HCC Traffic Light Toy - How to

Rev. 1 — 3 Nov 2018

Document information

Info	Content
Keywords	HCC, SNUG, IoT, Development, Integration
Abstract	This application note attempts to provide the reader with the necessary information to program their SNUG-2018 Traffic Light Toy, to expand on the basic capabilities provided during the Morning Tea exercise at SNUG Hamilton 2018.

Revision history

Rev	Date	Description
1	20181103	Initial version.

1. Overview

Hamilton City Council requested that Opito design a small exercise for SNUG participants to complete during morning tea of day 1 at the SNUG conference hosted in Hamilton.

Opito gladly developed the exercise, however due to time constraints the full potential of the electronic hardware provided was not exposed during the conference.

This application note will show how to expand on the capabilities of the hardware, while allowing the reader the opportunity to reprogram the device to meet their specific requirements and hopefully building something awesome!

Where possible we will include all web links and source code within this document as a package of supporting files invariably gets separated, making this document useless.

This app-note is targeted toward Windows users.

2. The Hardware

2.1 NodeMCU – WiFi enabled processor

Before we begin, I must confess to not having used this device previously, we purchased it off the internet and were attracted to it based on its capabilities and price. In general, the programming of it was reasonably straight forward if a little flaky, and the quality of boards was OK, although we had about a 20% OOB (Out of box) failure rate, we never tried to recover or repair these faulty boards, so the fix on them could be quite straight forward.

The module is known as a NodeMCU, this in turn has a processor fitted to the module called an ESP8266.

The NodeMCU comes by default with a scripting language called LUA. We wiped this immediately and installed a version of the programming language call Python, as we thought that this might be more widely used in the industry, and therefore the learning curve for those familiar with Python would be lower.

2.1.1 Connecting to your device

A prerequisite to program ESP8266-based modules is to establish a communications channel from your PC, to the device over USB. This in turn requires your system to detect the USB-to-Serial (aka USB-to-TTL, aka USB-to-UART) adapter on the ESP8266 module.

The chip on the NodeMCU that performs this USB to UART is manufactured by Silabs, they have a driver for your PC here.

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

2.1.2 Resetting your device back to factory defaults

You can put the code back to a factory version using this tool, we used the command line version of this tool to load Micropython on to the module.

https://github.com/marcelstoer/nodemcu-pyflasher/releases

2.1.3 Wiring

The eagle eyed among you would have noticed that we paralleled up a couple of resistors on the Green LED to get enough brightness out of them for the day, we did it this way as it was simpler to do this, rather than having a number of different value resistors for you all to sort through, you are welcome to change them for something more suitable should you wish to.

Additionally, as we were supplying the power from a couple of AA batteries we bypassed the on board voltage regulator, and “back fed” the power supply into the output of the regulator, this is not ideal as you run the risk of blowing up the chip if you supply too much voltage to it. We were never going to be able to this with the AA batteries so were fine. If you want to supply it with a proper power supply please use the VIN pin on the module.

