AdafruitGC$_{IOT_{C}ORE}$ Library Documentation

Release 1.0

Brent Rubell

Jun 07, 2021

Contents

Google Cloud IoT Core Client for CircuitPython

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython
- Adafruit CircuitPython JWT
- Adafruit CircuitPython Logging

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-gc-iot-core
```

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

```
sudo pip3 install adafruit-circuitpython-gc-iot-core
```

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-gc-iot-core
```

CHAPTER 3

Usage Example

Usage example 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

License

This library was written by Google for MicroPython. We've converted it to work with CircuitPython and made changes so it works with boards supported by CircuitPython and the CircuitPython API.

We've added examples for using this library to transmit board telemetry data along with sensor data to Google's Cloud Platform.

This open source code is licensed under the Apache license (see LICENSE) for details.

CHAPTER 7

Table of Contents

7.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/gc_iot_core_simpletest.py

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

import time
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_gc_iot_core import Cloud_Core, MQTT_API

### 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)
```

(continues on next page)

(continued from previous page)

```
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)
"""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 ###

# 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))
    # Subscribes to commands/# topic
    google_mqtt.subscribe_to_all_commands()

    # Publish to the default "events" topic
    google_mqtt.publish("testing", "events", qos=1)


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))


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))


```

(continues on next page)

(continued from previous page)

```
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))


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

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

# Initialize Google Cloud IoT Core interface
google_iot = Cloud_Core(esp, secrets)

# Optional JSON-Web-Token (JWT) Generation
# print("Generating JWT...")
# jwt = google_iot.generate_jwt()
# print("Your JWT is: ", jwt)

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

# Initialize Google MQTT API Client
google_mqtt = MQTT_API(client)

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

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

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

# Start a blocking message loop...
# NOTE: NO code below this loop will execute
# NOTE: Network reconnection is handled within this loop
while True:
```

(continues on next page)

(continued from previous page)

```
    try:
        google_mqtt.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        google_mqtt.reconnect()
        continue
    time.sleep(1)
```

7.2 adafruit_gc_iot_core

CircuitPython Google Cloud IoT Module

- Author(s): Brent Rubell, Google Inc.

7.2.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases
- Adafruit CircuitPython JWT Module: https://github.com/adafruit/Adafruit_CircuitPython_JWT
- Adafruit CircuitPython Logging Module: https://github.com/adafruit/Adafruit_CircuitPython_Logging

class adafruit_gc_iot_core.**Cloud_Core**(*esp*, *secrets*, *log=False*)
CircuitPython Google Cloud IoT Core module.

Parameters

- **esp** (*ESP_SPIcontrol*) – ESP32SPI object.
- **secrets** (*dict*) – Secrets.py file.
- **log** (*bool*) – Enable Cloud_Core logging, defaults to False.

client_id
Returns a Google Cloud IOT Core Client ID.

generate_jwt(*ttl=43200*, *algo='RS256'*)
Generates a JSON Web Token (https://jwt.io/) using network time. :param int jwt_ttl: When the JWT token expires, defaults to 43200 minutes (or 12 hours). :param str algo: Algorithm used to create a JSON Web Token.

Example usage of generating and setting a JSON-Web-Token: ..code-block:: python

jwt = CloudCore.generate_jwt() print("Generated JWT: ", jwt)

class adafruit_gc_iot_core.**MQTT_API**(*mqtt_client*)
Client for interacting with Google's Cloud Core MQTT API.

Parameters **mqtt_client** (*MiniMQTT*) – MiniMQTT Client object.

connect()
Connects to the Google MQTT Broker.

disconnect()
Disconnects from the Google MQTT Broker.

is_connected
Returns if client is connected to Google's MQTT broker.

loop()
Maintains a connection with Google Cloud IoT Core's MQTT broker. You will need to manually call this method within a loop to retain connection.

Example of "pumping" a Google Core IoT loop. ..code-block:: python

while True: google_iot.loop()

publish(*payload*, *topic='events'*, *subfolder=None*, *qos=0*)
Publishes a payload from the device to its Google Cloud IoT device topic, defaults to "events" topic. To send state, use the publish_state method.

Parameters

- **payload** (*float*) – Data to publish to Google Cloud IoT
- **payload** – Data to publish to Google Cloud IoT
- **payload** – Data to publish to Google Cloud IoT
- **topic** (*str*) – Required MQTT topic. Defaults to events.
- **subfolder** (*str*) – Optional MQTT topic subfolder. Defaults to None.
- **qos** (*int*) – Quality of Service level for the message.

publish_state(*payload*)
Publishes a device state message to the Cloud IoT MQTT API. Data sent by this method should be information about the device itself (such as number of crashes, battery level, or device health). This method is unidirectional, it communicates Device-to-Cloud only.

reconnect()
Reconnects to the Google MQTT Broker.

subscribe(*topic*, *subfolder=None*, *qos=1*)
Subscribes to a Google Cloud IoT device topic. :param str topic: Required MQTT topic. Defaults to events. :param str subfolder: Optional MQTT topic subfolder. Defaults to None. :param int qos: Quality of Service level for the message.

subscribe_to_all_commands(*qos=1*)
Subscribes to a device's "commands/#" topic. :param int qos: Quality of Service level for the message.

subscribe_to_config(*qos=1*)
Subscribes to a Google Cloud IoT device's configuration topic. :param int qos: Quality of Service level for the message.

subscribe_to_subfolder(*topic*, *subfolder*, *qos=1*)
Subscribes to a Google Cloud IoT device's topic subfolder :param str topic: Required MQTT topic. :param str subfolder: Optional MQTT topic subfolder. Defaults to None. :param int qos: Quality of Service level for the message.

unsubscribe(*topic*, *subfolder=None*)
Unsubscribes from a Google Cloud IoT device topic. :param str topic: Required MQTT topic. Defaults to events. :param str subfolder: Optional MQTT topic subfolder. Defaults to None.

unsubscribe_from_all_commands()
Unsubscribes from a device's "commands/#" topic. :param int qos: Quality of Service level for the message.

exception adafruit_gc_iot_core.**MQTT_API_ERROR**
Exception raised on MQTT API return-code errors.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index