FISH-PI V1.0 – By Michael van den Heever

http://Piworx.subzerobc.com

What the heck is FISH-PI?

Let's start with name shall we, I thought the name would be appropriate with it's relation to Britain or the UK for that matter. Since the Pi was created in the UK I aptly named my project after a traditional British dish. It just so happens my project has “allot” to do with the Raspberry Pi and real Fish.

Essentially FISH-PI is a soup of technologies enabling me with the ability to manage my salt water aquarium remotely via the internet. If you've ever owned a salt water aquarium, you will undoubtedly know that it's a very high maintenance hobby. Any form of automation is most welcome. One of the critical aspects of salt water aquariums and keeping corals is light management. Too little or too much and your in for growing more algae than you are corals.

There are also a few stages of lighting I control throughout the day as well as types of lighting. The amount of time certain lights are on can affect many factors with in the fish tank. I needed intelligent “control” here .. not simply a mechanical on and off timer switch ... in short I needed a Pi.

What I wanted to achieve in this project

1. Have remote control ability.
   (Manage my tank lights from any internet capable device anywhere in the world)
2. Some intelligence in that the code that controlled my lights knew what time of the day it was and adjusted my lighting accordingly. (This would enable my system to power up the correct light setup for the right time of day)
3. I wanted to know when a power cut took place and be notified via email.
4. From a thermal perspective I wanted to know what the temperature of the water was, as well as keep an eye on the thermal health of the Raspberry Pi.
5. I needed my lights to not only have an automated schedule but have the ability to flip to manual control for light calibration or testing.
6. Since I would have no screen attached to the Pi I wanted some form visual recognition that all was ok.
7. Visual confirmation via web-cam would be a bonus.
8. Finally I wanted all this packaged into a responsive website that ran on the Pi itself.

Now up front, I would like to iterate at this point that I wanted to “learn” through this process. I know many fine developers that would have easily been able to provide me with coding needed that would be refined and flawless. But it wouldn’t be half as satisfying as doing it yourself.

No matter how looney your ideas are for the Pi “PERSUE” them they will become a reality.

FISH-Pi's Capability and Function Outlined

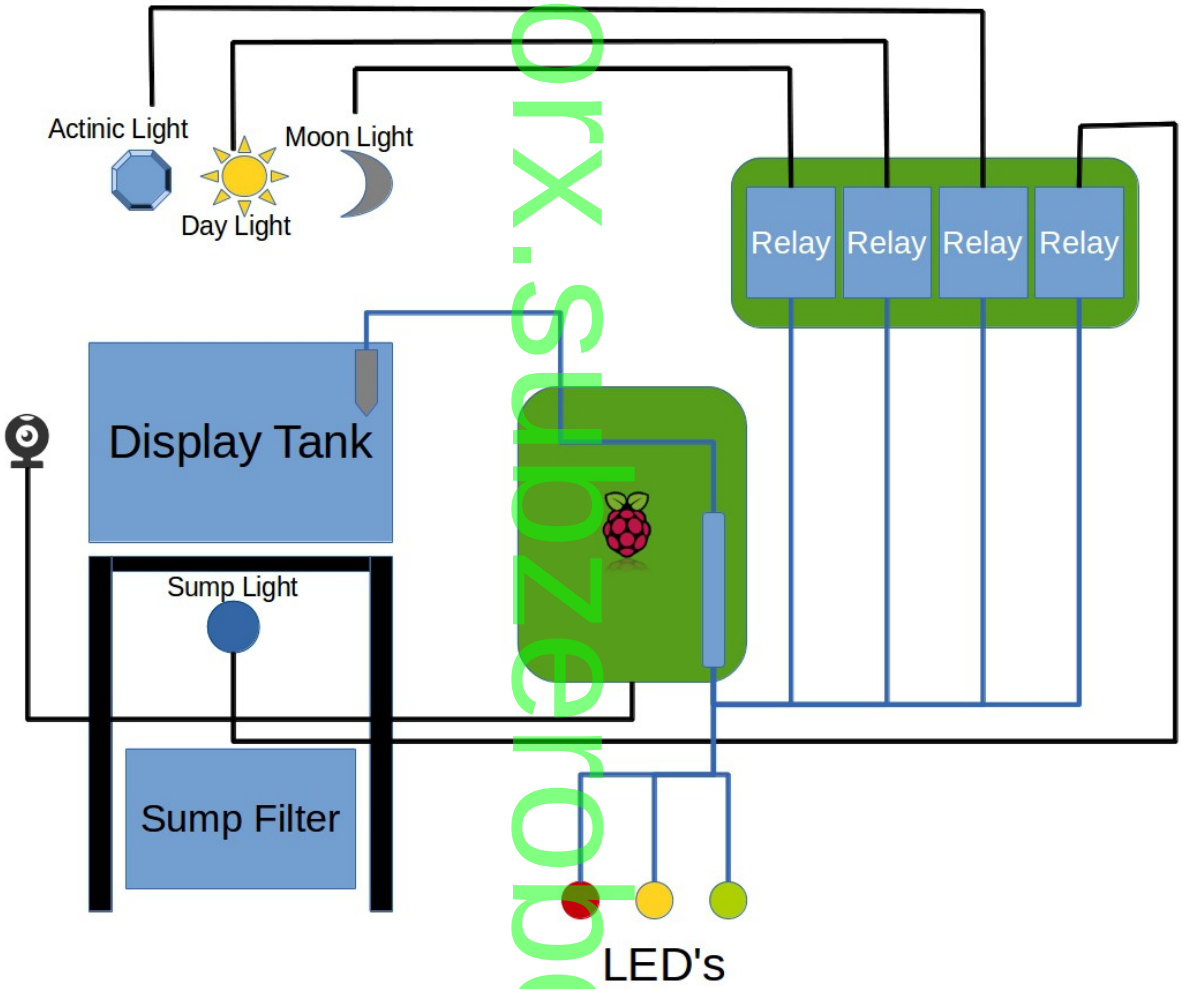
Fish-Pi at this point does the following. I have 4 types of lighting linked to my aquarium and one temperature probe. I have also made provision for a webcam to be able to visually confirm the health and state of my tank. Various python scripts control the state of the lighting through the day throught a pin state check (of either on or off) aswell as constant temperature metering via a probe aswell and

finally monitoring the temperature of the on board CPU.