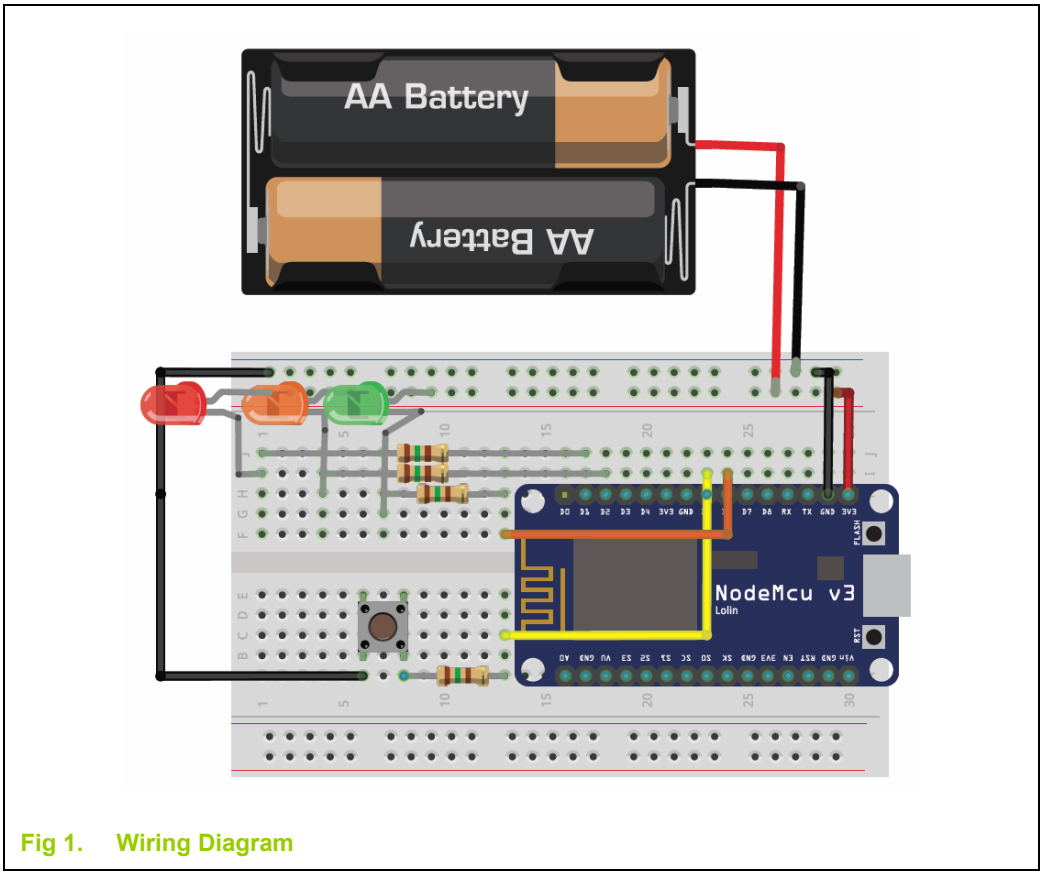

Fig 1. Wiring Diagram

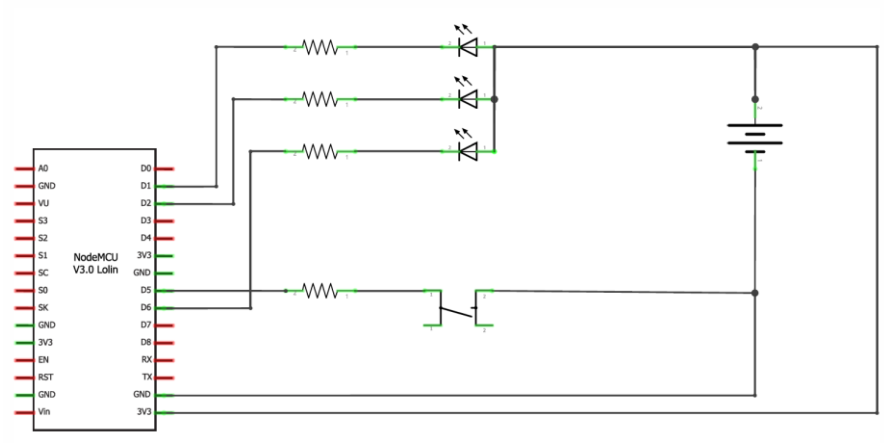

Fig 2. Wiring Schematic

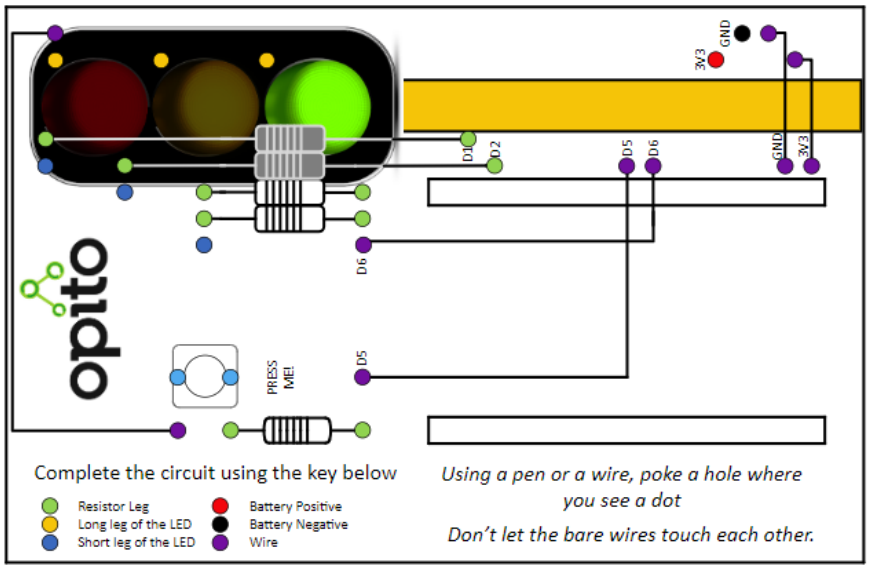

Fig 3. Wiring Overlay

3. Getting Started

A USB cable will power the module, you also don't need to have the LEDs and button connected to make it work, just the USB is enough to get you going, the LEDs might be handy for determining if something is working though.

3.1 Development Environment

Download and install your preferred development environment, we used Visual Studio Code, which is reasonably lightweight and full featured. https://code.visualstudio.com/

You are welcome to use an editor of your choice however, Atom is also popular https://atom.io/ as is Notepad++ (which is the best notepad replacement ever!) https://notepad-plus-plus.org/

Our examples below assume you are using Visual Studio Code.

After downloading and installing VS Code you have a window that looks like this. Click on the extensions icon on the left side bar, circled in the picture below.

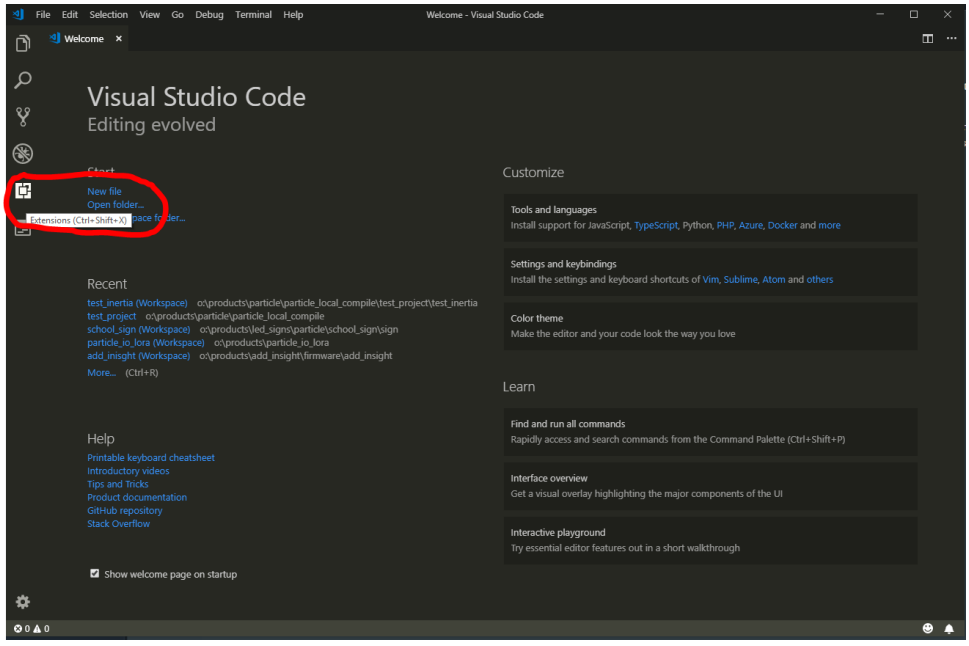

Fig 4. Visual Studio Code

Now install the pymakr extension by typing pymakr in the search box as illustrated.

