Programming Project

IoT Garage Door Opener

A fun project to introduce beginners to Github, Python scripting, Arduino sketch, REST API, and JSON

By Mark Nguyen

Oct 31, 2016

Version 0.5

Table of Contents

Change Log 1
Introduction 2
  Executive summary 2
  Project Goal – IoT Garage Door Opener 2
  Notice and Disclaimer 2
  Parts List 3
Instructions 4
  1. Assemble the board 4
  2. Create a GitHub account and fork project repository 6
  3. Setup Arduino IDE on your computer 6
  4. Prepare your Arduino sketch for the ESP8266 7
  5. Flashing your ESP8266 10
  6. Test your completed board 12
  7. Connect your board to your garage door 12
Check Point (the rest is optional) 13
Setup your Raspberry Pi 13
Checking the status of your sensor – a Python example 13
  Use case 13
  Building our script 14
  Putting it all together 15
OpenHAB 16
  Getting started 16
  Configuration 16
Future topics 17
Share 18
I have questions 18
Links to parts 18
Summary 20

List of Figures

Figure 1. Printed Circuit Board 4
Figure 2. PCB after soldering completed 5
Figure 3. Standalone Web GUI 8
Figure 4. Wiring diagram to flash ESP8266 10
Figure 5. Testing your board 12
Figure 6. Wiring to garage door opener 12

Change Log

ProgrammingProject.GarageDoor-v0.1 (January 8, 2016)
Initial Draft after ADS SE presentation
ProgrammingProject.GarageDoor-v0.2 (January 18, 2016)
Finished instructions section
ProgrammingProject.GarageDoor-v0.3 (January 21, 2016)
Added version 2 of PCB
ProgrammingProject.GarageDoor-v0.4 (January 22, 2016)
Distributed to ADS Operation SEs
ProgrammingProject.GarageDoor-GMU (October 31, 2016)
Modified for students attending CyberSecurity Youth Conference

Introduction

Executive summary

With advances in wireless technologies, the World has evolved to be more connected than ever. Physical devices are embedded with software and sensors that collect and exchange data, allowing for more direct integration of the physical world into computer-based systems. The next generation is growing up in this digital world, therefore it is important to foster their passion for these technologies. Through a fun home automation project, this document introduces the reader to examples of Python, Arduino code (for IoT applications), REST API, and JSON formatting. To complete this project, the reader will also be navigating through GitHub, discovering a wealth of learning resources.

Project Goal – IoT Garage Door Opener

The primary goal is to educate the reader by introducing them to programming tools and examples. The final product will be an IP connected garage door opener and sensor. We will be using a REST like API to control our device, allowing us to control it using other applications that speak REST (e.g. APIC-EM). The hope is for the reader to customize and create other projects. For example: motorized blinds, automatic plant watering, wireless switch, etc. Extra credit when they publish their collection in their own GitHub repository.

Notice and Disclaimer

This document is for educational purposes only. The project described in this document requires the use of a soldering iron. The author is not responsible for any injuries (such as burns or electrical shock) as a result of building, installing, modifying, or anything related to this document. The reader should take caution if they attempt to modify this project for applications that require more electric power. Although the relay supplied with this kit is rated for 10A, the PCB traces are too thin to support that much current. Board versions 1 (yellow) and 2 (green) are used for lower power applications. Version 3 of the board (red) has wider traces that will support more current (roughly 7A), but do so at your own risk and only if you know what you are doing. Please be safe!

This document assumes the ready has some basic knowledge of networking (such as IP addresses).

Parts List

Here are the parts that are supplied with this document. A short description for each component is provided for reference (links to parts provided at the end of this document).

- One Printed Circuit Board – this is a custom PCB produced by the author. There are two versions. You can tell which version you have by its color. Version 1 is yellow and version 2 is green.

- Three Resistors – two will be used as step-up resistors to keep their respective pins high by default (~2000 ohms). A third to reduce the voltage sent to the transistor (~600 ohms).

- One ESP8266-01 – this is an Arduino capable wireless board. We will program this with the appropriate code to control our switch. What is an Arduino? https://www.arduino.cc/en/Guide/Introduction.

- One FTDI FT232RL USB to TTL Serial – use this to program the ESP8266. You will need a micro USB cable to connect this to your computer.

- One 1N400x Diode – used to protect the transistor from damage due to the back-EMF pulse generated by the relay.

- One micro USB Type B female connector (surface mount) – This will be used to supply power to your PCB. Only Vcc and Ground will be wired.

- Two 2-pin terminal block connector – one pair will be connected to your garage door opener, the other pair connected to your magnetic sensor.

- One 5V SPST electromechanical relay – this is a switch that is controlled by an electric current. This will connect to the garage door.

- One 2x4 female header block – this is soldered onto your PCB, then the ESP8266 is attached on top of it. This makes it easier to remove your ESP8266 when you want to reprogram it.

- Two electrolytic capacitors – this acts as a buffer to keep your input and output power smooth. They will be in the range of 22 – 100uF.

- One AMS1117 3.3V voltage regulator (surface mount) – used to step down the 5V input to 3.3V for the ESP8266.

- One 2N2222 NPN transistor – used as a switch to signal the relay.

- One magnetic contact switch – input sensor to detect whether the garage door is open or closed. (optional) This was not provided in the kit.

Instructions

1. Assemble the board

This is where you get out your iron and solder the components onto the PCB. Figure 1 shows the supplied PCB (version 1 is yellow, version 2 is green). If you are new to soldering, I recommend watching a couple of YouTube videos to learn. Take your time and make sure you correctly place each component and solder them securely.

