

Introduction.

Mdhiggins has created a fantastic IR Blaster project that uses ModeMCU ESP8266 IOT micro controllers, that allows you to record infra red remote control commands from your existing remotes, then use various IoT home bridges to replay these to control your devices over local WiFi for things like Amazon Alexa, Apple Home, potentially Samsung SmartThings or your own locally hosted web page(what I'm doing).

The following is a step by step guide built by stresser1 based on information he sourced from Mdhiggins, various references, various esp tutorials, and guides to use PlatformIO and Arduino IDE.

The guide is based on the skill level of someone who has some IT background but this is the first time they've ever tried to do anything with home experimenter level micro controllers such as this.

Background

Many people have devices in their homes that are controlled solely by infra red remote controls. While there are bridging devices made by various little known manufacturers for 50-100 dollars or Logitech for hundreds of dollars; they have poor reviews and generally require you to expose your home automation to a 2rd cloud service beyond Amazon/Apple/Samsung etc. Some of us are uncomfortable using even the 1st layer of Amazon/Apple/Samsung home automation to begin with.

Enter the IR Blaster project. The components are readily available, cheap, customizable and best of all, you control how the data is transmitted and used. If you are like myself and don't want to have anything external to your home network have access to this project, you are able to do so.

Getting started

1. You will have to purchase components to build the hardware.
 - a) **ModeMCU ESP8266 WiFi IoT micro controllers.** The recommended is:
<https://www.amazon.com/gp/product/B01IK9GEQG/> - NodeMCU LUA CP2102 ESP-12E Internet WiFi Development Board Serial Wireless Module Internet for Arduino IDE/Micropython

I used a v3 board, which appears to be an odd duck. Due to misunderstanding some of the documentation I thought I had a 0.9 version, but it appears I have a 1.0 non standard version (<https://www.pcboard.ca/nodemcu-v3?search=nodemcu>) based on this (<https://frightanic.com/iot/comparison-of-esp8266-nodemcu-development-boards/>) information, while my board is not like the v3 board shown there, it looks identical to the v2(except slightly larger) which is probably some of the reason I got confused and thought the pic of the v3 was the v2 etc.

This likely caused some of my issues early on that Mdhiggins helped me through.

Regardless... the point of all this babbling is that there are various makes and models out there, so you will need to carefully research whatever you get, and you may need to try selecting 0.9 or 1.0 board when it comes time to upload the firmware.

b) **Prototyping equipment:**

1. **Breadboard**

I purchased the (<https://www.pcboard.ca/nodemcu-esp8266-breadboard>), note this is not a breadboard it's a breakout board. I am making this work, but honestly a real solder-less breadboard (<https://www.pcboard.ca/experimenters-solderless-breadboard.html?search=breadboard>) would work much more conveniently with appropriate wires

2. **Connecting Wires**

use solid core 22 awg wires – door bell wire works, or you can pick up some made wires like (<https://www.pcboard.ca/deluxe-breadboard-jumper-wires.html?search=jumper%20wires>) .

c) **Infra red receiver.**

You need this to capture your existing device IR remote control codes. You will only absolutely need one, but they aren't expensive so there is no reason not to get one per NodeMCU-esp8266. I got these ones that come with IR LEDs of unknown specs (https://www.amazon.ca/gp/product/B07FFQ9B9H/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1)

Mdhiggins recommends these (<https://www.amazon.com/gp/product/B00EFOQEUM/>)

d) **Infrared LEDs**

These need to be able to withstand a fairly high momentary amperage. The ones Mdhiggins recommends (<https://www.amazon.com/gp/product/B00ULB0U44/>) are spec'd to handle **1A** momentarily I don't know if the ones I picked out above will withstand that over time. I recommend these ones based on his and other people's success.

The reason these need to be able to handle high current, is so they can be put somewhere convenient to hit all the devices it will need to control from a decent distance, otherwise, you will need to use different resistors to lower the output (or just hope they don't burn out like me) and thus shorten the range. Some people hide the finished IR Blaster behind their devices and run an LED on a cable to it's IR receiver and affix them to it.

e) **2N2222 transistor.**

The transistor is important, because this allows the high powered LED to be driven by the higher current available right off the power input. The micro controller does not have the ability to output high current from the data pins, which will be used to send the signal. The transistor in this case would be described to work like an amplifier, in that you input the data signal, and it uses that to modulate the input voltage to the LED, outputting a much stronger identical signal. If you just connected the LED directly to the data pin it wouldn't be able to output as strong of a signal. As long as the transistor is an 2n2222 it will work. (<https://www.amazon.com/gp/product/B00R1M3DA4/>) (<https://www.pcboard.ca/2N2222-NPN-Transistor?search=2n2222>)

f) **Resistors**

You will need a 10 ohm and a 1k ohm resistor. The 1 k Ohm is used to connect the base or control pin on the transistor to the data output pin on the NodeMCU-esp8266. The 10 ohm resistor will protect the IR Led used to transmit the signal from the IR Blaster. Any resistors