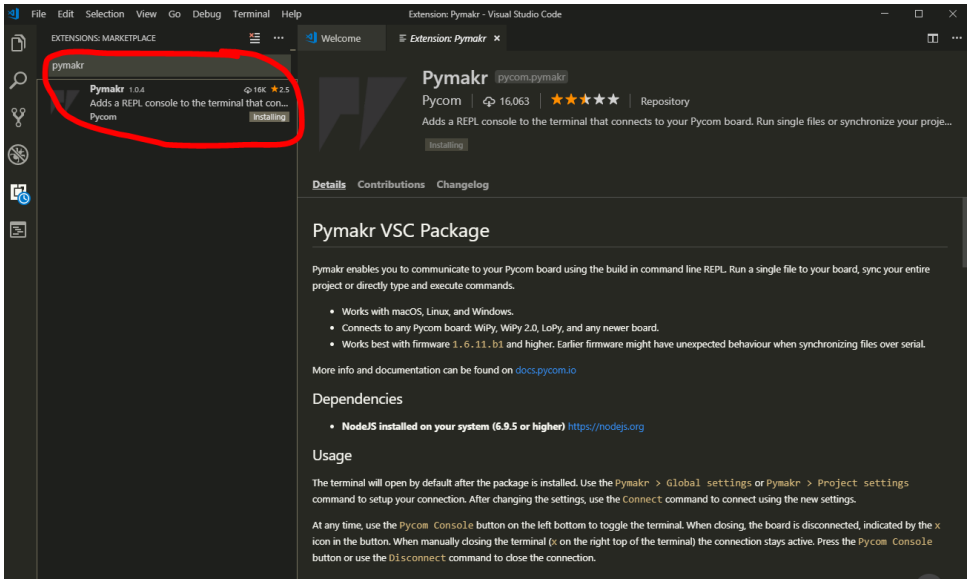

Fig 5. Installing pymakr extension

Once installed there are instructions on how to use it and the commands available on the right-hand window. When the installation is completed, VS code will need to be reloaded to activate the pymakr extension. The install button, changes to reload, click on this and we are all ready to go.

3.1.1 Creating a pymakr.conf file

We need to make a file named pymakr.conf in our project. This tells VS code where the USB communications port is located and how to connect to the module. If you want you are also able to connect to the module through the WiFi, however you will have to reconfigure your PC network interface to do this and connect to the module on 192.168.4.1, and it is outside the scope of this application note.

The first thing that we have to do is create a directory for our project to reside in, on our computer.

Step 1 is to click on the file explorer side bar, as shown below,

Step 2 is to open a folder where we will store our project. We created our project directory at Desktop\opito_traffic_light you can save it wherever you want of course.

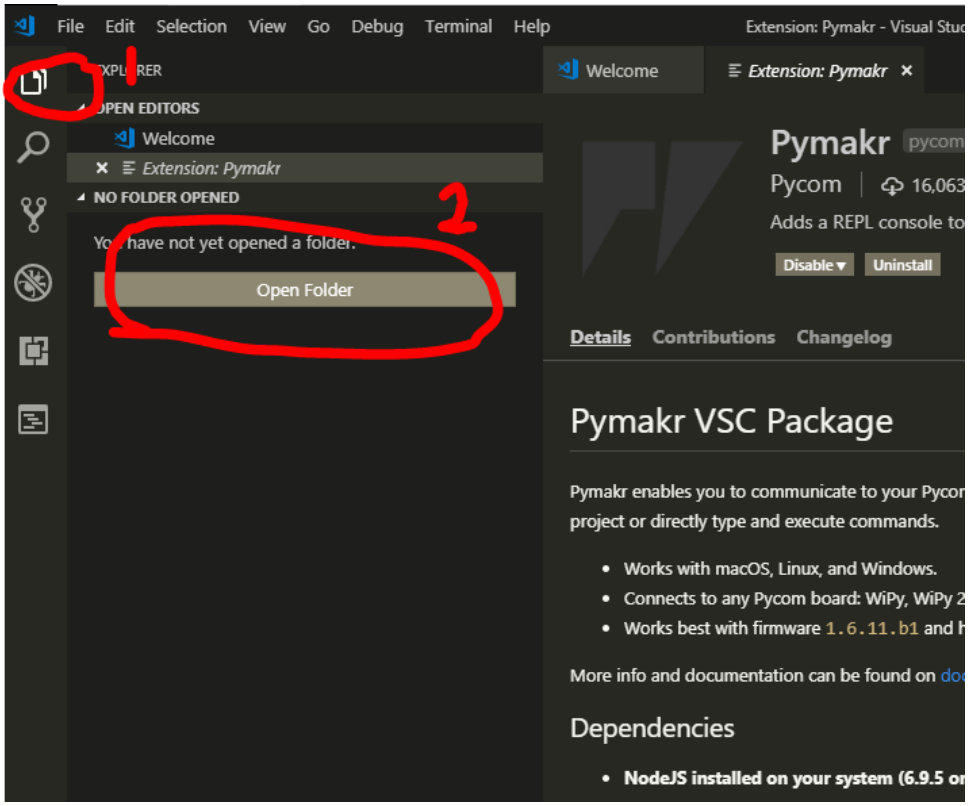

Fig 6. Creating a Project Directory

Step 3 create the pymakr.conf file.

