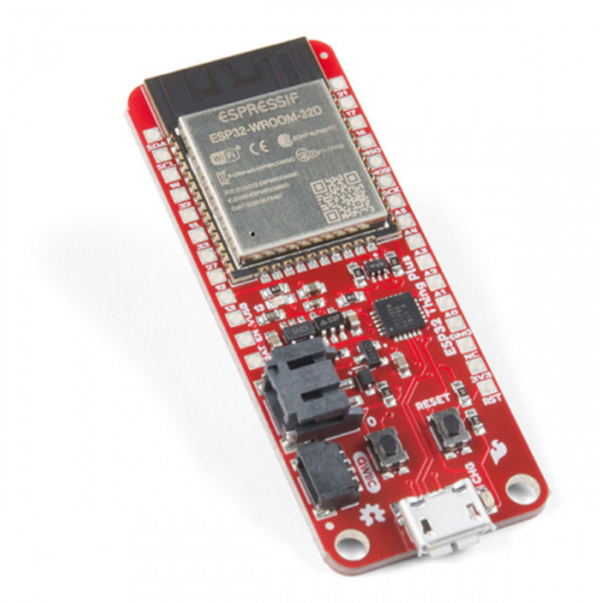
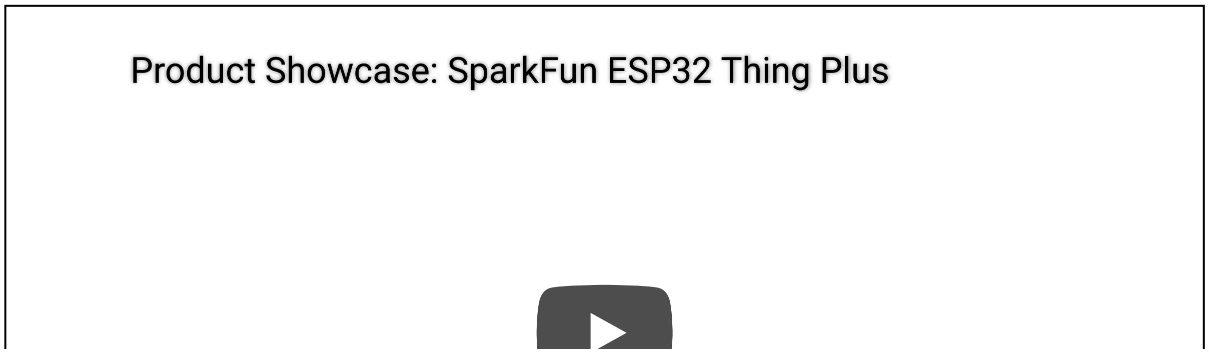
ESP32 Thing Plus Hookup Guide

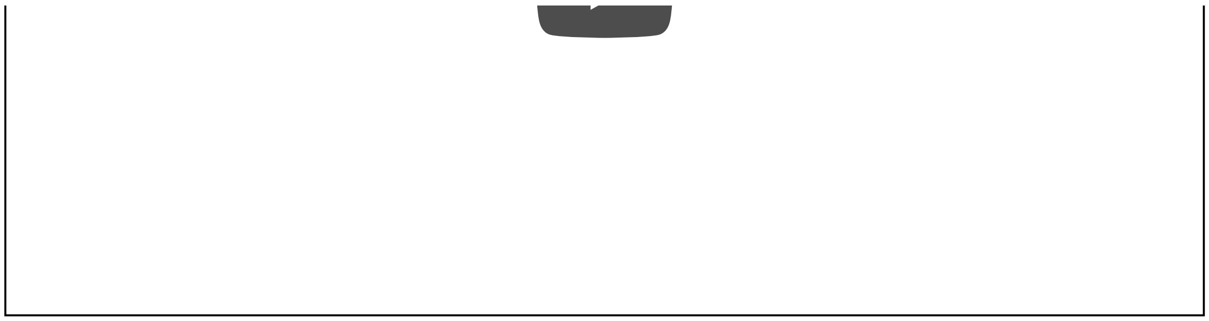
Introduction

The SparkFun ESP32 Thing Plus enjoys all the amenities of the ESP32 Thing, but with a few added sparkles. We've lengthened the board just a bit to accommodate a Qwiic connector for all your Qwiic breakout needs. We've also moved a few pins around to make the board compatible with the Adafruit Huzzah32 – ESP32 Feather Board such that you can use all of those lovely shields available out there! The ESP32 Thing plus also integrates a rich set of peripherals, ranging from capacitive touch sensors, Hall sensors, SD card interface, Ethernet, high-speed SPI, UART, I^{2}S and I^{2}C.



SparkFun Thing Plus - ESP32 WROOM
● WRL-14689





Not Yet Implemented The Arduino board definitions for the ESP32 are still a work in progress. There are a handful of peripherals and features that have yet to be implemented, including:
* Analog Input (`analogRead([pin])`)
* Analog Ouptut (`analogWrite([pin], [value])`)
* WiFi Server and WiFI UDP
* Real-Time Clock
* Touch-controller interface

These peripherals are available (if, also, still in their infancy) in the IoT Development Framework for the ESP32. If your application requires analog input, RTC, or any of the features above, consider giving the ESP-IDF a try!

Required Materials

Much of the ESP32 Thing Plus's functionality can be used by simply powering the board. To do so, you'll need a Micro-B USB Cable. The ESP32 Thing Plus's USB interface can be used to both power and program the chip. Once you're done programming the chip, a 5V Micro-B USB Wall Adapter can be used to power the board.

Wall Adapter Power Supply - 5V DC 2A (USB Micro-B)

TOL-12890

USB micro-B Cable - 6 Foot

CAB-10215

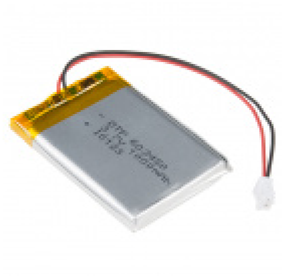
USB Wall Charger - 5V, 1A (Black)
◉ TOL-11456

USB Micro-B Cable - 6"
◉ CAB-13244

⚡**WARNING!**
Make sure your power supply is **5V**, NOT 5.1V. We have noticed a power spike in our 5.1V power supplies, that can damage the IC. Long cables can also generate a large enough voltage spike to damage the IC. We recommend keeping power supply cables shorter than 6 feet to minimize potential damage.