Some notes:

Solder smaller components first.

The diode and capacitors must be installed in the correct polarity. The shorter lead of Electrolytic capacitors represents negative (-) and the longer lead is positive (+). In figure 1, you will note that C1 and C2 are the capacitors and negative side is square solder pad. For the diode, the cathode (-) is represented by the line on the cylinder and has a square solder pad.

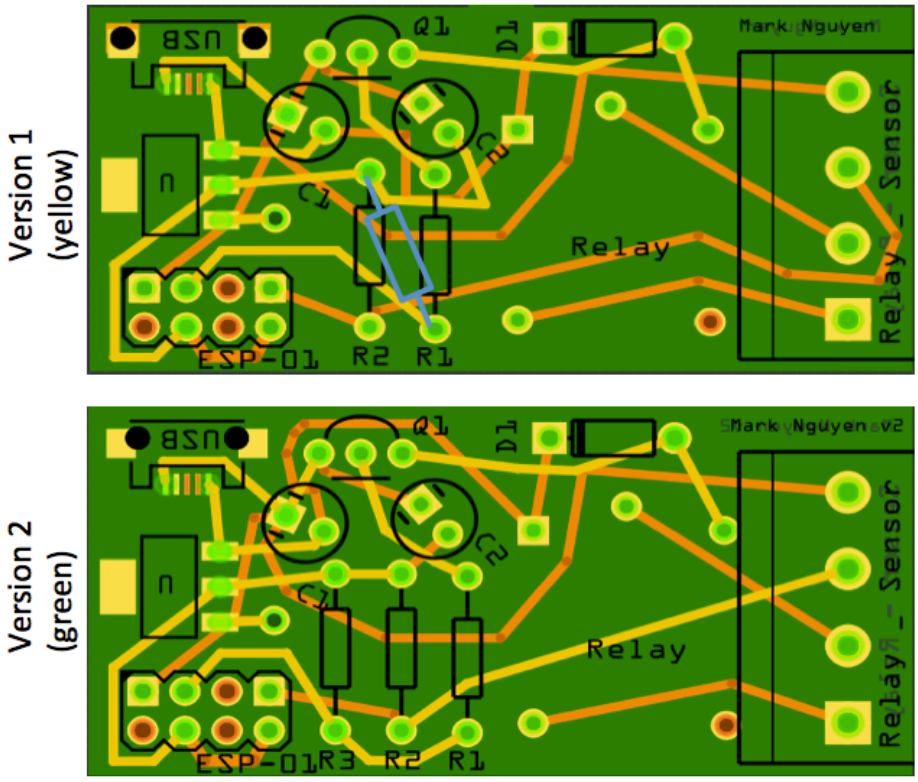
Keep in mind that the voltage regulator and micro USB connectors are surface mount components, and require a little more precision when soldering. The USB connector has two pins on the side that insert into PCB (not the 5 contact pins). Those with version1 boards will need to bend the side pins flat since they don't match up with the holes. Try to solder these two side pins to the square pads to help keep the connector securely fastened to the PCB.

This PCB has two layers, top and bottom. There are some holes that connect to the top and bottom layers. When soldering, make sure that you bridge both layers together that these holes. There is one hole without a component that needs to be bridge. This hole is located below C1 and to the right of the voltage regular. Fill this hole with enough solder to contact both sides.

Do not solder the ESP8266 onto the PCB! The female header block should be soldered on and the ESP8266 will be plugged onto the top of the header block.

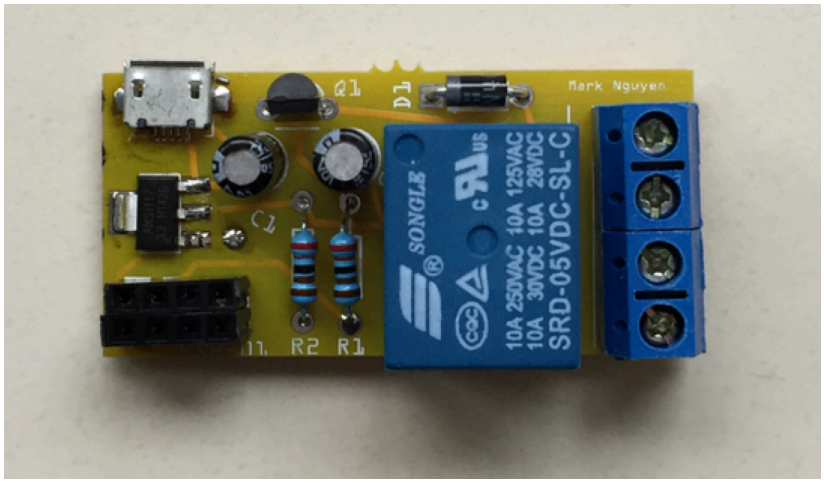
The first production of boards (version 1) was missing R3. Luckily there is an easy fix. Solder resistor 3 (680ohm) in the location (indicated in blue) in Figure 1. This can be done above resistor 1 and 2 or on the bottom of the board. There is enough room for two resistor leads to fit in the same hole. For version 2 of the board, solder R1 (2K ohm) and R2 (2K ohm) and R3 (680ohm) as depicted in the diagram.

Figure 1. Printed Circuit Board



One completed, your board should look like Figure 2 (this is version 1 with resistor 3 below the board). This picture does not show the ESP8266 plugged in yet. Do not plug in the ESP8266 until it is programmed. If you have a multi-meter, now would be a good time to use it to check your soldering joints.

