Adafruit ESP32 Feather V2

Created by Kattni Rembor

https://learn.adafruit.com/adafruit-esp32-feather-v2

Last updated on 2022-03-17 03:55:10 PM EDT
# Table of Contents

## Overview

### Pinouts
- Power
- ESP32
- STEMDA QT Connector
- LED and NeoPixel
- Buttons
- Logic Pins
- Bottom Row
- Top Row
- CP2102N USB to Serial Converter

## Low Power Usage

## Arduino IDE Setup

### Blink
- Pre-Flight Check: Get Arduino IDE & Hardware Set Up
- Start up Arduino IDE and Select Board/Port
- New Blink Sketch
- Verify (Compile) Sketch
- Upload Sketch
- Finally, a Blink!

### I2C Scan Test
- Common I2C Connectivity Issues
- Perform an I2C scan!
- Wiring the MCP9808

### WiFi Test
- WiFi Connection Test
- Secure Connection Example
- JSON Parsing Demo

### Factory Reset
- Factory Reset Example Code
- Factory Reset .bin
- The WebSerial ESPTool Method
- The esptool Method (for advanced users)
- Reset the board
- Older Versions of Chrome

### Downloads
- Files
- Schematic and Fab Print
One of the Adafruit star Feathers is the Adafruit HUZZAH32 ESP32 Feather ([https://adafruit.it/wcN](https://adafruit.it/wcN)) - with the fabulous ESP32 WROOM module, it makes quick work of WiFi and Bluetooth projects that take advantage of the Espressif popular chipset.

Recently we had to redesign this feather to move from the obsolete CP2104 to the CP2012N and one thing led to another and before you know it we made a completely refreshed design: the Adafruit ESP32 Feather V2.

The V2 is a significant redesign, enough so we consider it a completely new product. It still features the ESP32 chip but has many upgrades and improvements:

- Compared to the original Feather with 4 MB Flash and no PSRAM, the V2 has 8 MB Flash and 2 MB PSRAM
- Additional user button tactile switch on input pin 38
- Additional NeoPixel mini RGB LED with controllable power pin
- Additional STEMMA QT port for plug and play I2C connections
- USB Type C port instead of Micro B
- Separate controllable 3.3V power supply for STEMMA QT to allow for ultra low power consumption even with sensors are attached
- Designed for low power usage: [verified with a PPK (https://adafruit.it/YNf)](https://adafruit.it/YNf) to draw 70uA from the Lipoly battery in deep sleep and 1.2mA in light sleep.
- ESP32 Pico module is much smaller, allowing for clear marking of all breakout pads and additional mounting holes!
- Upgrade the USB to serial converter from CP2014 (2mbps max rate) to CP2102N which can handle 3 mbps.

However, in order to add the PSRAM, and use the new Pico module, which was small enough to allow all the fun extras, some of the breakout pads have changed, so here's what you need to know:

- The pin numbers for the I2C port (SDA, SCL), hardware UART (RX, TX), and SPI (SCK, MOSI, MISO) have changed. If your code has hardcoded use for those pins, you'll want to replace them either by the new numbers or change the code to use the 'pretty' names like SDA or SCK.
  When selecting the new Feather ESP32 V2 board in the Espressif board support package, the correct numbers will be substituted.
  Note the names are in the same spots, we haven't changed where the I2C/UART/SPI pins are located on the board, just which ESP32 pin numbers they are connected to in the module.
- The 'corner' pin next to TX has changed from pin 21 to 37. This pin is not used in any FeatherWings because its considered an 'extra pin'. It's also changed from a GPIO to input-only
- The remaining numbered pins and A0-A5 pins have not changed pin numbers.
The module nestled in at the end of this Feather contains a dual-core ESP32 chip, 8 MB of SPI Flash, 2 MB of PSRAM, a tuned PCB antenna, and all the passives you need to take advantage of this powerful new processor. The ESP32 has both WiFi and Bluetooth Classic/LE support. That means it's perfect for just about any wireless or Internet-connected project.

Because it's part of the Adafruit Feather eco-system, you can take advantage of the 50+ Wings (https://adafruit.it/wev) that we've designed to add all sorts of cool accessories. Plus that built in battery charging and monitoring you know and love with the ESP32 Feather is still there in this upgrade.
Features:

- ESP32 Dual core 240MHz Xtensa® processor - the classic dual-core ESP32 you know and love!
- Mini module has FCC/CE certification and comes with 8 MByte of Flash and 2 MByte of PSRAM - you can have huge data buffers
- Power options - USB type C or Lipoly battery
- Built-in battery charging when powered over USB-C
- LiPoly battery monitor with two 200K resistor divider
- Reset and User (I38) buttons to reset board and as a separate input
- High speed upload with auto-reset and serial debug with ultra-reliable CP2102N chipset.
- STEMMA QT connector for I2C devices, with switchable power, so you can go into low power mode.
- Charge/User LEDs + status NeoPixel with pin-controlled power for low power usage
- Low Power friendly! In deep sleep mode we can get down to 80~100uA of current draw from the Lipoly connection. Quiescent current is from the power regulator, ESP32 chip, and Lipoly monitor. Turn off the NeoPixel and external I2C power for the lowest quiescent current draw.
- Works with Arduino or MicroPython

Comes fully assembled and tested, with a USB interface that lets you quickly use it with the Arduino IDE or the low-level ESP32 IDF. We also toss in some header so you can solder it in and plug into a solderless breadboard. Lipoly battery and USB cable not included (but we do have lots of options in the shop if you'd like!)
The ESP32 Feather V2 comes with plenty of GPIO and other features. This page is a detailed look at everything you get with your new board!
Power

Power Pins

- GND - This is the common ground for all power and logic.
- BAT - This is the positive voltage to/from the JST connector for the optional lipoly battery.
- USB - This is the positive voltage to/from the USB-C connector if connected.
- EN - This is the 3.3V regulator’s enable pin. It’s pulled up, so connect to ground to disable the 3.3V regulator.
- 3V - This is the output from the 3.3V regulator. The regulator can supply 500mA peak but half of that is drawn by the ESP32, and it's a fairly power-hungry chip. So if you need a ton of power for stuff like LEDs, motors, etc. use the USB or BAT pins, and an additional regulator.
- RST - The reset pin is used for the reset button, but can also be used standalone. Tie it to ground to reset the board.

