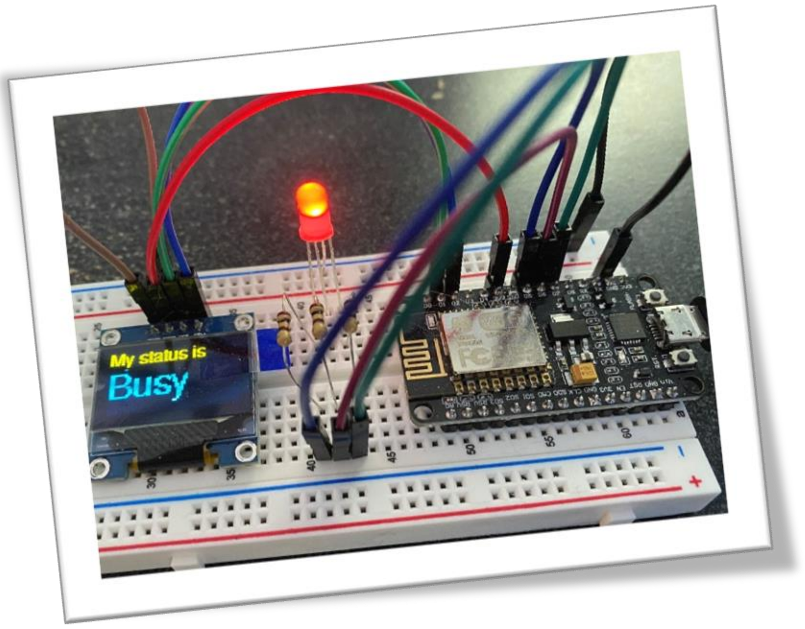
Busy Buddy v1

Introduction

Busy Buddy started off as a fun and extensible project that provides a way to visualize your Teams status, what is known as your "Presence" in Microsoft 365. Busy Buddy was able to focus on just the presentation side, the LED and screen, because of the fantastic work of Isaac Levin on his Presence Light project. This software, available via GitHub for Windows, Mac and Linux, and the Microsoft Store for Windows, is an application that monitors your Teams Presence status and notifies devices [like Busy Buddy] when that status changes. Out of the box, Presence Light talks to several commercially available smart lights and also provides a custom interface, which is how it talks to Busy Buddy. As such, you need to have Presence Light installed if you want to use Busy Buddy to indicate your Teams status.

That said, a funny thing happened while building Busy Buddy – I was able to make it more generic so that it could be used to indicate the status of pretty much anything you can dream up. It has a simple REST API which accepts a POST that includes a parameter for the status text ("Free", "Busy", etc.), as well as a parameter for the color. This means that you can send a simple POST message to Busy Buddy to, for example, indicate that your DevOps build pipeline passed or failed, or that your garage door was left open, or pretty much anything along those lines. All the text is configurable, as well as the LED colors, and you can do that without ever having to modify and compile the source code.

As is stated on the GitHub site, Busy Buddy is a DIY project intended for educational and personal use only. It is not intended for commercial use or to be relied upon for any professional or medical purposes. The creators and contributors of Busy Buddy are not responsible for any damage or injuries that may result from the use of this project. By using the Busy Buddy software and/or hardware plans, you acknowledge that you understand and assume all risks associated with its use. Please use caution and common sense when working on any DIY project.

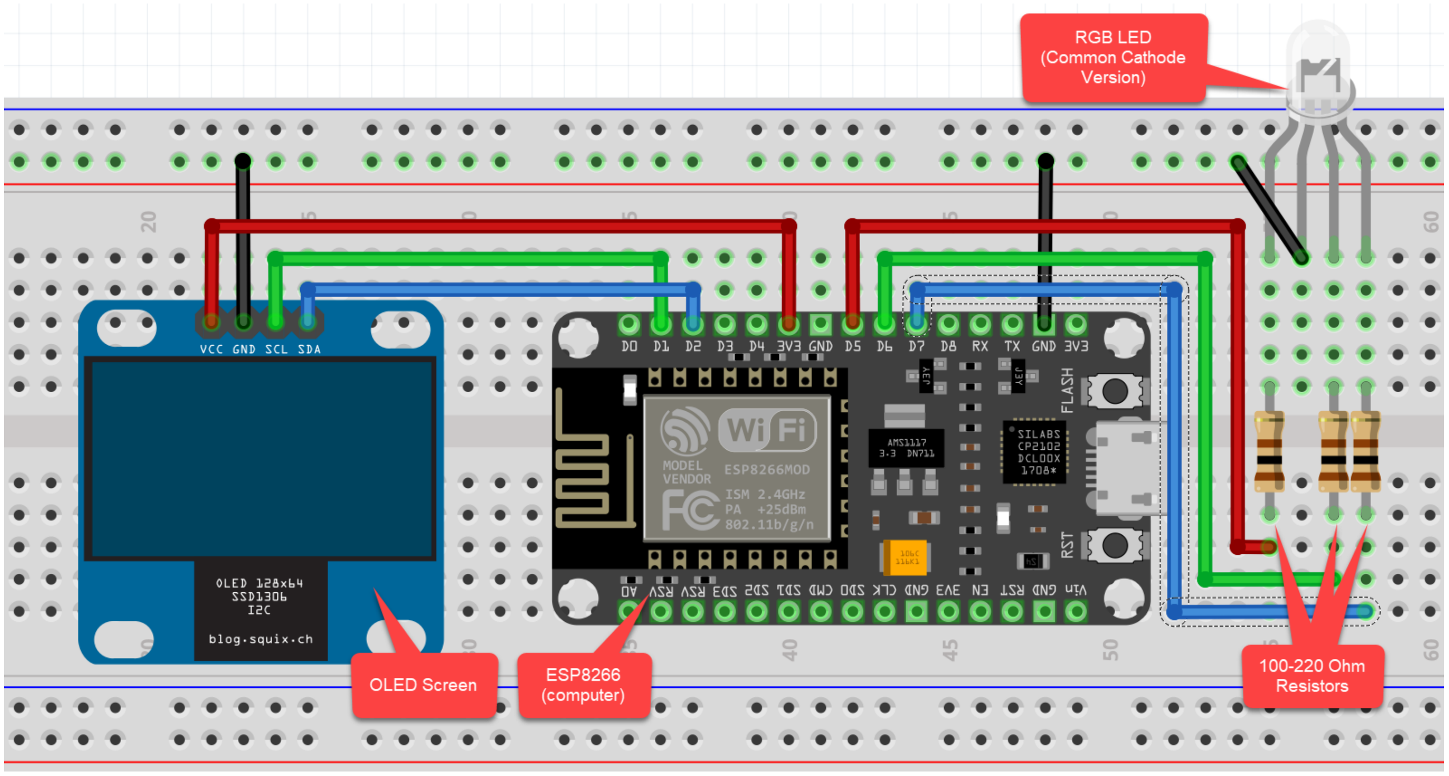


Building Busy Buddy

Overview

Now, if you found that photo of a bunch of wires and gadgets plugged into some kind of board with a bunch of holes in it a little bit scary, don't worry – I'll walk you through it 😊 That picture is just a project "breadboard"; it's an easy way for DIY tinkerers to play with electronics components and create circuits without having to solder wires together. If you don't have one, and aren't interested in getting one, no worries – we can get by without it, especially if we're just creating a Busy Buddy to use [not to tinker with].

When we take a more abstract look at what makes up Busy Buddy from the hardware perspective, there are just a handful of parts and a few wires.


RGB LED (Common Cathode Version)
VCC GND SCL SDA
OLED 128x64 SSD1306 I2C
blog.squix.ch
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
ESP8266MOD
FLASH
RST
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin
OLED Screen
ESP8266 (computer)
100-220 Ohm Resistors

The ESP8266 is the brains of the outfit, the "computer" (SOC – System On a Chip) and the little OLED screen is used for setup and to display our status message. The rest of the components, an RGB LED and the three(3) resisters, are used together so Busy Buddy can light up a colorful status LED in just about any color you like. RGB LEDs come in two types; Common Cathode and Common Anode, and the wiring is slightly different for each (that's the Common Cathode version in the above diagram and breadboard photo). Get the Common Cathode version if you can but the software will work with both and on the Busy Buddy setup screen you can select which of the two types of RGB LED you have.

The ESP8266 has a micro USB plug on it which is where it gets power from and is how we program it by plugging it into your PC (Windows or Mac). Once programmed, it just needs power and that can come from almost any USB charging adapter you have lying around.