Figure 2. PCB after soldering completed



2. Create a GitHub account and fork project repository

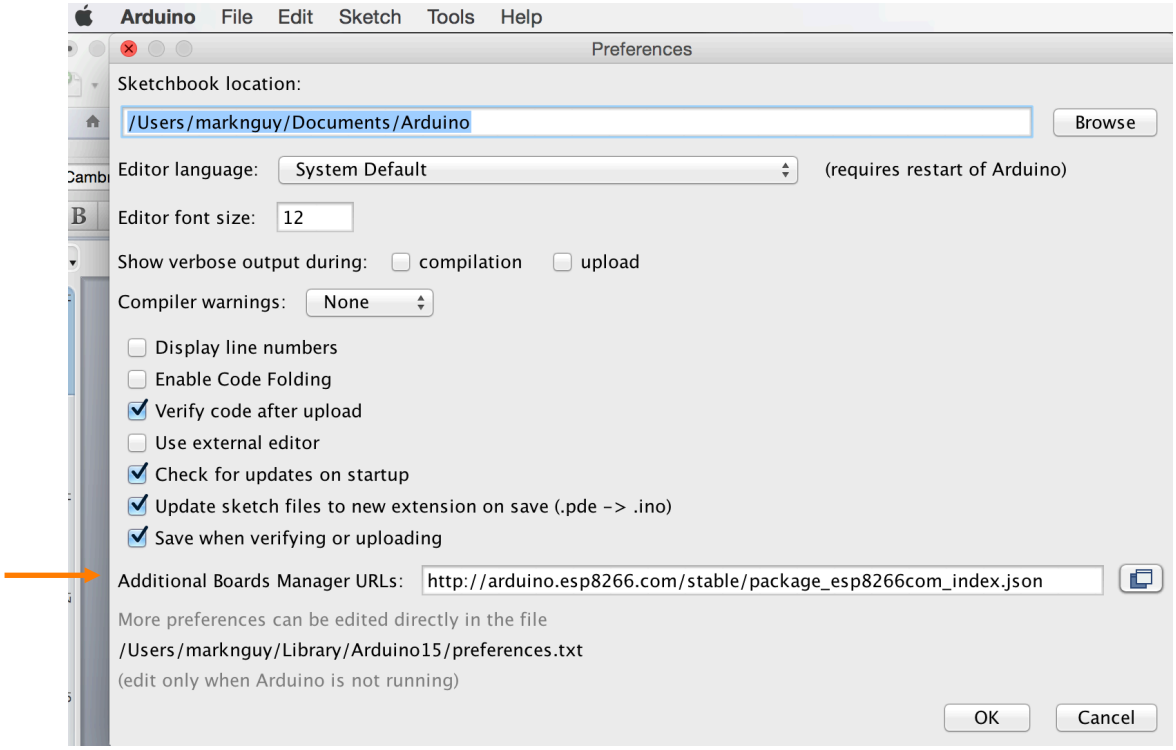
GitHub is much more than a version control system. It is a web-based VCS and social medial site. You can create and share your programs or copy others and track their changes. You can create a free account by following these simple instructions: https://help.github.com/articles/signing-up-for-a-new-github-account/. Once your account is active and you are logged on, go to the project repository at: https://github.com/marknguy/GarageDoorESP8266. On the upper right hand corner, select Fork. This will create a linked copy of the Garage Door project repository. You can track new changes that are made to my program. If you find mistakes in mine, you can fix it and make a pull request to the owner. You can create a new variant of the program and share it in the master repository. Very cool ☺.

3. Setup Arduino IDE on your computer

To push your program (aka flash) your code to your ESP8266, you must first install and setup Arduino IDE on your computer. IDE stands for Integrated Development Environment. An IDE is a software application that assists programmers develop code. It typically contains a source code editor, compiler, and debugger. Another example of an IDE is PyCharm, used for developing in Python. We will take a look at that later.

For now, go to https://www.arduino.cc/en/Main/Software and download and install the appropriate version for your computing platform. I'm using MacOS so my screen shots may be different than if you are running the Windows or Linux versions.

Once installed, go to Arduino->Preferences. In the field Additional Board Managers URLs, enter http://arduino.esp8266.com/stable/package_esp8266com_index.json.



Next, open Tools->Board-> and find and install the esp8266 platform. Once installed, go to the Tools-Board menu and select *Generic ESP8266 Module*.

You can find more information about this package here: https://github.com/esp8266/Arduino.

4. Prepare your Arduino sketch for the ESP8266

Let's look at our first piece of code. In Arduino terms, a *sketch* is a program for an Arduino. Download the *GarageDoor.ino* sketch from your GitHub repository and open it in Arduino. If you are anxious, at minimum, modify the `ssid` and `password` and proceed to step 5. For those who would like more details, continue reading.