As an alternative power source, the ESP32 Thing Plus includes support for single-cell lithium-polymer (LiPo) batteries, which plug into the board's white 2-pin JST connector. LiPos are perfect for projects on-the-go, or those that just need a little extra umph. The board includes a **LiPo charger** -- the rechargeable batteries can be juiced back up by plugging the Thing Plus into a 5V USB source.

Lithium Ion Battery - 2Ah
◉ PRT-13855



Lithium Ion Battery - 1Ah
◉ PRT-13813

Lithium Ion Battery - 400mAh
◉ PRT-13851

Lithium Ion Battery - 850mAh
◉ PRT-13854

Should you wish to make use of the board's qwiic functionality, you'll need a qwiic cable:

Qwiic Cable - 100mm
○ PRT-14427

Qwiic Cable - 500mm

Qwiic Cable - 500mm
◉ PRT-14429

Qwiic Cable - 200mm
◉ PRT-14428

Qwiic Cable - 50mm
◉ PRT-14426

Tools

To take advantage of the ESP32 Thing Plus's 28 external pins, you will need a soldering iron, solder, and general soldering accessories.

Break Away Headers - Straight
○ PRT-00116

Solder Lead Free - 100-gram Spool
◉ TOL-09325

Weller WLC100 Soldering Station
● TOL-14228

Suggested Reading

It may look intimidating, but the ESP32 Thing Plus -- especially when you take advantage of its Arduino compatibility -- is a perfect IoT foundation for electronics users of all experience levels. There are, however, a few concepts you should be familiar with before venturing further into this tutorial. If any of the concepts below sound foreign to you, consider reading through that tutorial first:

How to Solder: Through-Hole Soldering
This tutorial covers everything you need to know about through-hole soldering.

Serial Communication
Asynchronous serial communication concepts: packets, signal levels, baud rates, UARTs and more!

How to Power a Project
A tutorial to help figure out the power requirements of your project.

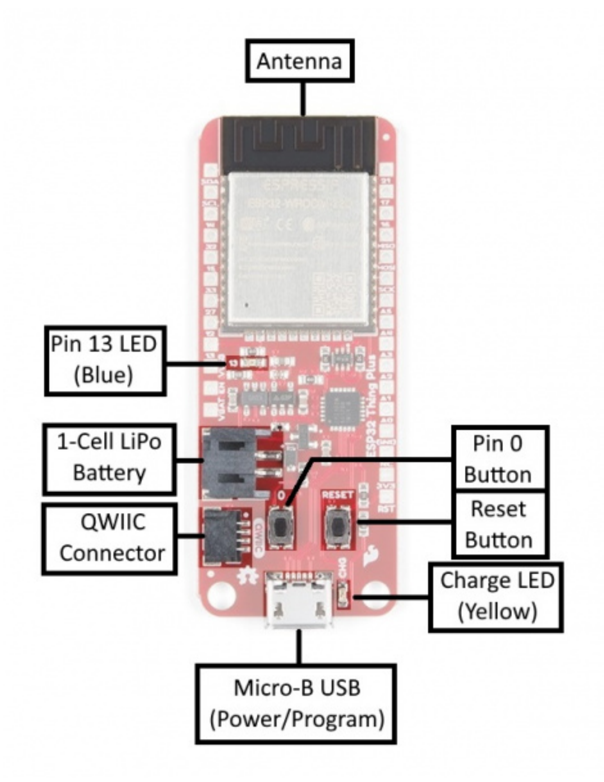
What is an Arduino?
What is this 'Arduino' thing anyway?

Hardware Overview

Espressif's ESP32 WROOM is a powerful, generic Wi-Fi+BT+BLE MCU module that targets a wide variety of applications. At the core of this module is the ESP32-D0WDQ6 chip which is designed to be both scalable and adaptive. It's laundry list of features include:

- Xtensa® dual-core 32-bit LX6 microprocessor
- Up to 240MHz clock frequency
- 16MB of flash storage
- 520kB internal SRAM
- Integrated 802.11 BGN WiFi transceiver
- Integrated dual-mode Bluetooth (classic and BLE)
- 2.3 to 3.6V operating range
- 21 GPIO
- 8-electrode capacitive touch support
- Hardware accelerated encryption (AES, SHA2, ECC, RSA-4096)
- 2.5 µA deep sleep current

The ESP32 Thing Plus is designed around the ESP32-WROOM module with everything necessary to run and program the microcontroller, plus a few extra goodies to take advantage of the chip's unique features.



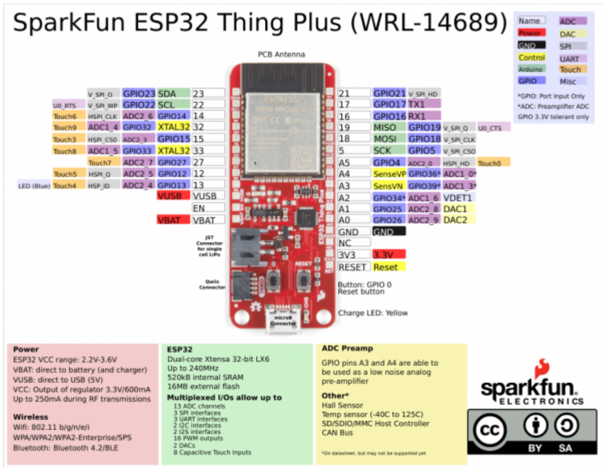
Peripherals and I/O

The ESP32 Thing Plus features your standard fare of hardware peripherals including:

- 13 analog to digital converter (ADC) channels
- 3 UARTs (only two are configured by default in the Arduino IDE, one UART is used for bootloading/debug)
- 3 SPI (only one is configured by default in the Arduino IDE)
- 2 I^2C (only one is configured by default in the Arduino IDE)
- 2 I2S Audio
- 2 digital-to-analog converter (DAC) channels
- 16 PWM outputs

And, thanks to the chip's **pin multiplexing** feature, those peripherals can be connected to just about any of the broken out I/O pins. Than means you decide which pins are RX, TX, MISO, MOSI, SCLK, SDA, SCL, etc.

There are, however, a few hardware features -- namely the ADC and DAC -- which are asigned to static pins. The graphical datasheet below helps demonstrate where you can find those peripherals (click to embiggen!).