The OLED screen connects to the ESP8266 with just four (4) wires; you'll want to be careful about following the directions here because depending on the manufacturer of these OLED screens, they sometimes rearrange the order of the four(4) pins that the wires connect to so, for example, instead of VCC, GND, SCL, SDA like in the diagram above, they might be GND, VCC, SCL, SDA instead. If you hook everything up and the screen stays blank though, double check the order of those pins first!

Bill of Materials (BOM)

Now for the shopping list, I've tried to give you two options – ordering from Amazon or ordering from AliExpress. I've ordered components from both and will happily do so again; I've had great experiences with AliExpress, and the prices are amazing, but it *will* take longer to get your parts (2-3 weeks usually). You'll also want to pay very close attention to what you're ordering, not because they're trying to scam you but because a single listing sometimes has several product variations, and you'll want to be sure you've selected the one you want. Don't assume that because the picture looks right, that the correct product/variation has been selected. Click it and confirm the price before adding to your cart.

The links and prices I've quoted are in US dollars so if you're outside of the US, check your local sources for equivalent products. I've also found all of these items at local electronics stores so if you have that option, check it out.

The must have parts:

- ESP8266 SOC [the "computer"]
    - Amazon: 3-pack ~$16USD
      https://www.amazon.com/KeeYees-Internet-Development-Wireless-Compatible/dp/B07HF44GBT/
    - Ali Express: 1 count ~ $3USD
      https://www.aliexpress.us/item/2251832478785371.html
- SSD1306 OLED [the display screen]
    - Amazon: 1 count ~ $7USD
      https://www.amazon.com/UCTRONICS-SSD1306-Self-Luminous-Display-Raspberry/dp/B072Q2X2LL/
    - Ali Express: 1 count ~ $3.50USD - be sure to select the correct **4-pin** yellow/blue display
      https://www.aliexpress.us/item/2251832709890624.html
- Wires [to directly connect the above components]
    - Amazon: 40 count female-to-female 4" ~ $6USD
      https://www.amazon.com/dp/B077N58HFK
    - Ali Express: 1 pack 40 count female-to-female 10cm [4"] ~ $2.50USD
      https://www.aliexpress.us/item/2251832638800607.html
- RGB LED – Common Cathode variety
    - Amazon: 1 pack of 100 count 5mm RGB LED ~ $9USD
      https://www.amazon.com/Tricolor-Multicolor-Lighting-Electronics-Components/dp/B01C19ENDM
    - Ali Express: 1 pack of 10 count 5MM RGB LED ~ $3USD
      Be sure to click on the "Diffused Cathode" or "Transparent Cathode" option!
      https://www.aliexpress.us/item/3256802620215390.html
- 100-220 Ohm resistor (depends on LED brightness desired)
    - Amazon: (100 count 200ohm ~ $6USD)
      https://www.amazon.com/EDGELEC-Resistor-Tolerance-Resistance-Optional/dp/B07HDGH1R1
    - Ali Express: (100 count, be sure to select 200R [for 200 ohm] ~ $3USD)
      https://www.aliexpress.us/item/3256801441680644.html

As you can see, there's quite a price difference [between Amazon and AliExpress] and you'll note that in some cases there's a difference in the minimum # of parts you can order (the smallest quantity of ESP8266 devices you can buy on Amazon is a 3-pack). If you're planning to make more than one Busy Buddy, probably not a big deal. Otherwise, especially if ordering from Amazon, you may want to see if you can split an order with some other local folks who want their own Busy Buddy – you'll certainly have more than enough wire and LEDs!

The optional parts:

- Project board (aka Breadboard)
  - Amazon: board & wires kit, some components won't be needed ~ $11USD
    https://www.amazon.com/dp/B09TX9CMG1
  - Ali Express: board without wires ~ $4USD
    https://www.aliexpress.us/item/3256801539488877.html
- Wires for breadboard
  - Amazon: 120 count 4" variety pack of female-to-female, male-to-male & male-female ~$7USD
    https://www.amazon.com/dp/B07GD1XFWV?th=1
- Basic soldering iron kit (only optional if you already have one)
  - Amazon: 60 watt soldering iron kit w/solder and stand ~$12USD
    https://www.amazon.com/Soldering-soldering-solder-adjustable-temperature/dp/B09DY7CCW5

If you're going to buy all of the above, you might want to consider just buying a "project kit" that includes most of the above parts at a lower cost, and then just buy the one or two items not in the kit. For example, this "weather station" kit from Amazon includes all of the above *except* the RGB LED, for less than $20USD. Sure, you'll end up with a couple of parts that you won't use (at least not for Busy Buddy) but you'll probably still come out ahead in terms of overall cost and shopping convenience. The kit isn't much cheaper on Ali Express so I'm just providing the US Amazon link:
https://www.amazon.com/Temperature-Humidity-Atmosphetic-BH1750FVI-YellowBlue/dp/B07GPBBY7F

The breadboard and related wires listed above are only needed if you want to start tinkering with this DIY electronics stuff – if your goal is to just assemble Busy Buddy and pop it in a case, you don't need the breadboard. Also, if you *do* decide to get the breadboard, you'll need some wires in addition to those in the "must have" list – you need wires with exposed ends you can poke into the holes in the breadboard. Because of that, I've listed a breadboard on Amazon that comes with a set of appropriate wires for about the same price as buying a breadboard and set of wires separately.

Since we do need to do some soldering with this project, I've also listed a very basic soldering iron kit which includes more than enough solder to do the job. You can spend a lot more on a soldering setup and if you're planning to do a lot more DIY electronics you may want to, but you can do a lot with a very basic tool like this one.

BTW, a quick note regarding the OLED screen; you'll see a number of varieties of this little display, some with a mix of yellow and blue text, some with all blue and some with white text. While I've chosen to use the yellow/blue variety in my example, they all behave the same, so it really comes down to your personal preference. The only difference that you *do* want to watch out for is the number of pins sticking out the back; **we want the 4-pin variety (GND, VCC, SCL, SDA)**, not the 7-pin version.

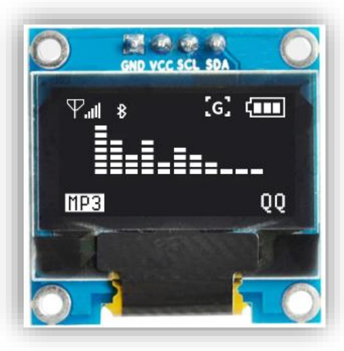


Figure 1: 4-pin OLED Display Unit

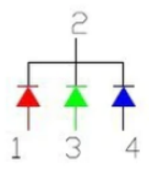
Hardware Step-by-Step

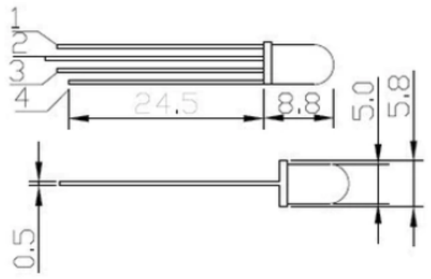
I'm going to assume that most readers are here to just *build the darn thing already* and call it a day. If that's you, and you've got your wires, ESP8266, OLED display, RGB LED and 3 resistors in hand, let's dive in and take it slow and easy.

First, let's take a bird's eye view at how this gets wired up. It's really pretty straightforward and although most of this document assumes you're using a Common Cathode RGB LED, we'll review the differences so you can be comfortable substituting the Common Anode version if that's what you have.

In both cases, the wiring for the OLED display is exactly the same so we'll take about that in a bit. Right now I want to focus on the RGB LED – this is also the only part that requires soldering when you do the final assembly, well, the LED and the 3 resistors connected to it [don't try and leave out the resisters – you'll almost certainly burn out the LED]. Remember that an RGB LED is really *three separate LEDs* in one package – Red, Green and Blue. Each has a separate wire (lead) to control it and by mixing the intensity of each we get all the color variations RGB LEDs are known for.

Now, if you have both types of RGB LEDs in your hand you'd be hard pressed to tell one from the other – they'd look identical and if we take a close look at schematics of the 4 pins on this LED, they *still* look the same!