We want to take advantage of the ESP8266's built-in web server to:

(a) Have a standalone user interface to control our garage door

(b) Provide an API to interact with other programs

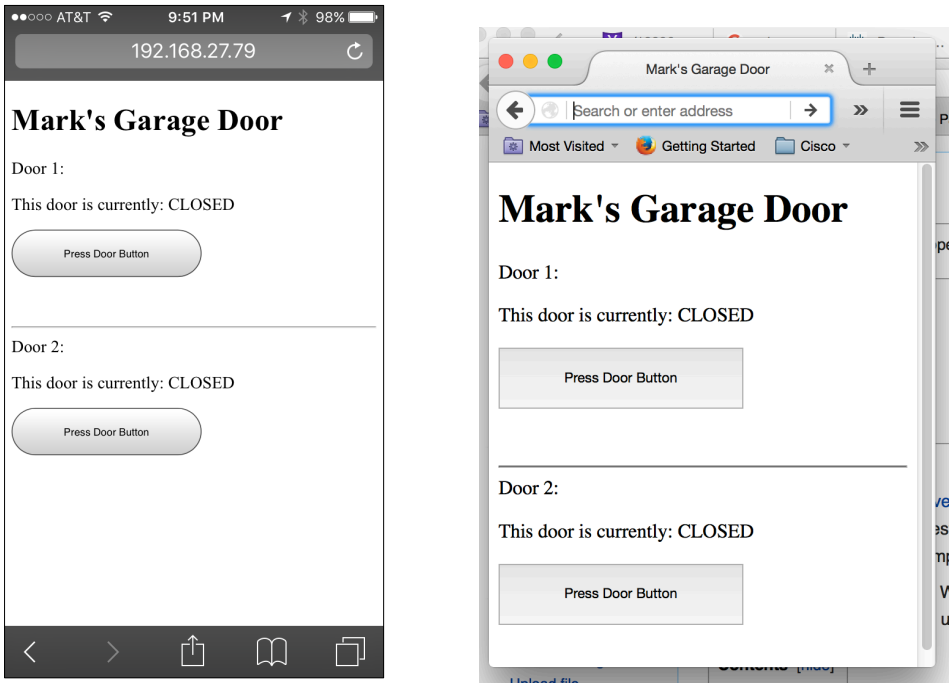
For (a), we want tell the ESP8266 to push out some basic HTML to the browser:

```
// Main page
  String m = "<!DOCTYPE html>\r\n<html>\r\n<head>\r\n<title>Mark's Garage Door</title>\r\n";
  m += "<h1>Mark's Garage Door</h1>";
  m += "<meta name=viewport content='width=400'>";
  m += "Door 1:<br>";
// calls a function that returns a String with the state of the switch
```

```
  m += GetSwitchState(client,3);
  m += "<form method=\"get\" action=\"/godoor1\"><button type=\"submit\" style='height:50px;
width:200px'>Press Door Button</button></form><br><br>\n";
  m += "<hr>";
  m += "</html>\n";
  client.print(m);
```

The webpage looks something like this on your smartphone or computer (this is a two door version):

Figure 3. Standalone Web GUI



If the user clicks on Press Door Button, their browser opens a relative link "/godoor1" on the ESP8266. Based on the sketch, when the ESP8266 receives an HTML request for "/godoor1", it will call two functions, `SendPulse()` and `WaitMessage()`.

```
// activates the door signal if called from Web page
  if (req.indexOf("godoor1") != -1) {
    SendPulse(2);
    WaitMessage(client);
    client.stop();
    return;
  }
```

`SendPulse()` tells the ESP8266 to raise the output of one of its GPIO pins to high for half a second. This simulates a button being pressed for a split second. GPIO stands for General Purpose Input Output. The GPIOs are what will signal the relay to open and close.

`WaitMessage()` outputs a JavaScript that displays a "Please Wait" countdown timer on the browser. This gives the garage door time to open or close.

```
// This page has a countdown timer to allow time for door to open or close
  void WaitMessage(WiFiClient cl) {
// Prepare the response
    String response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
    response += "<html><meta name=viewport content='width=400'>\r\n";
    response += "<script type=\"text/javascript\">\r\n";
    response += "window.onload = function() { setInterval(countdown, 1000); }\r\n";
    response += "function countdown() {\r\n";
    response += "  var num = parseInt(document.getElementById('timer').innerHTML);\r\n";
    response += "  if(--num < 0) location.href=document.referrer;\r\n";
    response += "  else document.getElementById('timer').innerHTML = num;\r\n";
    response += "}\r\n</script>\r\n";
    response += "<body>Activating the door - please wait...<br><br>\r\n";
    response += "This page will automatically refresh in <span id=\"timer\">14</span>
seconds...";
    response += "</body></html>\n";
    cl.print(response);
  }
```

Let's pause here and explain an important point. REUSE CODE WHEN POSSIBLE! I searched online on how to automatically redirect a browser back after a period of time. The above JavaScript I found by searching online and modifying for my needs. Never try to reinvent the wheel. There are lots of examples of code available for all sorts of tasks.

For (b), we want to create an API so that other applications can interact with our garage door opener. There are two possible operations that we can request of a garage door opener:

1) Activate the switch (this opens or closes the door)
2) Tell us if the door is open or closed

Since the ESP8266 has a built-in HTTP server, let's make our own REST API! REST stands for Representational State Transfer. In its most basic form, it communicates over HTTP requests. Check out the REST Basics (101) learning lab here: https://learninglabs.cisco.com/labs/tags/Coding.

Our garage door opener has only two operations, so we will keep it simple and perform these actions based on the URL in the HTTP request. When we see "door1" in the URL, we will trigger the switch. If the homeowner uses there browser to GET "http://<ip address>/door1", the relay will be triggered. Later, we will see an example of REST with a PUT request.

```
  if (req.indexOf("/door1") != -1) {
    SendPulse(2);
```

When we see “statusdoor1” in the URL, we will respond with the status of the sensor:

```
// Returns status of sensor in JSON format
  else if (req.indexOf("/statusdoor1") != -1) {
    if (digitalRead(3)) {
      s += "{ \"status\": \"OPEN\" }";
    }
    else {
      s += "{ \"status\": \"CLOSED\" }";
    }
    client.print(s);
```