Power
ESP32 VCC range: 2.2V-3.6V
VBAT: direct to battery (and charger)
VUSB: direct to USB (5V)
VCC: Output of regulator 3.3V/600mA
Up to 250mA during RF transmissions

Wireless
Wifi: 802.11 b/g/n/e/i
WPA/WPA2/WPA2-Enterprise/SPS
Bluetooth: Bluetooth 4.2/BLE

ESP32
Dual-core Xtensa 32-bit LX6
Up to 240MHz
520kB internal SRAM
16MB external flash

Multiplexed I/Os allow up to
13 ADC channels
3 SPI interfaces
3 UART interfaces
2 I2C interfaces
2 I2S interfaces
16 PWM outputs
2 DACs
8 Capacitive Touch Inputs

ADC Preamp
GPIO pins A3 and A4 are able to be used as a low noise analog pre-amplifier

Other*
Hall Sensor
Temp sensor (-40C to 125C)
SD/SDIO/MMC Host Controller
CAN Bus

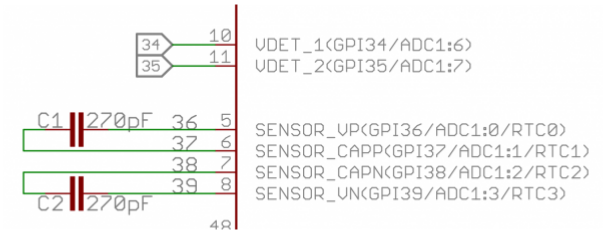
*On datasheet, but may not be supported yet

Click the image for a closer look.

One I^{2}C, two of the UART interfaces, and one of the SPI interfaces can be assigned to any pin your project requires.

❗ Input Only Pins: A2, A3, A4
Pins 34 (A2), 36 (A4), and 39 (A3) *cannot* be configured as outputs, but they can be used as either digital inputs, analog inputs, or for other unique purposes. Also note that they **do not have internal pull-up or pull-down resistors**, like the other I/O pins.

GPIO pins 36-39 are an integral part of the ultra low noise pre-amplifier for the ADC – they are wired up to 270pF capacitors, which help to configure the sampling time and noise of the pre-amp.


34
35
10
11
VDET_1(GPI34/ADC1:6)
VDET_2(GPI35/ADC1:7)
C1 270pF
36 5
37 6
38 7
39 8
C2 270pF
48
SENSOR_VP(GPI36/ADC1:0/RTC0)
SENSOR_CAPP(GPI37/ADC1:1/RTC1)
SENSOR_CAPN(GPI38/ADC1:2/RTC2)
SENSOR_VN(GPI39/ADC1:3/RTC3)

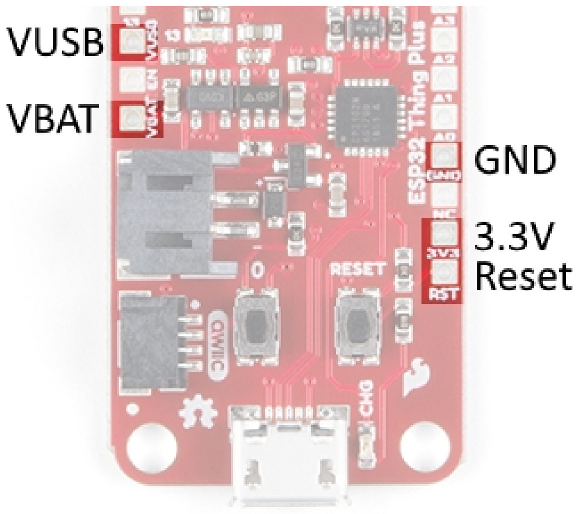
*From the ESP32 Thing Schematic: GPIO 36-39 are tied together with caps. GPIO 36-39 as well as pins 34 and 35 are **input only**!*

Powering the ESP32 Thing Plus

The two main power inputs to the ESP32 Thing Plus are **USB** and a **single-cell lithium-polymer (LiPo** battery. If both USB and the LiPo are plugged into the board, the onboard charge controller will charge the LiPo battery at a rate up to **500mA**.

⚡ The ESP32's operating voltage range is **2.2** to **3.6V**. Under normal operation the ESP32 Thing Plus will power the chip at **3.3V**. The I/O pins are **not 5V-tolerant**! If you interface the board with 5V (or higher) components, you'll need to do some **logic level shifting**.

The **3.3V regulator** on the ESP32 Thing Plus can **reliably supply up to 600mA**, which should be more than enough overhead for most projects. The ESP32 can pull as much as **250mA during RF transmissions**, but we've generally measured it to consume around **150mA** -- even while actively transmitting over WiFi. The output of the regulator is also broken out to the sides of the board -- the pin labeled "**3V3**". This pin can be used to supply external components.



In addition to USB and battery connectors, the **VBAT**, and **VUSB** pins are all broken out to the sides of the board. These pins can be used as an alternative supply input to the Thing Plus. The maximum, allowable voltage input to **VUSB is 5.8V**, and VBAT should not be connected to anything other than a LiPo battery. Alternatively, if you have a regulated voltage source between **2.2V** and **3.6V**, the "**3V3**" line can be used to directly supply the ESP32 and its peripherals.

Assembly Tips

The ESP32 Thing Plus ships without anything soldered into the header pins -- ensuring that you can mold the board to best fit your project. To use the chip's pins you'll need to solder *something* to the I/O and power rail vias broken out to either side of the board.

New to soldering? Check out our Through-Hole Soldering Tutorial for a quick introduction!

What you solder to the ESP32 Thing Plus's I/O pins is completely up to you. The header rows are breadboard-compatible, so you may want to solder male headers in.

ESP32 Thing Plus with soldered male headers.

Then plug it into the breadboard, hanging the USB and LiPo connectors off the end, and start wiring!

Alternatively, female headers (you may need two separate strips to solder all the pins), right-angle headers, or stranded wire are all good options, depending on your project's needs.

Software Setup

Note: This example assumes you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE.

Installation for the ESP32 Thing Plus is two-fold. Like the ESP32 Thing, you will want to install the board definitions via the Arduino Boards manager. In addition, you will also need to download and install the CP2104 USB Driver.

Installing Board Definition

Espressif has added support for the Arduino Boards Manager that includes a slew of great built-in examples. Instructions for installing via the board manager can be found at espressif's arduino-esp32 GitHub.

INSTALLATION INSTRUCTIONS USING ARDUINO IDE BOARDS MANAGER

For more information on installing boards via the Arduino Board Manager, check out the add-ons section of our Installing Arduino IDE tutorial.