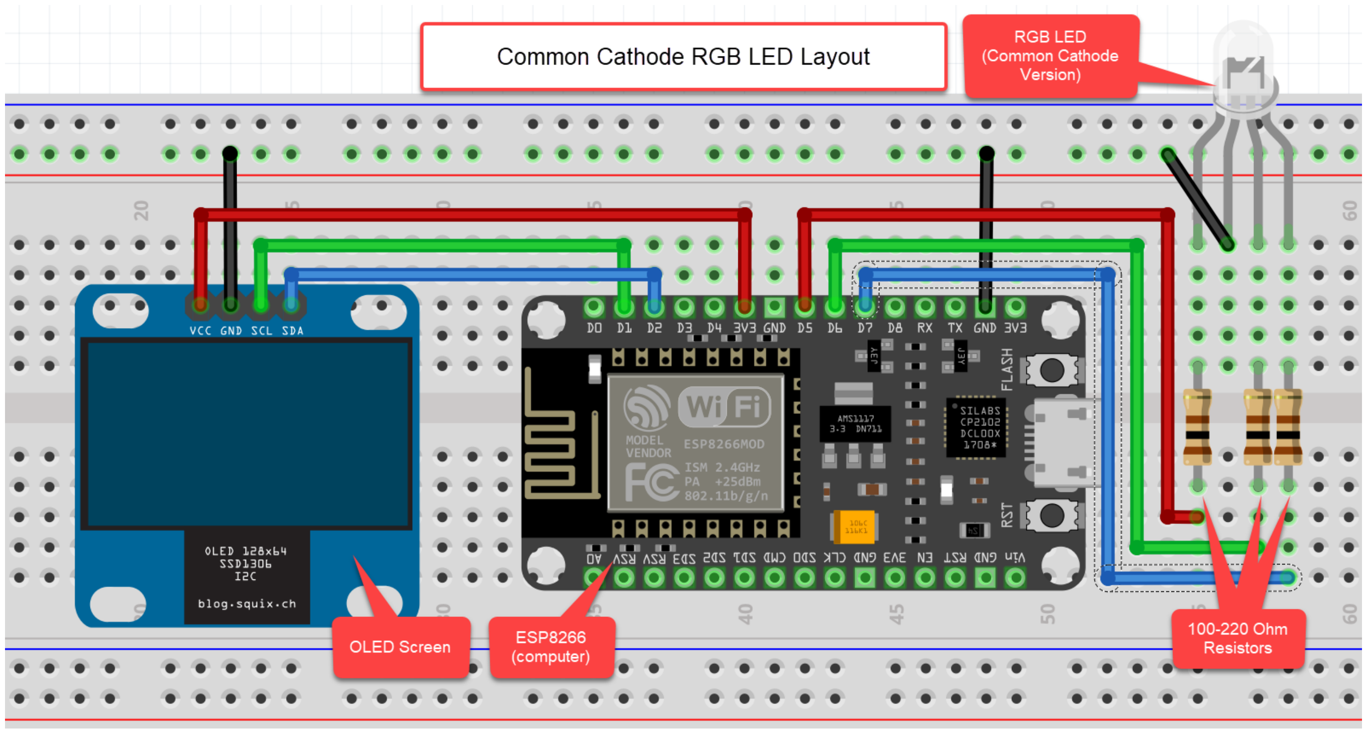


Sending a signal to pin 1 lights up the Red LED, pin 3 lights up Green and pin 4 lights up Blue. When we get into the programming, you'll see that we're sending a value of between 0 and 255 to these pins; 0 is "off" and 255 is "full power". Any value between 0 and 255 controls how bright the individual LEDs are; sending 255 to Red while sending 0 to Green and Blue will make the RGB LED show bright red. But, if you send 255 to Red, 0 to Green, and 255 to Blue, you'll see bright purple on the RGB LED – the combination of Red and Blue.

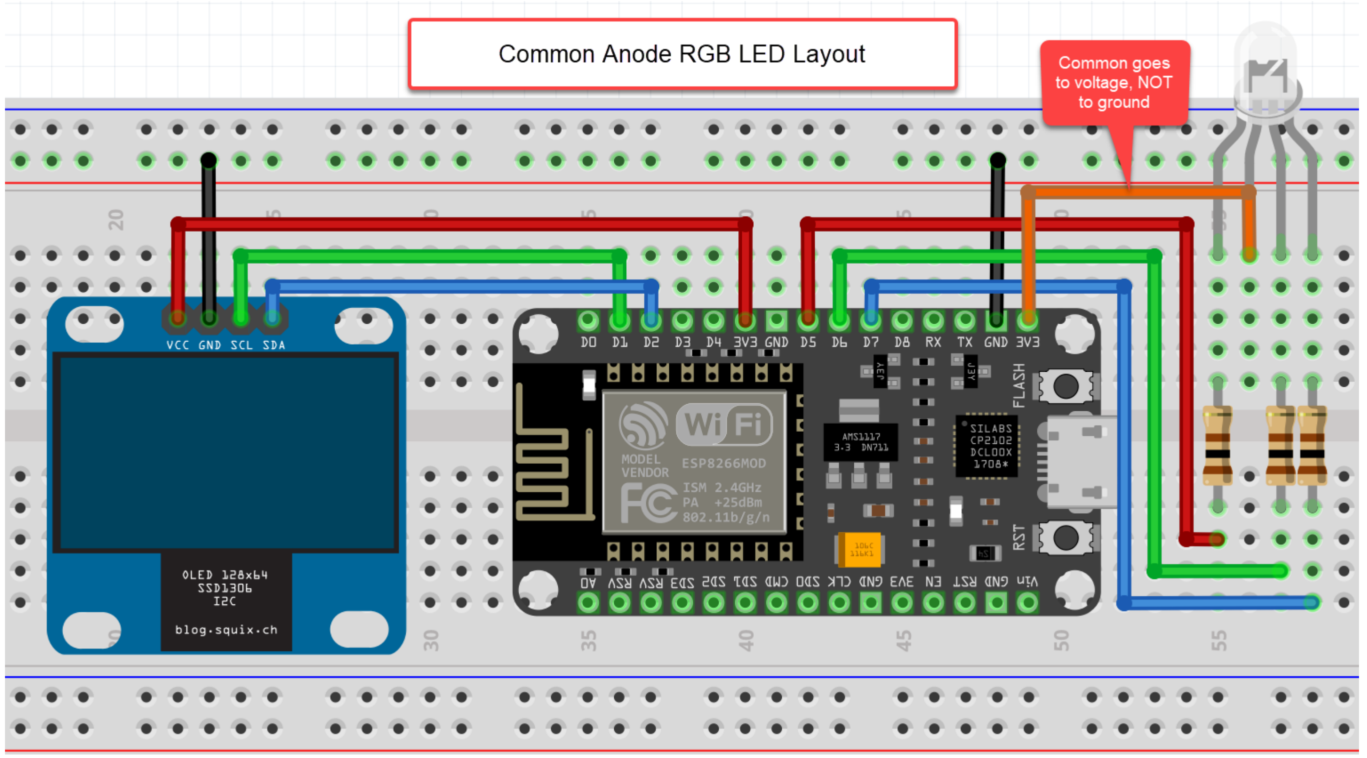
So, what about that other pin, pin #2? Well, that's "Common" pin I've been talking about and you'll notice that it's also the longest pin of the 4, making it easy to physically identify. If the RGB LED is "Common Cathode", then pin 2 will go a ground (gnd) or "negative" source and the other pins will be providing the positive voltage in the circuit. If, however, the RGB LED is a "Common Anode" type, pin 2 will need to go to a positive voltage source, one of the 3V pins on our ESP8266 SOC.

In either case, pins 1, 3, and 4 will connect to the same "signal" pins on the ESP8266 – no change to that wiring at all. What *will* change is how the software behaves; when you tell Busy Buddy that you're using a Common Cathode RGB LED, it will send a "High" signal to those pins which you can think of as "positive". When you're using the Common Anode version, and tell Busy Buddy on its setup page, those pins will now send a "Low" signal which you can think of as "negative" for our purposes.

Just remember, if you have a Common Cathode RGB LED, <u>*make sure*</u> the long wire, lead #2 in the diagram above, goes to negative/ground and if you're using Common Anode, that same lead #2 needs to go to positive/3V on the ESP8266. If you get it wrong, don't worry too much – it won't blow up or burn out, it just won't light up at all. Also, if you write it up correctly but forget to set the correct RGB LED type on the Busy Buddy settings page, it will still work but the colors will be odd … you might be trying for Red and instead get Green. If that happens, check your wiring and also check that setting. I'll remind you about that again later 😉


Common Cathode RGB LED Layout
RGB LED (Common Cathode Version)
VCC GND SCL SDA
OLED 128x64 SSD1306 I2C
blog.squix.ch
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
A0 RSV RSV SD3 SD2 SD1 CMD SDO CLK GND 3V3 EN RST GND Vin
FLASH
RST
OLED Screen
ESP8266 (computer)
100-220 Ohm Resistors

In the above layout for the Common Cathode version of the RGB LED, the Common lead is going to GND (ground) via the black wire (all the black wires in this diagram represent ground/negative).