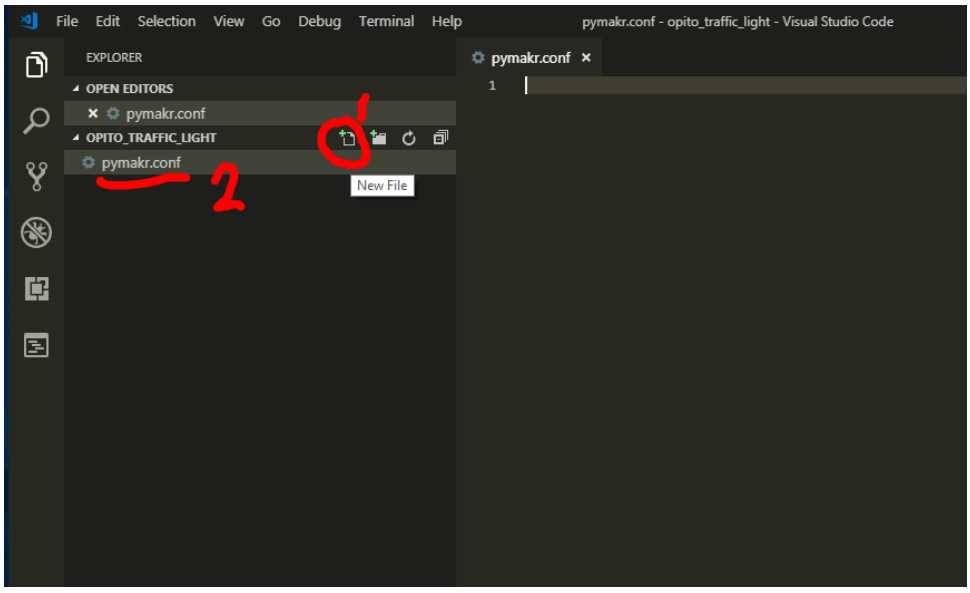

Fig 7. Creating a new file

Then paste the following code into the pymakr.conf file

```
//-----------BEGIN COPY (DON'T INCLUDE THIS LINE) -----------//
{
    "address": "COM93",
    "username": "micro",
    "password": "python",
    "sync_folder": "",
    "open_on_start": true,
    "safe_boot_on_upload": false,
    "sync_file_types": "py,txt,log,json,xml,html,js,css,mpy",
    "ctrl_c_on_connect": false,
    "sync_all_file_types": false,
    "auto_connect": true
}
//-----------END COPY (DON'T INCLUDE THIS LINE) -----------//
```

Please note – the address line is the comport number that your nodeMCU module has enumerated as on Windows – YOU WILL NEED TO EDIT THIS. It can be a little hard to find this comport number.

We find the easiest way to locate this information is to click on the Windows (10) start button and simply start typing "Bluetooth" it will bring up the option for "Bluetooth and other devices settings". Select this menu options and you should see a SiLabs CP2102 (COMxx) device listed there. This is the comport number you want.

Finally save the pymakr.conf file, and your window should look like this.

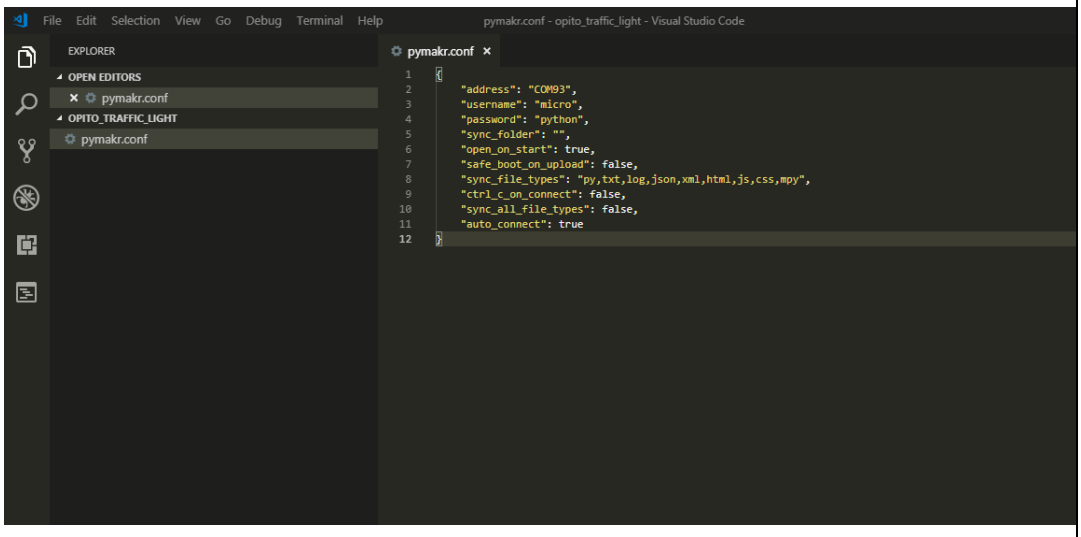

Fig 8. Pymakr.conf file

3.1.2 Creating main.py

Now we create our application file in the same manner we created the configuration file, we have to call it main.py as this is a special name for the module to run automatically on boot. There is another file called boot.py which you can use for configuration and setup data as well, but we are not using that in our demo.

