Lab 13 –Python / SenseHat (3)

1 Where is the Mosquitto broker?

1.1 When using the RPI to connect to a Mosquitto broker

We basically have two options:

- Connect to our locally installed broker (on your laptop)
- Connect to the broker in the lab (if you are in the lab) – "mosquitto.iktlab"

When installed, the Mosquitto broker will by default listen to port 1883 on all interfaces. That also means that your locally installed Mosquitto broker will be visible to all(!)

- You should consider setting "local services" for Mosquitto to "manual", and only run it when needed (windows)
- Enable/disable Mosquitto when not in use
- OR: You can edit the **mosquitto.conf** file to disable external listening
    - See **#bind_address** option – I have not tried it, but it ought to work

You may also choose to only run Mosquitto as a program (not a service).

1.2 How do you find the public IP address for the Mosquitto machine

1.2.1 IKT Lab broker

You don't need the IP address for the Mosquitto broker running at domain "mosquitto.iktlab". Here you just use "mosquitto.iktlab" is the hostname.

1.2.2 Local installations

On the machine where you have the broker installed:

Linux:

```
$ ifconfig
```

Look for **inet** addresses. The 127.0.0.1 is the localhost address.

Windows:

```
$ ifconfig
```

Look for the **IPv4 address**.

2 Advanced use of the paho API

```
[* Most of the code is included – You only need to do small changes *]
```

The following task is to report some of the environmental sensors of the RPI.

- A client on the RPI publishing sensor information
- A client on some machine (laptop) that subscribes to the topics
- The message payloads shall be json encoded (no extras, just the json encode info)

To use the more advanced parts of the paho API also means using callback functions. To do so effectively, one often uses event-loops. We will also need some sort of timer functions. To save you a little work, a simple event timer module is included (check it out).

- Using callback methods & asynch handling
- Use of event-loops
- Use of timer/timeout functionality

2.1 The Lab13ctrl & Lab13reader client

Two very basic clients that only reports the sensors and control the reporting.

These two programs are provided:

- make sure you understand the code
- see if you can improve the code (not difficult :-)

Lab13ctrl:

- Publish: True or False to the `start` topic (*retained* messages)
- Publish: "EXIT" to the `start` topic
- Publish: config info to the `config` topic

Best run from the command line[1]:

```
$ python Lab13ctrl.py
```

It may be run on any machine. If it runs on the same machine as the broker, then you may use "localhost".

[1] On the RPI it will have to be `python3`

Lab13reader:

- Subscribe: get data from the `temperature, pressure, humidity,` or `compass` topics
- Print the received data

Perhaps best to run from the command line[2]:

```
$ python Lab13reader.py
```

It may be run on any machine. If it runs on the same machine as the broker, then you may use "localhost".

Lab13common:

This is not a program, but a file to keep common definitions. Do investigate it.

[2] On the RPI it will have to be `python3`

2.2 The Lab13sensor.py client (on the RPI)

We need to write a python program that reads SenseHat sensors and publishes these to some defined topics. If you run this code on the Mosquitto server in the Lab, then **you must** replace dev-id with a unique identifier. And, the topics in

Topics:

TOPIC:	PAYLOAD:
dat235/lab13/dev-id/sensor/	
/temperature	json encoded temperature
/pressure	json encoded pressure
/humidity	json encoded humidity
/compass	json encoded direction
/config	json encoded [EV_<ev> + "ms"]
/start	json encoded **True** or **False**, **"EXIT"**

The EV_<ev> codes will be the EV_Temperature, EV_Pressure, EV_Humidity, EV_Compass string constants used in the SenseLogger code. See Lab13common.py. The "ms" is an integer that encoded the period (in milliseconds) between each reporting. A period of 0 (zero) means no reporting.

- "../**start**" messages shall be retained and have qos=1
- "../**config**" messages shall have qoS-1
- Other topics will have qos-0

- All payload data is json encoded
- All payload data received will be "encoded" as binary
  - The str() function does not mind
  - json in Python 3.6 and onwards automatically decodes this
  - json in Python 3.5.x (as in Raspbian) need explicit decoding

- Network coding
  - The whole of **on_message** and **on_connect** is enclosed in error catching code
  - **Errors in this code will be silently ignored**
  - It would be like your code running as the in `int(input("Int: "))` code below

Example: Silently ignore errors (also catches keyboard interrups)

```
def getInt():
    try:
        return(int(input("Int: ")))
    except:  # catches everything!
        return(None)
```

Main Lab12sensor.py logic (see code template):

```
Hello message
Initially() function
Subscribe to …/start topic & ../config topics
Setting up an EventTimer object

Main event loop:
  Client.loop
  Time.sleep

  If activestate:
      If timeout:
            Getevent & check tags
            If Temperature: publish & reschedule
            .. all events ..

  # activation command
  If CMD_Activate:
    If not activestate:
      activestate = True

      # check if period>0, publish and reschedule
      # do this for all sensors
  else:
    # go to passive state
    if activestate:
       cancel timers
       activestate = False

  # termination command
  if CMD_Terminate:
    break out of main event loop (which ends the program)

disconnect and terminate
```

Your task is the complete the missing code.