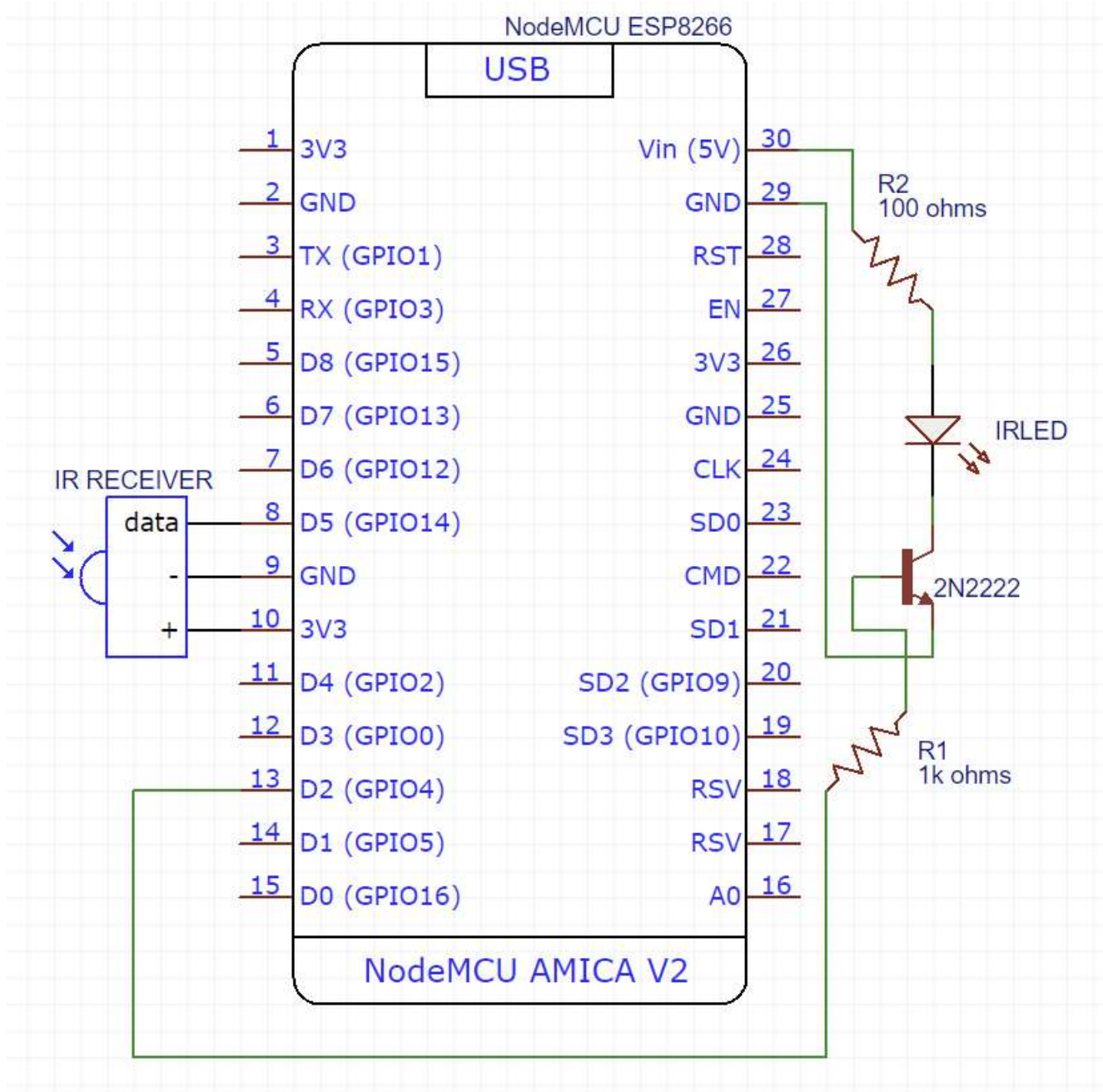
will work as long as they are the right ohms and can handle $\frac{1}{4}$ watt. Note the Amazon listing says 14watt, that's a formatting issue I believe.

(<https://www.amazon.com/gp/product/B00YX75O5M/>) (<https://www.pcboard.ca/56-value-resistor-assortment.html?search=resistors>)

NOTE in my pack, all the 10 ohm resistors were bad. Not a single one worked, so I adapted by putting 2 x 4.7 ohms in series. I've since lowered my resistor to 2.6 to more closely match the output of my remote... I strongly suspect I'm running into a limited current to the entire device situation, because I'm powering my device from my computer front port at the moment, this may be limiting me to 500ma, and therefore the resistor is further current limiting me and if I plug this into a wall wart I'll promptly burn out the ir led... I will have to do some more experimenting.

