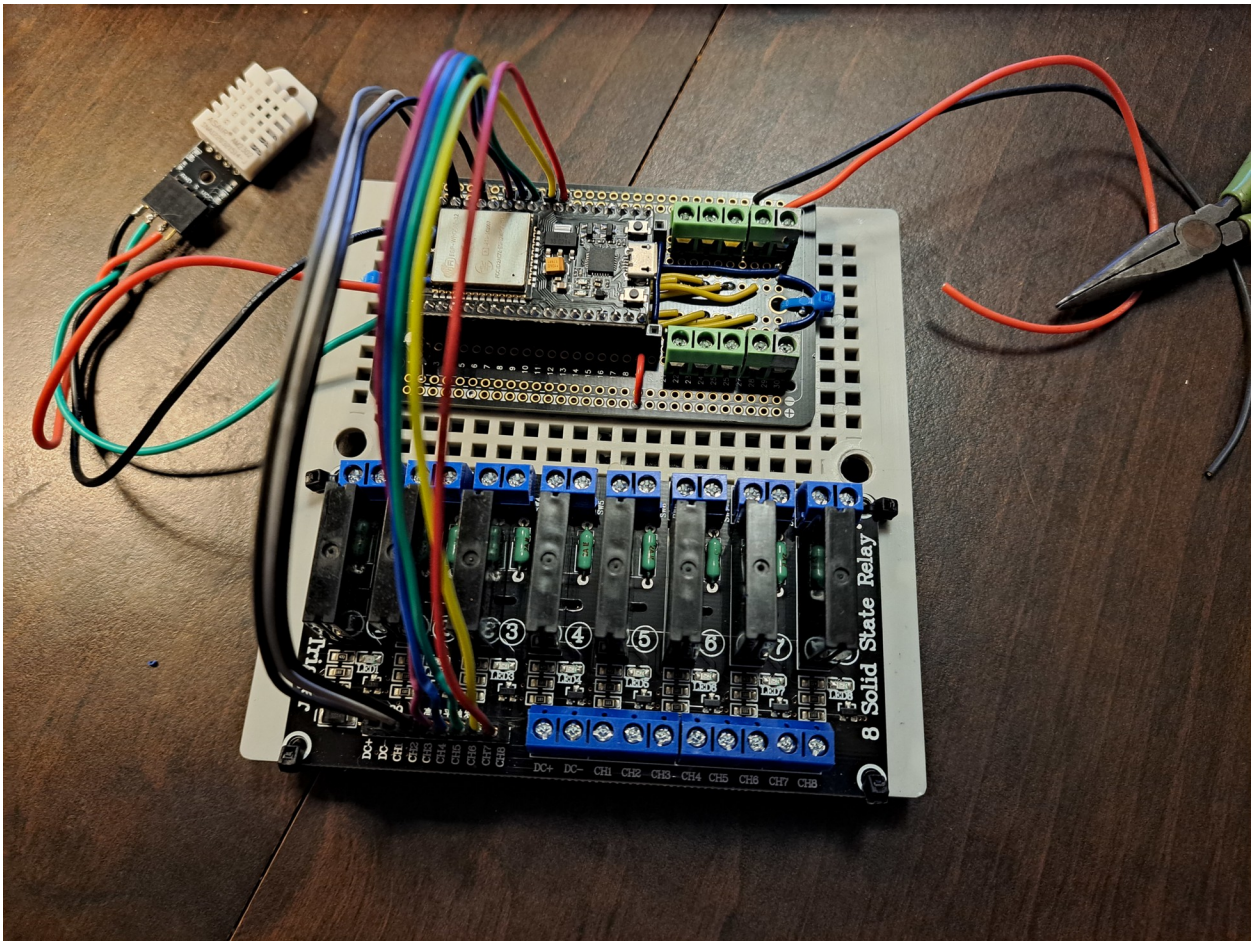
Climate Czar Combo Hub

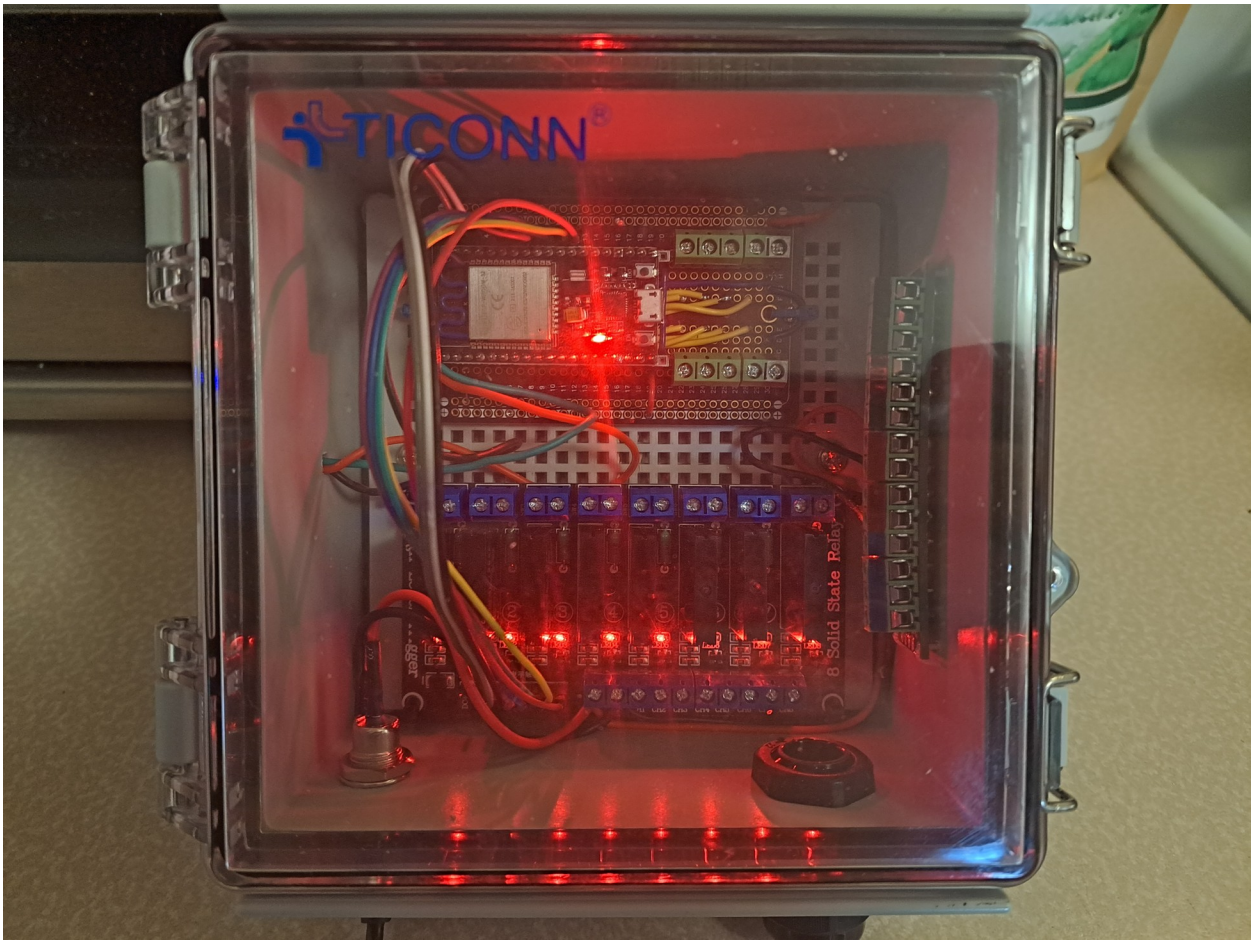
Don’t let the name fool you, there’s really a whole lot of nothing complex or high-tech going on here. Looking at the source code, you’ll see that it’s just roughly 350 lines of simplistic C++ code with a lot of that being comments. All of the heavy lifting is actually done in the libraries that are linked to it.

Basically, it’s just a 38 pin ESP32 based WiFi device that a Climate Czar server talks to which provides you with:

- 8 analog inputs for use with variable value sensors or on/off switches
- 8 digital outputs (3.3 volt logic) for toggling relays
- Built in DHT-22 temperature and humidity sensor
- Unlimited 1-wire DS18B20 temperature sensors
- HTTP API for reading sensors and toggling the digital outputs

The wiring of things is extremely basic, the only components outside of the ESP32 is a 4.7K pull up resistor from the 3.3 volt output pin to the 1-wire bus data pin, a 10µF electrolytic capacitor from the reset/enable pin to the ground, and the DHT-22 module. Everything else is just wires connecting the ESP32 pins to terminal blocks for input devices and the trigger pins of a relay board (preferably solid state relays).





If you notice the red, black, and green wires going through the left side of the utility box, that's because the DHT-22 is mounted to the outside so it can read the temperature and humidity level of my greenhouse. As you can see, this is a weather proof outdoor utility box, so it would do no good to have the humidity sensor inside of it.

On the right side of the utility box is a power bus that has six ground terminals, four 5-volt terminals, and four 3.3-volt terminals. The terminals on the ESP32 board are for the 8 analog inputs and two 1-wire bus terminals (located on the right end of both strips). The power supply bus is there because you need to power the DS18B20 sensors with 5 volts and the analog inputs require a voltage from 0 to 3.3 volts.