Having mentioned that I would not have a display attached, I have rather created a visual notification system using 3 LED's. RED – indicating the automatic schedule is off, YELLOW – indicates system activity, for example the temperature is taken every minute when this happens they yellow LED strobes. GREEN – indicating the automatic schedule is on.

I would like to make mention at this point, no to abandon simple LED communication, developing your own visual language through strobes and flashes can be quite fun and creative. It's also a great way to visual affirm the Pi has not frozen up.

A Quick Visual



Right let's take a look at whats involved in the diagram above. At the top you can see the three light sources (comprising in a total of 63w of blistering LED's). Actinic is the blue light spectrum (30W), Daylight is the white spectrum(30W) and moonlight(3W) needs no explanation. Down below I have a second tank called a Sump Filter that handles my water filtration, here you will see the last light in the group namely the Sump Light. All lights are wired to digital Relay and in turn is wired to the GPIO. Housed in the display tank is also a digital thermometer also connected to the GPIO. There is a

webcam as you can see, at the moment I am using a USB based camera for reasons I will provide a little further along in the article. Last but not lease are the three status indictor LED's below.

I will not be going into depth on the construction and wiring of the high power LED's that run over my tank, in this article I would like to focus more on that Raspberry Pi and the Web interface itself. However if anyone is interested in an article on building highpowered LED's systems using modules that can be ordered online, I would be more than happy to oblige.

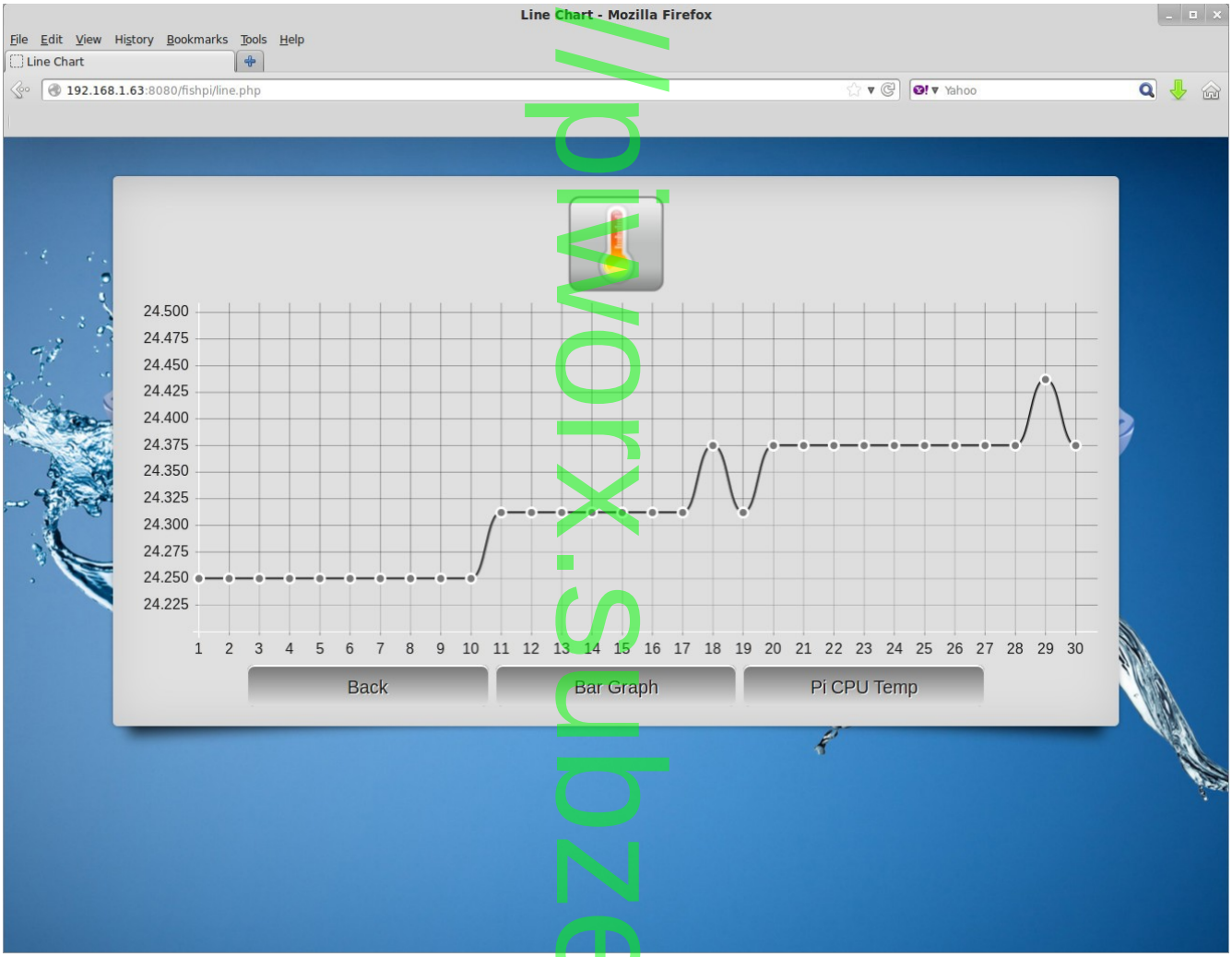
The Web Interface

I put in considerable effort into the graphic side of Fish-Pi using royalty free graphics and a bit of my own photoshoping skills. I have spent some time in the graphics arena so if you cant get your colours matched up I suggest you get a friend with artistic flair to give you a hand. Having said that let me give you some advice .. avoid the pretty stuff and get your framework working first. For example just get your buttons to work first then brush it all up when your done.

