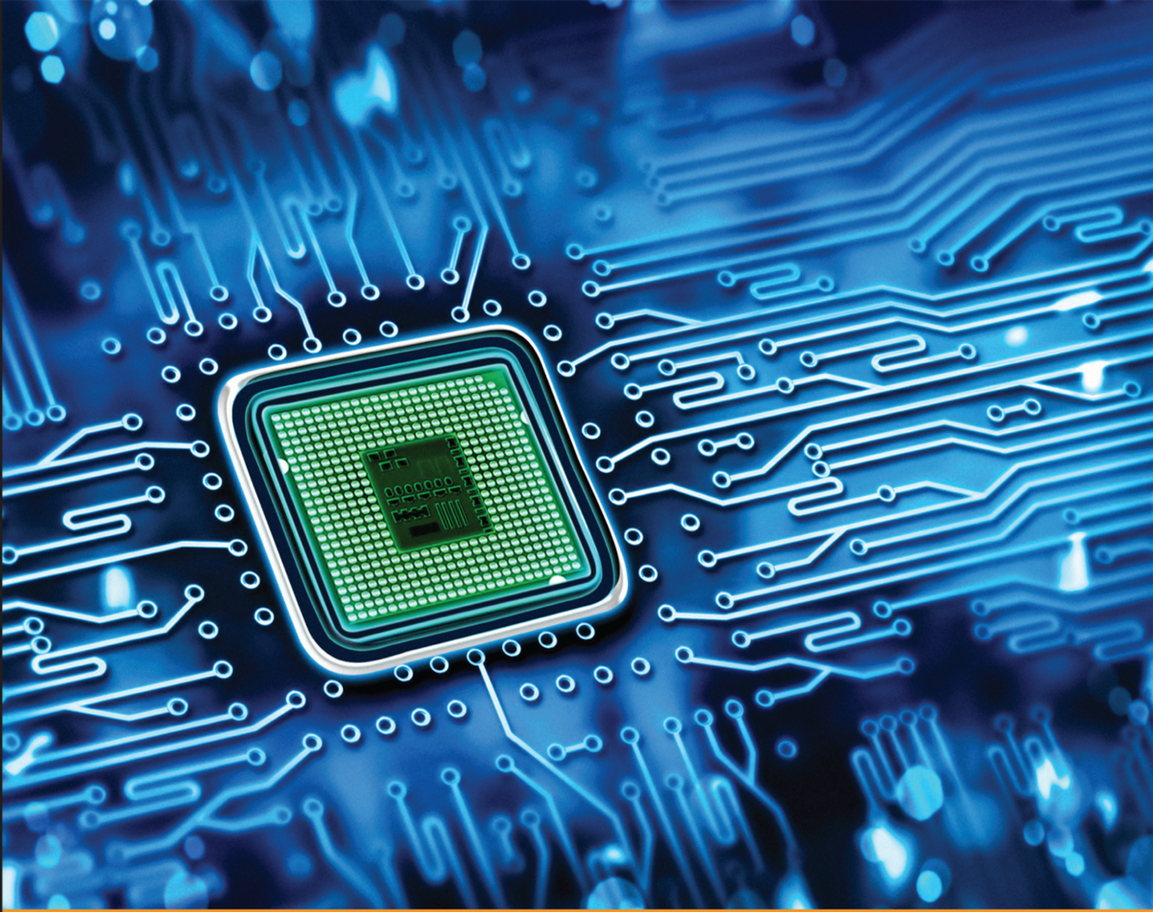


Community Experience Distilled

Internet of Things with ESP8266

Build amazing Internet of Things projects using the ESP8266 Wi-Fi chip

Marco Schwartz

[PACKT] PUBLISHING open source* community experience distilled

Internet of Things with ESP8266

Build amazing Internet of Things projects using the ESP8266 Wi-Fi chip

Marco Schwartz

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Internet of Things with ESP8266

First published: July 2016

Production reference: 1260716

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78646-802-4

`www.packtpub.com`

Credits

Author Marco Schwartz	**Project Coordinator** Kinjal Bari
Reviewer Catalin Batrinu	**Proofreader** Safis Editing
Commissioning Editor Pratik Shah	**Indexer** Pratik Shirodkar
Acquisition Editor Prachi Bisht	**Graphics** Kirk D'Penha
Content Development Editor Mamta Walkar	**Production Coordinator** Shantanu N. Zagade
Technical Editor Naveenkumar Jain	**Cover Work** Shantanu N. Zagade
Copy Editor Sneha Singh	

About the Author

Marco Schwartz is an electrical engineer, an entrepreneur, and a blogger. He has a master's degree in electrical engineering and computer science from Supélec, France, and a master's degree in micro engineering from the Ecole Polytechnique Fédérale de Lausanne (EPFL) in Switzerland.