At the bottom is an 8 channel solid state relay board where each relay is rated at 2 amps @ 250 volts. Personally, I would only use those for low current DC loads or triggering higher current relays for AC devices such as exhaust fans, heaters, pumps, lights, etc. This is the same board that I use in my pellet stove controller which switches AC devices, but everything in a pellet stove is really low current (less than 1 amp or 120 watts each).

My build is by no means dictating how you have to do things, it's just an example. The only part of it that is an absolute must is that you build it in a plastic enclosure since a metal one would block the WiFi connection. I've seen some ESP32 boards that have an external antenna connector, but I have no experience with one. You would need one of those if you absolutely must use a metal enclosure.

As for the power supply, I recommend at least a 5 volt, 2 amp switching wall-wart. These are cheap and rather small, the ones I use only take up one space on a power strip. The ESP32 really isn't a high current device, but it is often too much for most computer's USB port and will bog down. Using at least a 2 amp supply will power the ESP32 and 25 DS18B20 sensors perfectly fine according to my tests.

Using The HTTP API

The ESP32's Arduino networking library uses the HTTP 0.9 protocol, likely because its limited capabilities wouldn't benefit from anything newer. This isn't a problem for web browsers because they automatically adapt based on the response headers. However, command line utilities such as **curl** need to be forced by including **--http0.9** in its command line. If you send a command to your ESP32 and don't get anything back, it's more than likely because your command line program isn't automatically falling back to the older HTTP 0.9 protocol specified in the response headers.

Below are what would appear to be requests to pull the directory index file on a website, but the ESP32 just looks for these and acts accordingly. Say that your unit's IP address is 192.168.1.100 and you are using **curl** to make the requests, you would use the following commands.

curl -s --http0.9 http://192.168.1.100/humidity – Reads the humidity level from the DHT-22 sensor.

curl -s --http0.9 http://192.168.1.100/temperature/c – Reads the temperature from the DHT-22 sensor in Celcius.

curl -s --http0.9 http://192.168.1.100/temperature/f – Reads the temperature from the DHT-22 sensor in Farrenheit.

curl -s --http0.9 http://192.168.1.100/onewire/c?aa:bb:cc:dd:ee:ff:aa:bb – Reads a specific DS18B20 sensor by address on the 1-wire bus in Celcius.

curl -s --http0.9 http://192.168.1.100/onewire/c – Reads all DS18B20 sensors on the 1-wire bus in Celcius.

curl -s --http0.9 http://192.168.1.100/onewire/f?aa:bb:cc:dd:ee:ff:aa:bb – Reads a specific DS18B20 sensor by address on the 1-wire bus in Farrenheit.

curl -s --http0.9 http://192.168.1.100/onewire/f – Reads all DS18B20 sensors on the 1-wire bus in Farrenheit.

curl -s --http0.9 http://192.168.1.100/onewire/list – Reads all DS18B20 sensors on the 1-wire bus and shows them in a list of one device per line as "Address C-reading F-reading".

curl -s --http0.9 http://192.168.1.100/sensor1 – Reads the value of the analog input #1 (or replace the 1 with the number of the 8 sensors). Depending on the voltage to the port (0 to 3.3 volts) this value will be a number between 0 and 4095. You will need to process this value in your device read script to something that matches what you need in Climate Czar. Refer to the C++ source code for more information.

curl -s --http0.9 http://192.168.1.100/switch1/0 – Sets the status of the digital output #1 (or replace the 1 with the number of the 8 switches) to its off state. Change the zero at the end to a one to set the output port to its on state.

curl -s --http0.9 http://192.168.1.100/restart – Reboots the ESP32. There are also steps in the C++ code that checks the WiFi connection roughly once every 30 seconds and will reboot the ESP32 if the connection is lost.

curl -s --http0.9 http://192.168.1.100/stats – Returns the current system uptime and WiFi related information.

Refer to the code comments near the beginning of the C++ code to see how the 1-wire bus readings are returned and how to read a specific 1-wire sensor. Example hub read/write command scripts are also included in the latest Climate Czar server installation archive.

That's all there is to the HTTP API, it looks more complicated in the C++ code because of all the if-then-else branches. While this was designed around the Climate Czar server software, you could use it with anything else that lets you make HTTP requests. Keep in mind that I intentionally do not use the NVRAM to save and restore the relay states if the unit loses power. You are more than welcome to modify the code if you need that.

Parts That I Used

Below are links to parts that I used in my build, just in case you want to do the same.

ESP32 38 pin developer board - https://www.amazon.com/dp/B09J95SMG7

Breadboard format PCB - https://www.amazon.com/dp/B0BP28GYTV

Solid state relay board - https://www.amazon.com/dp/B0B4NS3RH3

5 volt 2 amp power supply - https://www.amazon.com/dp/B08722QC75

Weather proof utility box - https://www.amazon.com/dp/B0B87XSMVP