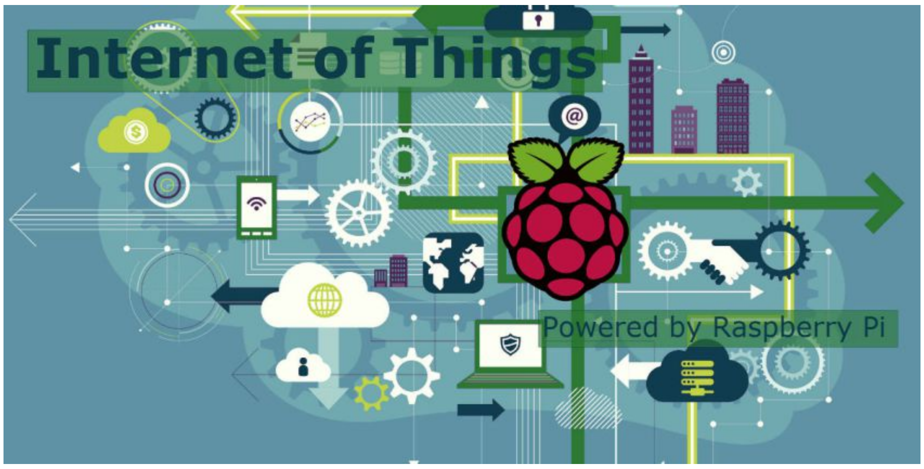
Web interface for Raspberry Pi Apps

John F. Moore October 2018

Table of Contents

Abstract
Defining the Problem
Creating a simple Application
Software Setup
Wiring Pi
Testing GPIO
Control GPIOs with a webinterface
GPIO Pin names
Raspberry Pi Web Server using Flask to Control GPIOs
Basic Raspberry Pi Setup
/home/pi/web-server/app.py
/home/pi/web-server/templates/main.html
Running the web interface
Simple and Intuitive Web Interface for your Raspberry PI
Closing Thoughts



Abstract

Building applications to run on the Raspberry Pi is getting to be a common occurrence. But how we interface to the applications is often an issue that is over looked.

For this talk we will explore how to build a web interface for turning on an off some LEDs. This will provide a framework for how to control an application using a Web interface.

Defining the Problem

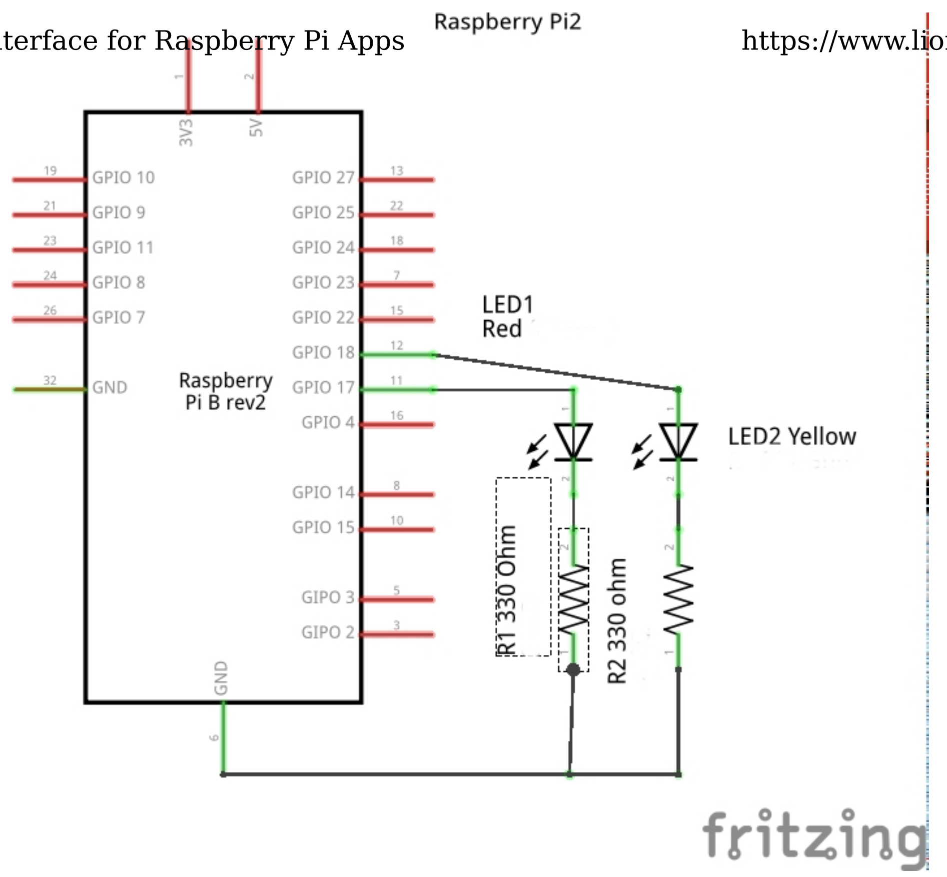
So you have just build an awesome server for your house using a Raspberry Pi and now you need to talk to it. But you do not want to log into the raspberry pi every time you want to run the application.