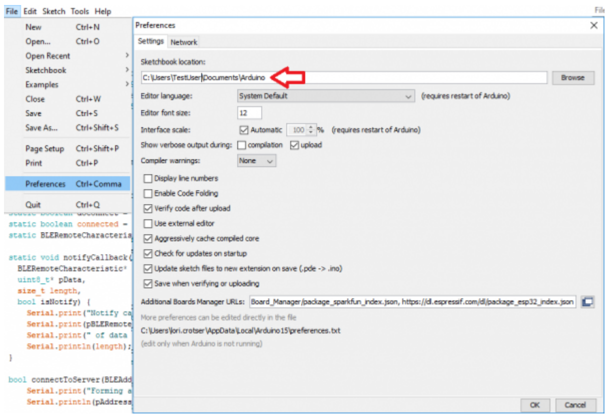
Installing Arduino IDE
MARCH 26, 2013
A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.

If you are familiar with installing boards via the Arduino IDE Boards Manager, the url to add is:

```
https://dl.espressif.com/dl/package_esp32_index.json
```

Note: If you have previously installed the ESP32 Arduino Core, we strongly recommend removing the associated folders before installing via the boards manager.

To remove previous arduino core installs for the esp32, start by finding your ***.../Arduino/hardware*** folder. This can be located by looking at your Sketchbook location under **File** > **Preferences**.



Sketchbook location: C:\Users\TestUser\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Show verbose output during: ☐ compilation ☑ upload

Compiler warnings: None

☐ Display line numbers

☐ Enable Code Folding

☑ Verify code after upload

☐ Use external editor

☑ Aggressively cache compiled core

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Additional Boards Manager URLs: Board_Manager/package_sparkfun_index.json, https://dl.espressif.com/dl/package_esp32_index.json

More preferences can be edited directly in the file

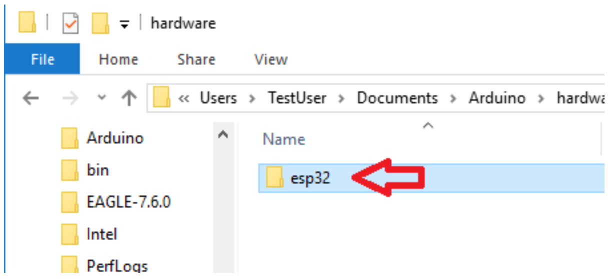
C:\Users\lori.crotser\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

```
static boolean connected =
static BLERemoteCharacteris

static void notifyCallback(
  BLERemoteCharacteristic*
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify ca
    Serial.print(pBLERemote
    Serial.print(" of data
    Serial.println(length);
}

bool connectToServer(BLEAdd
    Serial.print("Forming a
    Serial.println(pAddress
```

Click to enlarge.

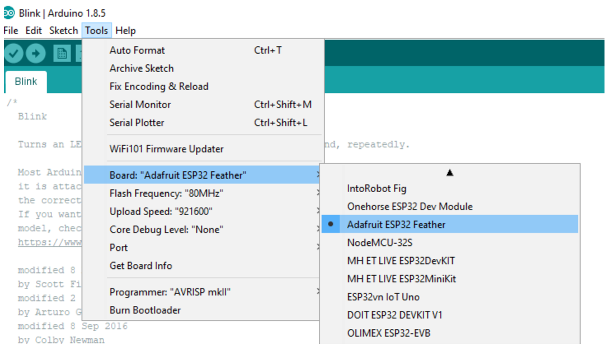
Go to this location in your finder and delete the **esp32** folder.



Once you have deleted the **esp32** folder, you can then install using the **Arduino Boards Manager**.

Selecting the Board Definition

Once installed, use the Adafruit ESP32 Feather board in the Arduino Board dropdown.



Board: "Adafruit ESP32 Feather"

Flash Frequency: "80MHz"

Upload Speed: "921600"

Core Debug Level: "None"

Programmer: "AVRISP mkII"

● Adafruit ESP32 Feather

*Selecting board definition from **Tools** drop down menu.*

Feeling adventurous?

If you are feeling strong or would like to have more control over your development environment, you can install Espressif's official ESP32 Arduino core. Installation is covered in our ESP32 Thing Hookup Guide.

Installing the CP2104 USB Driver

You will also need to install the SiLabs CP2104 Driver, which can be found here: USB to UART Bridge VCP Driver

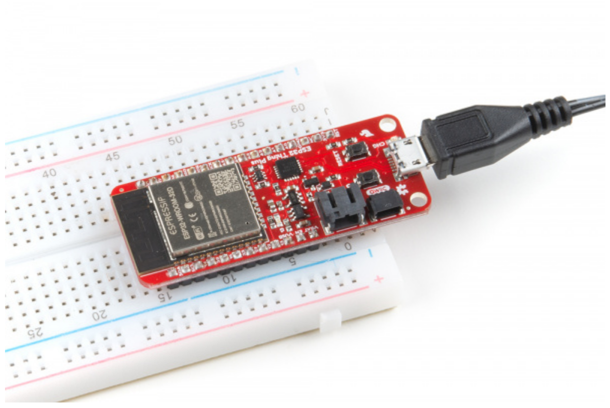
DOWNLOAD WINDOWS VCP DRIVER (ZIP)

DOWNLOAD MAC OSX VCP DRIVER (ZIP)

Note: If applicable, make sure you are using the proper driver files for your CPU architecture. This is usually indicated by a folder or file name with "*x86*" for 32-bit processors or "*x64*" for 64-bit processors.

Arduino Example: Blink

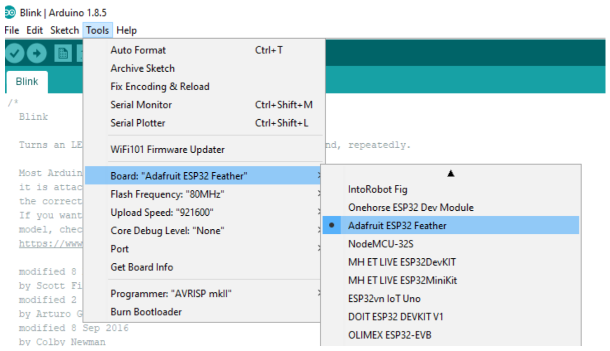
With the ESP32 Arduino core installed, you're ready to begin programming. If you haven't already, **plug the ESP32 Thing Plus into your computer** using a micro-B USB cable.