3.1.2.1 Main.py Simple

This code is the code that was loaded to your devices prior to leaving the conference, if you didn't get your code updated then you will want to load this code as a reference point.it will work without an active WiFi connection, which is probably what you want if you are setting it up at home for the first time.

```
//-----------BEGIN COPY (DON'T INCLUDE THIS LINE) -----------//
SNUG Demo  1 Nov 2018
#no copyright use as you see fit.
#no warranty, and no support, we strung it together quick.
#Regards
#Marshall
#
#
#       www.opito.io
#       021 748 703

import time
import sys; print(sys.version_info)
import esp
import urandom
import machine
import ubinascii
from machine import Pin
from machine import Timer

#pinouts for the Node.MCU module
#D0 = Pin(16, Pin.OUT)
#D1 = Pin(5, Pin.OUT)
#D2 = Pin(4, Pin.OUT)
#D3 = Pin(0, Pin.OUT)
#D4 = Pin(2, Pin.OUT) //also the blue LED on the Wifi
#D5 = Pin(14, Pin.OUT)
#D6 = Pin(12, Pin.OUT)
#D7 = Pin(13, Pin.OUT)
#D8 = Pin(15, Pin.OUT)
```

```
#RX = Pin(3, Pin.OUT)
#TX = Pin(1, Pin.OUT)
#A0 = Pin(ADC0, Pin.OUT)
#SD3 = Pin(10, Pin.OUT)
#SD2 = Pin(9, Pin.OUT)
#SD1 = Pin(MOSI, Pin.OUT)
#CMD = Pin(CS, Pin.OUT)
#SDO = Pin(MISO, Pin.OUT)
#CLK = Pin(SCLK, Pin.OUT)


BLU_LED = Pin(16, Pin.OUT)
RED_LED = Pin(5, Pin.OUT)
YEL_LED = Pin(4, Pin.OUT)
GRN_LED = Pin(12, Pin.OUT)
BUTTON = Pin(14, Pin.IN, Pin.PULL_UP)
WIFI_LED = Pin(2, Pin.OUT)

OFF_VAL = 3
RED_VAL = 2
YEL_VAL = 1
GRN_VAL = 0

tmr_blink = Timer(-1)
tmr_debounce = Timer(-1)
tmr_service = Timer(-1)     #used as an independent main() in case we block
trying to connect to WiFi (we still want our LEDs to blink away.)
tmr_turn_off_leds = Timer(-1)


#stupid python doesn't support static
debounce_count = 0
last_button = 0
button_state = 0
this_led = 0
last_led = 0
led_incremented = 0
leds_timeout = False
boot_complete = False
start_time = 0
hold_time = 0
random_led = False
first_run = True
```

```
def tmr_callback(self):
#    print ('tmr fired')
    BLU_LED.value(not BLU_LED.value())
    if BLU_LED.value() == 0:
        tmr_blink.init(period=50, mode=Timer.PERIODIC, callback=tmr_callback)
    else:
        tmr_blink.init(period=950, mode=Timer.PERIODIC, callback=tmr_callback)


def tmr_turn_off_leds_callback(self):
    global leds_timeout
    global last_led
    last_led = 27
    leds_timeout = True



def tmr_debounce_callback(self):
    global last_button
    global debounce_count
    global button_state
    global this_led
    global led_incremented
    global leds_timeout
    global start_time
    global hold_time
    global first_run

    if last_button == BUTTON.value():

        debounce_count = debounce_count + 1
        if debounce_count >= 5:
            debounce_count = 6
            button_state = not BUTTON.value()
            if led_incremented == 0:
                led_incremented = 1
                if BUTTON.value() == 0: #active low (comment this if you want
change of state.)
                    leds_timeout = False
                    start_time = time.time()
```

```
                    print("button pressed ", this_led)

                if BUTTON.value() == 1:
                    if first_run == True:
                        first_run = False
                    else:
                        print("button released ", this_led)
                        hold_time = time.time() - start_time
                        print("button held for " + str(hold_time) + " seconds")

                        if hold_time >= 2: #held down
                            random_led = True
                            print("random , set to " + str(random_led))
                            randomise()
                        else:
                            random_led = False
                            this_led = this_led + 1
                            if this_led >= 4:
                                this_led = 0


    else:
        debounce_count = 0
        led_incremented = 0

    last_button = BUTTON.value()


#stupid python doesn't have fwd decl so this has to be init'd here.
tmr_blink.init(period=500, mode=Timer.PERIODIC, callback=tmr_callback)
tmr_debounce.init(period=10, mode=Timer.PERIODIC,
callback=tmr_debounce_callback)


def randomise():
    global this_led
    global leds_timeout
    leds_timeout = False
    rand_num = urandom.getrandbits(2)

    if rand_num < 1.33:
        rand_num = 0
    elif rand_num > 1.33 and rand_num < 2.66 :
        rand_num = 1
    else:
```

```
        rand_num = 2

    this_led = rand_num
    if this_led > 2:
        this_led = 2

    #cycle
    start_time = time.time()
    rand_num = urandom.getrandbits(3)
    print(rand_num)
    while (time.time() - start_time < rand_num):
        time.sleep_ms(50)
        RED_LED.value(1)
        YEL_LED.value(1)
        GRN_LED.value(0)
        time.sleep_ms(50)
        RED_LED.value(1)
        YEL_LED.value(0)
        GRN_LED.value(1)
        time.sleep_ms(50)
        RED_LED.value(0)
        YEL_LED.value(1)
        GRN_LED.value(1)



def tmr_service_callback(self):
    global last_led
    global this_led
    global leds_timeout
    #this is on a 10mSec call back for general servicing.
    if leds_timeout == True:  #set from a callback timer when the button is
released.
        RED_LED.value(1)
        YEL_LED.value(1)
        GRN_LED.value(1)
    else:
        if this_led != last_led:
            last_led = this_led
            tmr_turn_off_leds.init(period=60000, mode=Timer.ONE_SHOT,
callback=tmr_turn_off_leds_callback)
            if this_led == RED_VAL:
                RED_LED.value(0)
                YEL_LED.value(1)
```

```
                GRN_LED.value(1)

            if this_led == YEL_VAL:
                RED_LED.value(1)
                YEL_LED.value(0)
                GRN_LED.value(1)

            if this_led == GRN_VAL:
                RED_LED.value(1)
                YEL_LED.value(1)
                GRN_LED.value(0)

            if this_led == OFF_VAL:
                RED_LED.value(1)
                YEL_LED.value(1)
                GRN_LED.value(1)


tmr_service.init(period=10, mode=Timer.PERIODIC, callback=tmr_service_callback)


def main():
    WIFI_LED.value(1)
    print("code ver 11")
    print("All done running off timers.")


if __name__ == "__main__":
    main()
```

//----------END COPY (DON'T INCLUDE THIS LINE) ----------//

3.1.2.2 Main.py With WiFi

This is the exact code that was running at SNUG, it was actually connecting to a WiFi Access point and counting the button presses, and we were able to turn on the LEDs on your boards remotely as well! However, it wasn't fully utilized at the time as we had some network capacity issues that needed to be ironed out. It was still really cool though so we include for your reference.