CONSTRUCTION

The schematic that Mdhiggins provided:

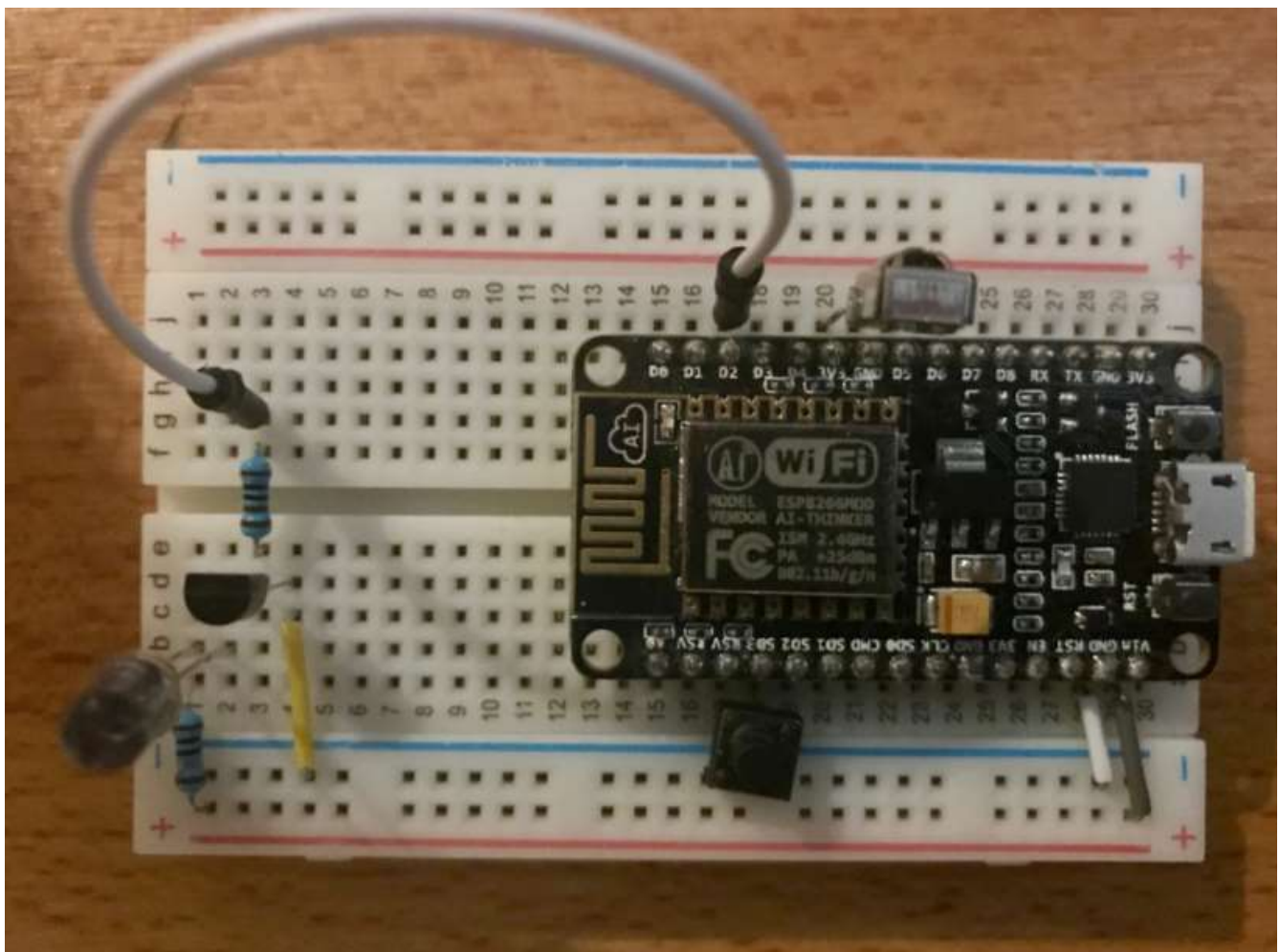


NOTE: Mdhiggins's schematic relies on using Vin(pin30) and Gnd(pin 29) assuming these are paralleled to the usb port. However, on the V3 I bought, these are not the same. On the V3 Vin only outputs 0.25v so there is probably power grooming circuit between the usb port and the vin so it can't be used to feed the transistor. Instead I used USBV and the adjacent ground between A0 and S3 to get 5volts. These are pins 18 for USB V and 17 for Gnd. On most boards those pins are not used however.

See the picture of the Mdhiggins's breadboard for orientation of components.

Notes:

1. The IR Receiver bulb is pointed away from the board
2. The transistor flat side is towards the board, the emitter (connection to negative) is the leg closest to the board. The base is the center pin which connects to the 1k resistor that connects to D2, the collector is the pin the negative side of the LED connects to.
3. the IR LED connects to a current limiting resistor. You will need to look up how to size this resistor based on the specs of your LED at 5volts. If you use the ones Mdhiggins recommends, then it should be 10 not 100 ohms.
4. The button shown in the middle is an (optional) normally open momentary switch that shorts pin sd3(GPIO10) to ground. When pressed and you trigger a reset, the device will reboot into AP mode so you can change the AP it's connecting to. This will be covered in the configuration section later.