In the control panel above as you can see I have buttons that control the various lights in a very simple format of ON or OFF with a Cycle lights option which basically goes into auto mode (I will elaborate on this more a little later on.) Down at the bottom I have a few orbs which basically show the live status of the lights by indicating ON or OFF. Very simply achieved by checking the state of the GPIO

pins on the GPIO writing the value out to a text file which the web interface reads.

At the top I have the current temperature being fed back. Below that I have two icons representing temperature data and webcam access. Lets take a look at what I was able to achieve with in the temperature arena.



Reading the temperature back on one minute interval I take a span of 30 “reads” from the temperature database which basically gives me an overview of the last half an hour. The same can be presented in Bar Graph format. I've also added teh Pi CPU Temperature inn this section.

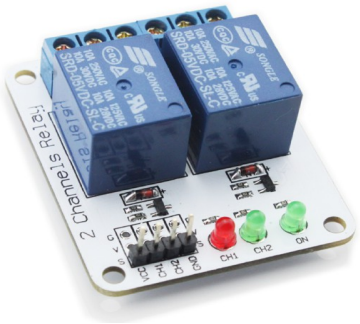
Down to the actual hardware components used.

1. Raspberry Pi Model B 512MB
2. 1 x 4.7K resistor and 3 x 270 ohm resistors
3. DS18B20 Digital Temperature Probe
4. 2 x Dual Relays
5. 30cm x 30cm 3mm Acyclic Sheet
6. Terminal block strip

The Relay Board

I would like to make a note here about the relay and also provide a picture of what it actually looks like, this is impont since these are “complete” as in “pre-built” relays. Do not get confused with the relay itself as a part which is literally just the blue block. These pre-built units are all ready to roll and just require hookup directly to the GPIO interface.

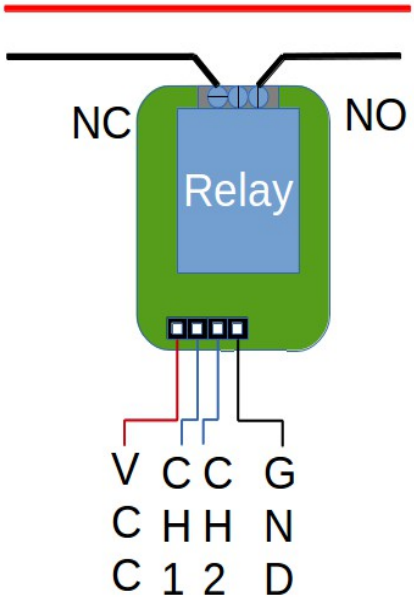
The wiring is quite simple, there are typically 4 pins on a dual relay.



Relay	**Pi GPIO**
VCC(positive 5v)	PIN 2
CH1(Channel1)	GPIO pin of your choice
CH2(Channel2)	GPIO pin of your choice
GND(Ground)	Any Ground Pin on the GPIO

Now hooking up the relay to the device you wish to control, in my case one of my high powered LED light units. The relay is essentially going to break the circuit to the LED (remember these are my highpowered LED lights not the 3 lower powered LED's for visual status. So lets take a look at the wiring hookup to make sense of it All. Im going to break the relay down to a single relay to make it simple to illustrate.

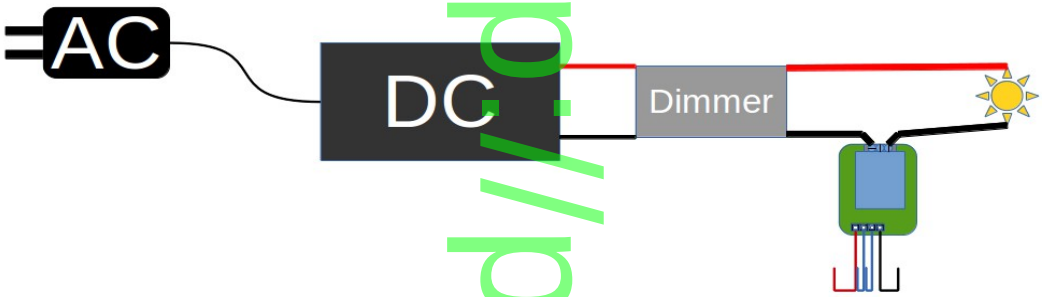
You have two choices here, you can either break the power of the AC line or the DC line of the light system your powering. I suggest breaking the DC line since AC is dangerous and you should seek the assistance of a qualified electrician if you intend to work on AC power.



These relays quite comfortably handling up to 10Amps AC power and I believe up to 30Amps DC. Specs vary here and there be sure to buy as per what you intend to control (ON / OFF)

You will notice a single relay has 3 positions, basically you can wire the device you want to control in one of two states. Either NC (Normally Closed or ON) or NO (Normally Open or OFF) in my case all my lights have been wired to the normally open or OFF position.