Common Anode RGB LED Layout
Common goes to voltage, NOT to ground
VCC GND SCL SDA
OLED 128x64 SSD1306 I2C
blog.squix.ch
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
A0 RSV RSV SD3 SD2 SD1 CMD SDO CLK GND 3V3 EN RST GND Vin
FLASH
RST

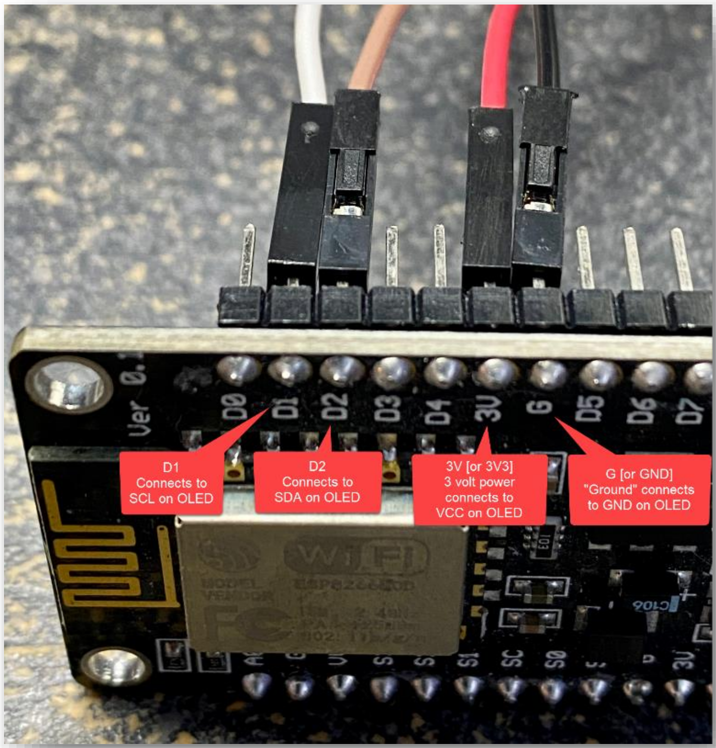
The only difference with the Common Anode version is that the Common lead (orange) is now going to a 3V pin on the ESP8266 [instead of GND]. In fact, no LED lead goes directly to ground in this version (nobody wants to be negative 🤣).

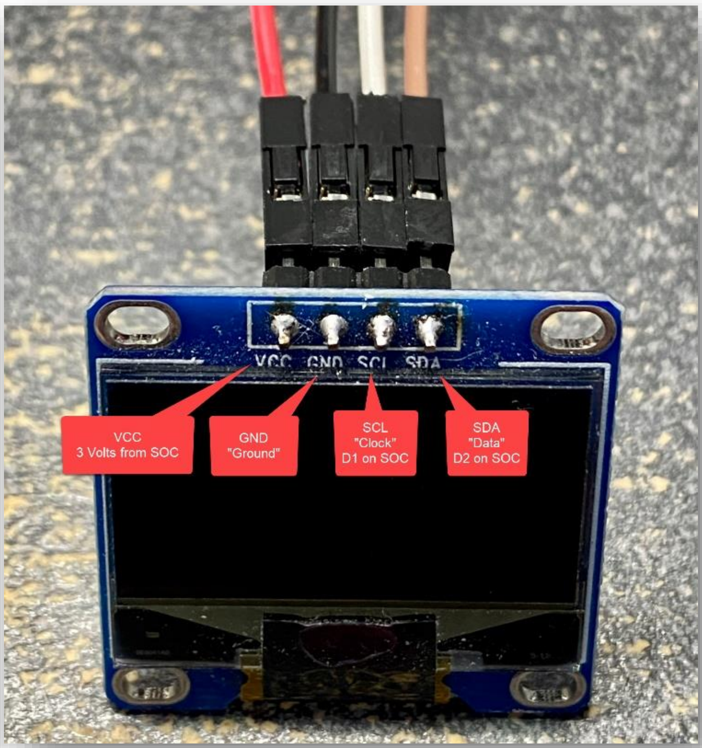
Now, let us talk about those three (3) resisters. Each LED lead has to connect to the ESP8266 signal pin via a resister. In fact, you'll almost never see a circuit of any kind involving an LED that doesn't have a resister – it provides protection for the LED and higher value resisters can also reduce how bright the LED can get. In our case, we just want the protection and so a resister of 100 to 220 OHMs ["OHM" is the unit of measure for resistance] will work fine. The only thing to keep in mind is that you want all three resisters to have the same value or you might get weird color results.

The way this will work is we'll connect one wire to a signal pin on the ESP8266 for each of our RGB LED leads, just like in the diagrams above; pin D5 for Red, D6 for Green and D7 for Blue. We'll need to solder the other end of that wire to one side of the resister (it doesn't matter which side) and then solder the other side of the resister to the corresponding lead on the RGB LED. If you get it wrong, don't panic; just move the wire to a correct corresponding pin on the ESP8266 (we didn't solder that side!). Once all 3 RGB leads are hooked up, we need to do the same thing for the Common lead *except* that Common doesn't get a resister and it needs to go to the ESP8266 *GND* pin for Common Cathode or the *3V* pin for Common Anode. Of course, if you're trying this on a breadboard first, ignore the soldering and just follow the diagram.

To connect up the OLED display, we're going to make four(4) connections by slipping the ends of the wires over the pins on both the ESP8266 and OLED devices, pressing them in as far as they'll go so they cover the pins completely. Which color wires you use for this really doesn't matter but for clarity, I'll use Red, Black, White and Brown to represent Power, Ground, Clock and Data respectively. The following table shows which color wire goes where on each device:

Wire	Purpose	ESP8266 SOC Connection	SSD1366 OLED Connection
Red	Power – 3 volts from the SOC	3V (sometimes 3V3)	VCC
Black	Ground (GND)	G (sometimes GND)	GND
White	Clock	D1	SCL
Brown	Data	C2	SDA

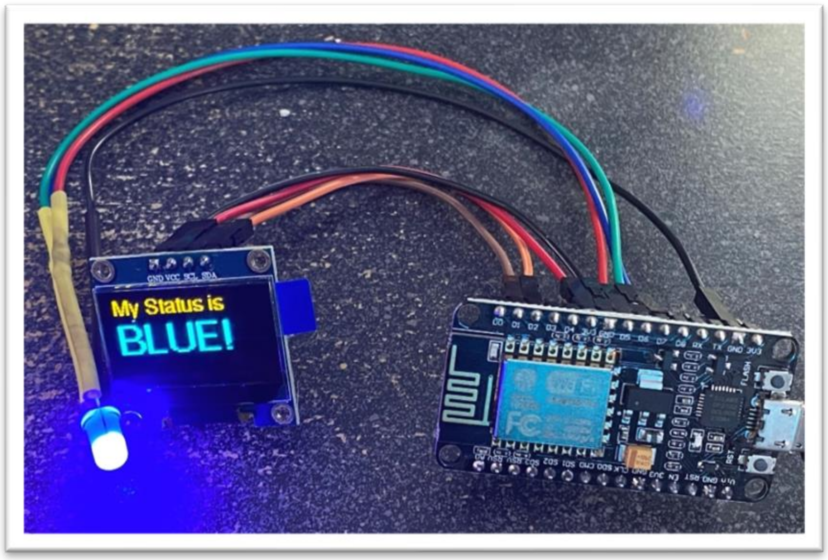

D1 Connects to SCL on OLED
D2 Connects to SDA on OLED
3V [or 3V3] 3 volt power connects to VCC on OLED
G [or GND] "Ground" connects to GND on OLED



Note that for the power (3V) and ground (G/GND) on the ESP8266, you'll find there are two or more sets of pins with the same labels. They're interchangeable so you can pick any 3V pin and any G/GND pin, whichever is most convenient.

If you've read this far and are thinking, "wow – I don't know if I'm up to soldering parts together", that's okay, but you'll want to get just a little practice with those steps first – we're not doing anything complicated but using a soldering iron arounds wires and electronics can cause an unexpected burn to you or overheat and damage a component if you're not careful. This link will cover the basics so you can continue here safely: https://www.makerspaces.com/how-to-solder/

The last point on the soldered connections is that it's usually a good idea to wrap those exposed wires with some electrical tape or heat shrink tubing. Almost any tape will work – this is very low voltage. The main thing you want to protect against is any of those bare wires touching each other, especially if you plan to mount your creation in a case of some sort. Don't be afraid to bend the LED leads a bit so you can protect them as needed.



