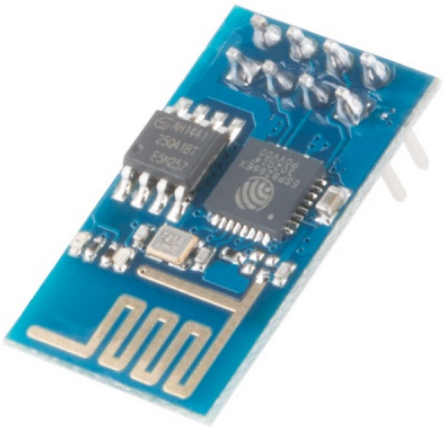
ESP8266 Thing Hookup Guide

Introduction

Over the past year, the ESP8266 has been a growing star among IoT or WiFi-related projects. It's an extremely cost-effective WiFi module, that – with a little extra effort – can be programmed just like any microcontroller. Unfortunately, the ESP8266 has mostly only been available in a tiny, modular form, which, with limited I/O and a funky pin-out, can be difficult to build a project around.



The original ESP8266 WiFi module. Great for piggybacking onto an Arduino, hard to build a project around.

SparkFun's new development board for the ESP8266 breaks out all of the module's pins, and comes equipped with a LiPo charger, power supply, and all of the other supporting circuitry it requires. We lovingly call it the Thing – it's the perfect foundation for your Internet of Things.

Covered in this Tutorial

This tutorial will familiarize you with all things SparkFun Thing. It's split into sections, which cover:

- Hardware Overview – A quick rundown of the Thing's **components and pinout**.
- Powering the Thing – The Thing can be powered through either USB or a LiPo battery.
- Programming the Thing – Interface a 3.3V FTDI Basic with the Thing to program it.
- Hardware Assembly – Tips and recommendations on what to solder to the Thing's I/O pins.
- Installing the ESP8266 Arduino Addon – The Thing can be programmed using Arduino! Just follow the instructions here to install the board definitions.
- Example Sketch: Posting to Phant – Our first example shows how you can use the Thing to post data to data.sparkfun.com.
- Example Sketch: AP Web Server – Set the Thing up as an access point and use it to serve web pages.
- Example Sketch: Goodnight Thing (Sleep Mode) – Put the Thing to sleep to save that sweet battery juice.
- Using the Arduino Addon – There are a few key differences between programming the Thing and any other Arduino board.

Required Materials

To follow along with this tutorial, and get up-and-running with the Thing, you may need a few extra tools and materials. This wishlist includes everything we use in this tutorial to program and use the Thing:

ESP8266 Thing Hookup Guide SparkFun Wish List

SparkFun ESP8266 Thing
WRL-13231
The SparkFun ESP8266 Thing is essentially a breakout and develop…

SparkFun Cerberus USB Cable - 6ft
CAB-12016
You've got the wrong USB cable. It doesn't matter which one you hav…

SparkFun FTDI Basic Breakout - 3.3V
DEV-09873
This is the newest revision of our [FTDI Basic](http://www.sparkfun.co…

(2) Arduino Stackable Header - 10 Pin
PRT-11376
This is a 10-pin female header, with extra long legs -- great for stackin…

Polymer Lithium Ion Battery - 850mAh

PRT-00341

These are very slim, extremely light weight batteries based on the ne…

Suggested Reading

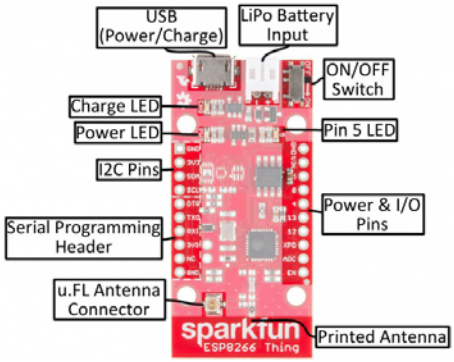
Before continuing on with this tutorial, you may want to familiarize yourself with some of these topics if they're unfamiliar to you:

- How to Power a Project
- Logic Levels
- Serial Communication
- How to Solder

Hardware Overview

The ESP8266 Thing is a relatively simple board. The pins are broken out to two parallel, breadboard-compatible rows. USB and LiPo connectors at the top of the board provide power – controlled by the nearby ON/OFF switch. And LEDs towards the inside of the board indicate power, charge, and status of the IC.

Here's a quick overview of the Thing's main components:



The Pinout

The Thing's I/O headers can be broken down into three sections:

Serial Programming Header

This six-pin header will be the main point of contact between the Thing and your development computer. The pinout of this header matches the extremely common "FTDI header." That means you can interface it with either a 3.3V FTDI Basic or a 3.3V I/O FTDI Cable to program and debug the Thing.

For a quick breakdown of the pins on this header, consult the table below. If a pin is directly tied to an ESP8266 I/O, it'll be noted:

Pin Label	ESP8266 I/O #	Notes
DTR		Performs auto-reset, and puts the ESP8266 into bootloader mode. Connects through a capacitor to RESET, and a buffer to the ESP8266's GPIO0.
TXO	7	ESP8266 UART1 data output.
RXI	8	ESP8266 UART1 data input.

3V3		By default, this pin does not supply the ESP8266 directly (a jumper on the back can change that).
NC		Not connected to anything on the Thing.
GND		Ground (0V).

I²C Header

I²C is a very popular communication protocol in the embedded world. Whether you want to hook the Thing up to a motion sensor, light sensor, digital-to-analog converter, or OLED display, I²C is often the protocol of choice.

This header includes four pins – all that should be required to connect an I²C device up to the Thing.

Pin Label	ESP8266 I/O #	Notes
GND		Ground (0V).
3V3		3.3V
SDA	2	Can either be used as ESP8266 GPIO2 or I²C serial data (SDA).
SCL	14	Can either be used as ESP8266 GPIO14 or I²C serial clock (SCL). Also used as the SPI clock (SCLK).

This pinout matches that of most of our I²C-based breakout boards, so you can piggyback them right on top of the Thing.