As mentioned before Im not going to elaborate on the lights I have put together since they are a subject on their own. But I will provide a simple diagram so you are able to see where the relay breaks the DC line. As per the diagram from left to right we are flowing from AC wall plug to DC Constant Current powersupply rated for the LED (from this point on were in DC) then through a Dimmer and finally the relay breaking the circuit before i get to the High Power LED itself. In my case I chose to break after the dimmer. From an efficiency perspective it would have been better to break the AC line, but I did not want to constantly spike my setups and risk damaging my power supplies.

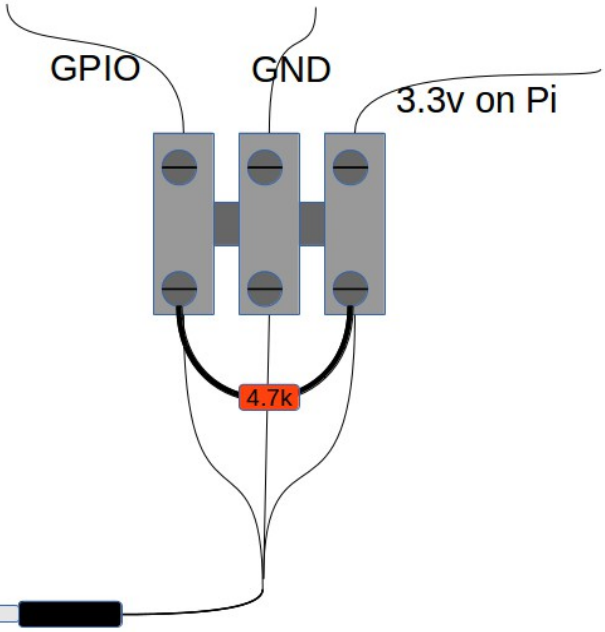


The Temperature Probe

I chose the DS18B20 becuase it seemed to be reliable and well constructed, even though its made of stainless steel I would still recommend coating it in a thin film of 100% silicone, if you dont immerse it in saltwater there is no need to do this.

I suggest you read the following article laid out by Adafruit to understand the use and hookup in detail.
http://learn.adafruit.com/adafruits-raspberry-pi-lesson-11-ds18b20-temperature-sensing/parts

This is where the 4.7K resistor comes into play. I didnt have one on hand so I simply put a few in series to get to the 4.7K mark. Resisters can be connected up in series to reach the resistance you require, use an ammeter to check how close you can get to the value. Falling short slightly or going over slightly yields no problems from my experience but try to get as close as possible.



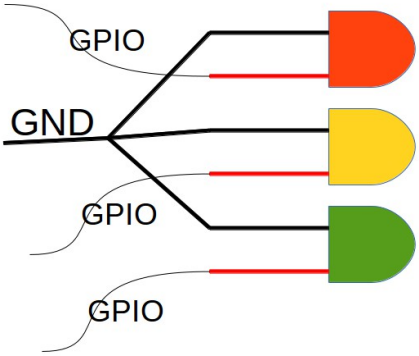
Here is a diagram to quickly simplify the hookup. I used a simple terminal block to connect the wires. To protect the 4.7k resistor. I placed it inside a strip of heat-shrink.

There is a good reason why I didnt just solder this section up. The GPIO pin for the temperature sensor HAS to be GPIO pin 4. The sensor is classed as a 1 wire sensor. You are able to attach multiple 1 wire sensors in parallel to the single GPIO 4 pin on the Raspberry Pi. The additional sensors will also not need their own 4.7K resistor. The terminal block makes adding more a very simple and neat exercise.

If you read the article up at Adafruit you will see that each 1 wire device has its own serial number or id and it will register itself under its own folder on the Raspberry Pi. As a result there is no clash. Just make sure you buy digital sensors and not analogue.

Three status LED's

Taking three simple 2.3-3.3 volt LED's to create a visual notification system presents no difficulty at all. The three LED's were wired each to their own GPIO pin, making them independently addressable. I found RED YELLOW and GREEN to be quite suitable and typical in world of signalling. The RED is used to signal that the system was up but the schedule or cycle was disabled, GREEN would be mean the opposite “Up and Running” and YELLOW would be used to strobe status. For example ever 1 minute when the temperature probe takes a reading the YELLOW LED flashes three times.



Simple but effective. Dont forget your resistors. 270 Ohms will do. An LED has no positive or negative. Is has an anode and a cathode. The resistor can be placed at ANY side of the led; but the cathode (K) of the led should path to the negative (gnd) of the supply, anode (A) towards the positive (+).

Webcam

For the webcam I am using the original Raspberry Pi webcam, if you do intend using the original camera it's very important to initialize your modules in the following order. Here's why ... participating in forums we found for 1 wire based sensors and the Raspberry Pi Camera to work we had to use the following order for ...