//----------BEGIN COPY (DON'T INCLUDE THIS LINE) ----------//

```
#SNUG Demo  1 Nov 2018
#no copyright use as you see fit.
#no warranty, and no support, we strung it together quick.
#Regards
#Marshall
#
#
#       www.opito.io
#       021 748 703



import time
import sys; print(sys.version_info)
import esp
import urandom
import machine
import network
import ubinascii
from machine import Pin
from machine import Timer
from umqtt.robust import MQTTClient


#pinouts for the Node.MCU module
#D0 = Pin(16, Pin.OUT)
#D1 = Pin(5, Pin.OUT)
#D2 = Pin(4, Pin.OUT)
#D3 = Pin(0, Pin.OUT)
#D4 = Pin(2, Pin.OUT) //also the blue LED on the Wifi
#D5 = Pin(14, Pin.OUT)
#D6 = Pin(12, Pin.OUT)
#D7 = Pin(13, Pin.OUT)
#D8 = Pin(15, Pin.OUT)
#RX = Pin(3, Pin.OUT)
#TX = Pin(1, Pin.OUT)
#A0 = Pin(ADC0, Pin.OUT)
#SD3 = Pin(10, Pin.OUT)
#SD2 = Pin(9, Pin.OUT)
#SD1 = Pin(MOSI, Pin.OUT)
#CMD = Pin(CS, Pin.OUT)
#SDO = Pin(MISO, Pin.OUT)
#CLK = Pin(SCLK, Pin.OUT)


BLU_LED = Pin(16, Pin.OUT)
RED_LED = Pin(5, Pin.OUT)
YEL_LED = Pin(4, Pin.OUT)
GRN_LED = Pin(12, Pin.OUT)
BUTTON = Pin(14, Pin.IN, Pin.PULL_UP)
WIFI_LED = Pin(2, Pin.OUT)
TOPIC = b"+/opito/snug/demo/set"
```

```
OFF_VAL = 3
RED_VAL = 2
YEL_VAL = 1
GRN_VAL = 0


tmr_print = Timer(-1)
tmr_connect = Timer(-1)
tmr_debounce = Timer(-1)
tmr_service = Timer(-1)     #used as an independent main() in case we block
trying to connect to WiFi (we still want our LEDs to blink away.)
tmr_turn_off_leds = Timer(-1)
tmr_send_mqtt = Timer(-1)

#stupid python doesn't support static
debounce_count = 0
last_button = 0
button_state = 0
this_led = 0
last_led = 0
led_incremented = 0
leds_timeout = False
boot_complete = False
start_time = 0
hold_time = 0
random_led = False
first_run = True
my_event = "blank"
flg_vote = False
flg_rdy_for_mqtt = False
wlan = network.WLAN(network.STA_IF)







CLIENT_ID = ubinascii.hexlify(machine.unique_id())
SERVER = "192.168.5.253"


def translate(value, leftMin, leftMax, rightMin, rightMax):
    leftSpan = leftMax - leftMin
    rightSpan = rightMax = rightMin

    scaledValue = float(value - leftMin) / float(leftSpan)

    return rightMin + (scaledValue * rightSpan)


# Receive topic
def opito_mqtt_receive_cb(topic, msg):
```

```
    global this_led
    global leds_timeout
    print((topic, msg))
    leds_timeout = False

    if msg == b"RED_ON":
        this_led = RED_VAL
        print("receive red: " + str(RED_VAL))

    if msg == b"YEL_ON":
        this_led = YEL_VAL
        print("receive yel")

    if msg == b"GRN_ON":
        this_led = GRN_VAL
        print("receive green")

    if msg == b"OFF":
        this_led = OFF_VAL
        print("receive off")


def tmr_turn_off_leds_callback(self):
    global leds_timeout
    global last_led
    last_led = 27
    leds_timeout = True



def tmr_send_mqtt_callback(self):
    opito_mqtt_vote()


def opito_mqtt_vote():
    global my_event
    global flg_vote
    global this_led
    my_event = "{\"vote\":" + str(this_led) + "}"

    flg_vote = True

def tmr_debounce_callback(self):
    global last_button
    global debounce_count
    global button_state
    global this_led
    global led_incremented
    global leds_timeout
    global start_time
    global hold_time
    global first_run
```

```
    if last_button == BUTTON.value():

        debounce_count = debounce_count + 1
        if debounce_count >= 5:
            debounce_count = 6
            button_state = not BUTTON.value()
            if led_incremented == 0:
                led_incremented = 1
                if BUTTON.value() == 0: #active low (comment this if you want
change of state.)
                    leds_timeout = False
                    start_time = time.time()
                    #tmr_send_mqtt.deinit()
                    print("button pressed ", this_led)

                if BUTTON.value() == 1:
                    if first_run == True:
                        first_run = False
                    else:
                        print("button released ", this_led)
                        hold_time = time.time() - start_time
                        print("button held for " + str(hold_time) + " seconds")

                        if hold_time >= 2: #held down
                            random_led = True
                            print("random , set to " + str(random_led))
                            randomise()
                        else:
                            random_led = False
                            print("voting")
                            this_led = this_led + 1
                            if this_led >= 4:
                                this_led = 0


                            if this_led < 3:
                                tmr_send_mqtt.init(period=5000,
mode=Timer.ONE_SHOT, callback=tmr_send_mqtt_callback)

    else:
        debounce_count = 0
        led_incremented = 0

    last_button = BUTTON.value()

#stupid python doesn't have fwd decl so this has to be init'd here.
tmr_debounce.init(period=10, mode=Timer.PERIODIC,
callback=tmr_debounce_callback)

def randomise():
    global this_led
    global leds_timeout
    leds_timeout = False
```

```
    rand_num = urandom.getrandbits(2)

    if rand_num < 1.33:
        rand_num = 0
    elif rand_num > 1.33 and rand_num < 2.66 :
        rand_num = 1
    else:
        rand_num = 2

    this_led = rand_num
    if this_led > 2:
        this_led = 2

    #cycle
    start_time = time.time()
    rand_num = urandom.getrandbits(3)
    print(rand_num)
    while (time.time() - start_time < rand_num):
        time.sleep_ms(50)
        RED_LED.value(1)
        YEL_LED.value(1)
        GRN_LED.value(0)
        time.sleep_ms(50)
        RED_LED.value(1)
        YEL_LED.value(0)
        GRN_LED.value(1)
        time.sleep_ms(50)
        RED_LED.value(0)
        YEL_LED.value(1)
        GRN_LED.value(1)
    tmr_send_mqtt.init(period=5000, mode=Timer.ONE_SHOT,
callback=tmr_send_mqtt_callback)



def tmr_service_callback(self):
    global last_led
    global this_led
    global leds_timeout
    #this is on a 10mSec call back for general servicing.
    if leds_timeout == True:  #set from a callback timer when the button is
released.
        RED_LED.value(1)
        YEL_LED.value(1)
        GRN_LED.value(1)
    else:
        if this_led != last_led:
            last_led = this_led
            tmr_turn_off_leds.init(period=60000, mode=Timer.ONE_SHOT,
callback=tmr_turn_off_leds_callback)
            if this_led == RED_VAL:
                RED_LED.value(0)
                YEL_LED.value(1)
```

```
                GRN_LED.value(1)

            if this_led == YEL_VAL:
                RED_LED.value(1)
                YEL_LED.value(0)
                GRN_LED.value(1)

            if this_led == GRN_VAL:
                RED_LED.value(1)
                YEL_LED.value(1)
                GRN_LED.value(0)

            if this_led == OFF_VAL:
                RED_LED.value(1)
                YEL_LED.value(1)
                GRN_LED.value(1)

    if boot_complete == True:
        WIFI_LED.value(last_button)


def tmr_connect_callback(self):
    global wlan
    print("checking WLAN")

    if wlan.isconnected() == False:
        do_connect()


tmr_service.init(period=10, mode=Timer.PERIODIC, callback=tmr_service_callback)
tmr_connect.init(period=30000, mode=Timer.PERIODIC,
callback=tmr_connect_callback)


def do_connect():
    print("actually reconnecting")
    global wlan
    global flg_rdy_for_mqtt
    ssid = "opito-2400"
    password = "123abcd123temp"

    wlan = network.WLAN(network.STA_IF)

    if wlan.isconnected() == True:
        print("Already connected")
        return

    print('connecting to network...')
    wlan.active(True)
    wlan.connect(ssid,password)
    while wlan.isconnected() == False:
        WIFI_LED.value(not WIFI_LED.value())
        time.sleep_ms(50)
```

```
        #pass

    print("connection successful")
    print('network config:', wlan.ifconfig())
    WIFI_LED.value(1);
    flg_rdy_for_mqtt = True

# Receive topic
def opito_mqtt_receive_cb(topic, msg):
    global this_led
    global leds_timeout
    print((topic, msg))
    leds_timeout = False

    if msg == b"RED_ON":
        this_led = RED_VAL
        print("receive red: " + str(RED_VAL))

    if msg == b"YEL_ON":
        this_led = YEL_VAL
        print("receive yel")

    if msg == b"GRN_ON":
        this_led = GRN_VAL
        print("receive green")

    if msg == b"OFF":
        this_led = OFF_VAL
        print("receive off")




def tmr_print_callback(self):
    print("Current LED is " + str(this_led))



def main():
    print("code ver 9")
    global flg_vote
    global flg_rdy_for_mqtt
    do_connect()


    opito_mqtt = MQTTClient(CLIENT_ID,SERVER)
    opito_mqtt.set_callback(opito_mqtt_receive_cb)
    opito_mqtt.connect()

    opito_mqtt.subscribe(TOPIC)
    print("Connected to %s, subscribed to %s topic" % (SERVER, TOPIC))

    opito_mqtt.publish(bytes(CLIENT_ID + "/opito/snug/demo/evt") ,b"online")
```

```
    boot_complete = True

    while True:

        if flg_vote == True:
            flg_vote = False
            print("sending MQTT event", (my_event))
            opito_mqtt.publish(bytes(CLIENT_ID + "/opito/snug/demo/evt"),
bytes(my_event,'utf-8'))

        if flg_rdy_for_mqtt == True:
            flg_rdy_for_mqtt = False
            print("hmm we are mqtt reconnecting")
            time.sleep_ms(50)
            opito_mqtt.disconnect()
            time.sleep_ms(50)
            opito_mqtt.connect()
            opito_mqtt.subscribe(TOPIC)
            print("Connected to %s, subscribed to %s topic" % (SERVER, TOPIC))

        opito_mqtt.check_msg()
        time.sleep(1)

    print("All done.")


if __name__ == "__main__":
    main()
```

```
//-----------END COPY (DON'T INCLUDE THIS LINE) -----------//
```

