Projekt zu Micropython: Interaktion mit der Web Applikation

Sebastian Stigler

0 Die Webapplikation

Die Web Applikation ist über Nachrichten, die mit MQTT geschickt werden, steuerbar.

Auf dem Raspberry Pi Zero ist ein MQTT-Broker installiert, der auf allen IP Adressen des Rechners auf zwei Ports lauscht. Auf Port `1883` erwartet er MQTT im Standardprotokoll (das ist der Port an den sich später der Microcontroller verbindet) und auf Port `18083` erwartet er MQTT über das Websocketprotokoll, das von der Webapplikation gesprochen wird.

Rufen Sie die Adresse http://localhost im Browser auf Ihrem Raspberry Pi Zero auf und verbinden Sie sich in der Eingabemaske über das Websocketprotokoll mit dem MQTT-Broker.

Auf der linken Seite der Applikation ist beschrieben, welche Topics es gibt und wie die zugehörigen Nachrichten auszusehen haben. **Beachten Sie Groß-/Kleinschreibung!**

Mit dem `ifconfig` können Sie die IP Adresse Ihres Raspberry Pi Zero feststellen.

1 Zusammensetzen der Steuerung und Web App mit MQTT

Hier werden jetzt aus den vorangegangenen Aufgaben die Steuerung der Web App zusammengesetzt.

- Mit Hilfe der Tasten soll dabei in der Applikation navigiert werden.
- Der Helligkeitssensor steuer den Tag/Nacht Modus.
- Die `Application` Klasse interagiert mit WLAN und MQTT und dient als Vermittler zwischen Microcontroller und der Web Applikation.

1.1 Navigation mit den Tasten in der Web App

Verwenden Sie in `joystick.py` die Datei `switches.py` aus Aufgabe 1!

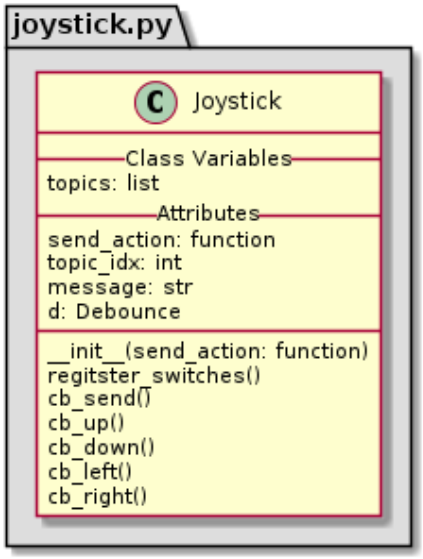

Abbildung 1: joystick.py

Klassenvariablen:

`topics` enthält eine Liste der Topics wie sie in der Webseite unter ***Structure of a topic*** und ***Room with applications*** beschrieben sind. Tragen Sie die Einträge in der selben Reihenfolge ein.

Benutzen Sie für die Einträge *Named Tuples*:

```
from ucollections import namedtuple

Topic = namedtuple('Topic', ['prefix', 'room', 'application'])

topic = Topic('haus', 'Kitchen', 'Oven')

print(topic.room)       # 'Kitchen'
print(topic[2])         # 'Oven'
print('/'.join(topic))  # 'haus/Kitchen/Oven'
```

Attribute:

`send_action` speichert die Funktion, die in `self.cb_send()` zum Senden benutzt wird. Diese Funktion wird später die MQTT Nachricht an die entsprechende Topic senden.

`topic_idx` ist der Index in `Joystick.topics`, für die aktuell ausgewählte Topic (siehe `cb_up()` und `cb_down()`).

`message` speichert die zu sendende Nachricht (siehe ***Messages*** in der Web App).

`d` enthält eine `Debounce` Instanz aus `switches.py`.

Methoden:

`__init__(send_action)` initialisiert die Attribute und ruft die `register_switches` Methode auf.