This is what the finished build looks like, other than putting it in some kind of case. In my case, I've used heat shrink tubing to cover up the solder connections – they even completely cover the resisters so the whole effect is very clean. I used longer wires for the RGB LED leads than for the OLED display, just to give me more flexibility on where the RGB LED will end up in the case design.

I'll include a full set of pictures of soldering up these leads at the end of this document so check that out if you need a little more guidance in this area.

With all those wires connected, _double check_ the power and ground (VCC->3v and GND->G) to make sure they're not reversed (it's not you, it's the manufacturers…tricky tricksters). Check the signal wires too and in particular double-check that pin D5 on the ESP8266 goes to lead #1 (red) on the RGB LED, D6 goes to lead #3 (green) and D7 goes to lead #4 (blue). With all that checked out, we're about ready to get this programmed, setup and operational!

If you plug it in right now for a "smoke test" (meaning, we *really* don't want to see any smoke), not much is going to happen and the screen will likely stay blank. The only sign of life you might see would be a brief flicker from the blue LED on the SOC and maybe the RGB LED – not much reward for your hard work!

So, jump down to the Software section and we'll breathe life into your new creation!

Software

We're almost there! Similar to the hardware build discussion though, there are two possible paths for getting the software built and installed on your new Busy Buddy; let's call them the Easy Path and the Geek Path. Also, just for the record and speaking as the geek who wrote the software, I recommend *everyone* take the Easy Path and then if you're just curious or if you want to help me make Busy Buddy even better, then step onto the Geek Path.

Easy Path

The easy path is simply this; I've already compiled the Busy Buddy software into something called a "binary image" and so you can quickly and easily install that onto your ESP8266 SOC with no programming knowledge, complicated tools, or much setup. The only things you'll need to do are:

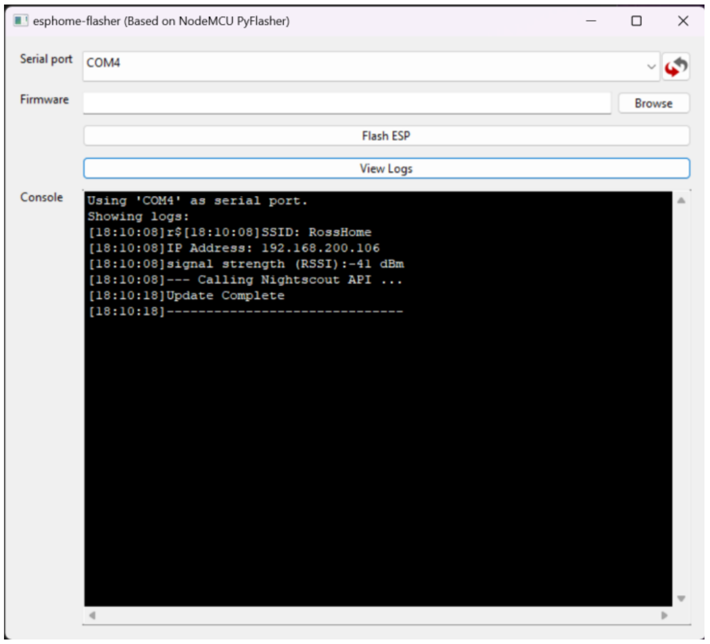
1. Download the latest version of the binary file from the GitHub repository where it's stored.
2. Download a "flasher" program that will take that binary file and load it ("flash" it) onto the SOC.

For step 1, you can download the binary file from here: https://github.com/VeryKross/BusyBuddy/tree/main/v1/bin

- Click on the "BusyBuddy_v1.bin" file listed there and on the subsequent screen, click on the "Download" button and save the file somewhere on your local computer that will be easy to find 😊

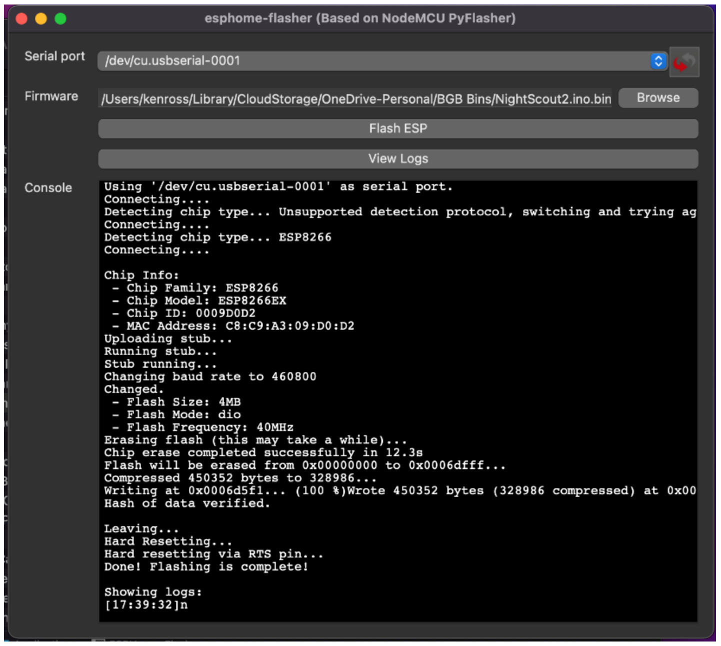
For step 2, I recommend using the free ESPHome-Flasher utility. They have versions available for Windows, Mac and Linux and there's really nothing to install; just download it and run it whenever you need to flash a program into memory [which we do]. You can get all versions of the utility from this page, just scroll down until you see the "Assets" section and pick the file that's right for you (note that if you're running Windows, it will be the ESPHome-Flasher-1.4.0-Windows-x64.exe file): https://github.com/esphome/esphome-flasher/releases

Okay, now that you have the utility we're just about ready to go! If you haven't done so already, make sure you have your ESP8266 SOC plugged into your PC with a USB cable. Then run the ESPHome-Flasher utility and you should see a screen something like this:



```
Using 'COM4' as serial port.
Showing logs:
[18:10:08]r$[18:10:08]SSID: RossHome
[18:10:08]IP Address: 192.168.200.106
[18:10:08]signal strength (RSSI):-41 dBm
[18:10:08]--- Calling Nightscout API ...
[18:10:18]Update Complete
[18:10:18]------------------------------
```

Windows version


esphome-flasher (Based on NodeMCU PyFlasher)
Serial port
/dev/cu.usbserial-0001
Firmware
/Users/kenross/Library/CloudStorage/OneDrive-Personal/BGB Bins/NightScout2.ino.bin
Browse
Flash ESP
View Logs
Console

```
Using '/dev/cu.usbserial-0001' as serial port.
Connecting....
Detecting chip type... Unsupported detection protocol, switching and trying ag
Connecting....
Detecting chip type... ESP8266
Connecting....

Chip Info:
 - Chip Family: ESP8266
 - Chip Model: ESP8266EX
 - Chip ID: 0009D0D2
 - MAC Address: C8:C9:A3:09:D0:D2
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
 - Flash Size: 4MB
 - Flash Mode: dio
 - Flash Frequency: 40MHz
Erasing flash (this may take a while)...
Chip erase completed successfully in 12.3s
Flash will be erased from 0x00000000 to 0x0006dfff...
Compressed 450352 bytes to 328986...
Writing at 0x0006d5f1... (100 %)Wrote 450352 bytes (328986 compressed) at 0x00
Hash of data verified.

Leaving...
Hard Resetting...
Hard resetting via RTS pin...
Done! Flashing is complete!

Showing logs:
[17:39:32]n
```

macOS version

So, first things first – we need to select which communications “port” your SOC is connected to. On Windows, this will show up as “COM” followed by a number (e.g. COM4). On Mac and Linux, the port shows up as a numbered device (see the screen shot above). It’s unlikely that you’ll see more than one listed but if you do, just pick one to try and if it doesn’t work, try again with the other (not very scientific, but you’ll find the right one pretty quickly).

Once you’ve selected the port, click the Browse button and locate the Busy Buddy binary file you downloaded earlier. With that selected, just click the “Flash ESP” button and you should see it run through a series of steps similar to what you see on the Mac screenshot above (technically, what you see in *that* screen shot is from another of my ESP8266 projects for blood glucose monitoring, but the steps are identical). The blue LED on the SOC will flicker while this is going on. It may take a minute or so to complete so just be patient. If it’s going to fail, it will likely do so immediately, and that will probably be from having the wrong port selected; if that’s the case, pick a different port and try again.