3.1.3 Close and Reopen VS Code

We found when writing this application note that it was necessary to close VS Code and relaunch it to get the pymakr extension to load properly.

Once VS Code restarts it opens another window at the bottom of the screen, this is called the terminal window, and is the communications between your PC and NodeMCU module.

3.1.4 Sending Code to your NodeMCU

Once we get this far, we are ready to connect to our module and send it some code. As python is an interpreted language, we don't need to run through a compile process, simply send it the code and we are ready to run.

At the bottom of the VS code window is a new small tool bar, the "pycom console" button allows the connection and disconnection of the module, it should have a tick beside it if its connected.

Step 1 Click on the "Pycom Console" button if it doesn't' have a tick beside it.

Step 2 You should see three >>> symbols appear at the console prompt, this is the module waiting to accept commands, if you know a little bit about python you can issue commands at this prompt.

Step 3 Press the “Upload” button, this sends your code to the module, all going well we have some code running!

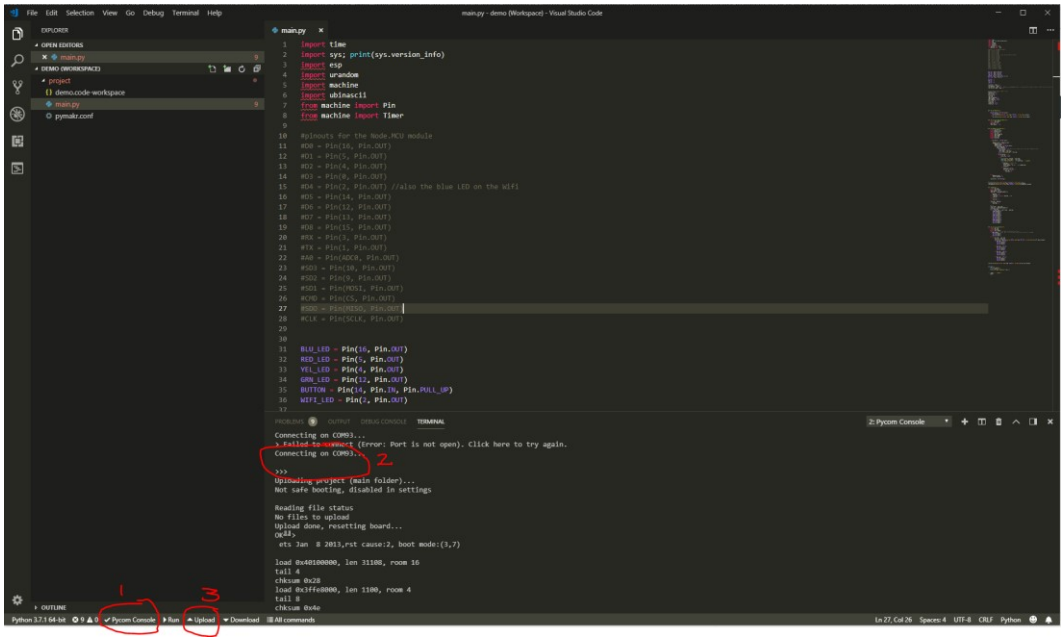

Fig 9. Connecting to your module

If your code is running properly you should see the LEDs turn on and a little bit of debug information appear at the console as shown below – each time you press (and release) the button you get a little bit of debug to provide some confidence that your code is running.