The answer is to create a web interface for your application and use the browser to access your server.

Creating a simple Application

To test the interface I need something to control using the GPIO pins. I will setup two LEDs on a bread board. Now there is no limit to what you can control once you understand how it is done.

I am going to connect a red LED to pin 11 then through a 330 ohm resistor to ground. I will connect a yellow LED to pin 12 through a 330 ohm resistor to ground. *Note Pin 11 is also GPIO 0 and pin 12 is GPIO 1. I will explain more later.*



LED Configuration

This is a very simple design, but the idea is that I can now see if the GPIO pins are on of off. I could just as easily have attached relays to the GPIO pins.

For this demo I am only going to switch on and off some LEDs.

Software Setup

I downloaded the Raspberry Pi software 2018-04-18-raspbian-stretch-lite.zip (https://www.raspberrypi.org/downloads/raspbian/) and copied the image to an SD card. I then followed a normal configuration, specifically turning on SSH and giving the board a static IP address. I prefer static IP addresses since they can be reliably mapped to a URL later. I then did a software update to make sure the system was current.

I prefer to enable the SSH to allow me to work on the raspberry pi from anywhere in my network. This way I can sit at my desk and control a remote raspberry pi without being physically at the computer. I typically use the command

```
ssh -X -l pi 192.168.1.137
```

Where 192.168.1.137 is the IP address of the raspberry pi. Sometime I assign a DNS name to the raspberry pi to make it easy to identify. For a windows I recommend the Putty program to connect using SSH.

Wiring Pi

The next step is to install an interface for the GPIO pins. This app Wiring Pi (http://wiringpi.com/download-and-install/), see link for installation instructions, allows you to control the GPIO from a command line. I ran this install as root user using the command:

```
git clone git://git.drogon.net/wiringPi
```

Controlling the GPIO pins from a command line means that you have an easier time interfacing this control into a web interface. You can also use this interface to allow control from inside your program.

Once the program is installed you can do the following test from the command line.

```
gpio -v
```

On my system it returned:

```
gpio version: 2.46
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Embest
  * Device tree is enabled.
  *--> Raspberry Pi 3 Model B Rev 1.2
  * This Raspberry Pi supports user-level GPIO access.
```

I then ran the command line test:

```
gpio readall
```

Which returned:

```
 +-----+-----+---------+------+---+---Pi 3
---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi
| BCM |

+-----+-----+---------+------+---+----++----+---+------+---------+-----
+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |
|     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |
|     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |
|     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15
| 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16
| 15  |
 |  17 |   0 | GPIO. 0 |  OUT | 0 | 11 || 12 | 1 | OUT  | GPIO. 1 | 1
| 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |
|     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4
| 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5
| 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |
|     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6
| 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10
| 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11
| 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31
| 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |
|     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26
| 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |
|     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27
| 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28
| 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29
| 21  |

+-----+-----+---------+------+---+----++----+---+------+---------+-----
+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi
 | BCM |
 +-----+-----+---------+------+---+---Pi 3
---+---+------+---------+-----+-----+
```

Testing GPIO

Before we proceeding we should test the gpio at the command line to make sure it works. First I need to assign the pins to outputs:

```
gpio mode 0 out  # 0 is the pin, out is output mode, as opposed to inpu
t
```

Next I want to turn on the red LED using the command line:

```
gpio write 0 1  # 0 is the pin, 1 is the high command
```

This should turn on the red LED. I can check by using the command:

```
gpio read 0
```

I can also test the yellow LED by using the same command.

```
gpio write 1 1
```

If everything is fine we can go on to the web interface.

Control GPIOs with a webinterface

The next step is to install the GPIO interface (https://github.com/brainfoolong/gpio-webinterface). For this install I run the install as the pi user. That is the user who will be executing the code. The main reason for running the app as the pi user instead of root is security.

For my install I only needed to use apt to install 2 additional packages, php-cli and php-mbstring. Next I use the git command to install the software.

```
git clone https://github.com/brainfoolong/gpio-webinterface.git
```

into /home/pi. This created the folder, **gpio-webinterface**.

I then ran the following command as the pi user.

```
php -S 0.0.0.0:4322 -t /home/pi/gpio-webinterface > /dev/null 2>&1 &
```

to start the service.