Power Connectors

The board can be powered from either of the following connectors.

- USB-C connector - The USB-C connector is located on the left end of the board. It is used for powering and programming the board, reading serial console output back to your computer, and charging a lipoly battery (if connected).
- JST lipoly battery connector - The lipoly battery connector, located to the right of the mounting hole in the upper left corner of the board, allows you to power the board via a lipoly battery, and, if also plugged into USB, charge the battery as well.
Battery Monitor

- LiPoly Battery Monitor - The lipoly battery monitor, located to the left of the STEMMA QT connector, towards the center of the board, has a two 200K resistor divider.
- BATT_MONITOR pin (GPI35) - This ADC pin can be used to read the data from the battery monitor. Basically perform an analog read and multiply by two, to get a rough sense of the lipoly battery voltage.

Charge LED

- CHG LED - This LED, located below the USB-C connector, and labeled CHG on the silk, lights up while the battery is charging. It is normal for it to possibly flicker while no battery is plugged in.

ESP32

The ESP32 Dual core 240MHz Xtensa® processor is located on the right end of the board. It comes with 8MB of flash and 2MB of PSRAM. There is also a tuned PCB antenna, and all the passives you need to take advantage of this powerful new processor. The ESP32 has both WiFi and Bluetooth Classic/LE support. That means it's perfect for just about any wireless or Internet-connected project.
At the top-left corner of the ESP32 module, is a STEMMA QT connector, labeled QT I2C on the silk. This connector allows you to connect a variety of sensors and breakouts (https://adafruit.it/NmD) with STEMMA QT connectors using various associated cables (https://adafruit.it/JRA).

You must enable the NEOPIXEL_I2C_POWER pin (GPIO 2) for the STEMMA QT connector power to work. Set it to be an output and HIGH in your code.

There is a NEOPIXEL_I2C_POWER (GPIO 2) pin that must be set to an output and HIGH for the STEMMA QT connector power to work.

For running in low power mode, you can disable (set output and LOW) the NEOPIXEL_I2C_POWER pin, this will turn off the separate 3.3V regulator that powers the QT connector's red wire

LED and NeoPixel
There are two controllable LEDs on the ESP32 Feather V2.

- NeoPixel RGB LED - This RGB LED, on `NEOPIXEL_PIN` or pin 0, can be controlled in code to light up any color of the rainbow. Treat it like a normal WS2812B.
- NeoPixel Power Pin - There is a `NEOPIXEL_I2C_POWER` or pin 2, which must be set to an output and HIGH for the NeoPixel to work. You can set this pin output and LOW for low power mode.
- Red LED - This little red LED, labeled #13 on the silk, is pin 13. It can be controlled in code like any LED, set high to turn on.

There is a `NEOPIXEL_I2C_POWER` (GPIO 2) pin that must be set HIGH for the NeoPixel LED to work.

**Buttons**

There are two buttons on the ESP32 Feather V2.

- SW38 - This is a user readable button switch to use as an input, labeled SW38 on the silk. You can read it on pin GPI 38. Note this is an input-only pin. There is a pull-up on board.
- Reset - The reset button, labeled reset on the silk, is used to reboot the board.
Logic Pins

These are all of the GPIO pins available on the ESP32 Feather V2. Check out the PrettyPins diagram above for more details.

Bottom Row

- A0 - This is also DAC2, as well as pin 26. It uses ADC2.
- A1 - This is also DAC1, as well as pin 25. It uses ADC2.
- A2 - This is also pin 34. It is input/ADC only. It uses ADC1.
- A3 - This is also pin 39. It is input/ADC only. It uses ADC1.
- A4 - This is also pin 36. It is input/ADC only. It uses ADC1.
- A5 - This is also pin 4. It uses ADC2.
- SCK - This is the SPI clock pin. It is also pin 5.
- MO - This is the SPI Microcontroller Out / Serial In (MOSI) pin. It is also pin 19.
- MI - This is the SPI Microcontroller In / Serial Out (MISO) pin. It is also pin 21.
- RX - This is the UART RX (receive) pin. It is also pin 8. Connect to the TX pin found on a breakout or device. This is separate than the 'debug UART' which is connected to the USB-to-Serial converter, so it will not interfere during upload.
- TX - This is the UART TX (transmit) pin. It is also pin 7. Connect to the RX pin found on a breakout or device. This is separate than the 'debug UART' which is connected to the USB-to-Serial converter, so it will not interfere during upload.
- 37 - This is also pin I37. It is input/ADC only. It uses ADC1.

Top Row

- IO13 - This is also pin 13. It uses ADC2.
- IO12 - This is also pin 12. It uses ADC2.
- IO27 - This is also pin 27. It uses ADC2.
- IO33 - This is also pin 33. It uses ADC1.
• IO15 - This is also pin 15. It uses ADC2.
• IO32 - This is also pin 32. It uses ADC1.
• IO14 - This is also pin 14. It uses ADC2.
• SCL - This is the I2C clock pin. It is also pin 20.
• SDA - This is the I2C data pin. It is also pin 22.

The pin numbers for the I2C port (SDA, SCL), hardware UART (RX, TX), and SPI (SCK, MOSI, MISO) have changed from the original Feather ESP32. If your code has hardcoded use for those pins, you'll want to replace them either by the new numbers or change the code to use the 'pretty' names like SDA or SCK.

When selecting the new Feather ESP32 V2 board in the Espressif board support package, the correct numbers will be substituted. Note the names are in the same spots, we haven't changed where the I2C/UART/SPI pins are located on the board, just which ESP32 pin numbers they are connected to in the module.

CP2102N USB to Serial Converter

The CP2102N USB to serial converter can handle 3 mbps max rate.

Low Power Usage

This microcontroller board can be used for low power usage thanks to the ESP32's multiple sleep modes.

There are three basic operating states to Espressif chips: normal, light sleep and deep sleep.
Normal power usage is as you expect: you can use the chip and run code as you like - connecting to WiFi, reading sensors, etc.

Light sleep is sort of a 'hibernation' - power usage is minimal and WiFi is disconnected, but the internal clock and memory is kept. That means you can wake up where you left off, in the middle of the code as desired. You'll still need to re-initialize any external hardware that got disconnected, and WiFi, but it's often faster than waking from a deep sleep.