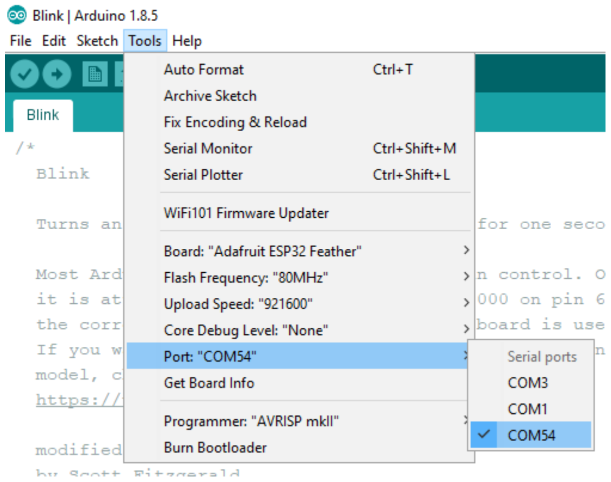
Once the board is plugged in (and drivers installed), it should be assigned a unique port identifier. On Windows machines, this will be something like `COM#` , and on Macs or Linux computers it will come in the form of `/dev/tty.usbserial-XXXXXX` .

Select the Board and Port

Make sure you have the Adafruit ESP32 Feather board definition selected under your **Tools** > **Board** menu.



Then select your ESP32 Thing Plus' serial port under the **Tools** > **Port** menu.



Board: "Adafruit ESP32 Feather"
Flash Frequency: "80MHz"
Upload Speed: "921600"
Core Debug Level: "None"
Port: "COM54"
Programmer: "AVRISP mkII"

You can also select the **Upload Speed: "921600"** baud -- the fastest selectable rate -- will get the code loaded onto your ESP32 the fastest, but may fail to upload once-in-a-while. (It's still way worth it for the speed increase!)

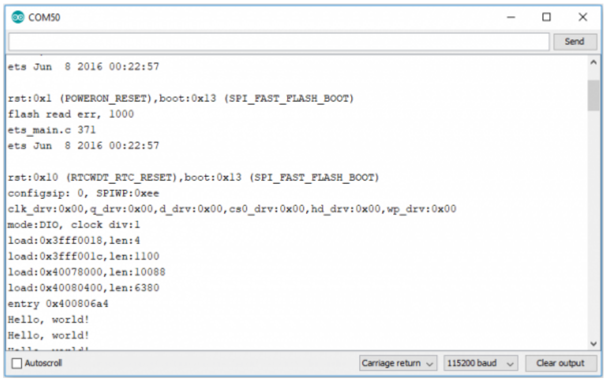
Loading Blink

To make sure your toolchain and board are properly set up, we'll upload the simplest of sketches -- Blink! The LED attached to **GPIO 13** is perfect for this test. Plus, with the ESP32 attached to your computer, this is a good time to test out serial communication. Copy and paste the example sketch below into a fresh Arduino sketch:

```
int ledPin = 13;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(115200);
}

void loop()
{
    Serial.println("Hello, world!");
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
}
```

With everything setup correctly, upload the code! Once the code finishes transferring, **open the serial monitor** and set the baud rate to **115200**. You should see `Hello, world!` 's begin to fly by.

If the blue LED remains dimly lit, it's probably still sitting in the bootloader. After uploading a sketch, you may need to **tap the RST button** to get your ESP32 Thing Plus to begin running the sketch.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4
Hello, world!
Hello, world!
```

You may also notice that when the ESP32 boots up it prints out a long sequence of debug messages. These are emitted every time the chip resets -- always at **115200 baud**.

Arduino Example: WiFi

The ESP32 Arduino core includes a handful of WiFi examples, which demonstrate everything from scanning for nearby networks to sending data to a client server. You can find the examples under the **File** > **Examples** > **WiFi** menu.

Here's another example using the WiFi library, which demonstrates how to connect to a nearby WiFi network and poll a remote domain (http://example.com/) as a client.

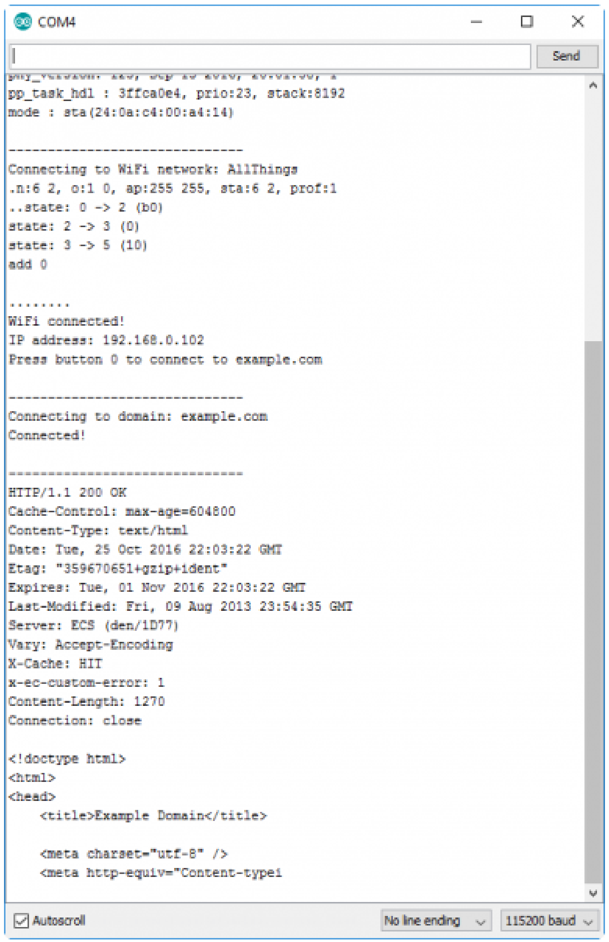
❗ **Please note:** Make sure you are connecting to the 2.4GHz band on your wireless router; the ESP32 is not (yet) compatible with 5GHz signals.