When the flash process has completed, the SOC will automatically reset and start running the Busy Buddy application! Next stop, setting up the software!

Geek Path

Who are we kidding? If you want the geek path, check out the src folder in the repo and load the sketch into your favorite Arduino IDE 😊 Be sure to add the ESP8266 devices if you haven’t already, and when selecting the board type, select “NodeMCU 1.0 (ESP-12E Module)”.

Other than that, it should all build and install onto your device without any issues. If you’re an experienced C/C++ developer, please excuse some of my code structure choices – it’s not always the most elegant as I’ve been dusting off yearly 30yrs of C/C++ neglect from my little grey cells while simultaneously learning the Arduino / ESP8266 platform and ecosystem 😊

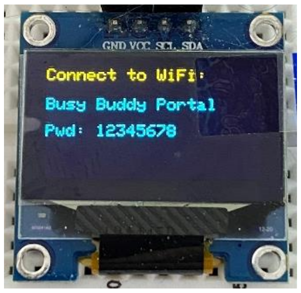
https://github.com/VeryKross/BusyBuddy

Busy Buddy Setup for 1st Run

Setting up Busy Buddy the first time requires a few manual steps but the device will prompt you through it and all you’ll need is your smartphone. Completing the following steps will have your Busy Buddy talking to Nightscout in no time:

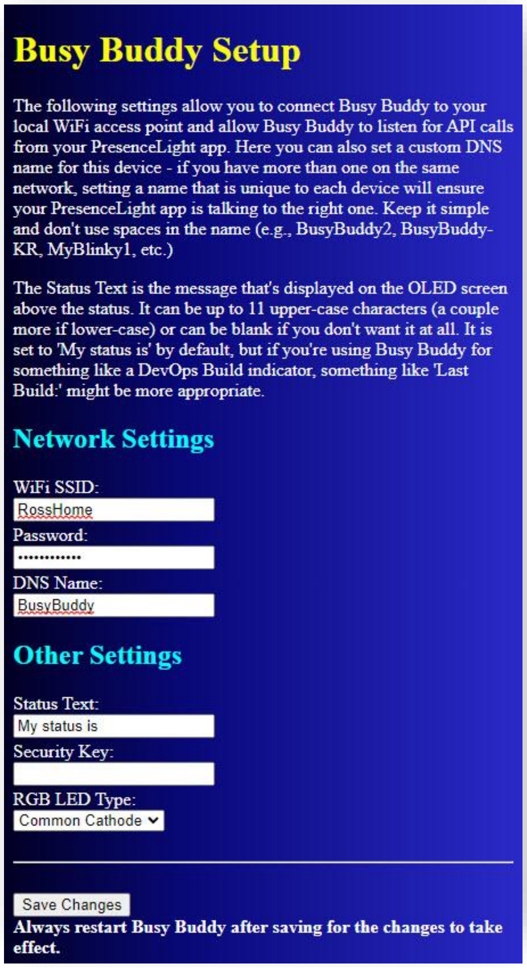
1. Connect your phone to Busy Buddy’s Wi-Fi Access Point
2. Open the Busy Buddy setup web page from a web browser on your phone
3. Fill in your Wi-Fi SSID and password so it knows how to connect to your network
4. Fill in your Nightscout URL (without the HTTPS:// prefix)
5. Tap Save and reboot Busy Buddy (just unplug and plug back in works too)

When Busy Buddy first woke up after being flashed, you should see something like this on the OLED display:



This not only tells us that the software is installed and running correctly (yay!) but it’s also telling you the next steps; open the Settings in your phone, go do the Wi-Fi settings, and look for a new network called “Busy Buddy Portal” and connect to it; you’ll need to provide the super-secret “12345678” password. Sometimes the connection can take up to a minute to complete so be patient. Note that network name probably *won’t* be “Busy Buddy Portal” exactly, look for “BusyBuddyPortal###” where ### is a random 3-digit number. It will show you the exact name to look for here on the screen.

Once you're fully connected, open a web browser on your phone and connect to http://10.10.10.10 – this is the address for the configuration web page during the initial setup.


Busy Buddy Setup

The following settings allow you to connect Busy Buddy to your local WiFi access point and allow Busy Buddy to listen for API calls from your PresenceLight app. Here you can also set a custom DNS name for this device - if you have more than one on the same network, setting a name that is unique to each device will ensure your PresenceLight app is talking to the right one. Keep it simple and don't use spaces in the name (e.g., BusyBuddy2, BusyBuddy-KR, MyBlinky1, etc.)

The Status Text is the message that's displayed on the OLED screen above the status. It can be up to 11 upper-case characters (a couple more if lower-case) or can be blank if you don't want it at all. It is set to 'My status is' by default, but if you're using Busy Buddy for something like a DevOps Build indicator, something like 'Last Build:' might be more appropriate.

Network Settings

WiFi SSID: RossHome
Password: ••••••••••••
DNS Name: BusyBuddy

Other Settings

Status Text: My status is
Security Key:
RGB LED Type: Common Cathode

Save Changes
Always restart Busy Buddy after saving for the changes to take effect.

You should see a web page very much like this one to the left. As long as you know your Wi-Fi name (the SSID) and the password, this should be pretty easy.

Besides the SSID and password, the other value you might want to change is the DNS Name. This is the name that you can use on your network to communicate with BusyBuddy to call its API to set the text and LED color. That POST API looks like this by default:

http://busybuddy.local/status?text=Busy&color=FF0000

If you change your DNS Name to "MyStatusBuddy" (note the lack of spaces), your API would change to:

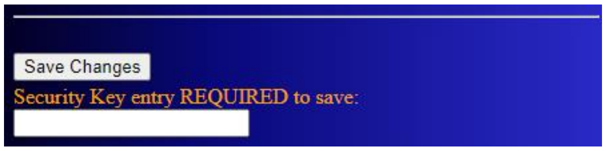
http://mystatusbuddy.local/status?text=Busy&color=FF0000

The ".local" part is always appended and it's mandatory. So, why change the DNS Name at all? Well, this allows you to have more than one Busy Buddy on your network at the same time. Maybe you're in an office setting with several co-workers who also have a Busy Buddy, or maybe you want one to show Teams status and another to show your DevOps build status. By giving them unique names, it makes it easy to reach them without having to resort to their unique IP addresses [which could change without your knowledge].

Speaking of co-workers, another optional change to consider is setting a Security Key. This isn't something I'd bother with at home, but in the rather geek-filled office in which I work, I know that if I don't padlock my Busy Buddy, some prankster is going to send "special" messages to my little device and it will be nearly impossible to figure out who's responsible. By setting a secret key, it blocks them from being able to mess with me…at least through Busy Buddy. Note that once you do set a key, that value will be required on any future API calls and so that will now look like:

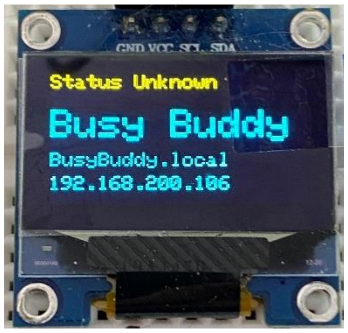
http://busybuddy.local/status?text=Busy&color=FF0000&key=secret

Also, to help ensure that would-be prankster doesn't just open your setup page and reset your key, once the Security Key has been set, any further changes to the setup page will require you to enter the existing key in order to save your changes. This means that even if your goal is to clear out the key, you'll need to clear the Security Key field, and then enter the existing/old key value into the prompt below the Save button.


Save Changes
Security Key entry REQUIRED to save:

Lastly, remember that if you are using an RGB LED with a Common Anode, select that option from the RGB LED Type drop-down list before saving (if you forget, and the colors act weird, come back and change it later). Oh, and if you're using Busy Buddy for build status, maybe change the Status Text to something like "Last Build:" 😊

Once you've finished filled in the various fields you can tap Save, restart Busy Buddy, and you should see something like the following:



This tells us that the software is up and running and waiting for its first status message. It will display its DNS Name (BusyBuddy.local) as a reminder in case you want to get back to the setup page (http://busybuddy.local), as well as its current IP address on your network (handy in case there's already another Busy Buddy with the same DNS Name and you need to get to your setup page to change your device's name (http://192.168.200.106).

Of course, *your* device will likely show a different IP address than this one so use whatever you see displayed if you need to access via IP.