Note the response will look like:

```
{ “status”: “OPEN” }
```

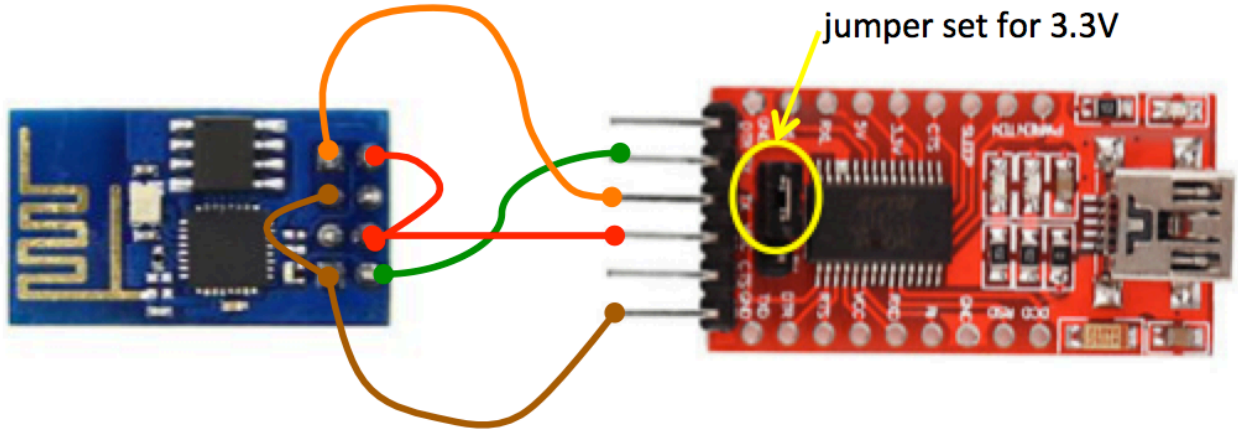
Guess what? You’re output is in JSON format! JSON stands for JavaScript Object Notation. It is a lightweight data-interchange format that is easy for humans to read and write. At its most basic form, JSON is a collection of name/value pairs. By outputting in JSON, other programs can easily parse through the data. In our case, it’s very simple: the variable “status” has either the value “OPEN” or “CLOSED”. We will use this JSON output later in a python script.

For more information about JSON, check out the Parsing JSON (202) learning lab: https://learninglabs.cisco.com/labs/tags/Coding.

5. Flashing your ESP8266

To upload your program to the ESP8266, you will use the USB to TTL Serial board that was supplied with your kit. Wire it to your ESP8266 according to the Figure 4. Make sure the USB programmer is set for 3.3V (as picture in the diagram), otherwise you will fry your ESP8266. Small alligator clip or hook clip or female-female breadboard jumper wires are useful for this temporary wiring.

Figure 4. Wiring diagram to flash ESP8266



Once your ESP8266 and serial board are connected, attach a micro USB cable to your serial board and connect the other end to your computer. Start the Arduino software. Load your program. Start the serial monitor (Tools->Serial Monitor). The serial monitor allows you to see the output from the ESP8266. You may have noticed the function `serial.println()` in your sketch. These were added to help you debug your application through the serial output.

Now you are ready to initiate the flashing operation. Click the upload button (upper left corner, looks like a right arrow). If your sketch compiles without any errors, the Arduino IDE will upload the code to your ESP8266. You will see a progress indicator at the bottom. The upload process takes about 10 seconds since we set the baud rate to 115200. Afterwards, you will see some output on your serial monitor stating that your ESP8266 is trying to connect to your WiFi network. Then it will display its IP address that it received from DHCP. Make a note of this address.

Note: This sketch intentionally uses DHCP instead of a static IP address. This is because most WiFi implementations block traffic from clients that set their own IP. Normally, if you want a static IP, you would assign it via your DHCP server (this is what I recommend). This also makes it easier to change your IP without having to re-flash your ESP8266. Having said that, if you still would like to assign a static IP to your board, add the following code:

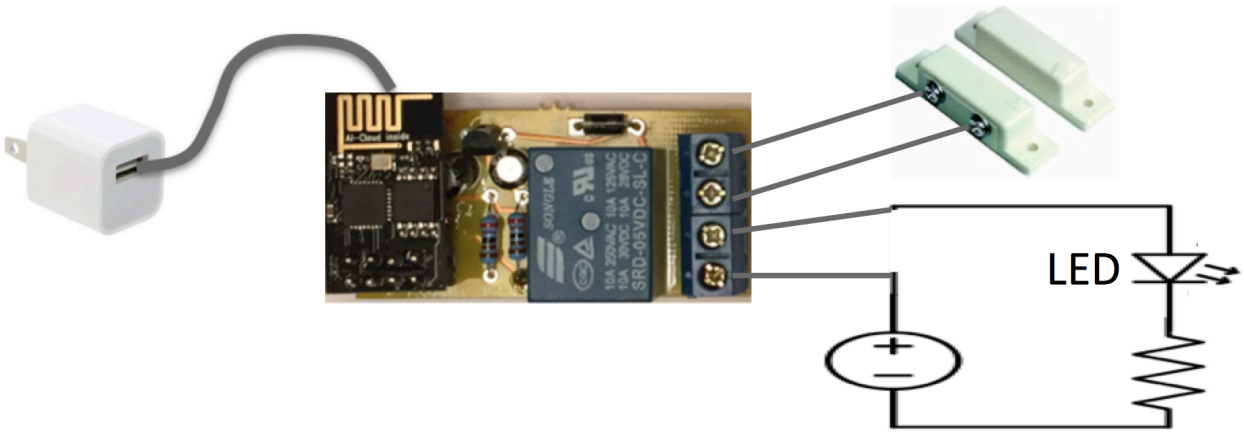
```
IPAddress ip(192,168,1,59);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);

void setup()
{
  WiFi.connect(ssid,password);
  WiFi.config(ip, gateway, subnet);
}
```

When the upload has completed, disconnect the USB cable and then the wires from your ESP8266. Plug your ESP8266 onto your PCB (on the 2x4 header block). Make sure the orientation points towards the center of the PCB, not away (see Figure 5).