`register_switches` registriert die vier Schalter für die Joystickpositionen im `Debounce` Objekt. Beim Aktiviern des jeweiligen Schalters wird der zugehörig `cb_*` Methode aufgerufen; beim Loslassen wird immer die `cb_send` Methode aufgerufen.

`cb_send` generiert aus dem aktuellen `topic_idx` die Topic und sendet diese, zusammen mit der aktuellen `message` an die registrierte `send_action`.

`cb_up` / `cb_dowm` wählen den Eintrag in `topics` Liste über den `topic_idx` (**up** erhöht und **down** verringert den Index) aus und setzen die `message` auf ***selected***.

`cb_left` schaltet die gewählte Applikation ein bzw. öffnet das gewählte Fenster oder die gewählte Tür indem die `message` entsprechend gesetzt wird.

`cb_right` schaltet die gewählte Applikation aus bzw. schließt das gewählte Fenster oder die gewählte Tür indem die `message` entsprechend gesetzt wird.

Testen und Abgabe

Testen Sie Ihren Code indem Sie folgende `send_action` in der Klasse registrieren:

```
def send_action(topic, message):
    """dummy send message"""
    print('[sending] %s: %s' % (topic, message))
```

Die Datei `joystick.py` enthält die Klasse `Joystick`.

1.2 Messen der Helligkeit um in der Web App zwischen Tag- und Nachtansicht zu wechseln

Verwenden Sie in `daynight.py` die Datei `bh1750.py` aus Aufgabe 2!

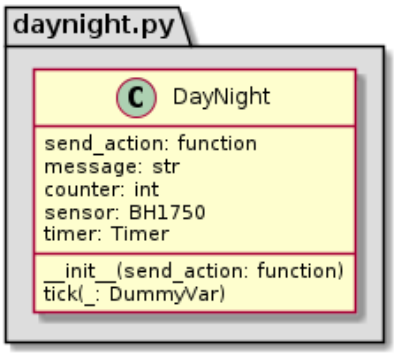

Abbildung 2: daynight.py

Attribute

`send_action` speichert die Funktion, die in `self.cb_send()` zum Senden benutzt wird. Diese Funktion wird später die MQTT Nachricht an die entsprechende Topic senden.

`message` speichert die zu sendende Nachricht (siehe ***Special Topic*** in der Web App).

`counter` wird benutzt, um spätestens jede Minute, den Tag/Nacht Status zu senden.

`sensor` speichert das `BH1750` Objekt des Helligkeitssensors.

`timer` ist ein `Timer` Objekt, das den Takt für das Einlesen des Sensors liefert.

Methoden:

`__init__(send_action)` initialisiert die Attribute. Der `Timer` soll seine Callback Methode jede Sekunde einmal aufrufen.

`tick(_)` dient als Callback Methode für den `Timer`. Sie führt folgende Schritte aus:

1. Helligkeit messen.
2. Ist es dunkler als 300 lx, so ist es Nacht, anderenfalls Tag.
3. Gab es eine Änderung (Tag->Nacht oder Nacht->Tag), so wird eine Nachricht mit dem aktuellen Tag/Nacht Status gesendet.
4. Wurde länger als eine Minute keine Nachricht gesendet, so wird ebenfallsder aktuelle Tag/Nacht Status gesendet.
5. Es darf an die eigentliche Nachricht mit Doppelpunkt getrennt der gemessen Helligkeitswert angehängt werden.

Testen und Abgabe

Testen Sie Ihren Code mit der `send_action` Funktion aus der vorherigen Aufgabe.

Abzugeben ist die Datei `daynight.py` mit den Klasse `DayNight`.

Bitte vermerken Sie die Namen und Matrikelnummern Ihrer Gruppenmitglieder sowie die Gruppennummer als Kommentar am Anfang der Datei.