Don't try and use that 10.10.10.10 IP address we first used for setup; that only works when you connect your phone's WiFi to Busy Buddy's published access point and it will only ask you to do that if it can't connect to the local WiFi. In fact, it will ask you to go through that processes any time it can't connect, like if you take Busy Buddy to a new location or if you experience an outage of some sort. If that does happen, you'll only need to give it the updated WiFi SSID & password information; all your other settings will still be there.

That really is all there is to it! Putting Busy Buddy into a nice case of some sort is probably a good idea so be creative there – you can use almost anything for a case whether that's just a small box from a craft store, a repurposed toy, or something you design and 3D print. Just make sure you have a cutout for the OLED screen and your status LED, and a smaller opening to plug the USB cable into your SOC.

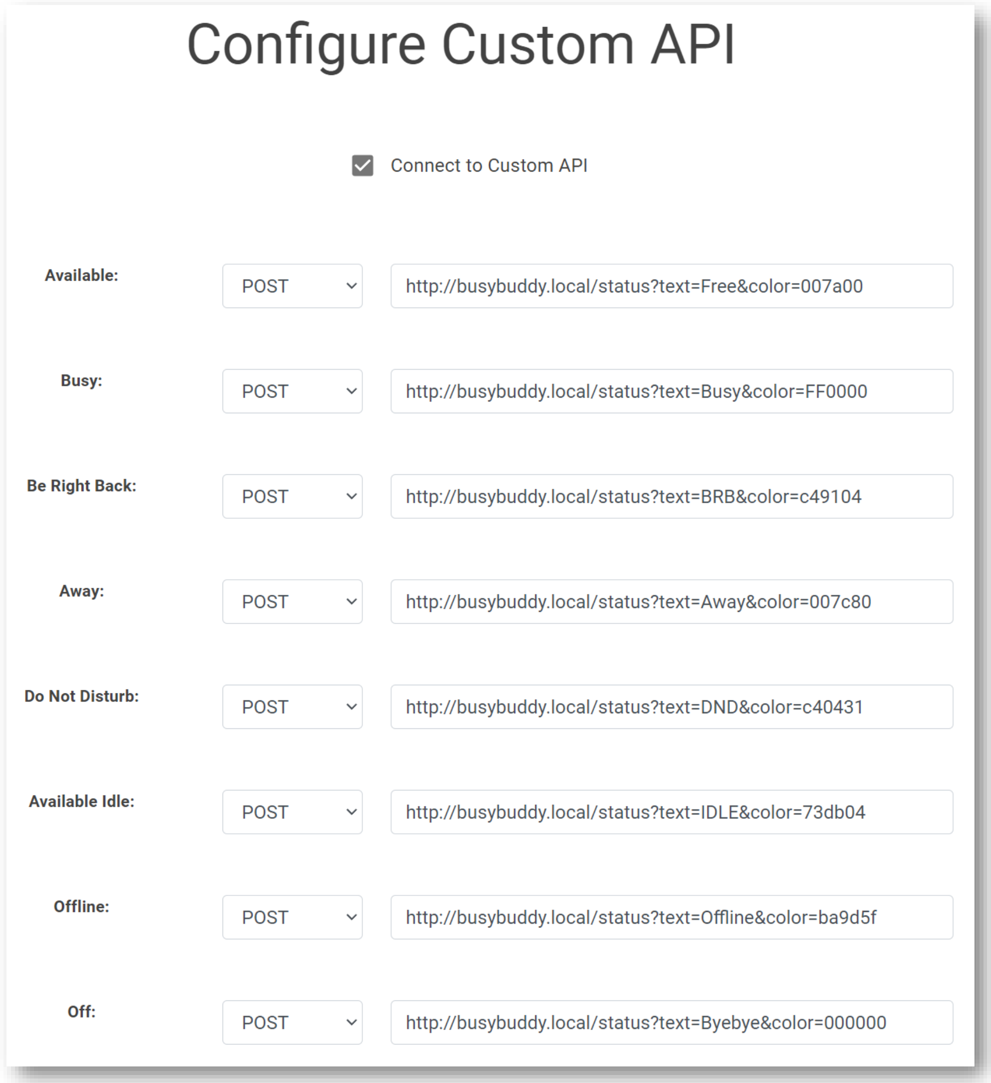
Once you're up and running, post a picture of your Busy Buddy creations to social media and spread the word! If you found this useful, I'm certain others will too, and they'll appreciate hearing about your experience putting it all together.

If you're using Busy Buddy for Teams status, see the next section of this document for how to get that setup. This will also cover the basics of how to use Busy Buddy as a general status device though its POST API.

If you run into any trouble that restarting Busy Buddy doesn't automatically fix, send along an email to busybuddy@hotkrossbits.com and I'll do my best to help out.

Using Busy Buddy for Teams Status (presence)

As mentioned at the start of this document, using Busy Buddy with Teams relies on you running the free Presence Light app which you can get from the Windows store. If you're a Mac or Linux user, you'll find a cross-platform version of the app on its GitHub site here.



Configure Custom API

☑ Connect to Custom API

Available:	POST	http://busybuddy.local/status?text=Free&color=007a00
Busy:	POST	http://busybuddy.local/status?text=Busy&color=FF0000
Be Right Back:	POST	http://busybuddy.local/status?text=BRB&color=c49104
Away:	POST	http://busybuddy.local/status?text=Away&color=007c80
Do Not Disturb:	POST	http://busybuddy.local/status?text=DND&color=c40431
Available Idle:	POST	http://busybuddy.local/status?text=IDLE&color=73db04
Offline:	POST	http://busybuddy.local/status?text=Offline&color=ba9d5f
Off:	POST	http://busybuddy.local/status?text=Byebye&color=000000

To setup Busy Buddy with Presence Light, you will want to use its Custom API feature. Make sure you have "Connect to Custom API" checked and that for each status you want to represent, you have "POST" selected as the action.

For each status you will provide a URL similar to what you see here. If you've changed your Busy Buddy DNS Name from "busybuddy" to something else, be sure to make that change in the URL.

Also, if you have set a Security Key, be sure to add "&key=*yourkey*" at the end of the URL.

Other than that, the "text" parameter sets the large text displayed on the OLED screen; keep it short and upper-case takes more room than lower-case letters.

The color is supplied as a single 6-character hexadecimal RGB value. It is really 3 2-character values with the first two being for the Red LED, the next two for the Green LED and the last two for the Blue LED – those three leads on our RGB LED. This kind of color representation is very standard and pretty much any color picking tool will provide you with the "Hex Code" for the selected color. An easy one to use can be found at https://colorpicker.me/ and when you copy the Hex Code color value, just be sure to leave off the # symbol at the front – you just want the six RGB values.

Feel free to copy the setup above as a starting point for your own meeting status indication. It closely follows the colors you typically see (red=busy, green=free, etc.). I took a little liberty with the "Away" color, mostly because yellow had a bit of a greenish tint with my LED and so wasn't as easy to distinguish from "Free". Have fun with it – who says purple can't be used for "Busy" 😊

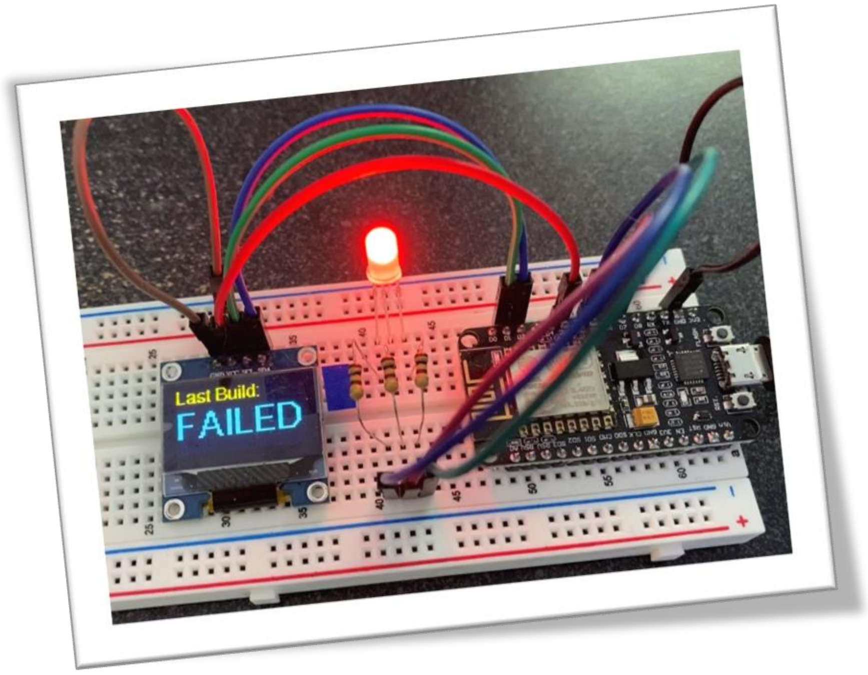
Using Busy Buddy for non-Teams Status

Using Busy Buddy for pretty much any status indication is simply a matter of calling its REST API with a POST and providing the Text, Color and, optionally, the Key parameters. That's it – Busy Buddy isn't really all that smart 😉

There are a ton of ways that you could set up an integration like that, and quite a few don't require you to write custom code. For example, using Power Automate, you could create a no-code Flow that is triggered by Azure DevOps or one of the many [many] other built-in triggers, and have it then call Busy Budy with a POST message based on the information you received in the initial Trigger.