SOFTWARE

Drivers – to use the USB port on the board, you will need the drivers for it.
(<https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers>)

Arduino IDE this is a piece of software used to program Arduino, with an additional board manager plugin to make it compatible with ESP based boards.

1. Download Arduino IDE (<https://www.arduino.cc/en/software>) WARNING do not install it via the windows store(selection called windows app – they mean the windows store not an app installer)... it won't function correctly and finding folders becomes a hassle since it puts it in temp folders and buries them in app data. Download the standard EXE.

2. Install Arduino Core for ESP 8266. This is fairly easy to do, you need to:

- a. open Arduino IDE,
- b. go to file, then choose preferences. A new window will open,
- c. paste the following link into the “Additional Boards Manager URLs”:

https://arduino.esp8266.com/stable/package_esp8266com_index.json

Note: If you need to add more board managers in the future, you can separate them with commas.

- d. restart Arduino IDE

Note: Aurdino IDE seems to be less than ideal. It seems the libraries are not fetched by referring to the .ino and the serial monitor causes the NodeMCU-esp8266 to crash when running commands.

Although, it appears that PlatformIO cannot fully erase flash memory (`pio run --target erase`) so you still need to use this to condition boards. Go into tools in Arduino IDE, in tools section, set the “Erase” option from “only sketch” to “all flash contents” then upload blink (this is explained more later) to clear any conflicting settings that can cause exceptions and reboot loop the NodeMCU-esp8266. This has been a problem multiple times... it might be because my boards are the “unofficial” v3 variants.

3. Install libraries: This is where things become murky and you'll want to use PlatformIO instead of Arduino IDE because PlatformIO will find the libraries needed automatically, while Arduino IDE will not. Over time the library names change and it becomes difficult to find the right ones, PlatformIO removes that from the equation.

Currently, Mdhiggins recommends installing these libraries:

ESP8266WebServer (is not in the library anymore I suspect it's covered in a combined ESP8255 and 32 library)

ESP8266WiFi (also not in the library)

ArduinoJson (available)

Time(available)

IrremoteESP8266(available)

1. In Arduino IDE, go to “Tools” then choose Manage Libraries to open the Library Manager.
2. filter for the above libraries and install them.
3. then restart Arduino IDE.

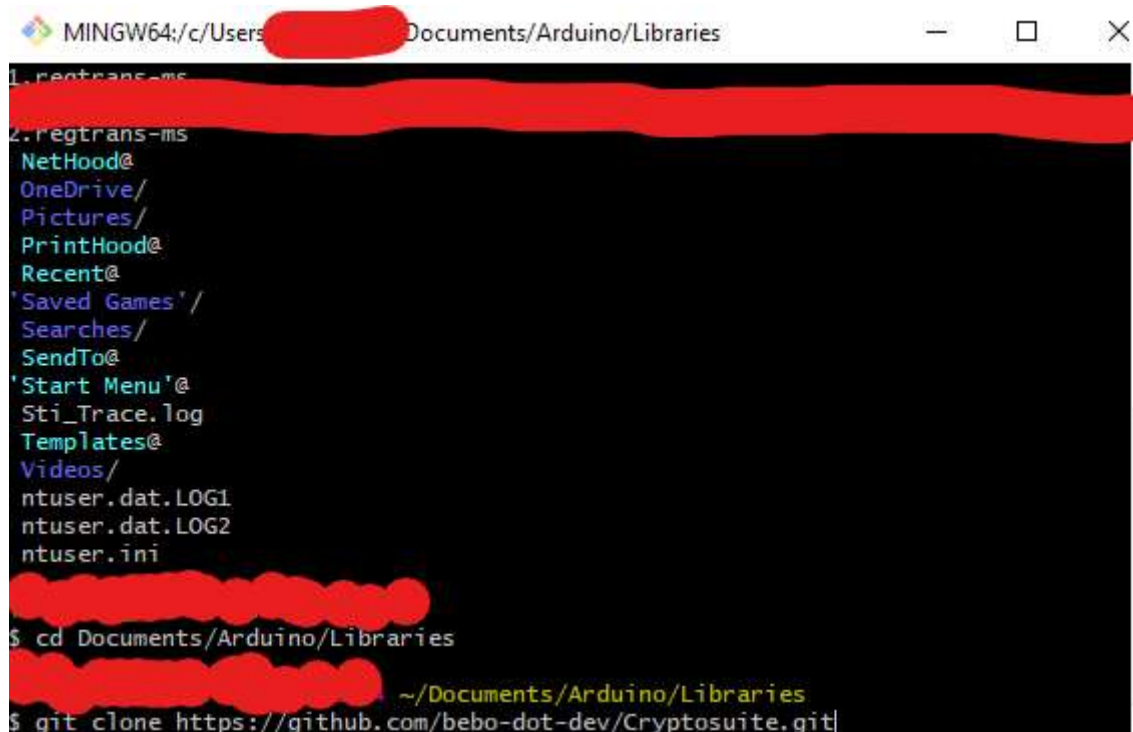
Next, to install **Cryptosuite** (This has to be installed manually, it won't be in the libraries) first you need to Clone the crypto suite from Git Hub to your local libraries.