1.3 Alles Zusammenführen

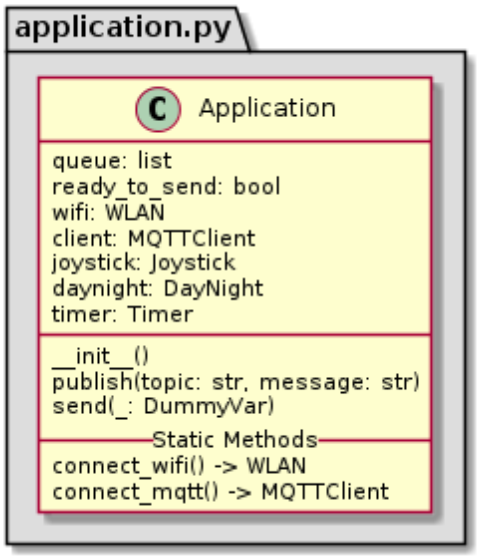

Abbildung 3: application.py

Attribute

`queue` ist eine Liste in die der `publish` Callback Werte am Ende einfügt (`.append(...)`) und der `send` Callback am Anfang (`.pop(0)`) ausließt.

`ready_to_send` zeigt an, ob der `send` Callback bereit ist, eine neue Nachricht zu senden.

`wifi` speichert das `WLAN` Objekt. Benutzen Sie die statische Methode `connect_wifi` um dieses Objekt zu erzeugen.

`client` enthält das `MQTTClient` Objekt. Benutzen Sie die statische Methode `connect_mqtt` um dieses Objekt zu erzeugen.

`joystick` speichert ein `Joystick` Objekt, das die `publish` Methode als `send_action` bekommt.

`daynight` speichert ein `DayNight` Objekt, das die `publish` Methode als `send_action` bekommt.

`timer` ruft periodisch alle 50 ms die `send` Methode auf.

Methoden

`__init__()` initialisiert alle Attribute in der angegebenen Reihenfolge.

`pubish(topic, message)` dient als Callback für das `Joystick` Objekt und das `DayNight` Objekt. Hier wird ein Tupel mit der `topic` und der `message` an die `qeueu` angehängt.

Schützen Sie das Anhängen mit `state = machine.disable_irq()` und `machine.enable_irq(state)`.

`send(_)` dient als Callback für das `Timer` Objekt. Die Funktion tut das folgende:

1. Wann immer die Queue einen Eintrag enthält und die Methode ist bereit ist, wird das folgende gemacht:
2. Setze die Methode auf nicht sendebereit.
3. Hole den ersten Eintrag aus der Queue und entpacke die `topic` und die `message`. Diese Aktion wird, wie bei `publish` geschützt.
4. Sende die `topic` und `message` via MQTT.
5. Setze die Methode wieder auf sendebereit.

Statische Methoden

`connect_wifi()` erzeugt ein `WLAN` Objekt indem eine Verbindung zu einem Wlan herstellt. Benutzen Sie die Variablen `config.SSID` und `config.WLAN_PASSWORD` aus der Datei `config.py`[1]

`connect_mqtt()` erzeugt das `MQTTClient` Objekt. Benutzen Sie auch hier die Einträge aus `config.py`.

Beispiel für eine statische Methode

```
class Beispiel:

    def normaleMethode(self):
        print(Beispiel.statischeMethode)

    @staticmethod
    def statischeMethode():
        return (
        "Ich habe keinen Parameter 'self' und es steht der Decorator "
        "'@staticmethod' vor der Definiton der Methode. "
        "Aufgrufen werde ich mit <Klassenname>.<Methodenname> "
        "und _NICHT_ mit <Instanzname>.<Methodenname>!"
    )
```

[1] Editieren Sie die `config.py` mit den Einträgen, die in der Vorlesung genannt werden.

Testen

Testen Sie Ihren Code mit der Web Applikation. Wenn nötig, dann können Sie in `send`, und in den `connect_*` Methoden eine **sinnvolle** `print` Ausgabe einfügen.

Fügen Sie in application.py die Zeile `app = Application()` am Ende mit ein, damit das Programm direkt mit einem `import application` gestartet werden kann.

Die Datei `application.py` enthält die Klasse `Application`.
