and qos level integer.
- **qos** (*int*) – Quality of Service level for the topic, defaults to zero. Conventional options are `0` (send at most once), `1` (send at least once), or `2` (send exactly once).

unsubscribe(*topic*)
Unsubscribes from a MQTT topic. :param str,list topic: Unique MQTT topic identifier string or list.

username_pw_set(*username*, *password=None*)
Set client's username and an optional password. :param str username: Username to use with your MQTT broker. :param str password: Password to use with your MQTT broker.

will_set(*topic=None*, *payload=None*, *qos=0*, *retain=False*)
Sets the last will and testament properties. MUST be called before *connect()*.

Parameters

- **topic** (*str*) – MQTT Broker topic.
- **payload** (*int*, *float*, *str*) – Last will disconnection payload. payloads of type int & float are converted to a string.
- **qos** (*int*) – Quality of Service level, defaults to zero. Conventional options are `0` (send at most once), `1` (send at least once), or `2` (send exactly once).

  Note: Only options `1` or `0` are QoS levels supported by this library.

- **retain** (*bool*) – Specifies if the payload is to be retained when it is published.

adafruit_minimqtt.adafruit_minimqtt.**set_socket**(*sock*, *iface=None*)
Legacy API for setting the socket and network interface. :param sock: socket object. :param iface: internet interface object

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index