Next you go to the web browser on any computer in the network and enter the URL:

```
http://192.168.1.137:4322
```

where 192.168.1.137 is the IP address, and :4320 says to open port 4322 of the raspberry pi.

Once you are there you will see a blank screen, so you start by selecting the menu icon in the upper left.

Which should open a menu like:

Select the Settings option. Now I will populate it with the 2 LEDs I have so it looks like this:

What we have done is assign a Name to each switch, then set the GPIO pin numbers as used by Wiring PI. and specify that the pin is an output. The settings section is just the location of the gpio command.

Once we have saved these settings we can return to the top page, by returning to the menu icon in the upper left, and going to GPIOs. This should bring up a screen like:

GPIO Pin names

One area of concern I found was the name of the different pins. Look at this chart.

You will notice that pin 11 is labeled **GPIO17 (GPIO_GEN0)**. What that means is that Wiring PI calls this pin 0. But some programs use some other names for the pins. For more information about GPIO pins have a look at Raspberry gPIo (https://learn.sparkfun.com/tutorials/raspberry-gpio/introduction) for a good introduction to GPIO pins and programming them. They have examples using both Python and Wiring PI.

Raspberry Pi Web Server using Flask to Control GPIOs

Next we will build a web page for manual control of the LEDs using python. This comes from Random Nerd Tutorials under Raspberry Pi Web Server using Flask to Control GPIOs (https://randomnerdtutorials.com/raspberry-pi-web-server-using-flask-to-control-gpios/).

Basic Raspberry Pi Setup

Lets install FLASK by following the instructions listed under Flask (https://randomnerdtutorials.com/raspberry-pi-web-server-using-flask-to-control-gpios/).

Now it is time to create the python script which will handle setting up a web server and initializing the GPIO ports. Create the folder */home/pi/web-server* and edit a file named *app.py* in the web-server folder.

/home/pi/web-server/app.py

```
Adapted excerpt from Getting Started with Raspberry Pi by Matt Richards
on

Modified by Rui Santos
Complete project details: http://randomnerdtutorials.com

'''

import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)

GPIO.setmode(GPIO.BCM)

# Create a dictionary called pins to store the pin number, name, and pi
n state:
pins = {
   17 : {'name' : 'GPIO 17', 'state' : GPIO.LOW},
   18 : {'name' : 'GPIO 18', 'state' : GPIO.LOW}
   }

# Set each pin as an output and make it low:
for pin in pins:
   GPIO.setup(pin, GPIO.OUT)
   GPIO.output(pin, GPIO.LOW)

@app.route("/")
def main():
   # For each pin, read the pin state and store it in the pins dictiona
ry:
   for pin in pins:
      pins[pin]['state'] = GPIO.input(pin)
   # Put the pin dictionary into the template data dictionary:
   templateData = {
      'pins' : pins
      }
   # Pass the template data into the template main.html and return it t
o the user
   return render_template('main.html', **templateData)

# The function below is executed when someone requests a URL with the p
in number and action in it:
@app.route("/<changePin>/<action>")
def action(changePin, action):
   # Convert the pin from the URL into an integer:
   changePin = int(changePin)
   # Get the device name for the pin being changed:
   deviceName = pins[changePin]['name']
   # If the action part of the URL is "on," execute the code indented b
```

```
glow:
   if action == "on":
      # Set the pin high:
      GPIO.output(changePin, GPIO.HIGH)
      # Save the status message to be passed into the template:
      message = "Turned " + deviceName + " on."
   if action == "off":
      GPIO.output(changePin, GPIO.LOW)
      message = "Turned " + deviceName + " off."

   # For each pin, read the pin state and store it in the pins dictiona
ry:
   for pin in pins:
      pins[pin]['state'] = GPIO.input(pin)

   # Along with the pin dictionary, put the message into the template d
ata dictionary:
   templateData = {
      'pins' : pins
   }

   return render_template('main.html', **templateData)

if __name__ == "__main__":
   app.run(host='0.0.0.0', port=9000, debug=True)
```

After saving this file, we need to create another folder named */home/pi/web-server /templates*. Now we need to create the file *main.html* in the folder templates.