1. install Git on your computer. (you can skip this and go to the PlatformIO because it can also clone repositories, but I did it this way. <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>)

Create a folder in the libraries directory (usually C:\users\\Documents\Arduino\Libraries\) called "Sha".

Find the cryptosuite project on Git Hub (<https://github.com/bebo-dot-dev/Cryptosuite>) click on the green Code button, press the clipboard icon next to the html link.

Open the Git terminal you installed on your local computer and navigate to the C:\Users\\Documents\arduino\Libraries folder using unix/linux commands.



```
MINGW64:/c/Users/[redacted]/Documents/Arduino/Libraries
1. regtrans-ms
2. regtrans-ms
NetHood@
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games' /
Searches/
SendTo@
'Start Menu'@
Sti_Trace.log
Templates@
Videos/
ntuser.dat.LOG1
ntuser.dat.LOG2
ntuser.ini
[redacted]
$ cd Documents/Arduino/Libraries
~/Documents/Arduino/Libraries
$ git clone https://github.com/bebo-dot-dev/Cryptosuite.git
```

Then use the command `git clone` and paste in the copied web address. This will clone the repository into a folder called `CryptoSuite`.

Copy the contents of it into the Sha folder you made there.

Then Clone `WiFiManager` with the same process, except don't copy it to the Sha folder.

Next we will install the Exception Decoder. This will be useful if we have a crash with a stack dump. This will allow us to decode it so that the developer (mdhiggins) can troubleshoot the code. (it will likely be something you're doing wrong though, as was the case with me)

1. Go to git hub, (<https://github.com/me-no-dev/EspExceptionDecoder>) download the EspExceptionDecoder-1.1.0.zip (or whatever version is the latest stable)
2. In your c:\users\\Documents\Arduino folder create a new folder called "tools"
3. unpack the trace decoder into the tools directory. It should look rather redundant:
c:\users\\Documents\Arduino\tools\EspExceptionDecoder\tool\EspExceptionDecoder.jar
4. restart Arduino IDE to install the tool.

It should now be under the tools menu as ESP Exception Decoder.

If you get a stack dump in the serial monitor, then you will need to copy it, and use the decoder. Bear in mind, that you need to compile the sketch to build an .elf file so the decoder can function. This is why you needed to install all the libraries even though we don't intend to use Arduino IDE to upload firmware.

First program

Now, We can condition a board. Platform IO lacks the ability to fully erase a board's flash, while Arduino IDE can. We can now plug an NodeMCU-esp8266 into our computer and test to make sure things are working.