If you need the extra I/O, instead of I²C, the SDA and SCL pins can be used as GPIO 2 and 14 respectively. The SCL pin also serves as the clock (SCLK) for the ESP8266's SPI interface.

General I/O Header

The rest of the power, control, and I/O pins are broken out on the other side of the board. They are:

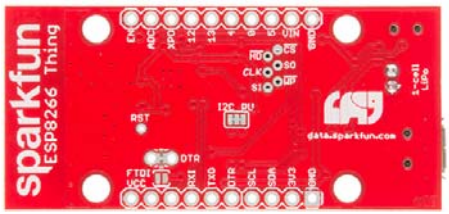
Pin Label	ESP8266 I/O #	Notes
GND		Ground (0V).
VIN		USB connected: ~4.5V output LiPo connected (no USB): ~3.7V output No supply: Can be used as a voltage supply input to the 3.3V regulator.
5	5	This pin is also tied to the on-board LED.
0	0	
4	4	
13	13	Hardware SPI MOSI
12	12	Hardware SPI MISO

XPD	16	Can be connected to reset to set the ESP8266 into deep sleep mode.
ADC	A0	A 10-bit ADC with a **maximum voltage of 1V**.
EN		ESP8266 enable pin. HIGH = on, LOW = off. Pulled HIGH on-board.

What happened to the rest of the GPIO pins? Why the eclectic pin-numbering scheme? We're taking what the ESP8266 gives us. Unfortunately, most of the remaining GPIO are connected to the on-board SPI flash memory IC, which stores the ESP8266's program memory and potentially other data.

Back of the Thing

Flipping the Thing over revels a few trace jumpers and test points, which you may find handy for your application. (Plus a friendly Phant.io logo, to remind you about our data storage service on data.sparkfun.com.)



Jumpers

Jumper Label	Default Setting	Notes
DTR	Closed	Allows for auto-reset while programming the ESP8266, but makes serial debugging difficult.
I2C PU	Closed	Connects 10kΩ pull-up resistors to the SDA and SCL pins.
FTDI VCC	Open	Connects the 3V3 pin on the serial header directly to the ESP8266's 3.3V supply.

Of these jumpers, the *DTR* one is the most commonly modified. The DTR output of the FTDI Basic is used for two purposes: to reset the ESP8266 and pull GPIO0 low (putting the chip in bootloader mode). Keeping this jumper closed enables programming, but makes debugging via the Serial Monitor difficult, as the board will reset into bootloader mode whenever the terminal opens. Using and modifying this jumper is discussed later in this tutorial.

The *FTDI_VCC* jumper defaults to open to ensure that, if a 3.3V Logic (5V power) FTDI Cable is used to program the Thing, 5V isn't accidentally delivered to the IC. Also, most 3.3V FTDI boards don't have a lot of juice to supply on the 3.3V bus (they often supply about 50mA max).

Test Points

These pins are made available just in case they become necessary to your project. The six pins bundled up together are connected to the Thing's on-board SPI flash memory, but if you really need the extra GPIO, or want to experiment with the pins, they're available.

The **RST** pin is more useful, but we didn't leave room to break it out – at least not directly. *RST* is tied through a 0.1µF capacitor to the *DTR* pin, to allow for automatic reset during programming. For many applications that need the *RST* pin, toggling the *DTR* pin works as well. Putting the ESP8266 into deep sleep is one such application.

Selecting the Antenna

The Thing's default WiFi antenna is a PCB trace antenna based on this TI app note. It's cost-effective and actually works really well!

If you need to connect a more sensitive antenna to the chip, a U.FL connector is also available on the board, but isn't connected by default to the ESP8266's antenna pin. To connect this antenna to the chip, you'll need to heat up the 0Ω resistor and rotate it 90°:

An (ugly, uncleaned) resistor swapped from printed antenna to U.FL antenna.

A soldering iron, pair of tweezers, (2) steady hands, and good set of eyes should be enough to accomplish the task.

Why Are There Unpopulated Parts?

We initially set out to make the Thing a secure, common-sensor base station. The empty pads you see are landing spots for three unique IC's:

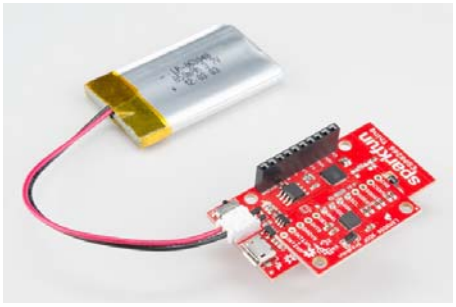
- ATECC108A – A "full turnkey Elliptic Curve Digital Signature Algorithm (ECDSA) engine", which can be used for unique serial numbers, hashing, key storage, or random numbers. A great start to securing your IoT project!
- TMP102 Temperature Sensor – A simple, 12-bit, digital temperature sensor.
- TSL2561 Light Sensor – A nifty luminosity/light sensor.
- Plus a few footprints for decoupling capacitors.

After a late change of heart, we decided to keep the board as low cost as possible (that's the ESP8266's best feature!), while leaving the option for later expansion. The pads are still there. If you want to add any of these components, hopefully all you should need is a hot air station (~~maybe~~ probably not a Heaterizer) and some tweezers.

Powering the Thing

The Thing provides for two methods of power delivery: USB or LiPo. The USB connector on the Thing is of the Micro-B variety. A micro-B cable plugged into either a computer USB port or a 5V USB Wall Wart can power the Thing.

Any of our single-cell LiPo batteries will also work to power the Thing – they all have the same 2-pin JST connector.



Add an 850mAh LiPo and an LSM9DS0 9DoF IMU to the Thing, to create an IoT motion sensor.

If both USB and LiPo are connected to the Thing, it'll take power from the USB port and charge the LiPo battery at up to 500mA.

Electrical Characteristics

The ESP8266's maximum voltage is 3.6V, so the Thing has an onboard 3.3V regulator to deliver a safe, consistent voltage to the IC. That means the ESP8266's I/O pins also run at 3.3V, you'll need to level shift any 5V signals running into the IC.