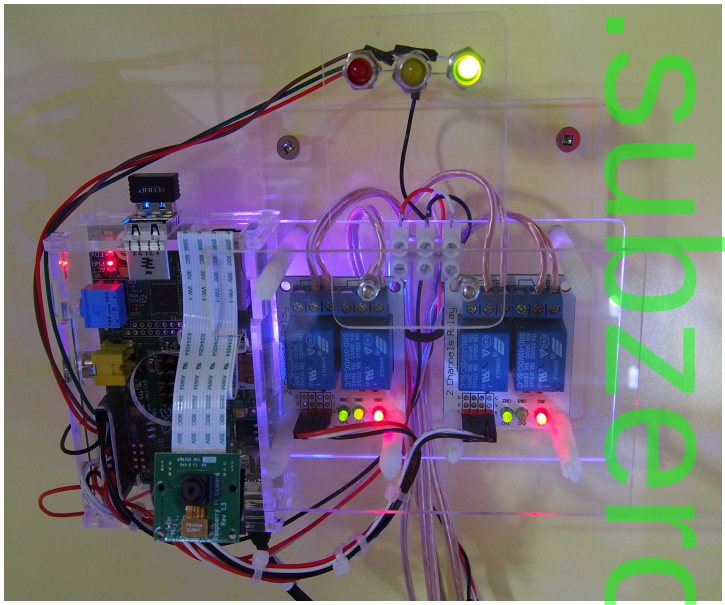
/etc/modules

```
w1-gpio
w1-gpio pullup=1
w1-therm
#i2c-dev
#i2c-bcm2708
spi-bcm2708
snd-bcm2835
```

If not it results in the camera freezing up. Some may have not experienced this but if you do I suggest you configure your modules to start up in that order. Of course I am not using I2c hence I have left them hashed out.

Right lets take a look at what all this looks like put together

At this point I still have the prototype body and setup. I plan to make an ornamental fish out of acrylic and produce a “more final” case for the whole system. Ill keep it all in clear acrylic so I can see all the LED's sparkle. It also help show fellow inventors what the components are. Currently I have the Pi-Camera installed, which will be replaced with a web cam instead.
The Pi-Cam will be moved to my next project a weather station.



All in all up to this point I'm more than satisfied as to what I was able to accomplish. This is not too electrically complex, I found more challenges in building the interfaces which is comprised of Raspian (My OS of choice) Python, PHP, CSS and Javascript.

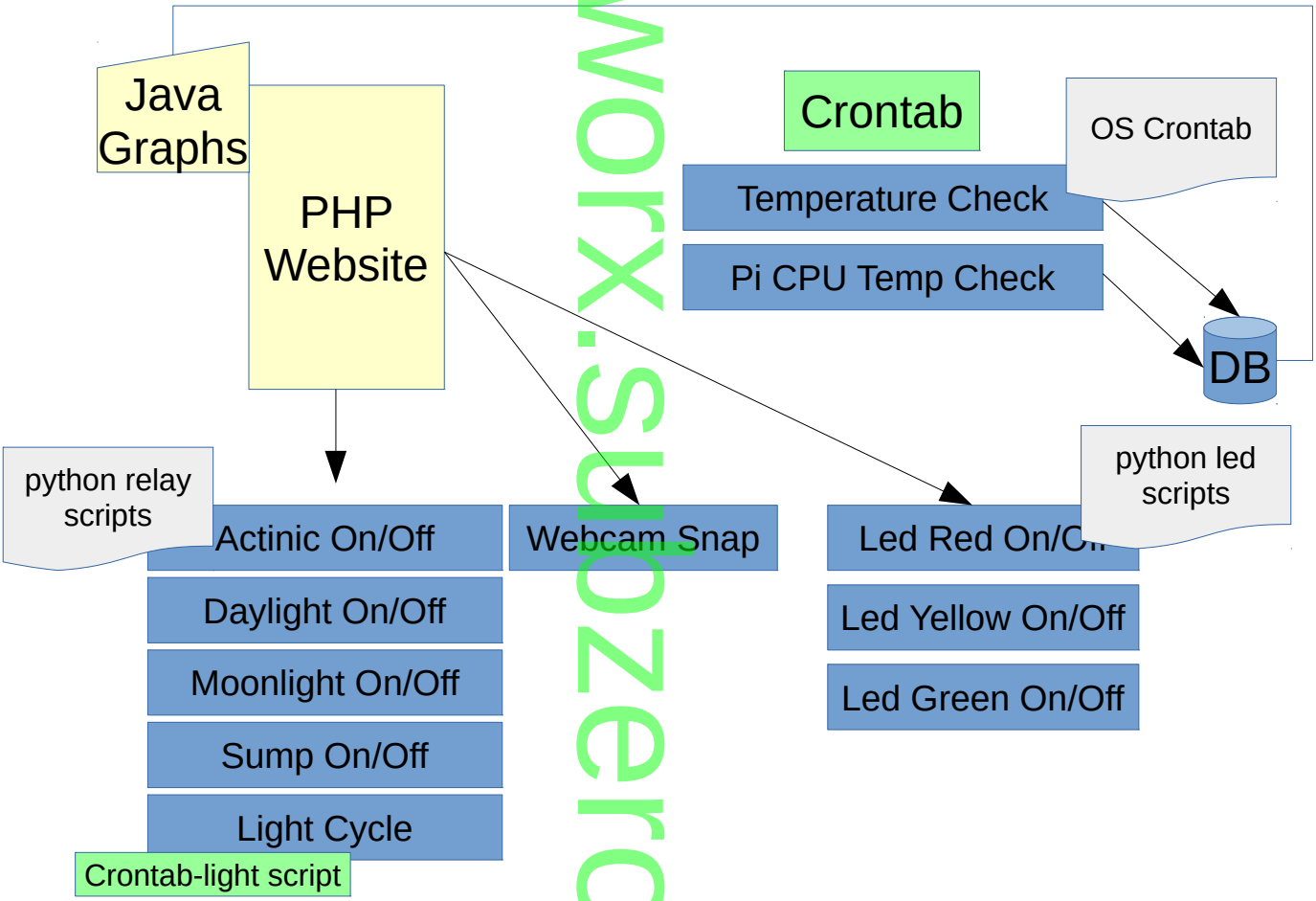
Let's move onto that shall we.

The Code, Routines and Web Tech

First off I would like to recommend that you find a way to manage your pin's, there is nothing worse than not knowing what's where and sending out a signal to the wrong pin causing all sorts of crazy errors.