He has more than five years of experience working in the domain of electrical engineering. Marco's interests gravitate around electronics, home automation, the Arduino and Raspberry Pi platforms, open source hardware projects, and 3D printing.

He has several websites about Arduino, including the Open Home Automation website, which is dedicated to building home automation systems using open source hardware.

Marco has written another book on home automation and Arduino, called *Home Automation With Arduino: Automate Your Home Using Open-source Hardware*. He has also written a book on how to build Internet of Things projects with Arduino, called *Internet of Things with the Arduino Yun*, Packt Publishing.

About the Reviewer

Catalin Batrinu graduated from the Politehnica University of Bucharest in Electronics, Telecommunications and Information Technology. He has been working as a software developer in telecommunications for the past 16 years.

He has worked with old protocols and the latest network protocols and technologies, so he has seen all the transformations in the telecommunication industry.

He has implemented many telecommunications protocols, from access adaptations and backbone switches to high capacity, carrier-grade switches on various hardware platforms from Wintegra and Broadcom.

Internet of Things came as a natural evolution for him and now he collaborates with different companies to construct the world of tomorrow that will make our life more comfortable and secure.

Using ESP8266, he has prototyped devices such as irrigation controllers, smart sockets, window shutters, Digital Addressable Lighting Controls, and environment controls, all of them being controlled directly from a mobile application over the cloud. Even an MQTT broker with bridging and a websockets server was developed for the ESP8266. Soon, all those devices will be part of our daily life, so we will all enjoy their functionality.

You can read his blog at `http://myesp8266.blogspot.com`.

Table of Contents

Preface v

Chapter 1: Getting Started with the ESP8266 1

How to choose your ESP8266 module 1
Hardware requirements 4
Hardware configuration 7
Installing the Arduino IDE for the ESP8266 10
Connecting your module to your Wi-Fi network 11
Summary 13

Chapter 2: First Projects with the ESP8266 15

Controlling an LED 15
Reading data from a GPIO pin 17
Grabbing the content from a web page 18
Reading data from a digital sensor 20
Summary 24

Chapter 3: Cloud Data Logging with the ESP8266 25

Hardware and software requirements 25
Hardware configuration 26
Testing the sensor 28
Logging data to Dweet.io 30
Displaying data using Freeboard.io 31
Summary 35

Chapter 4: Control Devices from Anywhere 37

Hardware and software requirements 38
Configuring the ESP8266 module and controlling an LED 39
Controlling the LED from a cloud dashboard 44
Controlling the lamp from anywhere in the world 45
Summary 47

Chapter 5: Interacting With Web Services 49
Hardware and software requirements 49
Getting weather data from Yahoo 51
Posting temperature and humidity data to Twitter 56
Creating a new Facebook post from the ESP8266 62
Summary 69
Chapter 6: Machine-to-Machine Communications 71
Hardware and software requirements 71
Simple machine-to-machine communication 73
Building a light-activated relay 82
Summary 87
Chapter 7: Sending Notifications from the ESP8266 89
Hardware and software requirements 89
Hardware configuration 91
Sending an e-mail notification 91
Sending data via text message 99
Receiving alerts via push notifications 103
Summary 107
Chapter 8: Controlling a Door Lock from the Cloud 109
Hardware and software requirements 109
Configuring the hardware 111
Configuring the ESP8266 board 112
Controlling the lock from the cloud 113
Receiving notifications when the lock is opened 114
Summary 119
Chapter 9: Building a Physical Bitcoin Ticker 121
What is Bitcoin? 121
Online Bitcoin services 122
Hardware and software requirements 125
Configuring the hardware 126
Testing the ticker 127
Adding alert LEDs to the ticker 132
Summary 134