The input to this regulator can either be delivered by USB, LiPo battery, or through the *VIN* pin.

> **Max Input Voltage**: If you supply power to the board through the VIN, make sure the voltage does not exceed **6V**. That's the maximum input voltage of the AP2112K-3.3V regulator the board uses.

Alternatively, if you have an external, regulated, supply you'd like to deliver directly to the ESP8266, you can supply that voltage through the *3V3* pin (on the I^2C header). While this voltage doesn't have to be 3.3V, it must be within the range of **1.7-3.6V**.

Current Ratings

On average, the Thing pulls about 80mA. WiFi transmits and receives can momentarily increase that draw. Here's a table, transcribed from the ESP8266 datasheet, with some of the more common current characteristics.

Parameter	Typical	Max	Unit
Transmit 802.11b (1 Mbps)	215		mA
Transmit 802.11b (11 Mbps)	197		mA
Transmit 802.11g (54 Mbps)	145		mA
Transmit 802.11n	135		mA
Receive 802.11b	60		mA
Receive 802.11g	60		mA
Receive 802.11n	62		mA
Standby	0.9		mA

Deep Sleep	10		µA
Maximum I/O Pin Drive Capability		12	mA

If your application requires maximum battery life, you'll likely need to make liberal use of the ESP8266's deep sleep functionality. That'll be covered later in this tutorial.

Programming the Thing

The ESP8266 has a built-in serial bootloader, which allows for easy programming and re-programming. You don't need a specialized, expensive programmer – just a simple, USB-to-Serial converter.

We use a 3.3V FTDI Basic to program the Thing, but other serial converters with 3.3V I/O levels should work (e.g. FTDI SmartBasic, FTDI Cable 5V VCC-3.3V I/O, FT231X Breakout. The converter does need a DTR line in addition to the RX and TX pins.

The FTDI Basic's 6-pin header matches up exactly to the Thing's 6-pin serial port header. To set up for programming, simply connect the FTDI directly to this port – take care to match up the DTR and GND pins!

If you're short on USB ports, the SparkFun Cerberus Cable might be just what you need.

If you solder female headers to the Thing, plugging a 6-pin row of right-angle male headers between the FTDI and header helps create a temporary programming interface.

Hardware Assembly

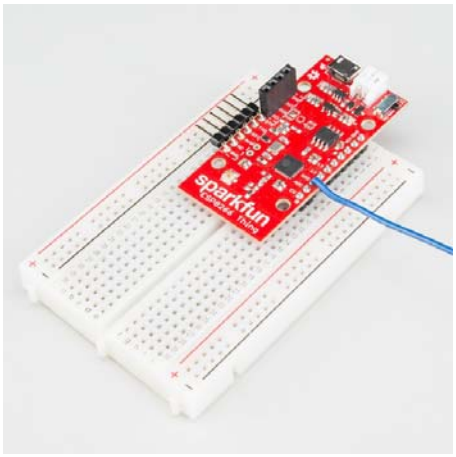
Oh. We're getting ahead of ourselves. To connect the FTDI programmer to your Thing you'll need to solder *something* to the Thing.

If you've never soldered before, this is a great time to start! These solder points are easy, through-hole pins, check out our How to Solder - Through-hole Soldering for help getting started.

What, exactly, you solder to the board depends both on how you'll use it in your project, and how you'll interface it with the programmer. When it comes to selecting a header (or wire) to solder, there are a variety of options. We've tried a lot of them with the Thing:

10-pin Stackable Headers make it convenient to both breadboard the Thing and jumper wire out of it.

Or you can mix and match headers to best fit your needs. Right-angle male headers may help to interface between the FTDI and the Thing. Straight male headers are a good choice for low-profile connections. Straight female headers may help with connecting to I^2C sensors. And, of course, wire can be soldered to any of the pins that have a long way to connect to something.



Once you've soldered up at least the programming port, you're ready to load some code onto the Thing. Let's blink some LEDs and IoT (Internet our Thing).

Installing the ESP8266 Arduino Addon

There are a variety of development environments that can be equipped to program the ESP8266. You can go with a simple Notepad/gcc setup, or fine-tune an Eclipse environment, use a virtual machine provided by Espressif, or come up with something of your own.

Fortunately, the amazing ESP8266 community recently took the IDE selection a step further by creating an Arduino addon. If you're just getting started programming the ESP8266, this is the environment we recommend beginning with, and the one we'll document in this tutorial.

This ESP8266 addon for Arduino is based on the amazing work by Ivan Grokhotkov and the rest of the ESP8266 community. Check out the ESP8266 Arduino GitHub repository for more information.

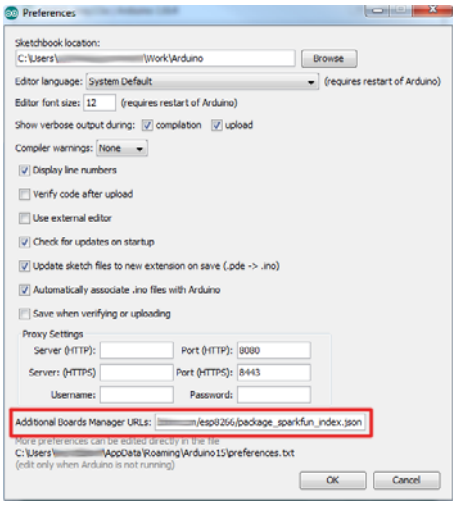
Installing the Addon With the Arduino Boards Manager

With the release of Arduino 1.6.4, adding third party boards to the Arduino IDE is easily achieved through the new board manager. If you're running an older version of Arduino (1.6.3 or earlier), we recommend upgrading now. As always, you can download the latest version of Arduino from arduino.cc.

To begin, we'll need to update the board manager with a custom URL. Open up Arduino, then go to the Preferences (**File** > **Preferences**). Then, towards the bottom of the window, copy this URL into the "Additional Board Manager URLs" text box:

```
https://raw.githubusercontent.com/sparkfun/Arduino_Boards/esp8266/IDE_Board_Manager/package_sparkfun_index.json
```

If you already have a URL in there, and want to keep it, you can separate multiple URLs by placing a comma between them.