I created the following git hub project called “pipins” which can be found on github using the following url https://github.com/michaeljvdh/pipins.git Follow the instructions found there on how to use it.

The following diagram will help understand how I put this all together and where the code would fit in.



To remove complication, I'm going to explain the above in simple terms and then provide snippets that can be reproduced according how you might prefer to do it, or perhaps take a different approach completely and re-write it using the same idea.

In a nutshell I have an Apache installation to provide a local web service, I have installed php support for Apache in order to run .php based files on Apache. My php website basically executes the relevant python scripts after pressing a button to cause an action (for example Daylight On or Off) would execute a python script (daylighton.py) that would trigger the appropriate relay.

Utilizing the Linux Operating Systems Crontab I have a my temperature python scripts run in one

minute intervals. These two scripts (Water Temperature and Pi CPU Temperature) constantly record their results to an Sqlite Database which in turns provides data for my Java Graphs Module to read. Additionally I have a webcam python script for taking a picture and an LED script providing different status indicators based on whats happening. Simple Right !!! Let's get into the mechanics.

Let's start with the simplest script : Led Status On/Off

These are very simple “on hand” functions which your main python code can call to power an LED on or off, to indicate whats going on with your scripts. There are two ways one can write this, either as one file with many functions or multiple files that perform a single action. The function method is far more efficient than a bunch of independent files. Lets take a look.

led.py

```
import RPi.GPIO as GPIO ## Import GPIO library

GPIO.setwarnings(False) ## Remove state warnings
GPIO.setmode(GPIO.BOARD) ## Use board pin numbering instead of typical GPIO assignment

GPIO.setup(26, GPIO.OUT) # Green LED
GPIO.setup(16, GPIO.OUT) # Yellow LED
GPIO.setup(22, GPIO.OUT) # Red LED

def greenon():
    GPIO.output(26, True) #Led On
def greenoff():
    GPIO.output(26, False) #Led Off

def yellowon():
    GPIO.output(16, True)
def yellowoff():
    GPIO.output(16, False)

def redon():
    GPIO.output(22, True)
def redoff():
    GPIO.output(22, False)
```

Note that I am using **GPIO.BOARD** numbering and not GPIO typical assignment there is a big difference here so please be sure to read up on this and be familiar with the command. This is where my git project 'pipins' comes in handy, as it shows the traditional assignment and the **GPIO.setmode(GPIO.BOARD)** pin assignment. A good article to help get to grips with this can be found at http://raspi.tv/2013/rpi-gpio-basics-4-setting-up-rpi-gpio-numbering-systems-and-inputs

Lets move on the relay scripts, the relay scripts simply do exactly what the led script does. By making a pin high or low (True or False) will activate the relay. Let's take a look at the daylighton.py script.

Again using GPIO.BOARD mode which will change physical pin 18 on the board which is actually GPIO 24 to the on/true/high value.

daylighton.py

```
import RPi.GPIO as GPIO ## Import GPIO library
```

```
GPIO.setwarnings(False) ## Remove state warnings
GPIO.setmode(GPIO.BOARD) ## Use board pin numbering instead of typical GPIO assignment
GPIO.setup(18, GPIO.OUT)

GPIO.output(18, True)
```

I have another script to do the opposite called daylightoff.py which of course sets the last line as **GPIO.output(18, False)** which would switch the relay to NO (normally open) for the daylight lights to be switched off.

Now I have 4 other lights to do this with ... and you will be asking why I have not created this like the led.py script with all the light modes in one python script. I broke it up because the lights scripts are being executed directly from the php website and NOT by another python script. This is the simplest method for a small project although there are other ways to do it ... I preferred to keep it simple.

So for the other lights I simply created an on & off python script for each light eg. actinicon.py, actinicoff.py, moonlighton.py, moonlightoff.py and so on.

Ok .. so we have the led script ready and all the relay scripts created.

Let's deal with the temperature next. This requires you to be familiar with the crontab function of Linux, I suggest you read up on it before attempting to work on it. Basically it's a file that enables you to run things at certain intervals from minutes to days/months and so on. In my instance I needed a temperature reading taken once a minute. Once logged in as pi(default user) type in **crontab -e**

This displays your crontab (schedule script) note that unless your user is logged in your crontab wont run. If you set your pi for autologin this solves the problem. It can be done without logging on and for Linux to perform certain crons at boot up .. but I only suggest that route once you have a good working knowledge of crontab in general. Setting your pi user for autologin and running **crontab -e** is the best route to start.

I added the following two lines manually in my crontab at the end of the file to run the following two scripts every minute. **Thermcheck.py** checks the DS18B20 probe (in the water) and **pi_cpu_temp.py** checks the temperature of the Rpi cpu. Both produce a text file output for simple reading from my website over and above being written to an Sqlite database for record keeping and graphs.

