AdafruitAWS IOT Library Documentation

Release 1.0

Brent Rubell

Mar 02, 2021

Contents

Amazon AWS IoT MQTT Client for CircuitPython.

Note: This library requires version >=1.4.0 of the Adafruit fork of the Arduino NINA-W102 firmware installed on your ESP32 Airlift/WiFi Co-Processor.

If you do not know how to do this, visit the Adafruit Learning System guide for this topic…

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-aws-iot
```

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

```
sudo pip3 install adafruit-circuitpython-aws-iot
```

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-aws-iot
```

CHAPTER 3

Usage Example

Library examples within examples/ folder.

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/aws_iot_simpletest.py

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

import time
import json
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
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_aws_iot import MQTT_CLIENT

### 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

# Get device certificate
try:
    with open("aws_cert.pem.crt", "rb") as f:
```

(continues on next page)

(continued from previous page)

```
        DEVICE_CERT = f.read()
except ImportError:
    print("Certificate (aws_cert.pem.crt) not found on CIRCUITPY filesystem.")
    raise

# Get device private key
try:
    with open("private.pem.key", "rb") as f:
        DEVICE_KEY = f.read()
except ImportError:
    print("Certificate (private.pem.key) not found on CIRCUITPY filesystem.")
    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)
# 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)

# Verify nina-fw version >= 1.4.0
assert (
    int(bytes(esp.firmware_version).decode("utf-8")[2]) >= 4
), "Please update nina-fw to >=1.4.0."

# 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)

### Code ###

topic = "circuitpython/aws"

# 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!")
```

(continues on next page)

(continued from previous page)

```
    print("Flags: {0}\n RC: {1}".format(flags, rc))

    # Subscribe to topic circuitpython/aws
    print("Subscribing to topic {}".format(topic))
    aws_iot.subscribe(topic)


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 topic.
    print("Subscribed to {0} with QOS level {1}".format(topic, granted_qos))

    # Create a json-formatted message
    message = {"message": "Hello from AWS IoT CircuitPython"}
    # Publish message to topic
    aws_iot.publish(topic, json.dumps(message))


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


def message(client, topic, msg):
    # This method is called when the client receives data from a topic.
    print("Message from {}: {}".format(topic, msg))


# Set AWS Device Certificate
esp.set_certificate(DEVICE_CERT)

# Set AWS RSA Private Key
esp.set_private_key(DEVICE_KEY)

# Connect to WiFi
print("Connecting to WiFi...")
wifi.connect()
print("Connected!")

# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

# Set up a new MiniMQTT Client
client = MQTT.MQTT(broker=secrets["broker"], client_id=secrets["client_id"])

# Initialize AWS IoT MQTT API Client
aws_iot = MQTT_CLIENT(client)
```

(continues on next page)

(continued from previous page)

```

# Connect callback handlers to AWS IoT MQTT Client
aws_iot.on_connect = connect
aws_iot.on_disconnect = disconnect
aws_iot.on_subscribe = subscribe
aws_iot.on_unsubscribe = unsubscribe
aws_iot.on_publish = publish
aws_iot.on_message = message

print("Attempting to connect to %s" % client.broker)
aws_iot.connect()

# Pump the message loop forever, all events
# are handled in their callback handlers
# while True:
#   aws_iot.loop()

# Start a blocking message loop...
# NOTE: NO code below this loop will execute
# NOTE: Network reconnection is handled within this loop
while True:
    try:
        aws_iot.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        aws_iot.reconnect()
        continue
    time.sleep(1)
```

6.2 adafruit_aws_iot

Amazon AWS IoT MQTT Client for CircuitPython

- Author(s): Brent Rubell

6.2.1 Implementation Notes

Hardware:

Software and Dependencies:

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

exception adafruit_aws_iot.**AWS_IOT_ERROR**
Exception raised on MQTT API return-code errors.

class adafruit_aws_iot.**MQTT_CLIENT**(*mmqttclient*, *keep_alive=30*)
Client for interacting with Amazon AWS IoT MQTT API.

Parameters

- **mmqttclient** (*MiniMQTT*) – Pre-configured MiniMQTT Client object.
- **keep_alive** (*int*) – Optional Keep-alive timer interval, in seconds. Provided interval must be 30 <= keep_alive <= 1200.

connect(*clean_session=True*)
Connects to Amazon AWS IoT MQTT Broker with Client ID. :param bool clean_session: Establishes a clean session with AWS broker.

disconnect()
Disconnects from Amazon AWS IoT MQTT Broker and de-initializes the MiniMQTT Client.

is_connected
Returns if MQTT_CLIENT is connected to AWS IoT MQTT Broker

loop()
Starts a synchronous message loop which maintains connection with AWS IoT. Must be called within the keep_alive timeout specified to init. This method does not handle network connection/disconnection.

Example of "pumping" an AWS IoT message loop: ..code-block::python

while True: aws_iot.loop()

loop_forever()
Begins a blocking, asynchronous message loop. This method handles network connection/disconnection.

publish(*topic*, *payload*, *qos=1*)
Publishes to a AWS IoT Topic. :param str topic: MQTT topic to publish to. :param str payload: Data to publish to topic. :param int payload: Data to publish to topic. :param float payload: Data to publish to topic. :param json payload: JSON-formatted data to publish to topic. :param int qos: Quality of service level for publishing.

reconnect()
Reconnects to the AWS IoT MQTT Broker

shadow_delete()
Publishes an empty message to the shadow delete topic to delete a device's shadow

shadow_get()
Publishes an empty message to shadow get topic to get the device's shadow.

shadow_get_subscribe(*qos=1*)
Subscribes to device's shadow get response. :param int qos: Optional quality of service level.

shadow_subscribe(*qos=1*)
Subscribes to all notifications on the device's shadow update topic. :param int qos: Optional quality of service level.

shadow_update(*document*)
Publishes a request state document to update the device's shadow. :param json state_document: JSON-formatted state document.

subscribe(*topic*, *qos=1*)
Subscribes to an AWS IoT Topic. :param str topic: MQTT topic to subscribe to. :param int qos: Desired topic subscription's quality-of-service.

static validate_topic(*topic*)
Validates if user-provided pub/sub topics adhere to AWS Service Limits. https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html :param str topic: Desired topic to validate

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index