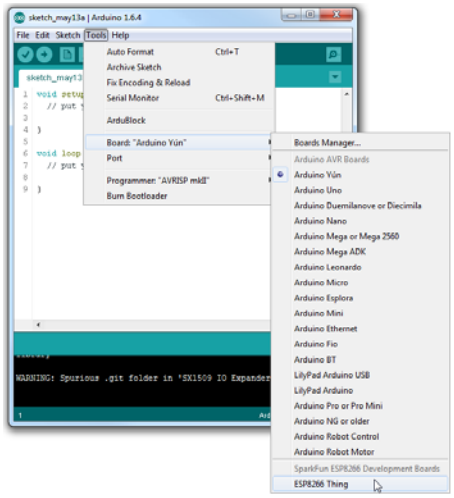


Hit OK. Then navigate to the Board Manager by going to **Tools** > **Boards** > **Boards Manager**. There should be a couple new entries in addition to the standard Arduino boards. Look for **SparkFun ESP8266 Boards**. Click on that entry, then select **Install**.

The board definitions and tools for the ESP8266 Thing include a whole new set of gcc, g++, and other reasonably large, compiled binaries, so it may take a few minutes to download and install (the archived file is ~110MB). Once the installation has completed, an Arduino-blue "INSTALLED" will appear next to the entry.

Selecting the ESP8266 Thing Board

With the Board addon installed, all that's left to do is select "ESP8266 Thing" from the **Tools** > **Boards** menu.



Then select your FTDI's port number under the **Tools** > **Port** menu.

Upload Blink

To verify that everything works, try uploading the old standard: Blink. Instead of blinking pin 13, like you may be used to though, toggle pin 5, which is attached to the onboard LED.

```
#define ESP8266_LED 5

void setup()
{
  pinMode(ESP8266_LED, OUTPUT);
}

void loop()
{
  digitalWrite(ESP8266_LED, HIGH);
  delay(500);
  digitalWrite(ESP8266_LED, LOW);
  delay(500);
}
```

If the upload fails, first make sure the ESP8266 Thing is turned on – the red "PWR" LED should be illuminated.

There are still some bugs to be fleshed out of the esptool, sometimes it may take a couple tries to successfully upload a sketch. If you continue to fail, try turning the ESP8266 Thing on then off, or unplug then replug the FTDI in. If you still have trouble, get in touch with our amazing tech support team.

Example Sketch: Posting to Phant

Our main goal for creating the ESP8266 Thing was finding a good platform to interact with our free online data storage service: data.sparkfun.com.

Here's a simple example sketch that posts four values to a test stream. Feel free to use that stream temporarily to make sure your Thing is working (don't abuse it please!).

Copy the code below, or download the example sketch.

Before uploading your code to the Thing, make sure you modify the `WiFiSSD` and `WiFiPSK` variables, setting them to the SSID and password of your WiFi network. The rest of the sketch should just work.

```
// Include the ESP8266 WiFi library. (Works a lot like the
// Arduino WiFi library.)
#include <ESP8266WiFi.h>
// Include the SparkFun Phant library.
#include <Phant.h>

//////////////////////
// WiFi Definitions //
//////////////////////
const char WiFiSSID[] = "WiFi_Network";
const char WiFiPSK[] = "WiFi_Password";

//////////////////////
// Pin Definitions //
//////////////////////
const int LED_PIN = 5; // Thing's onboard, green LED
const int ANALOG_PIN = A0; // The only analog pin on the Thing
const int DIGITAL_PIN = 12; // Digital pin to be read

////////////////
// Phant Keys //
////////////////
const char PhantHost[] = "data.sparkfun.com";
const char PublicKey[] = "wpvZ9pE1qbFJAjaGd3bn";
const char PrivateKey[] = "wzeB1z0xWNt1YJX27xdg";

/////////////////
// Post Timing //
/////////////////
const unsigned long postRate = 30000;
unsigned long lastPost = 0;

void setup()
{
  initHardware();
  connectWiFi();
  digitalWrite(LED_PIN, HIGH);
}

void loop()
{
  if (lastPost + postRate <= millis())
  {
    if (postToPhant())
      lastPost = millis();
    else
      delay(100);
  }
}

void connectWiFi()
{
  byte ledStatus = LOW;

  // Set WiFi mode to station (as opposed to AP or AP_STA)
  WiFi.mode(WIFI_STA);

  // WiFI.begin([ssid], [passkey]) initiates a WiFI connection
  // to the stated [ssid], using the [passkey] as a WPA, WPA2,
  // or WEP passphrase.
  WiFi.begin(WiFiSSID, WiFiPSK);

  // Use the WiFi.status() function to check if the ESP8266
  // is connected to a WiFi network.
```

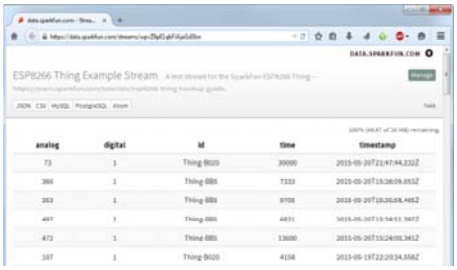
```
  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink the LED
    digitalWrite(LED_PIN, ledStatus); // Write LED high/low
    ledStatus = (ledStatus == HIGH) ? LOW : HIGH;

    // Delays allow the ESP8266 to perform critical tasks
    // defined outside of the sketch. These tasks include
    // setting up, and maintaining, a WiFi connection.
    delay(100);
    // Potentially infinite loops are generally dangerous.
    // Add delays -- allowing the processor to perform other
    // tasks -- wherever possible.
  }
}

void initHardware()
{
  Serial.begin(9600);
  pinMode(DIGITAL_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  // Don't need to set ANALOG_PIN as input,
  // that's all it can be.
}

int postToPhant()
{
  // LED turns on when we enter, it'll go off when we
  // successfully post.
  digitalWrite(LED_PIN, HIGH);

  // Declare an object from the Phant library - phant
  Phant phant(PhantHost, PublicKey, PrivateKey);

  // Do a little work to get a unique-ish name. Append the
  // last two bytes of the MAC (HEX'd) to "Thing-":
  uint8_t mac[WL_MAC_ADDR_LENGTH];
  WiFi.macAddress(mac);
  String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
                 String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
  macID.toUpperCase();
  String postedID = "Thing-" + macID;

  // Add the four field/value pairs defined by our stream:
  phant.add("id", postedID);
  phant.add("analog", analogRead(ANALOG_PIN));
  phant.add("digital", digitalRead(DIGITAL_PIN));
  phant.add("time", millis());

  // Now connect to data.sparkfun.com, and post our data:
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(PhantHost, httpPort))
  {
    // If we fail to connect, return 0.
    return 0;
  }
  // If we successfully connected, print our Phant post:
  client.print(phant.post());

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