* * * * * sudo python /yourpath/thermcheck.py
* * * * * sudo python /home/pi/Desktop/scripts/pi_cpu_temp.py

Note: Remember to run them with elevated rights by using sudo

Here is how I retrieve the local cpu temperature.

pi_cpu_temp.py (Fetches the cpu temperature and writes the value to a text file for general use)

```
import os
# Fetch Temperatrure Reading
x = os.system("/opt/vc/bin/vcgencmd measure_temp >/yourpath/pi_cpu_temp.txt")

# The original reading has to be stripped and cleaned up before actually getting degrees Celsius.
```

```
f = open("/yourpath/pi_cpu_temp.txt")
result = f.read()
v = result.replace("\n","")
w = v.replace("temp=","")
x = w.replace("'C","")
f.close()

# Finally the result is re-written cleaned up to the original text file
f = open("/yourpath/pi_cpu_temp.txt","w")
result = f.write(x)
f.close()
```

thermcheck.py

```
import time
import led

# Here the  led script is imported to pull the appropriate function from within.  In this case I
# have it flashing the yellow led 5 times, this shows me visually every minute when the crontab
# runs and data is being read and written to the temperature database.  Technically it also confirms
# that crontab is running and working.

for i in range(0,4):
        led.yellowon()
        time.sleep(0.05)
        led.yellowoff()
        time.sleep(0.05)

# The set of lines reads the value of the temperature sensor.
# Be sure to enter your temperature sensor serial value instead of mine eg. 28-0000053cdcb7
# the following setup will give you degrees C

tfile = open("/sys/bus/w1/devices/28-0000053cfcb7/w1_slave")
text = tfile.read()
tfile.close()
secondline = text.split("\n")[1]
thermdata = secondline.split(" ")[9]
therm = float(thermdata[2:])
therm = therm / 1000

# The next section reads the output from the pi_cpu_temp.py script which
# generated it's output as a .txt file and assigns it to variable called ***cpu*** so it can be written
# to the Sqlite database.

xfile = open("/yourpath/pi_cpu_temp.txt")
cpu = xfile.read()
xfile.close
print cpu

# Next the DS18B20's Temperature is written to a text file so the php website can easily read
# it for display on the website. (This can be done via the Sqlite DB again, keeping it simple)

wfile = open ('/yourpath/therm.txt','w')
wfile.write (str(therm))
wfile.close()
print therm
```

```
# The Database Portion – Basically a database is created called temp.db to which the DS18B20B
# and the Rpi's cpu temperature are written with a time stamp for future use.  You can move
# this import statement to the top of the file, I have it down here for illustration purposes.  Imports
# can be placed further on down the line, its just cleaner having them at the top. :)

import sqlite3
from time import ctime

createDb = sqlite3.connect('/yourpath/temp.db')
curs = createDb.cursor()

def createTable():
       curs.execute('CREATE TABLE temperature (id INTEGER PRIMARY KEY,time TEXT,temp INTEGER,cpu INTEGER)')

def addTemp(time, temp, cpu):
     curs.execute('INSERT INTO temperature (time,temp,cpu) VALUES (?,?,?)',(time,temp,cpu))

def writedata():
    time = ctime()
    addTemp(time,therm,cpu)
    createDb.commit()

try:
       createTable()
except:
       print ("Table already exists, table not created.")

writedata()
```

Ok .. so in summary the temperature readings are available in .txt files as well as written to the database for use in my JavaScript graphs.

But before we get to the web side of things there is one more script that is PIVOTAL to the whole operation. And that is the automatic light schedule script. If you looked at the menu of the web page you see that the lights can be on one of two modes. Either in cycle (automatic) or off (manual). While in cycle off mode you are free to switch lights on and off as you wish, however in cycle mode your subject to what time of the day it is. (Morning to Evening) and this will simply undo your manual settings and override using the schedule settings. Lets take a look at the script.

cyclon.py

```
__author__ = 'fishpi'

import RPi.GPIO as GPIO
import led

# The handy led functions script comes in here to ensure the GREEN led is on and the RED is off
# indicating that the schedule is active.

led.greenon()
led.redoff()

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(13, GPIO.OUT) #sump light
GPIO.setup(11, GPIO.OUT) #moon light
GPIO.setup(15, GPIO.OUT) #actinic light
GPIO.setup(18, GPIO.OUT) #daylight lighting

# The following controls which relays(lights) are on or off in respect to the time of day.
# The following lines are VERY useful in many projects to action certain things at a specific time of
# day. For this work you will need an internet connection for the correct time to be called.

def run():
    from datetime import datetime, time,date
    now = datetime.now()
    now_time = now.time()
    if now_time >= time(6,00) and now_time <= time(6,59):
        Print ("1") # These are here for testing purposes in case I need to test.
        GPIO.output(11, True)
        GPIO.output(15, False)
        GPIO.output(18, False)
    if now_time >= time(7,00) and now_time <= time(8,59):
        print ("2")
        GPIO.output(11, True)
        GPIO.output(15, True)
        GPIO.output(18, False)
    if now_time >= time(9,00) and now_time <= time(17,29):
        print ("3")
        GPIO.output(11, True)
        GPIO.output(15, True)
        GPIO.output(18, True)
    if now_time >= time(17,29) and now_time <= time(19,59):
        print ("4")
        GPIO.output(11, True)
        GPIO.output(15, False)
        GPIO.output(18, False)
    if now_time >= time(20,00) and now_time <= time(23,59):
        print ("5")
        GPIO.output(11, False)
        GPIO.output(15, False)
        GPIO.output(18, False)
run()
```

You will notice the cycle scripts runs once ... this is a problem since it needs to be continually checked to react according to the time? You guessed it ... it needs to be added to the crontab so it can be run every 2 minutes to evaluate what needs to be done. Question is how do we get it there. Python offers a solution to this called crontab library. This library allows you to inject a line and remove it, without disturbing your other manual entries. Lets take a look at how it's used.

cycleon_cron.py

```
__author__ = 'fishpi'

from crontab import CronTab
import led # That wonderful led functions script again

led.greenon()
led.redoff()

run = ('sudo python /yourpath/cycleon.py') # The actual light schedule script.
```

```
tab = CronTab(user='pi')

f = open ('/yourpath/cycle_status.txt','w')
f.write ("ON")
f.close

job = tab.new(run, comment='Schedule Enabled')
job.minute.every(2)
tab.write()

#print tab.render() #Remove HASH From Line Start To Test / See Crontab
```