```
#include <WiFi.h>

// WiFi network name and password:
const char * networkName = "YOUR_NETWORK_HERE";
const char * networkPswd = "YOUR_PASSWORD_HERE";

// Internet domain to request from:
const char * hostDomain = "example.com";
const int hostPort = 80;

const int BUTTON_PIN = 0;
const int LED_PIN = 5;

void setup()
{
  // Initilize hardware:
  Serial.begin(115200);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);

  // Connect to the WiFi network (see function below loop)
  connectToWiFi(networkName, networkPswd);

  digitalWrite(LED_PIN, LOW); // LED off
  Serial.print("Press button 0 to connect to ");
  Serial.println(hostDomain);
}

void loop()
{
  if (digitalRead(BUTTON_PIN) == LOW)
  { // Check if button has been pressed
    while (digitalRead(BUTTON_PIN) == LOW)
      ; // Wait for button to be released

    digitalWrite(LED_PIN, HIGH); // Turn on LED
    requestURL(hostDomain, hostPort); // Connect to server
    digitalWrite(LED_PIN, LOW); // Turn off LED
  }
}

void connectToWiFi(const char * ssid, const char * pwd)
{
  int ledState = 0;

  printLine();
  Serial.println("Connecting to WiFi network: " + String(ssid));

  WiFi.begin(ssid, pwd);

  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink LED while we're connecting:
```

```
    digitalWrite(LED_PIN, ledState);
    ledState = (ledState + 1) % 2; // Flip ledState
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.println("WiFi connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void requestURL(const char * host, uint8_t port)
{
  printLine();
  Serial.println("Connecting to domain: " + String(host));

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  if (!client.connect(host, port))
  {
    Serial.println("connection failed");
    return;
  }
  Serial.println("Connected!");
  printLine();

  // This will send the request to the server
  client.print((String)"GET / HTTP/1.1\r\n" +
               "Host: " + String(host) + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0)
  {
    if (millis() - timeout > 5000)
    {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }

  // Read all the lines of the reply from server and print them to Serial
  while (client.available())
  {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
  client.stop();
}
```

```
void printLine()
{
  Serial.println();
  for (int i=0; i<30; i++)
    Serial.print("-");
  Serial.println();
}
```

Make sure you fill in the `networkName` and `networkPswd` variables with the name (or SSID) and password of your WiFi network! Once you've done that and uploaded the code, open your **serial monitor**.



```
pp_task_hdl : 3ffca0e4, prio:23, stack:8192
mode : sta(24:0a:c4:00:a4:14)

------------------------------
Connecting to WiFi network: AllThings
.n:6 2, o:1 0, ap:255 255, sta:6 2, prof:1
..state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)
add 0

........
WiFi connected!
IP address: 192.168.0.102
Press button 0 to connect to example.com

------------------------------
Connecting to domain: example.com
Connected!

------------------------------
HTTP/1.1 200 OK
Cache-Control: max-age=604800
Content-Type: text/html
Date: Tue, 25 Oct 2016 22:03:22 GMT
Etag: "359670651+gzip+ident"
Expires: Tue, 01 Nov 2016 22:03:22 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (den/1D77)
Vary: Accept-Encoding
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270
Connection: close

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-typei
```

After your ESP32 connects to the WiFi network, it will wait for you to press the "0" button. Tapping that will cause the ESP32 to make an HTTP request to example.com. You should see a string of HTTP headers and HTML similar to the screenshot above.

Arduino Example: ESP32 BLE

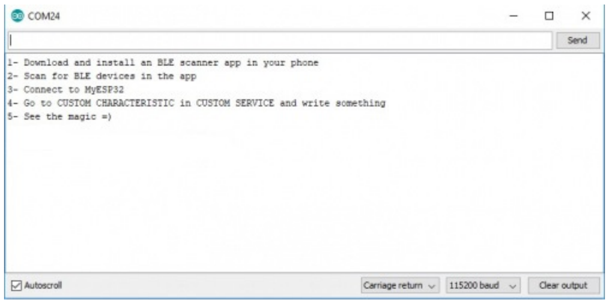
Both the board manager install and the ESP32 arduino core install come with Bluetooth examples that range from serial to serial to acting as a simple BLE device to functioning as either a Bluetooth server or client. Here we will briefly go over the `BLE_write` example that can be found in **Files** > **Examples** > **ESP32 BLE Arduino**. This example allows you to write messages on your phone that can then be read in a serial monitor on your computer.

This example works with a **BLE scanner** on your phone. A good, basic app is the BLE Scanner for iPhone or Android. Make sure to install the app to follow along with this example.

Compile and upload the following code, or if you wish, open the **BLE_write** example from the **Files** > **Examples** > **ESP32 BLE Arduino** menu. Make sure you have ***ESP32 Dev Module*** as your board and the correct port has been selected.

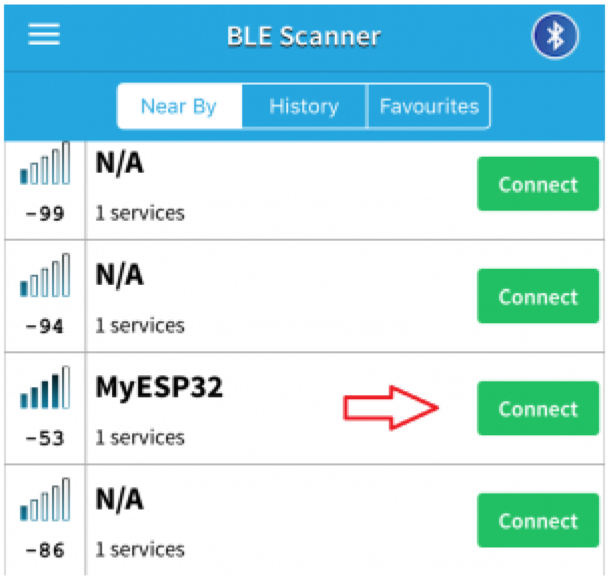
```
/*
    Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/master/
cpp_utils/tests/BLE%20Tests/SampleWrite.cpp
    Ported to Arduino ESP32 by Evandro Copercini
*/

#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/

#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"


class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string value = pCharacteristic->getValue();

      if (value.length() > 0) {
        Serial.println("*********");
        Serial.print("New value: ");
        for (int i = 0; i < value.length(); i++)
          Serial.print(value[i]);

        Serial.println();
        Serial.println("*********");
      }
    }
};

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

  Serial.println("1- Download and install an BLE scanner app in your phone");
  Serial.println("2- Scan for BLE devices in the app");
  Serial.println("3- Connect to MyESP32");
  Serial.println("4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something");
  Serial.println("5- See the magic =)");

  BLEDevice::init("MyESP32");
  BLEServer *pServer = BLEDevice::createServer();

  BLEService *pService = pServer->createService(SERVICE_UUID);

  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID,
                                         BLECharacteristic::PROPERTY_READ |
                                         BLECharacteristic::PROPERTY_WRITE
                                       );
  pCharacteristic->setCallbacks(new MyCallbacks());

  pCharacteristic->setValue("Hello World");
  pService->start();