/home/pi/web-server/templates/main.html

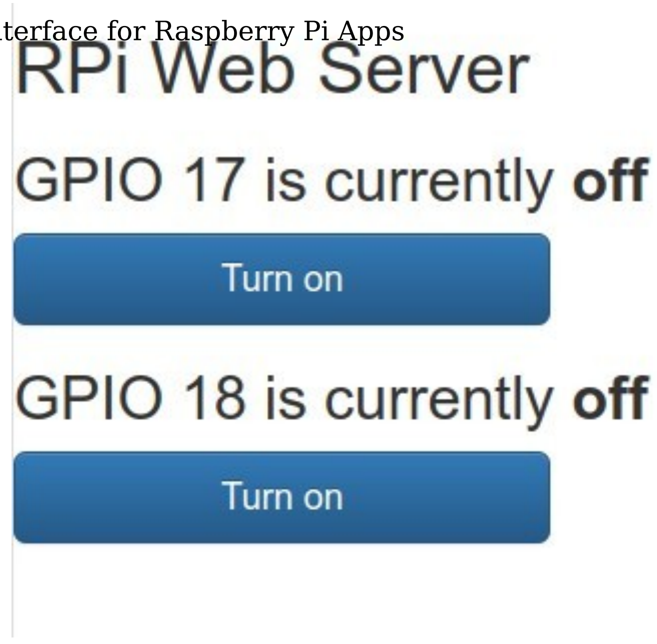
```
<!DOCTYPE html>

<head>
   <title>RPi Web Server</title>
   <!-- Latest compiled and minified CSS -->
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstr
ap/3.3.6/css/bootstrap.min.css" integrity="sha384-1q8mTJOASx8j1Au+a5WDV
nPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7" crossorigin="anonymous">
   <!-- Optional theme -->
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstr
ap/3.3.6/css/bootstrap-theme.min.css" integrity="sha384-fLW2N01lMqjakBk
x3l/M9EahuwpSfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r" crossorigin="anonymo
us">
   <!-- Latest compiled and minified JavaScript -->
   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/boot
strap.min.js" integrity="sha384-0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5
ELqxss1yVqOtnepnHVP9aJ7xS" crossorigin="anonymous"></script>
</head>

<body>
   <h1>RPi Web Server</h1>
   {% for pin in pins %}
   <h2>{{ pins[pin].name }}
   {% if pins[pin].state == true %}
      is currently <strong>on</strong></h2><div class="row"><div clas
s="col-md-2">
      <a href="/{{pin}}/off" class="btn btn-block btn-lg btn-default" r
ole="button">Turn off</a></div></div>
   {% else %}
      is currently <strong>off</strong></h2><div class="row"><div clas
s="col-md-2">
      <a href="/{{pin}}/on" class="btn btn-block btn-lg btn-primary" ro
le="button">Turn on</a></div></div>
   {% endif %}
   {% endfor %}
</body>
</html>
```

Running the web interface

Finally we cd to */home/pi/web-server*, and start this web server using the command:

```
sudo python app.py
```

To see the results open a web browser **http://:9000** which should display a page like this:


RPi Web Server

GPIO 17 is currently **off**

GPIO 18 is currently **off**

Simple and Intuitive Web Interface for your Raspberry PI

This interface comes from instructables Simple and Intuitive Web Interface (https://www.instructables.com/id/Simple-and-intuitive-web-interface-for-your-Raspbe/).

For this interface we are going to use PHP and CSS to interface the LEDs. The state of the led will be indicated by the color on the interface.

This interface also relies on the **Wiring PI library** installed above, so we do not need to install it again.

Since we have also setup the Apache web server we only need to confirm that we have PHP installed. The easy test is to use the web page:

```
<?php
    phpinfo();
?>
```

Which is named on this raspberry pi as phpinfo.php, lets have a look.

```
http://<ip-address>/phpinfo.php
```

Now that we know that php is active and working lets download and install the interface from the web site.

```
wget https://cdn.instructables.com/ORIG/FUQ/ZHUM/IBEWRSX4/FUQZHUMIBEWRS
X4.zip
mv FUQZHUMIBEWRSX4.zip web2.zip
```

Now you need to unpack in the web home folder.

```
cd /var/www/html
sudo unzip /home/pi/web2.zip
sudo mv Web\ 2.0 web2
```

Although we could run the page as is, I chose to modify the file for only 2 controls. To do this use this command:

```
sudo nano /var/www/html/web2/index.php
```

and change the following lines.

```
From:   $val_array = array(0,0,0,0,0,0,0,0);
To:     $val_array = array(0,0);

From:   for ( $i= 0; $i<8; $i++) {
To:     for ( $i= 0; $i<2; $i++) {
In two places
```

Now you can try it out by going to

```
http://<ip-address>/web2/index.php
```

It should look something like:

Closing Thoughts

We have seen how to create a web inteface to control operations on a raspberry pi through a we browser. The core thing to realize is that these web pages are using a system call, like gpio, to control hardware from inside a web page. The same ideas can be implemented to display data as well as control using php.

The small size and cost coupled with the large flexibility and capabilities of the Micro-controllers, like the Raspberry PI is what is driving the IOT (Internet Of Things) devices. With these tools you are able to create you very on IOT devices.

Written by John F. Moore (mailto:john.moore@lions-wing.net)

Last Revised: Mon Oct 15 17:04:43 EDT 2018

(http://www.w3.org/html/logo/)