Deep sleep is the lowest power but the tradeoff is that all memory and state is lost - the only thing that's running is the real time clock that can wake the chip up. When woken up, the chip starts as if it was physically reset - from the beginning of the code. This can be beneficial if you want to have a fresh start each time.

A rough guideline is:

- Normal power: 100mA+ can be as much power as need and spike during WiFi connection
- Light sleep: 2mA assuming all external hardware is de-powered
- Deep sleep: 100uA assuming all external hardware is de-powered

The Adafruit ESP32 Feather V2 has a NEOPIXEL_I2C_POWER pin that controls power to I2C and the NeoPixel LED. Disabling this pin by setting it to an output and LOW allows you to drop the power draw, even when you have I2C sensors or breakouts connected.

Here's a generic sketch we use for all our boards that has a macro-defined section for enabling and disabling all external powered elements. For example, if there's a power pin for NeoPixels, I2C port, TFT, etc...we turn that off before going into light or deep sleep! This will minimize power usage.

```cpp
#include <Adafruit_NeoPixel.h>

// While we wait for Feather ESP32 V2 to get added to the Espressif BSP,
// we have to select PICO D4 and UNCOMMENT this line!
//#define ADAFRUIT_FEATHER_ESP32_V2

// then these pins will be defined for us
#if defined(ADAFRUIT_FEATHER_ESP32_V2)
#define PIN_NEOPIXEL 0
#define NEOPIXEL_I2C_POWER 2
#endif
```

©Adafruit Industries
#if defined(PIN_NEOPIXEL)
    Adafruit_NeoPixel pixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
#endif

void setup() {
    Serial.begin(115200);

    // Turn on any internal power switches for TFT, NeoPixels, I2C, etc!
    enableInternalPower();
}

void loop() {
    LEDon();
    delay(1000);

    disableInternalPower();
    LEDoff();
    esp_sleep_enable_timer_wakeup(1000000); // 1 sec
    esp_light_sleep_start();
    // we'll wake from light sleep here

    // wake up 1 second later and then go into deep sleep
    esp_sleep_enable_timer_wakeup(1000000); // 1 sec
    esp_deep_sleep_start(); // we never reach here
}

void LEDon() {
    #if defined(PIN_NEOPIXEL)
        pixel.begin(); // INITIALIZE NeoPixel
        pixel.setBrightness(20); // not so bright
        pixel.setPixelColor(0, 0xFFFFFF);
        pixel.show();
    #endif
}

void LEDoff() {
    #if defined(PIN_NEOPIXEL)
        pixel.setPixelColor(0, 0x0);
        pixel.show();
    #endif
}

void enableInternalPower() {
    #if defined(NEOPIXEL_POWER)
        pinMode(NEOPIXEL_POWER, OUTPUT);
        digitalWrite(NEOPIXEL_POWER, HIGH);
    #endif

    #if defined(NEOPIXEL_I2C_POWER)
        pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
        digitalWrite(NEOPIXEL_I2C_POWER, HIGH);
    #endif

    #if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
        // turn on the I2C power by setting pin to opposite of 'rest state'
        pinMode(PIN_I2C_POWER, INPUT);
        delay(1);
        bool polarity = digitalRead(PIN_I2C_POWER);
        pinMode(PIN_I2C_POWER, OUTPUT);
        digitalWrite(PIN_I2C_POWER, !polarity);
        pinMode(NEOPIXEL_POWER, OUTPUT);
        digitalWrite(NEOPIXEL_POWER, HIGH);
    #endif
}

void disableInternalPower() {
    #if defined(NEOPIXEL_POWER)
        pinMode(NEOPIXEL_POWER, OUTPUT);
    #endif
}
The best way to really test power draw is with a specialty power meter such as the Nordic PPK 2

Nordic nRF-PPK2 - Power Profiler Kit II
The Power Profiler Kit II is a standalone unit, which can measure and optionally supply currents all the way from sub-uA and as high as 1A on all Nordic DKs, in...
https://www.adafruit.com/product/5048

When running the above code and monitoring with a PPK, you'll get a graph like this:

The big pulse is normal mode, you can see the ESP32 booting up, loading code, and then pausing 1 second. Then there's a big drop for one sec to light sleep, and finally one more 1 second pause at deep sleep.
Power Draw for ESP32 Feather V2

The following graphs show the power draw for the ESP32 Feather V2 in normal power mode, light sleep mode, and deep sleep mode.

Normal Power Mode

![Graph showing power draw in normal power mode]

The power draw, running normally (without WiFi), is 40mA.

Light Sleep Mode

![Graph showing power draw in light sleep mode]

The power draw in light sleep mode is 1.1mA.
Deep Sleep Mode

The power draw in deep sleep mode is 70\textmu A.

Arduino IDE Setup

We primarily recommend using the ESP32 chipsets with Arduino. Don't forget you will also need to install the SiLabs CP2104 Driver if you are using an ESP32 board with USB-to-Serial converter! (There's no harm in doing it, so we recommend even if you aren't)

Install Arduino IDE

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using version 1.8 or higher for this guide

Arduino IDE Download

https://adafruit.it/f1P

Install CP2104 / CP2102N USB Driver

Many ESP32 boards have a USB-to-Serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac, Windows and Linux.

Click here to download the CP2104 USB Driver

https://adafruit.it/vrf
Install ESP32 Board Support Package from GitHub

For this board we recommend you don't use 'release' version of Espressif's board support package because the current release doesn't include board support.

Instead we will install the "very latest" by following these instructions (https://adafru.it/YYB) (scroll down for mac and linux as well)

Basically, install by git cloning the esp32 board support to get the very latest version of the code

In the Tools → Board submenu you should see ESP32 Arduino (in sketchbook) and in that dropdown it should contain the ESP32 boards along with all the latest ESP32 boards.

Look for the Adafruit Feather ESP32 V2.

The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. 921600 should work fine, but if you're having issues, you can drop down lower.
Blink

The first and most basic program you can upload to your Arduino is the classic Blink sketch. This takes something on the board and makes it, well, blink! On and off. It's a great way to make sure everything is working and you're uploading your sketch to the right board and right configuration.

When all else fails, you can always come back to Blink!