1. Open Arduino IDE.
2. Go to "Tools", "Board Manager" and in the new window filter for "esp8266" Install it.
3. mouse over the ESP8266 entry below the board manager, and choose the appropriate board.
4. in the sketch paste the following code which will blink the LED on and off every second, and write to the serial monitor:

```
/*
ESP8266 Blink
Blink the blue LED on the ESP8266 module
*/

#define LED 2 //Define blinking LED pin

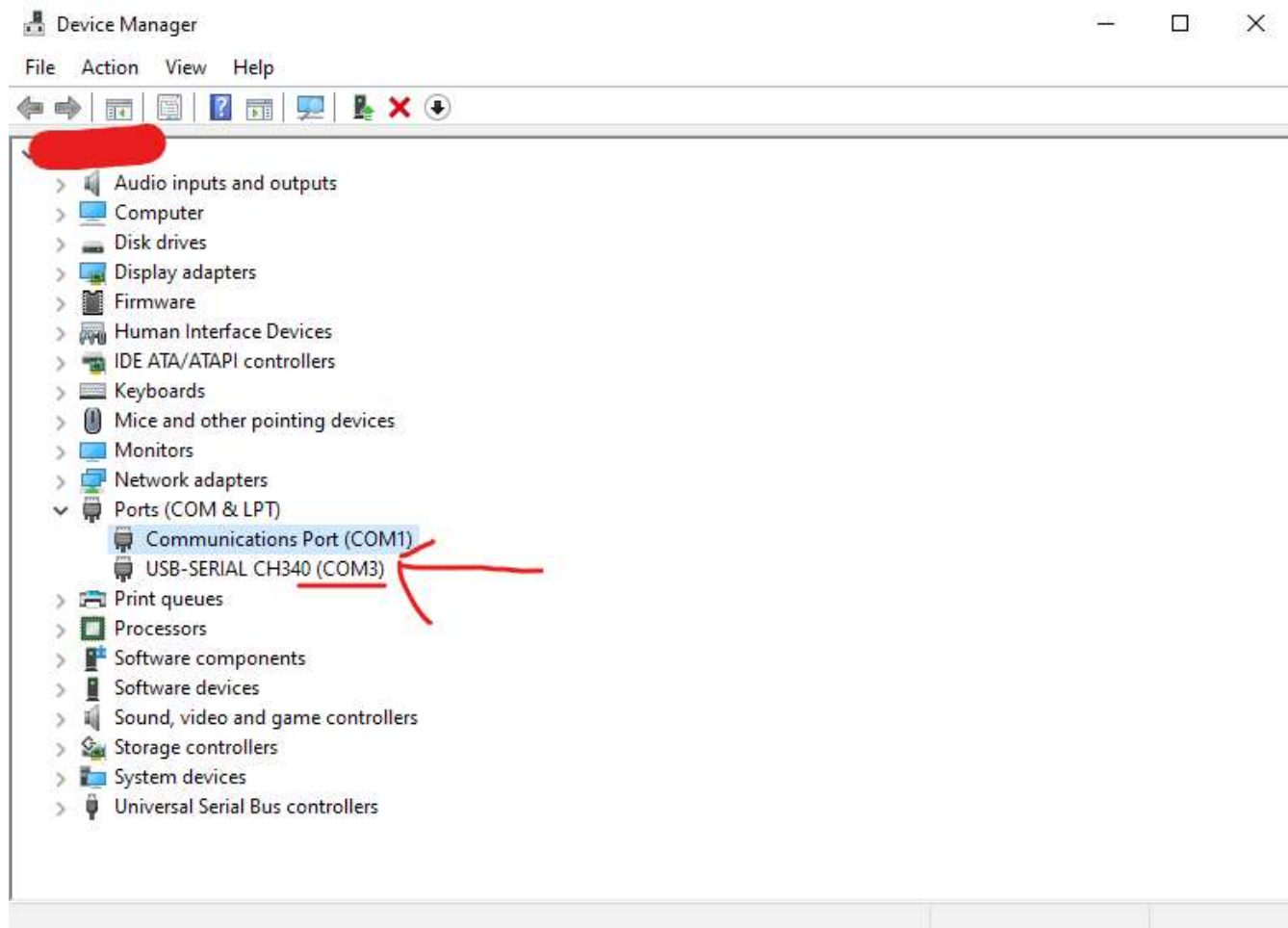
void setup() {
  pinMode(LED, OUTPUT); // Initialize the LED pin as an output
  Serial.begin(115200);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED, LOW); // Turn the LED on (Note that LOW is the voltage level)
  Serial.println("on");
  delay(1000); // Wait for a second
  digitalWrite(LED, HIGH); // Turn the LED off by making the voltage HIGH
```



```
Serial.println("off");  
delay(1000); // Wait for two seconds  
}
```

5. Then go to tools again, and choose “erase flash” and set it to “All Flash Contents” This is important for conditioning boards. If you put a sketch onto the board and it's not quite right, loading a new sketch in PlatformIO may not clear the flash properly and cause random crashes and reboot loops.

6. Plug in your NodeMCU-esp8266 that you want to put this program onto. Open Device Manager on your computer to figure out what Com Port the drivers will establish the board on. Once you know, go to tools, and set the port to the right com, in my case it is COM 3.



Then go to Sketch>Upload to compile and load this sketch into the board. Once it's done compiling it will load the sketch and reset. At this point your board LED should be blinking on and off, changing every 1000 ms.

7. Lets now view the serial monitor; go to tools, and choose Serial Monitor, a new window will open and you will see the device printing On/Off repeatedly as it blinks the led on and off. Press the reset button on the board to see the boot up sequence.

This board is now ready for use.

Note: you cannot have the Serial Monitor Running while uploading.

PlatformIO.

1. install Visual Studio Code (<https://code.visualstudio.com/>)
2. once that is installed, then install PlatformIO (<https://platformio.org/>)
 - Open VSCode Extension Manager
 - Search for official PlatformIO IDE extension
 - Install PlatformIO IDE.

Review the quick start guide. (<https://docs.platformio.org/en/latest/integration/ide/vscode.html#quick-start>)

NO REALLY, Go through the quick start guide, it has a lot of functions you will need that we will be talking about in a bit.

IRBlaster Project – Compile and Upload

Finally we are here, you've built the board, you've installed the suite of software you need to troubleshoot and condition boards, and test ran a blinky light firmware that's probably annoying you right now blinking in your face...

Now we want to upload MdHiggins's IR Blaster program and start doing the cool things you wanted to do when you started reading 10 pages ago.