6. Test your completed board

Now the fun part. Connect the magnetic contact switch to the pins of the terminal block for the sensor. Optional, connect a simple circuit with an LED to the pins of the terminal block for the garage door opener. If you don't have an LED, that's' fine, you will know the relay is working when you hear it click. See figure 5.

Figure 5. Testing your board



Power your board using a 5Volt/1Amp USB charger. The same kind of charger you use for your phone (with a micro USB cable). The board takes about 5 seconds to boot. Open the URL: *http://<ip-address>/* where the ip address is the one you saw in your serial monitor (the last octet may have incremented). You should see a page similar to Figure 3.

When you click "Press door button", you will hear an audible click as your relay activates for a half second. If you have an LED attached, you will see the LED blink. The webpage should switch to a "Please wait" page, than go back to the main page after 14 seconds.

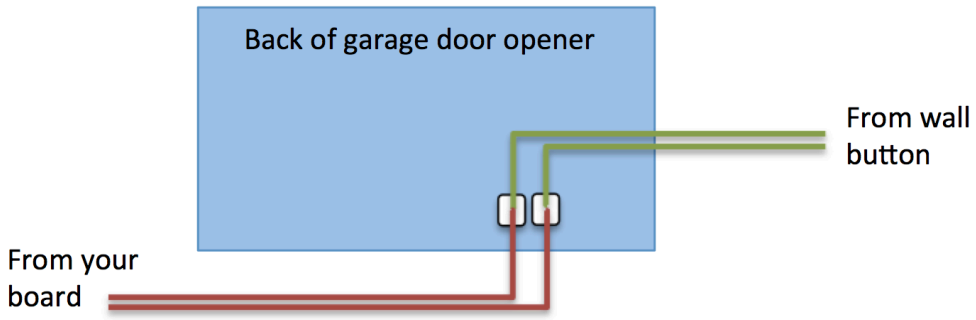
The status of the door (open or closed) will match the status of the magnetic sensor. If you place the contacts together, it will close the circuit and the sensor will read CLOSED. Conversely, if you pull the contacts apart, the circuit will open and the sensor will read OPEN. Try doing this a few times and refresh your screen each time to see the status change.

Try going to *http://<ip-address>/statusdoor1* and *http://<ip-address>/door1* to see the behavior. Other programs will use these links to interact with the board.

7. Connect your board to your garage door

Wire the board to your garage door opener in parallel with the wall button. See Figure 6. The magnetic sensor should be wired the same way as you had it in Figure 5. Plug in your board. Now you have a functioning IP accessible garage door opener!

Figure 6. Wiring to garage door opener



Check Point (the rest is optional)

At this point, we are done with the hardware and we can operate our new device as a standalone system through its built-in web browser. However, what if we want to be able to open our garage door from anywhere in the world? What about connecting this to a home automation server? This requires a home automation server. We can use any server, but for our example, we will use linux – specifically a Raspberry Pi. We chose a Raspberry Pi because they are relatively inexpensive and has plenty of processing power for our needs.

Setup your Raspberry Pi

The installation of a Raspberry Pi has been well documented so I will not try to re-write it. You can find instructions here: https://www.raspberrypi.org/help/. If you are not already familiar with linux, it would be beneficial to go through the fundamentals: https://www.raspberrypi.org/documentation/linux/.

Checking the status of your sensor – a Python example

Use case

Let's say that we want to query a sensor for its status, than report the result to a server. For example, perhaps we have a weather sensor that measures temperature, humidity, pressure, and wind speed. We want to check it every 15 minutes to record its results to plot on a graph. The results would come back in an easy to parse format such as JSON. For example, it might look like this:

```
{
  "temp" = 65,
  "humidity" = 40,
  "pressure" = 29.5,
  "wind" = 9
}
```

We know what the temperature is by parsing our output for the value in the field "temp".

We can push this temperature value to our server using a REST PUT function. Let's examine how this works by example in our script.

Building our script

The first line in a python script points to the location of the python executable. Should look something like this:

```
#!/usr/bin/python
```

Next we typically have a description of what the code does, followed by the libraries used by the code. Libraries are other pieces that we can reuse over and over again. Our library import statement looks like this:

```
import json, requests, time
```

The `json` library allows us to parse JSON data formats. The `requests` library helps us perform HTTP requests. The `time` library provides us some time related functions such as adding a pause.

After we import our libraries, we define some variables. Python supports basic data types such as integers or strings as well as more complex data structures such as arrays. First let's start by defining the URLs we will use.

```
url1 = 'http://192.168.1.59/statusdoor1'

OHurl1 = 'http://192.168.1.50:8080/rest/items/Garage1Sensor/state'

perform = True
```

Python does not require you to define the data type, it automatically knows. Python knows that the variable `url1` is a *string* because of the quotes. It knows the variable `perform` is of type *boolean* because of the **True** keyword.
We know from before that the URL `http://<ESP8266-ip-address>/statusdoor1` will respond with the status of our sensor (in JSON format).
`http://<server-ip-address>:8080/rest/items/Garage1Sensor/state` is the URL for the REST API that contains the state of *Garage1Sensor*. This URL is just an example. Application developers will publish the appropriate REST API for you to call to update data.
We will use the `perform` variable as a conditional check for our while loop.