Pre-Flight Check: Get Arduino IDE & Hardware Set Up

This lesson assumes you have Arduino IDE set up. This is a generalized checklist, some elements may not apply to your hardware. If you haven't yet, check the previous steps in the guide to make sure you:

- Install the very latest Arduino IDE for Desktop (not all boards are supported by the Web IDE so we don't recommend it)
- Install any board support packages (BSP) required for your hardware. Some boards are built in defaults on the IDE, but lots are not! You may need to install plug-in support which is called the BSP.
- Get a Data/Sync USB cable for connecting your hardware. A significant amount of problems folks have stem from not having a USB cable with data pins. Yes, these cursed cables roam the land, making your life hard. If you find a USB cable that doesn't work for data/sync, throw it away immediately! There is no need to keep it around, cables are very inexpensive these days.
• Install any drivers required - If you have a board with a FTDI or CP210x chip, you may need to get separate drivers. If your board has native USB, it probably doesn't need anything. After installing, reboot to make sure the driver sinks in.
• Connect the board to your computer. If your board has a power LED, make sure its lit. Is there a power switch? Make sure its turned On!

The Feather ESP32 V2 has no power LED or power switch. Make sure you download the CP2104 / CP2102N USB Driver if you haven't already. Check out the IDE Setup page (https://adafru.it/YYC) for details.

Start up Arduino IDE and Select Board/Port

OK now you are prepared! Open the Arduino IDE on your computer. Now you have to tell the IDE what board you are using, and how you want to connect to it.

In the IDE find the Tools menu. You will use this to select the board. If you switch boards, you must switch the selection! So always double-check before you upload code in a new session.

![Arduino IDE Tools Menu](https://example.com/arduino-tools.png)
New Blink Sketch

OK lets make a new blink sketch! From the File menu, select New

Then in the new window, copy and paste this text:

```cpp
int led = LED_BUILTIN;

void setup() {
    // Some boards work best if we also make a serial connection
    Serial.begin(115200);
    // set LED to be an output pin
    pinMode(led, OUTPUT);
}

void loop() {
    // Say hi!
    Serial.println("Hello!");
    digitalWrite(led, HIGH);  // turn the LED on (HIGH is the voltage level)
    delay(500);  // wait for a half second
    digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
    delay(500);  // wait for a half second
}
```

Note that in this example, we are not only blinking the LED but also printing to the Serial monitor, think of it as a little bonus to test the serial connection.

One note you'll see is that we reference the LED with the constant `LED_BUILTIN` rather than a number. That's because, historically, the built in LED was on pin 13 for Arduinos. But in the decades since, boards don't always have a pin 13, or maybe it could not be used for an LED. So the LED could have moved to another pin. It's best to use `LED_BUILTIN` so you don't get the pin number confused!

The red LED on the Feather ESP32 V2 is available as `LED_BUILTIN`, as well as `13`.
Verify (Compile) Sketch

OK now you can click the Verify button to convert the sketch into binary data to be uploaded to the board.

Note that Verifying a sketch is the same as Compiling a sketch - so we will use the words interchangeably.

During verification/compilation, the computer will do a bunch of work to collect all the libraries and code and the results will appear in the bottom window of the IDE.

If something went wrong with compilation, you will get red warning/error text in the bottom window letting you know what the error was. It will also highlight the line with an error.

For example, here I had the wrong board selected - and the selected board does not have a built in LED!
Here's another common error, in my haste I forgot to add a `;` at the end of a line. The compiler warns me that it's looking for one - note that the error is actually a few lines up!

Turning on detailed compilation warnings and output can be very helpful sometimes - Its in Preferences under "Show Verbose Output During:" and check the Compilation button. If you ever need to get help from others, be sure to do this and then provide all the text that is output. It can assist in nailing down what happened!

On success you will see something like this white text output and the message Done compiling. in the message area.
Upload Sketch

Once the code is verified/compiling cleanly you can upload it to your board. Click the Upload button.

The IDE will try to compile the sketch again for good measure, then it will try to connect to the board and upload a the file.

This is actually one of the hardest parts for beginners because it's where a lot of things can go wrong.

However, let's start with what it looks like on success! Here's what your board upload process looks like when it goes right:

Often times you will get a warning like this, which is kind of vague:

No device found on COM66 (or whatever port is selected)
An error occurred while uploading the sketch

This could be a few things.
First up, check again that you have the correct board selected! Many electronics boards have very similar names or look, and often times folks grab a board different from what they thought.

Second, make sure you selected the right port! If you have the wrong port or no port selected, Arduino doesn't know where to look for your board.

If both of those are correct, the next step is to enable verbose upload messages.

Before continuing we really really suggest turning on Verbose Upload messages, it will help in this process because you will be able to see what the IDE is trying to do. It's a checkbox in the Preferences menu.

Now you can try uploading again!

This time, you should have success!

After uploading this way, be sure to click the reset button - it sort of makes sure that the board got a good reset and will come back to life nicely.

Finally, a Blink!

OK it was a journey but now we're here and you can enjoy your blinking LED. Next up, try to change the delay between blinks and re-upload. It's a good way to make sure your upload process is smooth and practiced.
I2C Scan Test

'A lot of sensors, displays, and devices can connect over I2C. I2C is a 2-wire 'bus' that allows multiple devices to all connect on one set of pins so it's very convenient for wiring!

When using your board, you'll probably want to connect up I2C devices, and it can be a little tricky the first time. The best way to debug I2C is go through a checklist and then perform an I2C scan

Common I2C Connectivity Issues

- Have you connected four wires (at a minimum) for each I2C device? Power the device with whatever is the logic level of your microcontroller board (probably 3.3V), then a ground wire, and a SCL clock wire, and a SDA data wire.
- If you're using a STEMMA QT board - check if the power LED is lit. Its usually a green LED to the left side of the board.
- Does the STEMMA QT/I2C port have switchable power or pullups? To reduce power, some boards have the ability to cut power to I2C devices or the pullup resistors. Check the documentation if you have to do something special to turn on the power or pullups.
- If you are using a DIY I2C device, do you have pullup resistors? Many boards do not have pullup resistors built in and they are required! We suggest any common 2.2K to 10K resistors. You'll need two: one each connects from SDA to positive power, and SCL to positive power. Again, positive power (a.k.a VCC, VDD or V+) is often 3.3V
- Do you have an address collision? You can only have one board per address. So you cannot, say, connect two AHT20's to one I2C port because they have the
same address and will interfere. Check the sensor or documentation for the address. Sometimes there are ways to adjust the address.