```
    //Serial.print(line); // Trying to avoid using serial
  }

  // Before we exit, turn the LED off.
  digitalWrite(LED_PIN, LOW);

  return 1; // Return success
}
```

After loading the code onto your Thing, it will begin to run. The status LED connected to pin 5 will initially blink at about 2 Hz. After the Thing connects to your network, the green LED will turn solid and post to the test stream. At that point the LED will go dark, only blinking every 30s-or-so as the Thing posts to Phant again.

If the LED never stops blinking, your Thing is probably having trouble connecting to the WiFi network. Make sure the SSID and PSK variables are set correctly.

Four values are posted to the Phant stream: the reading from the ADC pin, a digital reading from pin 12, the Thing's ID ("Thing" appendd with the last two MAC bytes), and a time variable loaded from the `millis()` function. Load up the test stream to check for your Thing's signature there!



Example screenshot from our communal data.sparkfun.com stream.

Read through the comments in the code to get a line-by-line breakdown of what's going on in the sketch.

Example Sketch: AP Web Server

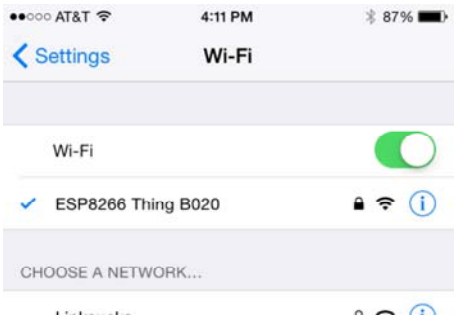
Not only can the ESP8266 connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This example demonstrates how to turn the ESP8266 into an **access point** (AP), and serve up web pages to any connected client.

Copy and paste the code from below, or download it here.

```
#include <ESP8266WiFi.h>

//////////////////////
// WiFi Definitions //
//////////////////////
const char WiFiAPPSK[] = "sparkfun";

//////////////////////
// Pin Definitions //
//////////////////////
const int LED_PIN = 5; // Thing's onboard, green LED
const int ANALOG_PIN = A0; // The only analog pin on the Thing
const int DIGITAL_PIN = 12; // Digital pin to be read

WiFiServer server(80);

void setup()
{
  initHardware();
  setupWiFi();
  server.begin();
}

void loop()
{
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(req);
  client.flush();

  // Match the request
  int val = -1; // We'll use 'val' to keep track of both the
                // request type (read/set) and value if set.
  if (req.indexOf("/led/0") != -1)
    val = 0; // Will write LED low
  else if (req.indexOf("/led/1") != -1)
    val = 1; // Will write LED high
  else if (req.indexOf("/read") != -1)
    val = -2; // Will print pin reads
  // Otherwise request will be invalid. We'll say as much in H
TML

  // Set GPIO5 according to the request
  if (val >= 0)
    digitalWrite(LED_PIN, val);

  client.flush();

  // Prepare the response. Start with the common header:
  String s = "HTTP/1.1 200 OK\r\n";
  s += "Content-Type: text/html\r\n\r\n";
  s += "<!DOCTYPE HTML>\r\n<html>\r\n";
  // If we're setting the LED, print out a message saying we d
id
  if (val >= 0)
  {
    s += "LED is now ";
    s += (val)?"on":"off";
```

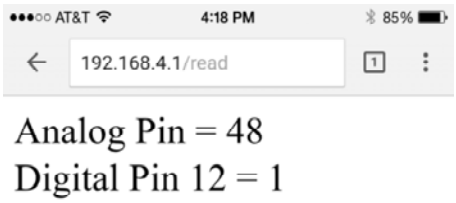
```
  }
  else if (val == -2)
  { // If we're reading pins, print out those values:
    s += "Analog Pin = ";
    s += String(analogRead(ANALOG_PIN));
    s += "<br>"; // Go to the next line.
    s += "Digital Pin 12 = ";
    s += String(digitalRead(DIGITAL_PIN));
  }
  else
  {
    s += "Invalid Request.<br> Try /led/1, /led/0, or /read.";
  }
  s += "</html>\n";

  // Send the response to the client
  client.print(s);
  delay(1);
  Serial.println("Client disonnected");

  // The client will actually be disconnected
  // when the function returns and 'client' object is detroyed
}

void setupWiFi()
{
  WiFi.mode(WIFI_AP);

  // Do a little work to get a unique-ish name. Append the
  // last two bytes of the MAC (HEX'd) to "Thing-":
  uint8_t mac[WL_MAC_ADDR_LENGTH];
  WiFi.softAPmacAddress(mac);
  String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
                 String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
  macID.toUpperCase();
  String AP_NameString = "ESP8266 Thing " + macID;

  char AP_NameChar[AP_NameString.length() + 1];
  memset(AP_NameChar, AP_NameString.length() + 1, 0);

  for (int i=0; i<AP_NameString.length(); i++)
    AP_NameChar[i] = AP_NameString.charAt(i);

