AdafruitRequests Library Documentation

Release 1.0

ladyada

Jan 03, 2020

Contents

A requests-like library for HTTP commands.

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

Note: This library is not available on PyPI yet. Install documentation is included as a standard element. Stay tuned for PyPI availability!

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-requests
```

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

```
sudo pip3 install adafruit-circuitpython-requests
```

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-requests
```

CHAPTER 3

Usage Example

Usage examples are within the *Simple test* subfolder of 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/requests_simpletest.py

```
# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# 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)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
```

(continues on next page)

(continued from previous page)

```
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "http://httpbin.org/get"
JSON_POST_URL = "http://httpbin.org/post"

print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()

print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()

data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()

json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

6.2 adafruit_requests

A requests-like library for web interfacing

- Author(s): ladyada, Paul Sokolovsky

6.2.1 Implementation Notes

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

micropython-lib consists of multiple modules from different sources and authors. Each module comes under its own licensing terms. Short name of a license can be found in a file within a module directory (usually metadata.txt or setup.py). Complete text of each license used is provided at https://github.com/micropython/micropython-lib/blob/master/LICENSE

author='Paul Sokolovsky' license='MIT'

Software and Dependencies:

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

class adafruit_requests.**Response**(*sock*)
The response from a request, contains all the headers/content

close()
Close, delete and collect the response data

content
The HTTP content direct from the socket, as bytes

iter_content(*chunk_size=1*, *decode_unicode=False*)
An iterator that will stream data by only reading 'chunk_size' bytes and yielding them, when we can't buffer the whole datastream

json()
The HTTP content, parsed into a json dictionary

text
The HTTP content, encoded into a string according to the HTTP header encoding

adafruit_requests.**delete**(*url*, ***kw*)
Send HTTP DELETE request

adafruit_requests.**get**(*url*, ***kw*)
Send HTTP GET request

adafruit_requests.**head**(*url*, ***kw*)
Send HTTP HEAD request

adafruit_requests.**parse_headers**(*sock*)
Parses the header portion of an HTTP request/response from the socket. Expects first line of HTTP request/response to have been read already return: header dictionary rtype: Dict

adafruit_requests.**patch**(*url*, ***kw*)
Send HTTP PATCH request

adafruit_requests.**post**(*url*, ***kw*)
Send HTTP POST request

adafruit_requests.**put**(*url*, ***kw*)
Send HTTP PUT request

adafruit_requests.**request**(*method*, *url*, *data=None*, *json=None*, *headers=None*, *stream=False*, *timeout=1*)
Perform an HTTP request to the given url which we will parse to determine whether to use SSL ('https://') or not. We can also send some provided 'data' or a json dictionary which we will stringify. 'headers' is optional HTTP headers sent along. 'stream' will determine if we buffer everything, or whether to only read only when requested

adafruit_requests.**set_socket**(*sock*, *iface=None*)

Helper to set the global socket and optionally set the global network interface. :param sock: socket object. :param iface: internet interface object

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index