- Does your board have multiple I2C ports? Historically, boards only came with one. But nowadays you can have two or even three! This can help solve the "hey but what if I want two devices with the same address" problem: just put one on each bus.
- Are you hot-plugging devices? I2C does not support dynamic re-connection, you cannot connect and disconnect sensors as you please. They should all be connected on boot and not change. (Only exception is if you're using a hot-plug assistant but that'll cost ya (https://adafruit.it/XBY))
- Are you keeping the total bus length reasonable? I2C was designed for maybe 6" max length. We like to push that with plug-n-play cables but really please keep them as short as possible! (Only exception is if you're using an active bus extender (https://adafruit.it/XBZ)).

The Feather ESP32 V2 does not have I2C pullup resistors - make sure any sensor you attach to the I2C port has resistors or add them yourself.

The Feather ESP32 V2 has a NEOPIXEL_I2C_POWER pin that must be pulled HIGH to enable power to the STEMMA QT port. Without it, the QT port will not work!

**Perform an I2C scan!**

**Install TestBed Library**

To scan I2C, the Adafruit TestBed library is used. This library and example just makes the scan a little easier to run because it takes care of some of the basics. You will need to add support by installing the library. Good news: it is very easy to do it. Go to the Arduino Library Manager.

Search for TestBed and install the Adafruit TestBed library
Now open up the I2C Scan example

```cpp
#include <Adafruit_TestBed.h>
extern Adafruit_TestBed TB;
#define DEFAULT_I2C_PORT &Wire
// Some boards have TWO I2C ports, how nifty. We should scan both
#if defined(ARDUINO_ADAFRUIT_KB2040_RP2040) ||
   defined(ARDUINO_ADAFRUIT_ITSYBITSY_RP2040) ||
   defined(ARDUINO_ADAFRUIT_FEATHER_RP2040) ||
   defined(ARDUINO_ADAFRUIT_QTPY_RP2040) ||
   defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) ||
   defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
#define SECONDARY_I2C_PORT &Wire1
#endif
void setup() {
  Serial.begin(115200);
  // Wait for Serial port to open
  while (!Serial) { 
    delay(10);
  }
  delay(500);
  Serial.println("Adafruit I2C Scanner");
#if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) ||
   defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
  // ESP32 is kinda odd in that secondary ports must be manually
  // assigned their pins with setPins()!
  Wire1.setPins(SDA1, SCL1);
#endif
#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32_V2)
  // turn on the I2C power by setting pin to opposite of 'rest state'
  pinMode(PIN_I2C_POWER, INPUT);
  delay(1);
  bool polarity = digitalRead(PIN_I2C_POWER);
  pinMode(PIN_I2C_POWER, OUTPUT);
  digitalWrite(PIN_I2C_POWER, !polarity);
#endif
#if defined(ADAFRUIT_FEATHER_ESP32_V2)
```
// Turn on the I2C power by pulling pin HIGH.
pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
digitalWrite(NEOPIXEL_I2C_POWER, HIGH);
#endif
}

void loop() {
  // Serial.println("\n");
  Serial.println("\n");

  Serial.print("Default port ");
  TB.theWire = DEFAULT_I2C_PORT;
  TB.printI2CBusScan();

#if defined(SECONDARY_I2C_PORT)
  Serial.print("Secondary port ");
  TB.theWire = SECONDARY_I2C_PORT;
  TB.printI2CBusScan();
#endif

  delay(3000); // wait 3 seconds
}

Wire up I2C device

While the examples here will be using the Adafruit MCP9808 (https://adafruit.it/Trb), a high accuracy temperature sensor, the overall process is the same for just about any I2C sensor or device.

The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.

[Image of Adafruit MCP9808]

Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027

Wiring the MCP9808

The MCP9808 comes with a STEMMA QT connector, which makes wiring it up quite simple and solder-free.
Simply plug a STEMMA QT to STEMMA QT cable into the port on the Feather V2, and a port on the MCP9808 breakout.

Now upload the scanning sketch to your microcontroller and open the serial port to see the output. You should see something like this:

WiFi Test

Thanksfully if you have ESP32 sketches, they'll 'just work' with variations of ESP32. You can find a wide range of examples in the File->Examples->Examples for Adafruit Metro ESP32-S2 subheading (the name of the board may vary so it could be "Examples for Adafruit Feather ESP32 V2" etc)
Let's start by scanning the local networks.

Load up the WiFiScan example under Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan

And upload this example to your board. The ESP32 should scan and find WiFi networks around you.

For ESP32, open the serial monitor, to see the scan begin.

For ESP32-S2 and -C3, don't forget you have to click Reset after uploading through the ROM bootloader. Then select the new USB Serial port created by the ESP32-S2. It will take a few seconds for the board to complete the scan.
If you cannot scan any networks, check your power supply. You need a solid power supply in order for the ESP32 to not brown out. A skinny USB cable or drained battery can cause issues.

WiFi Connection Test

Now that you can scan networks around you, its time to connect to the Internet!

Copy the example below and paste it into the Arduino IDE:

```cpp
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*
Web client
This sketch connects to a website (wifitest.adafruit.com/testwifi/index.html) using the WiFi module.

This example is written for a network using WPA encryption. For WEP or WPA, change the Wifi.begin() call accordingly.

This example is written for a network using WPA encryption. For WEP or WPA, change the Wifi.begin() call accordingly.

created 13 July 2010
by dlf (Metodo2 srl)
modified 31 May 2012
by Tom Igoe
*/
#include <WiFi.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID"; // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD"; // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0; // your network key Index number (needed only for WEP)

int status = WL_IDLE_STATUS;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
// IPAddress server(74,125,232,128); // numeric IP for Google (no DNS)
char server[] = "wifitest.adafruit.com"; // name address for adafruit test
char path[] = "/testwifi/index.html";

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
WiFiClient client;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
  }
```

©Adafruit Industries
// attempt to connect to Wifi network:
Serial.print("Attempting to connect to SSID: ");
Serial.println(ssid);

WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println(""即
Serial.println("Connected to WiFi");
printWifiStatus();
Serial.println("Starting connection to server...");
// if you get a connection, report back via serial:
if (client.connect(server, 80)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.print("GET "); client.print(path); client.println(" HTTP/1.1");
    client.print("Host: "); client.println(server);
    client.println("Connection: close");
    client.println();
}
}

void loop() {
    // if there are incoming bytes available
    // from the server, read them and print them:
    while (client.available()) {
        char c = client.read();
        Serial.write(c);
    }
    // if the server's disconnected, stop the client:
    if (!client.connected()) {
        Serial.println();
        Serial.println("disconnecting from server.");
        client.stop();
        // do nothing forevermore:
        while (true) {
            delay(100);
        }
    }
}

void printWifiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your board's IP address:
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI): ");
    Serial.print(rssi);
    Serial.println(" dBm");
}
NOTE: You must change the \texttt{SECRET_SSID} and \texttt{SECRET_PASS} in the example code to your WiFi SSID and password before uploading this to your board.

```c
// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID";    // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD";    // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0;    // your network key index number (needed only for WEP)
```

After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password

```
Attempting to connect to SSID: Transit
........
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-57 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 11 Nov 2020 20:51:30 GMT
Content-Type: text/html
Content-Length: 70
Last-Modified: Thu, 16 May 2019 18:21:16 GMT
Connection: close
ETag: "5cddaa1c-46"
Accept-Ranges: bytes

This is a test of Adafruit WiFi!
If you can read this, its working :)
```

Secure Connection Example

Many servers today do not allow non-SSL connectivity. Lucky for you the ESP32 has a great TLS/SSL stack so you can have that all taken care of for you. Here's an example of a using a secure WiFi connection to connect to the Twitter API.

Copy and paste it into the Arduino IDE:

```c
// SPDX-FileCopyrightText: 2015 Arturo Guadalupi
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT
```
/*
This example creates a client object that connects and transfers
data using always SSL.

It is compatible with the methods normally related to plain
connections, like client.connect(host, port).

Written by Arturo Guadalupi
last revision November 2015
*/

#include <WiFiClientSecure.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID"; // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD"; // your network password (use for WPA, or
use as key for WEP)
int keyIndex = 0; // your network key Index number (needed
only for WEP)

int status = WL_IDLE_STATUS;
// if you don’t want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128); // numeric IP for Google (no DNS)
#define SERVER "cdn.syndication.twimg.com"
#define PATH   "/widgets/followbutton/info.json?screen_names=adafruit"

// Initialize the SSL client library
// with the IP address and port of the server
// that you want to connect to (port 443 is default for HTTPS):
WiFiClientSecure client;

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  if (client.connect(SERVER, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println("GET " PATH " HTTP/1.1");
    client.println("Host: " SERVER);
    client.println("Connection: close");
    client.println();
  }
}
void loop() {
    // if there are incoming bytes available
    // from the server, read them and print them:
    while (client.available()) {
        char c = client.read();
        Serial.write(c);
        bytes++;
    }

    // if the server's disconnected, stop the client:
    if (!client.connected()) {
        Serial.println();
        Serial.println("disconnecting from server.");
        client.stop();
        Serial.print("Read "); Serial.print(bytes); Serial.println(" bytes");
        // do nothing forevermore:
        while (true);
    }
}

void printWifiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your board's IP address:
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI):");
    Serial.print(rssi);
    Serial.println(" dBm");
}

As before, update the ssid and password first, then upload the example to your board.

Note we use `WiFiClientSecure client` instead of `WiFiClient client;` to require a SSL connection! This example will connect to a twitter server to download a JSON snippet that says how many followers adafruit has
This example is a little more advanced - many sites will have API's that give you JSON data. We will build on the previous SSL example to connect to twitter and get that JSON data chunk

Then we'll use ArduinoJSON (https://adafru.it/Evn) to convert that to a format we can use and then display that data on the serial port (which can then be re-directed to a display of some sort)

First up, use the Library manager to install ArduinoJSON (https://adafru.it/Evo).

Then load the example JSONdemo by copying the code below and pasting it into your Arduino IDE.

```
// SPDX-License-Identifier: MIT

/*
This example creates a client object that connects and transfers data using always SSL, then shows how to parse a JSON document in an HTTP response.

It is compatible with the methods normally related to plain connections, like client.connect(host, port).
*/
```

© Adafruit Industries
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>

// uncomment the next line if you have a 128x32 OLED on the I2C pins
//#define USE_OLED
// uncomment the next line to deep sleep between requests
//#define USE_DEEPSLEEP

#if defined(USE_OLED)
// Some boards have TWO I2C ports, how nifty. We should use the second one sometimes
#if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) || defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
#define OLED_I2C_PORT &Wire1
#else
#define OLED_I2C_PORT &Wire
#endif
#endif

#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, OLED_I2C_PORT);

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID"; // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD"; // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0; // your network key Index number (needed only for WEP)

int status = WL_IDLE_STATUS;
// if you don’t want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128); // numeric IP for Google (no DNS)

#define SERVER "cdn.syndication.twimg.com"
#define PATH   "/widgets/followbutton/info.json?screen_names=adafruit"

t{
  //Initialize serial and wait for port to open:
  Serial.begin(115200);

  // Connect to WPA/WPA2 network
  WiFi.begin(ssid, pass);

  #if defined(USE_OLED)
    setupI2C();
    delay(200); // wait for OLED to reset
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
      Serial.println(F("SSD1306 allocation failed"));
      for(;;){} // Don't proceed, loop forever
    }
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.clearDisplay();
    display.setCursor(0,0);
  #else
    // Don't wait for serial if we have an OLED
    while (!Serial) {
      // wait for serial port to connect. Needed for native USB port only
      delay(10);
    
    Serial.println(F("Starting the sketch.
    "));
  
    // Connect to WiFi
    WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.print("Connecting to ");
      Serial.println(ssid);
    }
    
    // Print the IP address of the sketch
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    
    // Initialize the OLED
    display = Adafruit_SSD1306(128, 32, OLED_I2C_PORT);
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println("Hello, world!");
    display.display();
  
    // Set the verbose level to 2 for detailed logging
    verbose_level = 2;
  