Each time we make a change to the code we update the code version string so that when we update the module we know that the code is the new version, we had a few issues connecting to the device when we initially started and this provided a confidence that the code was what we thought

```
def main():
    WIFI_LED.value(1)
    print("code ver 11")
    print("All done running off timers.")
```

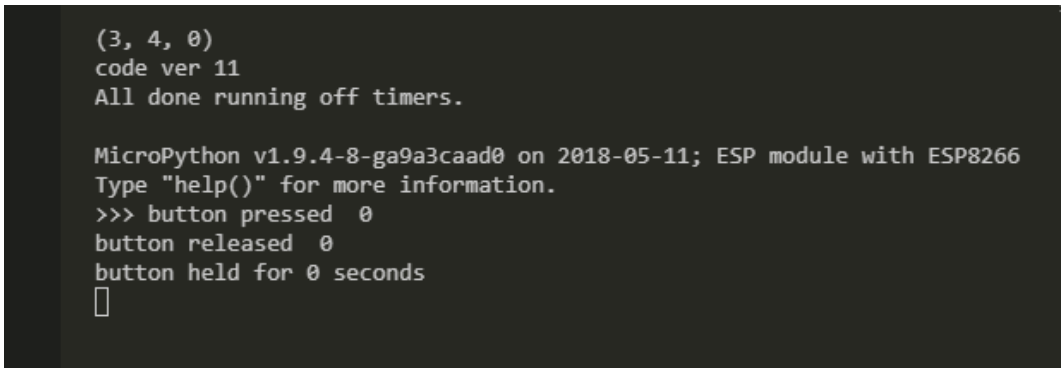

Fig 10. Code up and running

And that's it! Hopefully this is enough to get you going, its by no means an exhaustive document. There are a heap of forums and related resources on the web, and we have included a few of the links we found useful in the references section below.

Feel free to expand on the example code, as you see fit, and control your watering systems, traffic lights, garage doors, or anything else.

Should you need to get in touch with us we can offer limited support.

Marshall Brown

marshall@opito.io

021 748703

07 957 0188

4. All Done

All Done !

5. References

5.1 ESP8266

The chip you are actually programming on the NodeMCU module

https://en.wikipedia.org/wiki/ESP8266

5.2 NodeMCU

Link to the NodeMCU website

http://nodemcu.com/index_en.html

Link to more resources on Wikipedia

https://en.wikipedia.org/wiki/NodeMCU

Node MCU pinouts

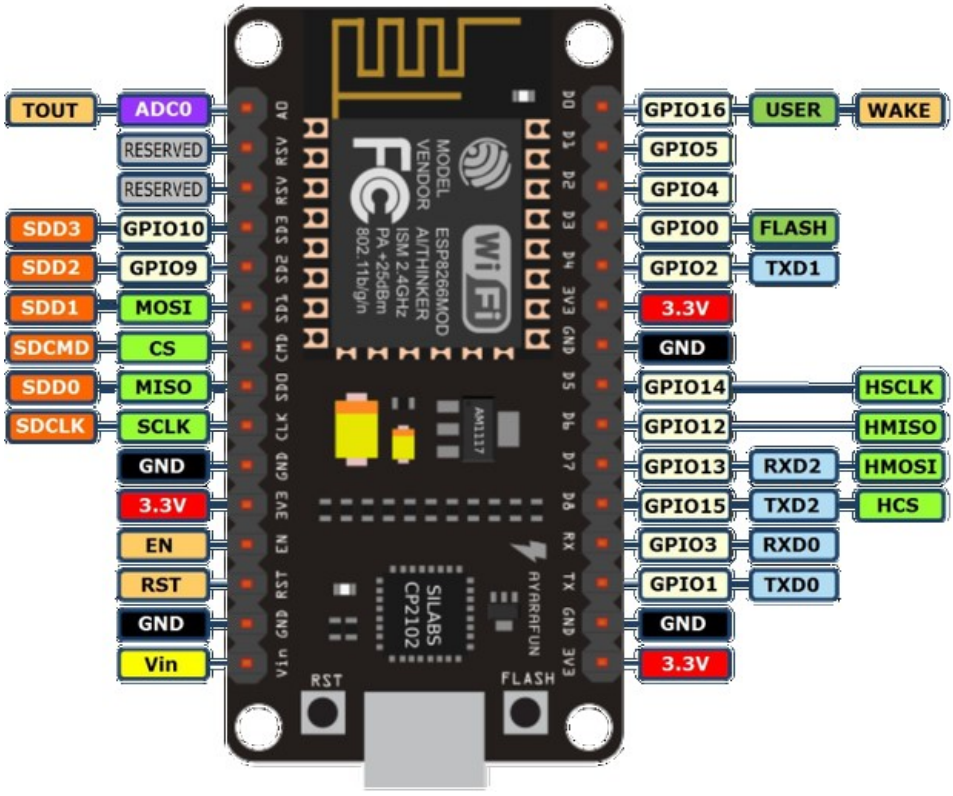

Fig 11. NodeMCU pinout

5.2.1 Aliexpress

To purchase additional units this is where we purchased them from. Make sure you get the ones with the CP2102 USB to serial device on them.

https://www.aliexpress.com/item/V3-Wireless-module-NodeMcu-4M-bytes-Lua-WIFI-Internet-of-Things-development-board-based-ESP8266-for/32469441553.html

5.3 Micropython

The programming language we used on the demo
https://micropython.org/

The binary image loaded into the NodeMCU module we used the latest build esp8266-20180511-v1.9.4.bin at the time of writing.

Here is the link to all the images available (there might be a newer one)
https://micropython.org/download#esp8266

Quick reference to the feature set and API for extending our examples.
http://docs.micropython.org/en/latest/esp8266/quickref.html

How to install Micropython on your NodeMCU module
http://docs.micropython.org/en/latest/esp8266/tutorial/intro.html#intro

5.4 Visual Studio Code

We use Visual Studio Code for writing the software. There is an extension called pymakr which allows you to connect and upload the code to the module. There are many other extensions which can aide you in Syntax highlighting etc, however this is left as an exercise for the reader.
https://code.visualstudio.com/

5.5 Fritzing

Simple breadboard drawing tool for laying out your designs
http://fritzing.org/home/

6. Contents