After running this python script a cronjob is added to the crontab and configured to run **cycleon.py** every 2 minutes. Problem solved. To remove this entry from the crontab another script needs to be executed. Lets take a look. Notice in the above script I have written "ON" to a text file, this **cycle_status.txt** file is in turn read by the PHP code on my website to show me if the cycle is on or off, you will see the same in the next script.

cycleoff_cron.py

```
__author__ = 'fishpi'

import led # That wonderful led functions script again

led.redon()
led.greenoff()

from crontab import CronTab

run = ('sudo python /yourpath/cycleon.py')
tab = CronTab(user='pi')

f = open ('/yourpath/cycle_status.txt','w')
f.write ("OFF")
f.close

tab.remove_all(run, comment='Schedule Enabled')
tab.write()

#print tab.render()  #Remove HASH from Line Start To Test / See Crontab
```

The last line in the cron python scripts are hashed out, these will actually print your crontab out after the update is done so you can see if it worked, it's there purely for troubleshooting. Otherwise you would have to run **crontab -e** after each test to see the result.

An important note here about the adding and removing a cron the run variable HAS to be identical or it cannot remove what it added by way of recognition.

Status Checking

The next problem I faced was how to display which lights where on and which were off on the web page, it had to be live and current. I did this by simply having the web page executing a status.py script every time it's accessed or refreshed. This script produces text files with the current state. The web

page then reads these files to display if a pin is ON or OFF. Let's take a look.

status.py

```
import RPi.GPIO as GPIO ## Import GPIO library

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD) ## Use board pin numbering

GPIO.setup(11, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)

loff = "OFF" # Variable for lights off
lon = "ON" # Variable for lights on

# The following status.txt files are produced for the website to easily read the state of the pin.
# I simply read the relative file using php and display it below the icon on my webpage.

if GPIO.input(23) == False:
        f = open ('/yourpath/actinic_status.txt','w')
        f.write (loff)
        f.close
        print ("Actinic Lights Are OFF") # Actinic
if GPIO.input(23) == True:
        f = open ('/yourpath/actinic_status.txt','w')
        f.write (lon)
        f.close
        print ("Actinics Lights Are ON") # Actinic
if GPIO.input(18) == False:
        f = open ('/yourpath/daylight_status.txt','w')
        f.write (loff)
        f.close
        print ("Daylights Are OFF") # Daylight
if GPIO.input(18) == True:
        f = open ('/yourpath/daylight_status.txt','w')
        f.write (lon)
        f.close
        print ("Daylights Are ON") # Daylight
if GPIO.input(11) == False:
        f = open ('/yourpath/moonlight_status.txt','w')
        f.write (loff)
        f.close
        print ("Moonnlights Are OFF") # Moon
if GPIO.input(11) == True:
        f = open ('/yourpath/moonlight_status.txt','w')
        f.write (lon)
        f.close
        print ("Moonlights Are ON") # Moon
if GPIO.input(13) == False:
        f = open ('/yourpath/sumplight_status.txt','w')
        f.write (loff)
        f.close
        print ("Sump Lights Are OFF") # Sump
if GPIO.input(13) == True:
        f = open ('/yourpath/sumplight_status.txt','w')
        f.write (lon)
```

```
    f.close
    print ("Sump Lights Are ON") # Sump
```

The webcam

This was the easiest and the best part of it all. I created a bash script that uses the raspistill tool to take a picture and place it where my webpage can pick the image up to display it. Here is the line you need to take a photograph with the resolution 640x480.

This script in can be run in two ways, type in **sh ./webcamshot.sh** or type in **chmod a+x webcamshot.sh** which will make it executable. It can now be executed typing in **./webcamshot.sh**

webcamshot.sh

```
#!/bin/bash
sudo raspistill -vf -hf -q 100 -n -o /var/www/fishpi/images/webcam/image.jpg -w 640 -h 480
```

If you have purchased the Raspberry Pi Camera you will be familiar with the script above and would also have enabled the camera using raspi-config. If you need help I suggest you read http://www.raspberrypi.org/camera

Power Failure Notification

Being familiar with the cronjob system now, being notified of a power failure is really easy. Simply follow online guides in setting up mail on your raspberry pi and then manually add the following script to your crontab. Using ***crontab -e*** and start your line with @reboot ***command*** this will naturally execute my script in a reboot event.

```
@reboot sudo echo "Power Failure Message" | mail -s "PI-ALERT Possible Power Failure" michael@subzerobc.com
```

Useful Cron Tip: *Do not add your manual crontab entries below lines that are injected via the python crontab script library, always place them above. The python crontab library seems to work from bottom up, it is possible to lose lines below entries placed by the python crontab injection library.*

The Web Page

The web page uses php, css, javascript and of course it's not possible to display all the code in this article. Hence I have created a github repository which give you access to the entire development so that you are able to deconstruct or inspect to see how I put it together. The code and website can be picked up at https://github.com/michaeljvdh/fishpi.git

The graph is particularly impressive and is an open source project that's worth investigating. http://www.chartjs.org/ The trick is knowing how to get your data into the graphs using php and Sqlite. I have provided a document on how to do this. The document can be found in the fishpi.git project folder. I suggest you clone it and go through it in it's entirety.

This project has not only been an experiment but has actually become my best friend in regards to managing my saltwater aquarium. I am currently in the process of upgrading all the code and adding new features to the entire project. Thank you all for your time and I hope this provided some food for thought. *Fish Pi 2 is on it's way - happy hacking.*