  #endif
}


```c
} #endif
// attempt to connect to Wifi network:
Serial.print("Attempting to connect to SSID: ");
Serial.println(ssid);
#if defined(USE_OLED)
  display.clearDisplay(); display.setCursor(0,0);
  display.print("Connecting to SSID\n");
  display.println(ssid);
  display.display();
#endif
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("\nConnected to WiFi");
#endif
printWifiStatus();
}

uint32_t bytes = 0;

void loop() {
  WiFiClientSecure client;
  client.setInsecure(); // don't use a root cert

  Serial.println("\nStarting connection to server...");
  #if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
    display.print("Connecting to "); display.println(SERVER);
    display.display();
  #endif

  // if you get a connection, report back via serial:
  if (client.connect(SERVER, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println("GET " PATH " HTTP/1.1");
    client.println("Host: " SERVER);
    client.println("Connection: close");
    client.println();
  }
  // Check HTTP status
  char status[32] = {0};
  client.readBytesUntil(\r\n, status, sizeof(status));
  if (strcmp(status, "HTTP/1.1 200 OK\n") != 0) {
    Serial.print(F("Unexpected response: "));
    Serial.println(status);
    #if defined(USE_OLED)
      display.println("Connection failed, code: ");
      display.println(status);
      display.display();
    #endif
    return;
  }
  // wait until we get a double blank line
  client.find(\n\n\n\n, 4);
  // Allocate the JSON document
```
// Use arduinojson.org/v6/assistant to compute the capacity.
const size_t capacity = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(8) + 200;
DynamicJsonDocument doc(capacity);

// Parse JSON object
DeserializationError error = deserializeJson(doc, client);
if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.c_str());
    return;
}

// Extract values
JsonObject root_0 = doc[0];
Serial.println(F("Response: "));
const char* root_0_screen_name = root_0["screen_name"];
long root_0_followers_count = root_0["followers_count"];
Serial.print("Twitter username: "); Serial.println(root_0_screen_name);
Serial.print("Twitter followers: "); Serial.println(root_0_followers_count);
#if defined(USE_OLED)
display.clearDisplay(); display.setCursor(0,0);
display.setTextSize(2);
display.println(root_0_screen_name);
display.println(root_0_followers_count);
display.display();
display.setTextSize(1);
#endif

// Disconnect
client.stop();
delay(1000);
#if defined(USE_DEEPSLEEP)
#if defined(USE_OLED)
display.clearDisplay();
display.display();
#endif // OLED
#if defined(NEOPIXEL_I2C_POWER)
digitalWrite(NEOPIXEL_I2C_POWER, LOW); // off
#else

digitalWrite(NEOPIXEL_I2C_POWER, LOW); // off
#endif
 // wake up 1 second later and then go into deep sleep
esp_sleep_enable_timer_wakeup(10 * 1000UL * 1000UL); // 10 sec
esp_deep_sleep_start();
#else

delay(10 * 1000);
#endif

void setupI2C() {
    #if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) ||
defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
    // ESP32 is kinda odd in that secondary ports must be manually
    // assigned their pins with setPins()!
    Wire1.setPins(SDA1, SCL1);
    #endif

    #if defined(NEOPIXEL_I2C_POWER)
    pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
digitalWrite(NEOPIXEL_I2C_POWER, HIGH); // on
    #endif

    #if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
    // turn on the I2C power by setting pin to opposite of 'rest state'
pinMode(PIN_I2C_POWER, INPUT);
delay(1);
    bool polarity = digitalRead(PIN_I2C_POWER);
pinMode(PIN_I2C_POWER, OUTPUT);
digitalWrite(PIN_I2C_POWER, !polarity);
#endif

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI): ");
  Serial.print(rssi);
  Serial.println(" dBm");
}

By default it will connect to the Twitter banner image API, parse the username and followers, and display them.

```
Attempting to connect to SSID: Transit
........
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-54 dBm

Starting connection to server...
connected to server
Response:
Twitter username: adafruit
Twitter followers: 176400
```

Factory Reset

Your microcontroller ships running a factory demo. It's lovely, but you probably had other plans for the board. As you start working with your board, you may want to return to the original code to begin again, or you may find your board gets into a bad state. Either way, this page has you covered.

Factory Reset Example Code

If you're still able to load Arduino sketches, you can load the following sketch onto your board to return it to its original state.
```
#include "WiFi.h"
#include <Adafruit_TestBed.h>
extern Adafruit_TestBed TB;
#define NEOPIXEL_I2C_POWER 2
#define NEOPIXEL_PIN 0

// the setup routine runs once when you press reset:
void setup() {
  Serial.begin(115200);

  // turn on the QT port and NeoPixel
  pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
  digitalWrite(NEOPIXEL_I2C_POWER, HIGH);

  // TestBed will handle the neopixel swirl for us
  TB.neopixelPin = NEOPIXEL_PIN;
  TB.neopixelNum = 1;
  TB.begin();
  TB.setColor(0xFF0000);
  delay(50);
  TB.setColor(0x00FF00);
  delay(50);
  TB.setColor(0x0000FF);

  // Set WiFi to station mode and disconnect from an AP if it was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
}

// the loop routine runs over and over again forever:
uint8_t wheelColor = 0;
void loop() {
  if (wheelColor == 0) {
    // Test I2C!
    Serial.print("I2C port ");
    TB.theWire = &Wire;
    TB.printI2CBusScan();

    // Test WiFi Scan!
    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.print("WiFi AP scan done...");
    if (n == 0) {
      Serial.println("no networks found");
    } else {
      Serial.print(n);
      Serial.println(" networks found");
      for (int i = 0; i < n; ++i) {
        // Print SSID and RSSI for each network found
        Serial.print(i + 1);
        Serial.print(" ");
        Serial.print(WiFi.SSID(i));
        Serial.print(" ( ");
        Serial.print(WiFi.RSSI(i));
        Serial.print(")");
        Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? "*" : " ");
        delay(10);
      }
    }
  }
  Serial.println(" ");
}
```
Your board is now back to its factory-shipped state! You can now begin again with your plans for your board.

## Factory Reset .bin

If your board is in a state where Arduino isn't working, you may need to use these tools to flash a .bin file directly onto your board.

There are two ways to do a factory reset. The first is using WebSerial through a Chromium-based browser, and the second is using `esptool` via command line. We highly recommend using WebSerial through Chrome/Chromium.

First you'll need to download the factory-reset.bin file. Save the following file wherever is convenient for you. You'll need access to it for both tools.

Click to download feather-esp32-v2-factory-reset.bin

[https://adafru.it/YRb](https://adafru.it/YRb)

Now that you've downloaded the .bin file, you're ready to continue with the factory reset process. The next two sections walk you through using WebSerial and `esptool`.

