AdafruitMiniMQTT Library Documentation

Release 1.0

Brent Rubell

Jan 31, 2020

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

```
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket

from adafruit_minimqtt import MQTT

### WiFi ###

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
```

(continues on next page)

(continued from previous page)

```
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2) # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

### 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 = 'aio_user/feeds/temperature'

### Code ###

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

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

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

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

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

# Connect to WiFi
```

(continues on next page)

(continued from previous page)

```
wifi.connect()

# Set up a MiniMQTT Client
client =  MQTT(socket,
               broker = secrets['broker'],
               username = secrets['user'],
               password = secrets['pass'],
               network_manager = wifi)

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

print('Attempting to connect to %s'%client.broker)
client.connect()

print('Subscribing to %s'%mqtt_topic)
client.subscribe(mqtt_topic)

print('Publishing to %s'%mqtt_topic)
client.publish(mqtt_topic, 'Hello Broker!')

print('Unsubscribing from %s'%mqtt_topic)
client.unsubscribe(mqtt_topic)

print('Disconnecting from %s'%client.broker)
client.disconnect()
```

6.2 adafruit_minimqtt

MQTT Library for CircuitPython.

- Author(s): Brent Rubell

6.2.1 Implementation Notes

Software and Dependencies:

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

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

class adafruit_minimqtt.**MQTT**(*socket*, *broker*, *port=None*, *username=None*, *password=None*, *network_manager=None*, *client_id=None*, *is_ssl=True*, *log=False*, *keep_alive=60*)
MQTT Client for CircuitPython :param socket: Socket object for provided network interface :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 bool log: Attaches a logger to the MQTT client, defaults to logging level INFO. :param int keep_alive: KeepAlive interval between the broker and the MiniMQTT client.

attach_logger(*logger_name='log'*)
Initializes and attaches a logger to the MQTTClient. :param str logger_name: Name of the logger instance

connect(*clean_session=True*)
Initiates connection with the MQTT Broker. :param bool clean_session: Establishes a persistent session.

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

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

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

is_sock_connected
Returns if the socket is connected.

is_wifi_connected
Returns if the ESP module is connected to an access point, resets module if False

last_will(*topic=None*, *message=None*, *qos=0*, *retain=False*)
Sets the last will and testament properties. MUST be called before connect(). :param str topic: MQTT Broker topic. :param str message: Last will disconnection message. :param int qos: Quality of Service level. :param bool retain: Specifies if the message is to be retained when it is published.

loop()
Non-blocking message loop. Use this method to check incoming subscription messages.

This method does NOT handle networking or network hardware management, use loop_forever or handle in code instead.

loop_forever()
Starts a blocking message loop. Use this method if you want to run a program forever. Code below a call to this method will NOT execute. Network reconnection is handled within this call.

mqtt_msg
Returns maximum MQTT payload and topic size.

ping()
Pings the MQTT Broker to confirm if the broker is alive or if there is an active network connection.

publish(*topic*, *msg*, *retain=False*, *qos=0*)
Publishes a message to a topic provided. :param str topic: Unique topic identifier. :param str msg: Data to send to the broker. :param int msg: Data to send to the broker. :param 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.

Example of sending an integer, 3, to the broker on topic 'piVal'. .. code-block:: python

mqtt_client.publish('topics/piVal', 3)

Example of sending a float, 3.14, to the broker on topic 'piVal'. .. code-block:: python

mqtt_client.publish('topics/piVal', 3.14)

Example of sending a string, 'threepointonefour', to the broker on topic piVal. .. code-block:: python

mqtt_client.publish('topics/piVal', 'threepointonefour')

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

reconnect_socket()
Re-establishes the socket's connection with the MQTT broker.

reconnect_wifi()
Reconnects to WiFi Access Point and socket, if disconnected.

set_logger_level(*log_level*)
Sets the level of the logger, if defined during init. :param string log_level: Level of logging to output to the REPL.

subscribe(*topic*, *qos=0*)
Subscribes to a topic on the MQTT Broker. This method can subscribe to one topics or multiple topics. :param str topic: Unique MQTT topic identifier. :param int qos: Quality of Service level for the topic, defaults to zero. :param tuple topic: Tuple containing topic identifier strings and qos level integers. :param list topic: List of tuples containing topic identifier strings and qos.

Example of subscribing a topic string. .. code-block:: python

mqtt_client.subscribe('topics/ledState')

Example of subscribing to a topic and setting the qos level to 1. .. code-block:: python

mqtt_client.subscribe('topics/ledState', 1)

Example of subscribing to topic string and setting qos level to 1, as a tuple. .. code-block:: python

mqtt_client.subscribe(('topics/ledState', 1))

Example of subscribing to multiple topics with different qos levels. .. code-block:: python

mqtt_client.subscribe([('topics/ledState', 1), ('topics/servoAngle', 0)])

unsubscribe(*topic*)
Unsubscribes from a MQTT topic. :param str topic: Unique MQTT topic identifier. :param list topic: List of tuples containing topic identifier strings.

Example of unsubscribing from a topic string. .. code-block:: python

mqtt_client.unsubscribe('topics/ledState')

Example of unsubscribing from multiple topics. .. code-block:: python

mqtt_client.unsubscribe([('topics/ledState'), ('topics/servoAngle')])

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index