- 1) Go to Git Hub, IR Blaster by Mdhiggins and choose the green code button, press the clipboard button to grab the repository link. <https://github.com/mdhiggins/ESP8266-HTTP-IR-Blaster>
- 2) Open VSC (Visual Studio Code) and click on the Platform IO Home button – You found it in the quick reference guide right? (little house on the tool bar along the bottom)
- 3) Click on the alien head on the tool bar on the left. This will open the quick access for PlatformIO.
- 4) Go to miscellaneous and choose “Clone Git Project” a box at the top of the screen will open and ask for the repository link, paste it in.
- 5) Press enter and choose a location to put the local clone. I suggest creating a folder far away from Arduino IDE's folder. Perhaps Documents/PlatformIO/
- 6) choose open, then yes I trust the authors.
- 7) In the project explorer, expand the src folder and click on IRController.ino
- 8) You will get an error about C/C++ intelliSense, you can ignore this.
- 9) In the ino you can see a bunch of include statements then a set of user settings you can modify for your use.


```

1  #include <FS.h> // This needs to be first, or it all crashes and burns
2
3  #include <IRremoteESP8266.h>
4  #include <IRsend.h>
5  #include <IRrecv.h>
6  #include <IRutils.h>
7  #include <ESP8266WiFi.h>
8  #include <WiFiManager.h> // https://github.com/tzapu/WiFiManager WiFi Configuration
9  #include <ESP8266mDNS.h> // Useful to access to ESP by hostname.local
10
11 #include <ArduinoJson.h>
12 #include <ESP8266WebServer.h>
13 #include <ESP8266HTTPClient.h>
14 #include <ArduinoOTA.h>
15 #include "sha256.h"
16
17 #include <Ticker.h> // For LED status
18 #include <TimeLib.h>
19
20 #include <LittleFS.h>
21
22 // User settings are below here
23 //+=====
24 const bool getExternalIP = true; // Set to false to disable querying external IP
25
26 const bool getTime = true; // Set to false to disable querying for the time
27 const int timeZone = -5; // Timezone (-5 is EST)
28
29 const bool enableMDNSServices = true; // Use mDNS services, must be enabled for ArduinoOTA
30
31 const bool bypassLocalAuth = true; // Allow local traffic to bypass HMAC check
32
33 const unsigned int captureBufSize = 1024; // Size of the IR capture buffer.
34
35 const bool toggleRC = true; // Toggle RC signals every other transmission
36
37 #if defined(ARDUINO_ESP8266_WEMOS_D1R1) || defined(ARDUINO_ESP8266_WEMOS_D1MINI) || defined(ARDUINO_ESP8266_WEMOS_D1)
38 const uint16_t pinr1 = D5; // Receiving pin (GPIO14)
39 const uint16_t pins1 = D6; // Transmitting preset 1 (GPIO12)
40 const uint16_t configpin = D2; // Reset Pin (GPIO4)
41 const uint16_t pins2 = 5; // Transmitting preset 2
42 const uint16_t pins3 = 12; // Transmitting preset 3
43 const uint16_t pins4 = 13; // Transmitting preset 4
44 #else
45 const uint16_t pinr1 = 14; // Receiving pin
46 const uint16_t pins1 = 4; // Transmitting preset 1
47 const uint16_t configpin = 10; // Reset Pin
48 const uint16_t pins2 = 5; // Transmitting preset 2
49 const uint16_t pins3 = 12; // Transmitting preset 3
50 const uint16_t pins4 = 13; // Transmitting preset 4
51 #endif
52 //+=====
53 // User settings are above here

```

10. set your options, make sure the time zone is correct. More information on these options is available on the Git Hub. I personally only changed:

getExternalIP to false, since I don't want traffic outside my local network accessing this and I don't intend to use Amazon or other services,

timeZone to ensure the time is correct and I don't end up with WiFi connection issues.

bypassLocalAuth to false to make it require a pass code to accept commands even on the local network. This is more for access control than for security as the passcode is only protected by your WiFi security. If you are using external links you are advised to use HMAC for authentication. Note you may want this set to true if you are using HMAC during your setup phase, as once HMAC is enabled, it is very hard to troubleshoot if you can't run local commands. In the future versions there will be MQTT implemented as an alternative authentication method. <https://mqtt.org/>

If you need to change these parameters you can usually get away with re uploading over top of the existing and it won't even reset your wifi information unless you first upload blink with the erase all flash function set in Arduino IDE.

Lower down is a spot that says “//Do not modify these values, these are placeholders that WiFi manager will override.” You can change these to reflect your network, so when you configure via WiFiManager you don't have to delete out all the ip addresses or retype your gateway/dns server every time you set one of these up. Mdhiggins just doesn't want people to think setting those will bypass the wifimanager config step – they will not, but setting these can save you a few seconds configuring the AP connection info in WiFiManager on first boot.

First connect to your router and set up a reservation or static IP for the blaster to use. Record this information as well as the gateway and DNS. On most home routers DNS and Gateway are both the same and usually end in “xxx.xxx.xxx.1”.

11. compile and upload the code

- a. click the alien head again, and under miscellaneous choose “PlatformIO Core CLI”
- b. in the new terminal window in the bottom right quadrant, type “PIO run -t upload.
- c. the program will compile and be uploaded to the board.
- d. once it's done uploading, press the plug on the bottom bar to open PlatformIO's serial monitor

12. connect and configure the board to connect to your Local Access Point (AP)

- a. on a wifi device connect to the new SSID “IR Controller Configuration”. If it doesn't automatically open a browser to the WiFi Manager configure page, open one and go to 192.168.4.1. On the serial monitor you should see the traffic and a countdown until it reboots.
- b. click on the configure button. It will pause as it draws the local SSIDs it can sense. If it stays blank but you've seen the local SSIDs in serial monitor, you may need to refresh your browser.
- c. click on the correct SSID, give it the WiFi password to connect, and fill out the information per the router and the IP address you created in step 11. User ID is your amazon user id if you want to use Alexa. Leave it blank otherwise. And Passcode is the code you need to send (in the

clear) to trigger an IR code if you are coming in from external, and/or local traffic if you have set `bypasslocalauth` to false.

d. click the save button and in the serial monitor you should see that it received the information as required, and will switch over to connect to your AP.