Our first function call is to get the status of the sensor. We use the `get()` function in the `requests` library. The function call looks like this:

```
response1 = requests.get(url=url1)
```

The item inside the parenthesis is called the *argument*. We are passing `url1` as the argument in our function call. In English terms, this line can be explained as: Do an HTTP GET request to

URL http://192.168.1.59/statusdoor1 and place the response data into the variable called `response1`.
We know that `response1` contains data in JSON formatted text, which allows humans to easily read and parse it. However, computers are different. They prefer data to be indexed into arrays for easy retrieval later. This is where the `json` library comes in.

```
data1 = json.loads(response1.text)
```

In English terms, this reads: Take the text format of `response1` and load it into a json parser, than store the data into an array called `data1`.
In our garage door project, we have a simple example with one name/value pair. Therefore, we just need to look for the value for the field "status" in our array.

```
status1 = data1["status"]
```

In English terms: Look in the array `data1` for an index called "status"; retrieve the corresponding value and place it in the variable `status1`.
We know that `status1` will contain the value of either "OPEN" or "CLOSED". Now we just need to push that value to our server.

```
requests.put(OHurl1,data=status1)
```

We are passing two arguments to the requests.put() function. In English terms: Do an HTTP PUT request to the URL `'http://192.168.1.50:8080/rest/items/Garage1Sensor/state'` with the data stored in the variable `status1`.

Putting it all together

We add a while loop and a pause and our script looks like this:

```
#!/usr/bin/python

# Polls the ESP8266 sensor continuously for door contact status.
# This script provides an example of pull data using HTML calls.  The
# pull is using a REST-like method, but not truly REST (because the web
# ESP8266 does not implement full REST.  The script then parses the response
# for the appropriate date (in case the status of the sensor).  Then the
# script pushes the data to the server using a RESTful call (in this case the
# server does support REST).  This process is repeats over and over unless
# there is an issue with the ESP8266.

import json, requests, time

# URLs that request door sensor status
url1 = 'http://192.168.27.79/statusdoor1'

# REST API for OpenHab server
OHurl1 = 'http://10.1.1.70:8080/rest/items/Garage1Sensor/state'

# boolean to check if a valid response is received from ESP8266
perform = True
```

```
# loop
while(perform):
# reads response from query
 response1 = requests.get(url=url1)
# parses the JSON data
 data1 = json.loads(response1.text)
# retrieves the resultant data associated with parameter “status”
 status1 = data1["status"]
# checks sensor state
 if ((status1 == 'OPEN')|(status1 == 'CLOSED')):
# REST put call to server
    requests.put(OHurl1,data=status1)
    time.sleep(20)
 else:
    perform = False
```

We can now use this python script to poll our board for the sensor status and upload that information to another application. One such application is OpenHAB, an open source home automation software.

OpenHAB

As you learn to write your own scripts, you can take advantage of OpenHAB’s add-ons and APIs. Since the installation of OpenHAB is already well documented on their website, I won’t restate it here. However, I will give you some pointers to help you get started. Also I’ll share some configuration examples that you can use to tie in your new garage door opener.

Getting started

For general information about OpenHAB, visit their website: www.openhab.org.

Download the version for your platform: http://www.openhab.org/getting-started/downloads.html.

OpenHAB runs great on Raspberry Pi. If you choose this platform, follow the instructions here: http://www.openhab.org/getting-started/. I have found that this works better than installing through apt-get.

Installation of the demo is optional, but I would recommend it as it gives you some great examples to start with. Remember - don’t re-invent the wheel!

Download the mobile app for your smart device. There is an app for both Android and Apple.

Configuration

You will be setting up your own custom sitemap. For the garage door project, you might have something like this in your *sitemap* file:

```
Frame {
  Text label="Garage" icon="garage" {
    Text item=Garage1Sensor  icon="garagedoor"
      Switch item=Garage1Switch label="Activate Door 1" icon="none" mappings=[ON="Press button"]
}
```

The corresponding *items* file would have this included:

```
/* Garage */
Switch Garage1Switch         "Garage Door 1"  { http=">[ON:POST:http://192.168.1.59/godoor1]" }
Contact Garage1Sensor        "Garage Door 1 is [MAP(en.map):%s]"
```

The python script in our previous section would populate the variable `Garage1Sensor`.

If we want to bypass the python script (which was created solely as a learning example), we can do that directly with OpenHAB. OpenHAB supports REST calls and JSON. To do that, we change our line in our *items* file to:

```
String Garage1Sensor         "Garage Door 1 is [MAP(en.map):%s]"
{http="<[http://192.168.1.59/statusdoor1:30000:JS(ESP8266GetSensor.js)]" }
```

And we create a new file in the *transforms* directory called *ESP8266GetSensor.js* that contains:

```
JSON.parse(input).status;
```

Now OpenHAB will on its own fetch the status of the sensor (via HTTP GET) and parse the JSON data.

Try to do other things such as embed video feeds or weather forecasts onto your OpenHAB page. If you have smart devices such as Nest or Ring or Hue, you can add them too.

You should be able to control your garage door from your smart device anywhere in the world (as long as you connect to the OpenHAB cloud per instructions on their website).