  WiFi.softAP(AP_NameChar, WiFiAPPSK);
}

void initHardware()
{
  Serial.begin(115200);
  pinMode(DIGITAL_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  // Don't need to set ANALOG_PIN as input,
  // that's all it can be.
}
```

After uploading this sketch, find another device that you can connect to a WiFi network – phone, laptop, etc. Look for a network called “Thing-XXXX”, where XXXX is the last 2 bytes of the Thing’s MAC address.



The sketch sets the network's password to "sparkfun".

After connecting to your Thing's AP network, load up a browser and point it to `192.168.4.1/read`. The Thing should serve up a web page showing you its ADC and digital pin 12 readings:



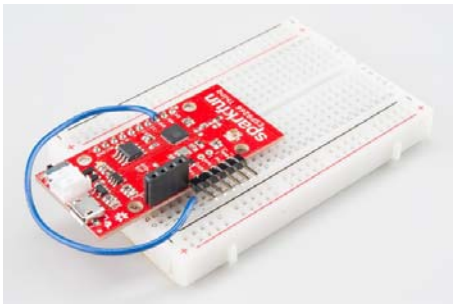
After that, give `192.168.4.1/led/0` and `192.168.4.1/led/1` a try, and keep an eye on the Thing's green LED while you do.

As always, check through the code comments to get a line-by-line breakdown of what's going on.

Example Sketch: Goodnight Thing (Sleep Mode)

One of the Thing's most unique features is the integrated support for LiPo batteries. Unfortunately, the ESP8266 is still a pretty power hungry device. If you want your project to run off a battery for more than a few hours, you have two options: get a *huge* battery or cleverly put the Thing to sleep.

To use the Thing's sleep capability, you'll need to wire the *XPD* pin up to the ESP8266's reset line. *RST* isn't broken out, but the *DTR* pin will work for our application as well.



Wire up from the XPD pin to DTR to set up sleep-ability.

Watch out! The ESP8266 can't be programmed while the *XPD* pin is connected to *DTR*. Make sure you disconnect the two pins before trying to upload a sketch.

After you tell the ESP8266 to sleep, it'll wait a specified number of microseconds, then trigger the XPD pin to toggle the reset line. When the ESP8266 wakes up, it'll begin back at the start of the sketch.

Let's riff off the first example in this tutorial – posting to Phant – but take advantage of sleeping to greatly increase the battery life. Here's some example code (or click here to download):

Once again, make sure you modify the `WiFiSSID` and `WiFiPSK` variables near the top of the sketch.

```
// Include the ESP8266 WiFi library. (Works a lot like the
// Arduino WiFi library.)
#include <ESP8266WiFi.h>
// Include the SparkFun Phant library.
#include <Phant.h>

//////////////////////
// WiFi Definitions //
//////////////////////
const char WiFiSSID[] = "WiFi_Network";
const char WiFiPSK[] = "WiFi_Password";

//////////////////////
// Pin Definitions //
//////////////////////
const int LED_PIN = 5; // Thing's onboard, green LED
const int ANALOG_PIN = A0; // The only analog pin on the Thing
const int DIGITAL_PIN = 12; // Digital pin to be read

////////////////
// Phant Keys //
////////////////
const char PhantHost[] = "data.sparkfun.com";
const char PublicKey[] = "wpvZ9pE1qbFJAjaGd3bn";
const char PrivateKey[] = "wzeB1z0xWNt1YJX27xdg";

// Time to sleep (in seconds):
const int sleepTimeS = 30;

void setup()
{
  initHardware();
  connectWiFi();
  digitalWrite(LED_PIN, HIGH);
  while (postToPhant() != 1)
  {
    delay(100);
  }
  digitalWrite(LED_PIN, LOW);
  // deepSleep time is defined in microseconds. Multiply
  // seconds by 1e6
  ESP.deepSleep(sleepTimeS * 1000000);
}

void loop()
{
}

void connectWiFi()
{
  byte ledStatus = LOW;

  // Set WiFi mode to station (as opposed to AP or AP_STA)
  WiFi.mode(WIFI_STA);
  // WiFI.begin([ssid], [passkey]) initiates a WiFI connection
  // to the stated [ssid], using the [passkey] as a WPA, WPA2,
  // or WEP passphrase.
  WiFi.begin(WiFiSSID, WiFiPSK);

  // Use the WiFi.status() function to check if the ESP8266
  // is connected to a WiFi network.
  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink the LED
```

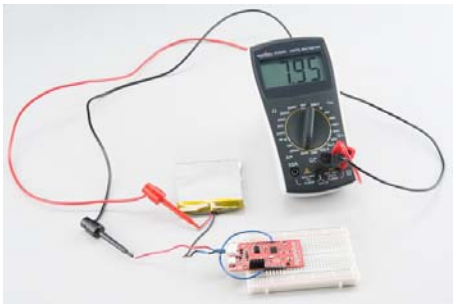
```
    digitalWrite(LED_PIN, ledStatus); // Write LED high/low
    ledStatus = (ledStatus == HIGH) ? LOW : HIGH;

    // Delays allow the ESP8266 to perform critical tasks
    // defined outside of the sketch. These tasks include
    // setting up, and maintaining, a WiFi connection.
    delay(100);
    // Potentially infinite loops are generally dangerous.
    // Add delays -- allowing the processor to perform other
    // tasks -- wherever possible.
  }
}

void initHardware()
{
  Serial.begin(9600);
  pinMode(DIGITAL_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  // Don't need to set ANALOG_PIN as input,
  // that's all it can be.
}

int postToPhant()
{
  // LED turns on when we enter, it'll go off when we
  // successfully post.
  digitalWrite(LED_PIN, HIGH);

  // Declare an object from the Phant library - phant
  Phant phant(PhantHost, PublicKey, PrivateKey);

  // Do a little work to get a unique-ish name. Append the
  // last two bytes of the MAC (HEX'd) to "Thing-":
  uint8_t mac[WL_MAC_ADDR_LENGTH];
  WiFi.macAddress(mac);
  String macID = String(mac[WL_MAC_ADDR_LENGTH - 2], HEX) +
                 String(mac[WL_MAC_ADDR_LENGTH - 1], HEX);
  macID.toUpperCase();
  String postedID = "Thing-" + macID;