13. Now you can connect your computer back to your AP, and in your browser navigate to the IP address you gave your IR Blaster.

Note, add `/?pass= <passcode>` if you set one and `bypasslocalauth` is set to true or it will not allow you to access it.

14. Assuming you have your IR Blaster wired correctly with the IR receiver, you should be able to start recording codes, to start recording codes point your remote control you want to emulate at the receiver at fairly close distance (1") and press it a couple of times with a pause between. Refresh the page and you should see it listing the received codes. If it doesn't quite get all of a code it will be an odd size and be unknown... look for a brand name and a code to be sure. Sometimes there will be a device that does not fit a known standard, to deal with those, see the readme on the git hub.

Codes Received

Received	Command	Type	Length	Address
18:19:05	E0E007F8	SAMSUNG	32	0X7
18:19:05	FCF87D96	UNKNOWN	30	0X0
18:19:05	7DBAE47F	UNKNOWN	7	0X0
18:19:05	E0E007F8	SAMSUNG	32	0X7
18:19:05	7616B72B	UNKNOWN	33	0X0

GPIO 14 Receiving
GPIO 4 Transmitter 1
GPIO 5 Transmitter 2
GPIO 12 Transmitter 3
GPIO 13 Transmitter 4

1073381ms uptime; EPOCH 1625008755 / 1625008757 (2)

This was several pushes of the same button, but you can see the codes were different except for the 2 identified as Samsung.

Once you have a code you can craft a URL to trigger it to transmit that IR code to the device you want to control.

ie. <http://192.168.2.152/msg?code=E0E040BF:SAMSUNG:32&pulse=3&pdelay=10>

Will send 3 pulses of the code “E0E040BF:SAMSUNG:32”, 10ms apart. This timing is similar to my remote as near as I can tell. My remote sends pulses as long as you hold the remote, so 3 seems a good number.

You can add or remove parameters by using an ampersand to add on codes.

Code words available are:

pass - password required to execute IR command sending

code - IR code such as A90:SONY:12

address - (optional) Additional address data for NEC codes. Hex format

pulse - (optional) Repeat a signal rapidly. Default 1

pdelay - (optional) Delay between pulses in milliseconds. Default 100

repeat - (optional) Number of times to send the signal. Default 1. Useful for emulating multiple button presses for functions like large volume adjustments or sleep timer

rdelay - (optional) Delay between repeats in milliseconds. Default 1000

out - (optional) Set which IRsend present to transmit over. Default 1. Choose between 1-4.

Corresponding output pins set in the blueprint. Useful for a single ESP8266 that needs multiple LEDs pointed in different directions to trigger different devices

Example: `http://xxx.xxx.xxx.xxx:port/msg?code=A90:SONY:12&pass=yourpass`

Note: The recorded codes are not stored long term on the device, you're just using it to get the data so you can build your own triggers later. The blaster will just transmit the code given to it. It won't check to see if a code sent to it was recorded prior, it will just transmit any codes given to it on demand.

15. Next try out your device by making URLs that trigger different codes to control the device.

Once you are at this point the rest of the readme will make it easy to add on additional functions such as connecting this to Amazon Alexa, or as I'm doing, creating a locally hosted web page as a universal remote control.

TROUBLESHOOTING

To see if the IR LED is actually transmitting, you can use the camera on your smart phone, any digital camera will work generally. To test, first point your normal remote at your camera and press any button. You should see the remote's IR LED pulsing on the camera in purple light that is invisible to the naked eye, if you don't and you know the remote works, try a different camera. Once you have a digital camera that can see IR in purple, you can then trigger a code on the IRBlaster and check to see if you get purple light from it. A single pulse will be a fraction of a second, so I suggest 5 or more pulses to ensure you see it.

If you are having errors compiling in PlatformIO, hit the garbage can in the bottom bar to clear out the build folder. It may have something in a bad state causing it to fail. It should always compile.

If you are still having problems with stability, use Arduino IDE to condition the board by loading the Blink program (page 9 – first program)

Reset WiFi

Once the AP has been set, if you want to go back to the configuration menu, short pin GPIO10(SD3) to ground, and then press the reset button while this pin is grounded. - The program only checks for this pin to be grounded on boot, it does not check in the main loop (Note this pin doesn't work on my v3 boards during start up, So I switched the configuration pin to GPIO12 and set output 3 to GPIO 16 since it was using 12.)

Mdhiggins comments that this is likely due to the V3 keeping a live current going through the default config pin (10 in this case.)