  BLEAdvertising *pAdvertising = pServer->getAdvertising();
  pAdvertising->start();
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
}
```

Once you have uploaded your code, open a Serial Monitor set at **115200 baud** so you can see the message that we will write.

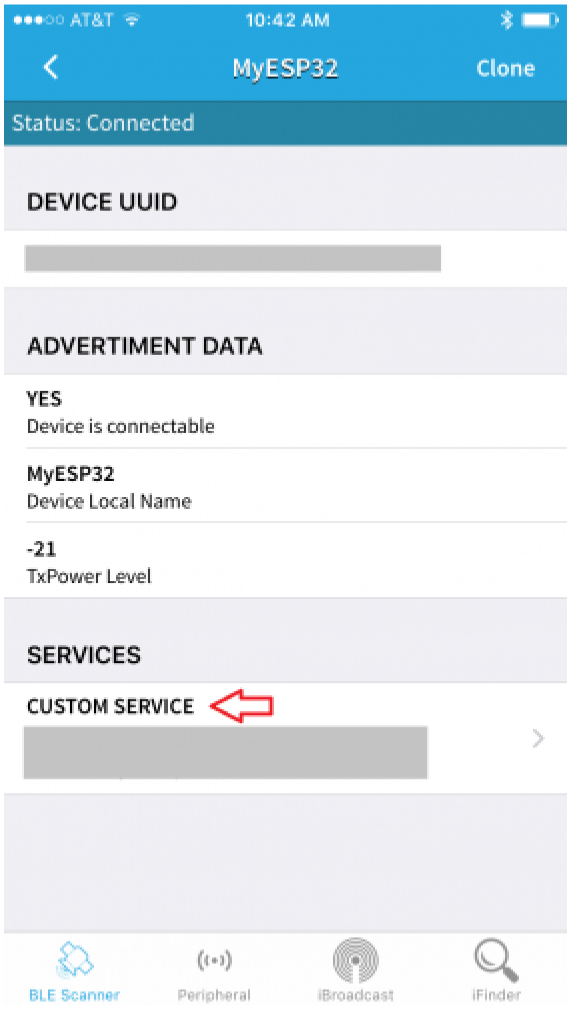


```
1- Download and install an BLE scanner app in your phone
2- Scan for BLE devices in the app
3- Connect to MyESP32
4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something
5- See the magic =)
```

Then open your **BLE Scanner** app on your phone. You will see multiple options; scroll through these and connect to **MyESP32**.



Now we need to drill down to the communication capability we want. Once you are connected to **MyESP32**, you will be taken to the following page. Select **CUSTOM SERVICE**.



●●●○○ AT&T 10:42 AM

MyESP32

Clone

Status: Connected

DEVICE UUID

[redacted]

ADVERTIMENT DATA

YES
Device is connectable

MyESP32
Device Local Name

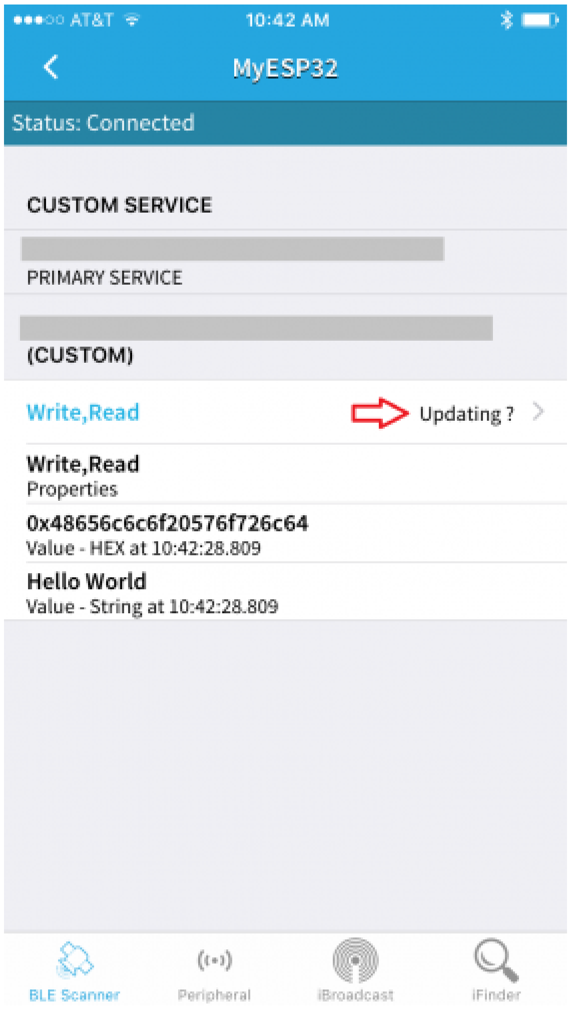
-21
TxPower Level

SERVICES

CUSTOM SERVICE

[redacted]

The next page will show you communications and options for doing so. Select **Write,Read**.



●●●○○ AT&T 10:42 AM

MyESP32

Status: Connected

CUSTOM SERVICE

[redacted]

PRIMARY SERVICE

[redacted]

(CUSTOM)

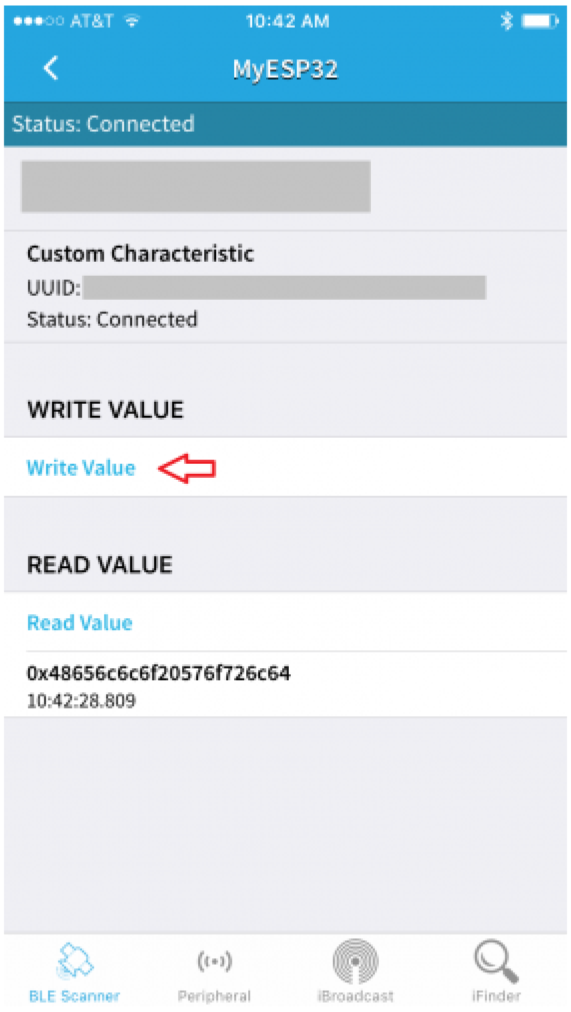
Write,Read Updating ?

Write,Read
Properties

0x48656c6c6f20576f726c64
Value - HEX at 10:42:28.809

Hello World
Value - String at 10:42:28.809

Finally, we can choose the option that allows us to write a message. Select **Write Value**.


●●●○○ AT&T 10:42 AM

MyESP32

Status: Connected

[redacted]

Custom Characteristic
UUID: [redacted]
Status: Connected

WRITE VALUE

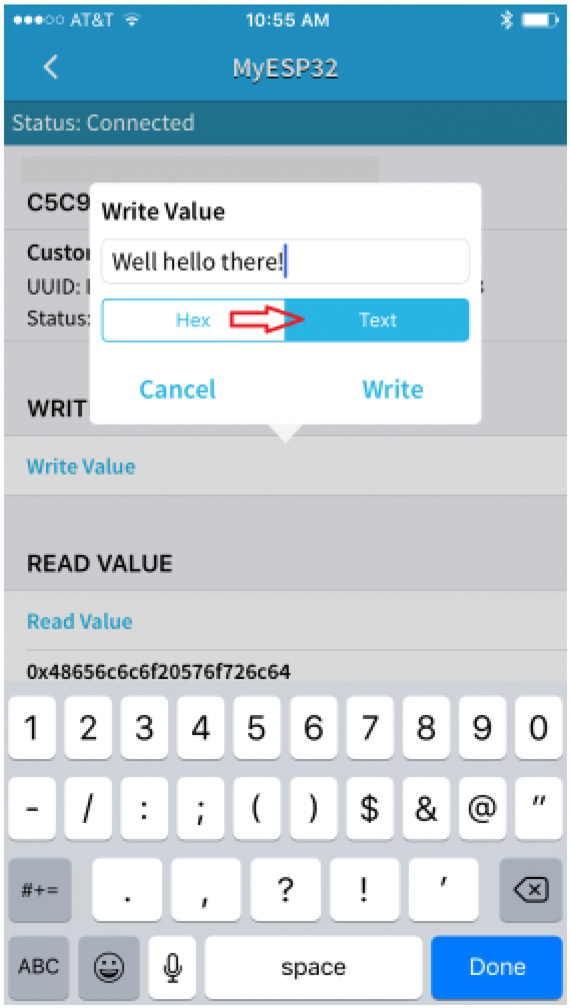
Write Value

READ VALUE

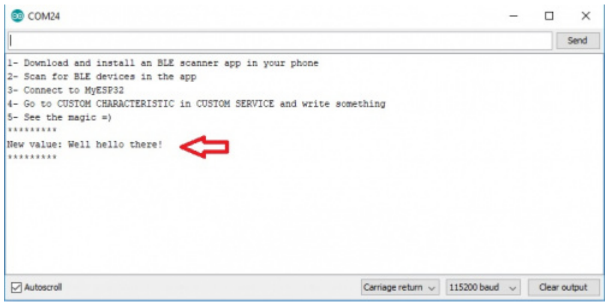
Read Value

0x48656c6c6f20576f726c64
10:42:28.809

Now we can write our message. Make sure you choose **Text**, write yourself a message, and click the **Write** button.



So now what? Go have a look at your serial monitor. You should see "**New value:**" with your message.