  // Add the four field/value pairs defined by our stream:
  phant.add("id", postedID);
  phant.add("analog", analogRead(ANALOG_PIN));
  phant.add("digital", digitalRead(DIGITAL_PIN));
  phant.add("time", millis());

  // Now connect to data.sparkfun.com, and post our data:
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(PhantHost, httpPort))
  {
    // If we fail to connect, return 0.
    return 0;
  }
  // If we successfully connected, print our Phant post:
  client.print(phant.post());

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

```
  // Before we exit, turn the LED off.
  digitalWrite(LED_PIN, LOW);

  return 1; // Return success
}
```

This sketch accomplishes the same feat as that in our first example – it posts some data to a data.sparkfun.com stream every 30 seconds. But there's one huge difference: sleep.

Notice there's nothing in the `loop()`. The program halts when the `ESP.deepSleep(30000000)` is called. After 30 seconds, when the ESP8266 wakes up, it'll start running code back at the beginning of `setup()`.

If you put a multimeter in-line to measure the current draw, the Thing would pull about 80mA for the ~5 seconds it takes to connect and post to Phant. Then, while it sleeps for about 30 seconds, the current draw would be around 8mA. An average of about 18mA per second.



In deep sleep mode, the Thing pulls about 8mA (that's mostly the power LED).

Most of the current draw in sleep mode is from the power LED indicator – if you want to save even more juice, you may consider removing the LED (or current-limiting resistor) or even cutting a trace. (Hint: cut the trace running through middle of the 'R' in "PWR".)

Using the Arduino Addon

If you've used Arduino in the past, there will be some new programming schemes to get used to in ESP8266 land.

Pin Mappings

As with any other Arduino, the pin mappings printed on the board match the pin you read or write to. The SDA and SCL pins can be referenced as 2 and 14 respectively.

There's only one analog input pin, labeled *ADC*. To read the *ADC* pin, make a function call to `analogRead(A0)`. Remember that this pin has a weird maximum voltage of 1V – you'll get a 10-bit value (0-1023) proportional to a voltage between 0 and 1V.

Yielding

This is one of the most critical differences between the ESP8266 and a more classical Arduino microcontroller. The ESP8266 runs a lot of utility functions in the background – keeping WiFi connected, managing the TCP/IP stack, and performing other duties. Blocking these functions from running can cause the ESP8266 to crash and reset itself. To avoid these mysterious resets, **avoid long, blocking loops in your sketch**.

If you have a long loop in your sketch, you can **add a `delay([milliseconds])`** call within, to allow the critical background functions to execute. The ESP8266's `delay()` funciton, while of course delaying for a set number of milliseconds, also makes a quick call to the background functions.

The amazing creators of the ESP8266 Arduino libraries also implemented a **`yield()`** function, which calls on the background functions to allow them to do their thing. As an example, if your sketch is waiting for someone to press a button attached to pin 12, creating a loop like this will keep the ESP8266 from crashing:

```
pinMode(12, INPUT_PULLUP); // Set pin 12 as an input w/ pull-u
p
while (digitalRead(12) == HIGH) // While pin 12 is HIGH (not a
ctivated)
    yield(); // Do (almost) nothing -- yield to allow ESP8266
background functions
Serial.println("Button is pressed!"); // Print button pressed
message.
```

ESP8266WiFi Class

This is the ESP8266, so the WiFi class will probably be included in just about every sketch there is. If you've used the Arduino WiFi library before, the ESP8266 WiFi library will be very similar, there's just a few key differences:

- To **include the ESP8266 WiFi library** call `#include <ESP8266WiFi.h>` *not* `<WiFi.h>`.
- To **connect** to a network, like the normal WiFi library, call `WiFi.begin(NetworkSSID, NetworkPassword)`. You can also set the ESP8266 up as a WiFi access point by calling `WiFi.softAP(AP_SSID, AP_Password)`.
- To set the ESP8266's **mode**, which can be access point (AP), station (STA), or combo (the ESP8266 can do both at the same time!), call `WiFi.setMode([mode])` with either `WIFI_AP`, `WIFI_STA`, or `WIFI_STA_AP` as the parameter.

The examples earlier in this tutorial should have demonstrated all of these differences.

Libraries Available/Not Available and the Differences

A lot of the core Arduino libraries have been re-written to work for the ESP8266, including:

- **Wire** – The ESP8266 should work with any I²C sensor you can throw at it – just use the same Wire API calls you're used to. There are a few differences:
  - Pin definition: The ESP2866 doesn't actually have any hardware I²C pins – those labeled on the Thing are the default, but you can actually use any two pins as SDA and SCL. Calling `Wire.begin()` will assume pins 2 and 14 are SDA and SCL, but you can manually set them to any other pin by calling `Wire.begin([SDA], [SCL])`.
- **SPI** – The ESP8266 Thing can control an SPI bus using function calls made standard by the Arduino SPI library.
  - An additional function to set the frequency – `SPI.setFrequency([frequency])` – is added. You may need to call that in your setup to slow the clock down from its default

value. For example, `SPI.setFrequency(1000000)` will set the SPI clock to 1MHz.
- The MISO, MOSI, and SCLK SPI pins are hard-coded and can't be moved, they are:

Pin Number	SPI Function
12	MISO
13	MOSI
14 (SCL)	SCLK
15	CS

Using the Serial Monitor

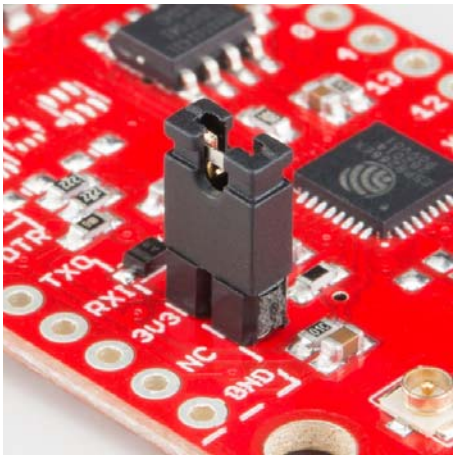
GPIO0 – while perfectly capable as a digital I/O – serves a secondary purpose as a bootload/run-mode controller. When the ESP8266 boots up, it looks at GPIO0's value to either enter the bootloader or start running the current program:

GPIO0 Value	ESP8266 Mode
HIGH (3.3V)	Run Program
LOW (0V)	Bootloader

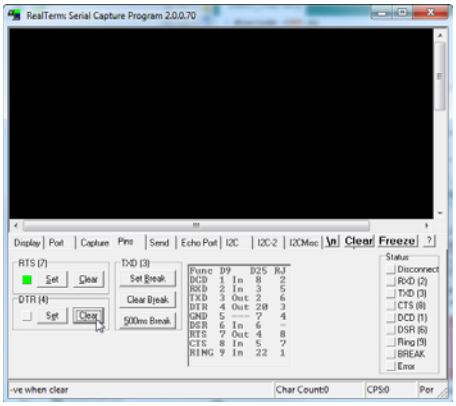
To make it easy to program the ESP8266, we've tied GPIO0 to DTR (along with RST). When your programmer begins to upload a sketch, it'll pull DTR low, in turn setting GPIO0 low and making the ESP8266 enter bootloader mode.

Unfortunately, when you open a serial terminal, DTR usually goes low again. So every time you open the Arduino serial monitor, it'll cause the ESP8266 to enter bootloader mode, instead of run-program mode. If you open up the serial monitor, and all you see is a line of gibberish, you've probably booted the ESP8266 into bootloader mode.

There are a few ways around this. We've added the DTR jumper on the bottom of the board. You can cut the trace on the back and install a 2-pin male header combined with a 2-pin jumper. If the jumper is present, the board will be able to be programmed. Removing the jumper will enable serial terminal mode.



Or you can find a serial terminal program that allows control of the DTR pin directly. RealTerm allows for this control – navigate to the “Pins” tab, and click “Clear” next to “DTR.”



Unfortunately, this Windows-only solution is the only terminal program we’ve found so far with such control. Your best bet may be to try to avoid serial debugging whenever possible – that’s what LED’s are for, right? (Tongue only kind-of in cheek.)

Resources & Going Further

An astoundingly awesome community has grown around the ESP8266. We owe them big time for the amazing Arduino addon they’ve cooperatively built. For all of your ESP8266 needs, we recommend checking out the esp8266.com Community Forum. In addition to that, here are a few ESP8266-related resources we’ve found incredibly helpful:

- ESP8266 GitHub User Repos – Tons of incredible tools can be found here. From Crosstool (to compile your own Xtensa GCC, G++, etc.) to the ESP8266 Arduino GitHub Repo
- ESP8266 Community Wiki – Related to the community forum, there’s a good amount of information available in this wiki.
- NodeMCU Firmware and the NodeMCU Flasher – NodeMCU is a popular firmware for the ESP8266. It implements a LUA-based interpreter on the ESP8266 MCU.
- Espressif Board Forums – Espressif, the manufacturers of the ESP8266, have a forum of their own. You can sometimes find updated software development kit downloads, or other helpful links here.
- Espressif GitHub Repos – Espressif is also somewhat active on GitHub. They host a couple versions of the SDK here.

The ESP8266 Thing is open source hardware! If you need, or just want to look at, the PCB design files, you can find them in our ESP8266 Thing GitHub repository.

Going Further

Need a little project inspiration, now that you’ve got your ESP8266 Thing up-and-running? Maybe some of these related SparkFun tutorials will help spur some ideas:

Are You Okay? Widget

Weather Station Wirelessly Connected to Wunderground

Build your own open source, official Wunderground weather station that updates every 10 seconds over Wifi via an Electric Imp.

Use an Electric Imp and accelerometer to create an "Are You OK" widget. A cozy piece of technology your friend or loved one can nudge to let you know they're OK from half-a-world away.

Pushing Data to Data.SparkFun.com

A grab bag of examples to show off the variety of routes your data can take on its way to a Data.SparkFun.com stream.

Using AT&T's M2X With the CC3000

A set of tutorials and examples to show how to connect an Arduino and CC3000 to AT&T's M2X data streams. We show how to post, fetch, and delete data. The final lesson is controlling an LED from an M2X stream.

With it’s deep sleep ability, the Thing is a great foundation for a WiFi-based weather station, or a friendly, huggable, interactive plushy.

Стандарт Электрон Связь

Мы молодая и активно развивающаяся компания в области поставок электронных компонентов. Мы поставляем электронные компоненты отечественного и импортного производства напрямую от производителей и с крупнейших складов мира.

Благодаря сотрудничеству с мировыми поставщиками мы осуществляем комплексные и плановые поставки широчайшего спектра электронных компонентов.

Собственная эффективная логистика и склад в обеспечивает надежную поставку продукции в точно указанные сроки по всей России.

Мы осуществляем техническую поддержку нашим клиентам и предпродажную проверку качества продукции. На все поставляемые продукты мы предоставляем гарантию .

Осуществляем поставки продукции под контролем ВП МО РФ на предприятия военно-промышленного комплекса России , а также работаем в рамках 275 ФЗ с открытием отдельных счетов в уполномоченном банке. Система менеджмента качества компании соответствует требованиям ГОСТ ISO 9001.

Минимальные сроки поставки, гибкие цены, неограниченный ассортимент и индивидуальный подход к клиентам являются основой для выстраивания долгосрочного и эффективного сотрудничества с предприятиями радиоэлектронной промышленности, предприятиями ВПК и научно-исследовательскими институтами России.

С нами вы становитесь еще успешнее!

Наши контакты:

Телефон: +7 812 627 14 35

Электронная почта: sales@st-electron.ru

Адрес: 198099, Санкт-Петербург,
Промышленная ул, дом № 19, литера Н,
помещение 100-Н Офис 331