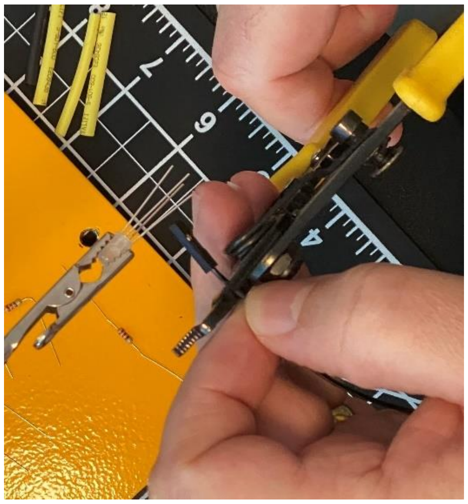
There are probably also ways to trigger calls to Busy Buddy through ITTT, GitHub Actions, and many others. Of course, if you are using a cloud-based system, you might need to find a way to make your local Busy Buddy visible to the cloud so you can reach your local device's API – something like a small reverse proxy or a similar setting in your home router.

However you do it, the API itself is super simple and intended to be flexible enough to use Busy Buddy in a lot of simple notification scenarios. Have fun!



Soldering the RGB LED “wiring harness”

If you’re already familiar with the basic techniques for soldering electronics you’ll probably not find much new or exciting here. But, if you’re new to soldering, this step-by-step of how I prepared the wired and soldered them to the resisters and RGB LED may help you avoid some mistakes. Keep in mind that if you *do* make a mistake, you’re really not going to hurt anything (well, unless you melt the LED…); just unsolder, retrace your steps, and try again.

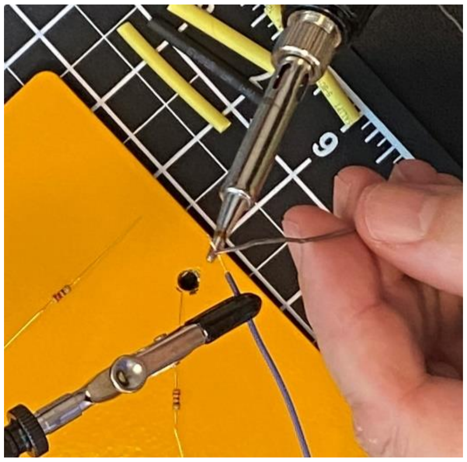


The first thing I did was to pick out four(4) wires that I’d sacrifice for this project. I needed a “female” end that would connect to the pins on the ESP8266 but it didn’t matter what the other end looked like because I was going to cut it off.

In this case, I happened to pick a set of wires with female connectors on both ends and so here you see me cutting one off, close to the connector so I have the maximum length of wire left.

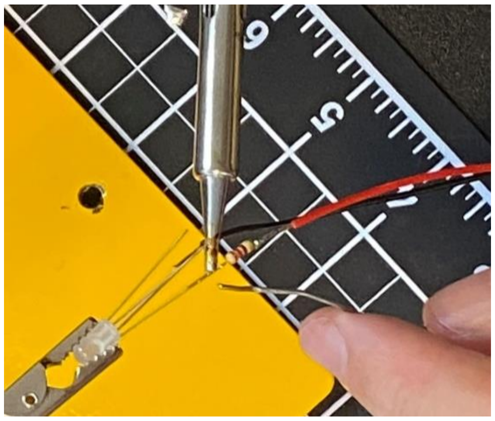
After cutting off the connector, I stripped off about ½ inch (1cm) of the insulation so just the bare wire stuck out. These I twisted together so they were more like a braid than loose wires sticking out randomly.

Here I’m comparing that length of bare wire with the length of the lead on the resister; the resister leads are much longer than I need and so I’m going to cut the lead on each side of resister so that it’s just a little longer than the bare wire that I’m going to be soldering to it.



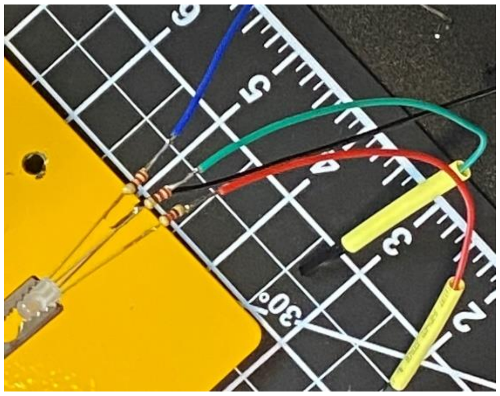
With all the resisters cut to size, I’m ready to “tin the wires”. This just involves heating up the bare wire with my soldering iron until it’s just hot enough that the solder will melt against it. This lets me pre-coat the bare wires with a small amount of solder so that later when I want to solder it to the resister things will go much faster and provide a better connection.

When you do this, notice that the heated wire acts almost like a wick, pulling a small amount of solder into the strands. Once you see that happen, you’re done!



Here you can see that I've already soldered the wire to the resister and am now attaching the other side of the resister to lead #1 (red) of our RGB LED. Prior to this point I've also "tinned" the RGB LED leads; even though they're a solid wire [not strands like our red wire], getting a small mount of solder to adhere to the lead before laying the resister against it will really help this part go faster – it's already awkward trying to juggle a hot soldering iron, solder, the LED and the resister; anything to help make the components "stick" quickly is welcome!

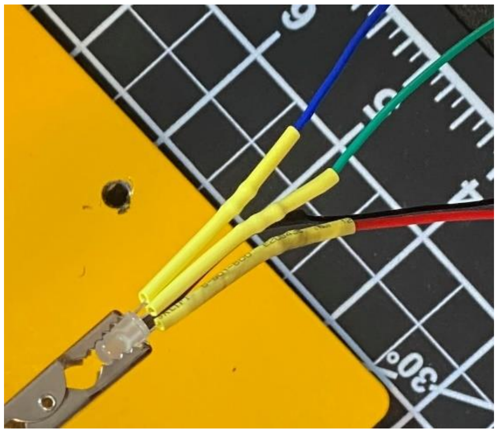
With that prep work, I just need to heat up the LED lead, lay the resister lead against it, and apply a small amount of solder to complete the "weld".



You can see the nearly finished job here with all three resisters soldered in place, as well as the black wire that's soldered directly to the LED Common lead.

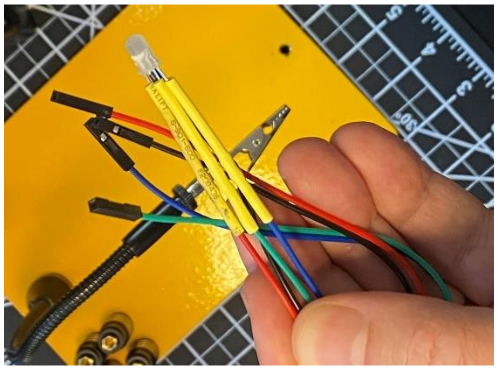
You can also see the yellow heat shrink tubes that will be used later to cover up the soldered areas and protect against those bare wires touching anything they shouldn't.

I should point out that if you're going to use heat shrink tubing, be sure to slip it over your writes *before* you finish soldering the other end 😉



Here you can see the end result of the heat shrink being applied; you just slip them up over the area you want to protect and apply a little heat from a lighter, heat gun or even a blow dryer.

If you're not familiar with it, you can buy it in all sorts of different diameters, lengths and colors. Just cut it to the length you need before completing your solder job.



Finally we have the finished assembly, ready to be plugged into the pins on the ESP8266. Since this is a Common Cathode RGB LED, the black wire will go to ground (GND), the red wire to D5, the green to D6 and the blue to D7.