```
1- Download and install an BLE scanner app in your phone
2- Scan for BLE devices in the app
3- Connect to MyESP32
4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something
5- See the magic =)
*********
New value: Well hello there!
*********
```

Autoscroll [✓] · Carriage return · 115200 baud

This is just a quick walk through of one of the provided examples. We recommend looking through the rest of the provided samples and playing with the code to see what may work for your application. For more information on Bluetooth technology and how it works, check out our Bluetooth Basics Tutorial.

Resources and Going Further

For more resources related to the ESP32 Thing Plus, check out the links listed here:

- Schematic (PDF)
- Eagle Files (ZIP)
- Datasheets
  - Graphical Datasheet (PDF)
  - ESP32-Wroom 32 (PDF)
  - ESP32 (PDF)
- CP2104 USB Drivers

    - Windows (ZIP)
    - Mac OSX (ZIP)
- ESP32 Thing Plus GitHub Repo
- SFE Product Showcase

Espressif has some great resources built around the ESP32:

- ESP32.com -- Great forums, where you can discuss ESP32 news, get help, or show off your project.
- Espressif ESP32 Resource Page -- A great source for the latest datasheets, reference manuals, and software tools.
- Espressif GitHub Repositories -- Espressif is efforting to be as open-source as possible. Among other things, you'll find repositories for Arduino ESP32 support, SDK's, and even a proof-of-concept NES emulator hosted on their GitHub page.
- ESP-IDF -- IoT Development Framework -- If you want to take your development environment up a step from Arduino, this should be your first step into ESP32 software development. Some notes on the IDF:
    - The IDF is well-documented. Check out the set up guides (for Windows, Mac or Linux) for help getting your environment set up. For help with the API's and data structures, check out esp32.info.
    - There are a handful of example applications, including a **BLE Advertising** example, which works as a proof-of-concept for the ESP32's Bluetooth support.
    - Use the ESP-IDF project template, once you're ready to start creating applications of your own.

For more ESP32 related tutorials, check out the following.

Using Artnet DMX and the ESP32 to Drive Pixels
In this tutorial, we'll find out how to use Resolume Arena, a popular video jockey software, to control custom-made ArtNet DMX fixtures.

How to Load MicroPython on a Microcontroller Board
This tutorial will show you how to load the MicroPython interpreter onto a variety of development boards.

LuMini Ring Hookup Guide
The LuMini Rings (APA102-2020) are the highest resolution LED rings available.

LuMini 8x8 Matrix Hookup Guide
The LuMini 8x8 Matrix are the highest resolution LED matrix available.

If you need some project inspiration, check out some of these IoT-focused projects and get making!

SparkFun Blocks for Intel® Edison - 9 Degrees of Freedom Block

A quick overview of the features of the 9 Degrees of Freedom Block.

SparkFun Blocks for Intel® Edison - ADC V20

A quick overview of the features of the ADC Block.

SparkFun LoRa Gateway 1-Channel Hookup Guide

How to setup and use the LoRa Gateway 1-Channel in Arduino.

Introduction to MQTT

An introduction to MQTT, one of the main communication protocols used with the Internet of Things (IoT).