### The WebSerial ESPTool Method

We highly recommend using WebSerial ESPTool method to perform a factory reset and bootloader repair. However, if you'd rather use `esptool` via command line, you can skip this section.

This method uses the WebSerial ESPTool through Chrome or a Chromium-based browser. The WebSerial ESPTool was designed to be a web-capable option for programming ESP32 boards. It allows you to erase the contents of the microcontroller and program up to four files at different offsets.
You will have to use a Chromium browser (like Chrome, Opera, Edge...) for this to work, Safari and Firefox, etc. are not supported because we need Web Serial and only Chromium is supporting it to the level needed.

Follow the steps to complete the factory reset.

If you're using Chrome 88 or older, see the Older Versions of Chrome section at the end of this page for instructions on enabling Web Serial.

Connect

You should have plugged in only the ESP32 that you intend to flash. That way there's no confusion in picking the proper port when it's time!

In the Chrome browser visit https://adafruit.github.io/Adafruit_WebSerial_ESPTool/ (https://adafru.it/PMB). You should see something like the image shown.

Press the Connect button in the top right of the web browser. You will get a pop up asking you to select the COM or Serial port.

Remember, you should remove all other USB devices so only the ESP32 board is attached, that way there's no confusion over multiple ports!
The Javascript code will now try to connect to the ROM bootloader. It may timeout for a bit until it succeeds. On success, you will see that it is Connected and will print out a unique MAC address identifying the board.

Once you have successfully connected, the command toolbar will appear.

Erase the Contents

To erase the contents, click the Erase button. You will be prompted whether you want to continue. Click OK to continue or if you changed your mind, just click cancel.
You'll see "Erasing flash memory. Please wait..." This will eventually be followed by "Finished." and the amount of time it took to erase.

Do not disconnect! Immediately continue on to programming the ESP32.

Do not disconnect after erasing! Immediately continue on to the next step!

Program the ESP32

Programming the microcontroller can be done with up to four files at different locations, but with the board-specific factory-reset.bin file, which you should have downloaded earlier, you only need to use one file.

Click on the first Choose a file.... (The tool will only attempt to program buttons with a file and a unique location.) Then, select the *-factory-reset.bin file you downloaded in Step 1 that matches your board.

Verify that the Offset box next to the file location you used is (0x) 0.

Once you choose a file, the button text will change to match your filename. You can then select the Program button to begin flashing.
A progress bar will appear and after a minute or two, you will have written the firmware.

Once completed, you can skip down to the section titled Reset the Board.

The **esptool** Method (for advanced users)

If you used WebSerial ESPTool, you do not need to complete the steps in this section!

Alternatively, you can use Espressif’s esptool program ([https://adafruit.it/E9p](https://adafruit.it/E9p)) to communicate with the chip! esptool is the ‘official’ programming tool and is the most common/complete way to program an ESP chip.

Install ESPTool.py

You will need to use the command line / Terminal to install and run esptool.

You will also need to have pip and Python installed (any version!).

Install the latest version using pip (you may be able to run pip without the 3 depending on your setup):

```bash
pip3 install --upgrade esptool
```

Then, you can run:

```bash
esptool.py
```
Test the Installation

Run `esptool.py` in a new terminal/command line and verify you get something like the below:

```
$ esptool.py
esptool.py v3.0
usage: esptool [-h] [--chip {auto,esp8266,esp32,esp32s2,esp32s3beta2,esp32c3}]
             [--port PORT] [--baud BAUD]
             [--before {default_reset,no_reset,no_reset_no_sync}]
             [--after {hard_reset,soft_reset,no_reset} [--no-stub]]
             [--trace] [--override-vids dio {11.0V,1.8V,0V}]
             [--connect-amounts CONNECT_ATTEMPTS]
             [load_ram,dump_mem,read_mem,write_mem,write_flash,run,image_info,
             make_image,elf2image,read_mac,chip_id,flash_id,read.flash_status,
             write.flash_status,read.flash,verify_flash,erase.flash,erase.region,
             version,get_security_info]
```

Connect

Run the following command, replacing the `COM88` identifier after `--port` with the `COMxx`, `/dev/cu.usbmodemxx` or `/dev/ttySxx` you found above.

```
esptool.py --port COM88 chip_id
```

You should get a notice that it connected over that port and found an ESP32.

Installing the Factory Test file

Run this command and replace the serial port name, `COM88`, with your matching port and `*-factory-reset.bin` with file you just downloaded

```
esptool.py --port COM88 write_flash 0x0 *-factory-reset.bin
```
Don't forget to change the `--port` name to match.

There might be a bit of a 'wait' when programming, where it doesn't seem like it's working. Give it a minute, it has to erase the old flash code which can cause it to seem like it's not running.

You'll finally get an output like this:

![Output of esptool.py](image)

Once completed, you can continue to the next section.

**Reset the board**

Now that you've reprogrammed the board, you need to reset it to continue. Click the reset button to launch the new firmware.

In the event that pressing the reset button does not restart the board, unplug the board from USB and plug it back in to get the new firmware to start up.

The NeoPixel LED on the Feather ESP32 V2 will light up blue, followed by a repeating rainbow swirl.

You've successfully returned your board to a factory reset state!

**Older Versions of Chrome**

As of chrome 89, Web Serial is already enabled, so this step is only necessary on older browsers.
We suggest updating to Chrome 89 or newer, as Web Serial is enabled by default.

If you must continue using an older version of Chrome, follow these steps to enable Web Serial.

If you receive an error like the one shown when you visit the WebSerial ESPTool site, you’re likely running an older version of Chrome.

You must be using Chrome 78 or later to use Web Serial.

To enable Web Serial in Chrome versions 78 through 88:

Visit chrome://flags from within Chrome. Find and enable the Experimental Web Platform features

Restart Chrome

Downloads

Files

- ESP32 datasheet (https://adafru.it/YQA)
- CP2102N datasheet (https://adafru.it/YQB)
- EagleCAD PCB files on GitHub (https://adafru.it/YQC)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/YQD)
- PrettyPins pinout PDF (https://adafru.it/YQE) on GitHub (https://adafru.it/YQE)
- PrettyPins pinout SVG (https://adafru.it/YQF)