Future topics

Bridging with Apple HomeKit – this would allow you to control devices using Apple devices (and Siri voice commands). I will document this once I have time, but if you're adventurous, go here:
https://github.com/nfarina/homebridge
and
https://github.com/tommasomarchionni/homebridge-openHAB

For Android users, HABDroid allows you to do voice commands.

https://github.com/openhab/openhab/wiki/HABDroid

Share

Please share your home automation projects. Creating something new is a great way to learn and practice your programming skills (or learn how to copy other people's examples). Here are some ideas:

Motorized blinds
Automatic plant watering machine (measures moisture in soil)
Someone can send a tweet to you, which changes the color of your outdoor Christmas lights
Voice control throughout the house (create your own Jarvis)

Here are different types of sensors to get your imagine flowing:
https://www.adafruit.com/categories/35.

I have questions

If you have questions, please post them in the GitHub repository for this project (under issues). https://github.com/marknguy/GarageDoorESP8266/issues. That's one of the social aspects of GitHub. This allows others to comment and help or find a solution to that someone else may have already come across.

Links to parts

In case you would like to build more of these, I've included the Amazon links to these components. I chose Amazon only to identify the parts for you - you may find these components cheaper from other sources, so shop around (e.g. ebay). In some cases, you can't buy single units. If you want just single units, you would have to go to electronic sites such as digikey.

One PCB
I've uploaded my design to this fabrication site. If you would like my PCB design file in Gerber format (so that you can order from a different fab), send me an email.
http://dirtypcbs.com/view.php?share=16156&accesskey=9219c039df56d57e0b2e95994e515850

Three Resistors. Two 2200ohms and one 680ohms.
http://www.amazon.com/Stackpole-Ohm-Resistors-Watt-Pieces/dp/B00B5R9BNO/ref=sr_1_11?ie=UTF8&qid=1453427844&sr=8-11&keywords=resistors+2.2k
http://www.amazon.com/E-Projects-Resistors-Watt-680R-Pieces/dp/B00BVORBN8/ref=sr_1_1?ie=UTF8&qid=1453427894&sr=8-1&keywords=resistors+680

One ESP8266-01

http://www.amazon.com/Diymall%C2%AE-Esp8266-Serial-Wireless-Transceiver/dp/B00O34AGSU/ref=sr_1_1?ie=UTF8&qid=1453426383&sr=8-1&keywords=esp8266+esp-01

One FTDI FT232RL USB to TTL Serial
http://www.amazon.com/Qunqi-FT232RL-Serial-Adapter-Arduino/dp/B014Y1IMNM/ref=sr_1_5?ie=UTF8&qid=1453426435&sr=8-5&keywords=ftdi+ft232rl

One 1N4007 Diode
http://www.amazon.com/Plastic-Encapsulated-Silicon-Rectifier-Diodes/dp/B00N3Y753U/ref=sr_1_2?ie=UTF8&qid=1453426496&sr=8-2&keywords=1n4007

One micro USB Type B female connector (surface mount)
http://www.amazon.com/Phone-Repair-Parts-Female-Socket/dp/B00EZB7LJS/ref=sr_1_1?ie=UTF8&qid=1453426644&sr=8-1&keywords=micro+usb+type+b+female+connector

Two 2-pin terminal block connector
http://www.amazon.com/30Pcs-5-08mm-Pitch-Terminal-Connector/dp/B00SWOL5XW/ref=sr_1_1?ie=UTF8&qid=1453427571&sr=8-1&keywords=2-pin+terminal+block+connector

One 5V SPST relay
http://www.amazon.com/uxcell%C2%AE-240VAC-125VAC-28VDC-JQC-3F/dp/B008SO6BDK/ref=sr_1_1?ie=UTF8&qid=1453426897&sr=8-1&keywords=5v+spst+relay

One 2x4 female header pitch 2.54mm
http://www.amazon.com/10pcs-Female-Header-2-54mm-Straight/dp/B00QBH8B4S/ref=sr_1_1?ie=UTF8&qid=1453426807&sr=8-1&keywords=2x4+female+header+pitch+2.54mm

Two electrolytic capacitors between 22 – 100 uF (25V or less otherwise they will be too big)
http://www.amazon.com/98Pcs-Cylinder-Aluminum-Electrolytic-Capacitors/dp/B00HR6MMJW/ref=sr_1_6?ie=UTF8&qid=1453427233&sr=8-6&keywords=electrolytic+capacitor+16v+22uF

One AMS1117 3.3V voltage regulator (surface mount)
http://www.amazon.com/AMS1117-3-3V-Voltage-Regulator-AMS1117-3-3V/dp/B00898NGL0/ref=sr_1_2?ie=UTF8&qid=1453427287&sr=8-2&keywords=AMS1117+3.3v

One 2N2222 NPN transistor
http://www.amazon.com/2N2222-Plastic-Encapsulate-Power-Transistors-600mA/dp/B00R1M3DA4/ref=sr_1_1?ie=UTF8&qid=1453427377&sr=8-1&keywords=2n2222

One magnetic contact switch
http://www.amazon.com/Magnetic-Sensor-Window-Warning-Contact/dp/B00ISPBJOK/ref=sr_1_3?ie=UTF8&qid=1453427473&sr=8-3&keywords=magnet+contact

Raspberry Pi (optional if you want to use OpenHAB)
https://www.amazon.com/CanaKit-Raspberry-Clear-Power-Supply/dp/B01C6EQNNK/ref=sr_1_4?s=pc&ie=UTF8&qid=1477920415&sr=1-4&keywords=raspberry+pi+3

Summary

This project was intended to spark interest in learning more about programming and engineering. We went through a fun and practical example of home automation. These basic types of scripts are very applicable to network automation as well. My hope is that many of you will continue to research and learn on your own.