Chapter 10: Wireless Gardening with the ESP8266 135
Hardware and software requirements 135
Hardware configuration 137
Creating alerts to water your plant 139
Monitoring the temperature and humidity 145
Automating your gardening 147
Summary 149
Chapter 11: Cloud-Based Home Automation System 151
Hardware and software requirements 151
Hardware configuration 152
Controlling your home from a dashboard 154
Creating a cloud alarm system 160
Automating your home 163
Summary 173
Chapter 12: Cloud-Controlled ESP8266 Robot 175
Hardware and software requirements 175
Hardware configuration 180
Testing the motors 182
Connecting the robot to the cloud 185
Controlling the robot from a dashboard 188
Summary 190
Chapter 13: Building Your Own Cloud Platform to Control ESP8266 Devices 191
Hardware and software requirements 191
Hardware configuration 192
Creating a cloud server 193
The aREST cloud server code 196
Deploying the server 198
Connecting the ESP8266 board to your cloud server 201
Summary 204
Index 205

Preface

The **Internet of Things** (**IoT**) is an exciting field that proposes to have all the devices that surround us connected to the Internet and interacting with us, but also between each other. It's estimated that there will be 50 billion IoT devices in the world by the year 2020.

On the other hand, the ESP8266 chip is a small, cheap (less than $5), and powerful Wi-Fi chip that is also really easy to program. Therefore, it is just the perfect tool to build inexpensive and nice IoT projects. In this book, you are going to learn everything you need to know on how to build IoT projects using the ESP8266 Wi-Fi chip.

What this book covers

Chapter 1, *Getting Started with the ESP8266*, will teach all you need to know about how to choose your ESP8266 board and upload your first sketch to the chip.

Chapter 2, *First Projects with the ESP8266*, will explain the basics of the ESP8266 by making some real simple projects.

Chapter 3, *Cloud Data Logging with the ESP8266*, will dive right into the core of the topic of the book, and build a project that can log measurement data on the cloud.

Chapter 4, *Control Devices from Anywhere*, will reveal how to control devices from anywhere in the world using the ESP8266.

Chapter 5, *Interacting With Web Services*, will show how to use the ESP8266 to interact with existing web platforms such as Twitter.

Chapter 6, *Machine-to-Machine Communications*, will explain how to make ESP8266 chips talk to each other via the cloud, to build applications that don't require human intervention.

Chapter 7, *Sending Notifications from the ESP8266*, will show how to send automated notifications from the ESP8266, for example, via text message or email.

Chapter 8, *Controlling a Door Lock from the Cloud,* will use what we learned so far in the book to build our first application: a door lock that can be controlled remotely.

Chapter 9, *Building a Physical Bitcoin Ticker*, will use the ESP8266 for a fun project: making a physical display of the current price of Bitcoin.

Chapter 10, *Wireless Gardening with the ESP8266*, will dive into a more complex, by learning how to automate your garden with the ESP8266.

Chapter 11, Cloud-Based Home Automation System, will show how to build the essential blocks of an home automation system using the ESP8266.

Chapter 12, *Cloud-Controlled ESP8266 Robot*, will explain how to use the ESP8266 to control a mobile robot from anywhere in the world.

Chapter 13, Building Your Own Cloud Platform to Control ESP8266 Devices, will reveal how to build our own cloud platform for your ESP8266 projects.

What you need for this book

For this book, you will need to have the Arduino IDE, which we will use for all the projects of the book. You will learn how to install it and configure it in the first chapter of the book.

The chapters of the book were also written with a progressive complexity, so even if you don't know a lot about Arduino and/or the ESP8266 you will be able to learn as you progress through the chapters. However, previous experience in programing (especially in C++ and/or JavaScript) is recommend for this book.

Who this book is for

This book is for those who want to build powerful and inexpensive IoT projects using the ESP8266 Wi-Fi chip, including those who are new to IoT, or those who already have experience with other platforms such as Arduino.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
void loop() {
Serial.print("Connecting to ");
Serial.println(host);
// Use WiFiClient class to create TCP connections
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
  }
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Open **Boards Manager** from the **Tools | Board** menu and install the **esp8266** platform, as shown."

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `http://www.packtpub.com/sites/default/files/downloads/InternetofThingswithESP8266_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Getting Started with the ESP8266

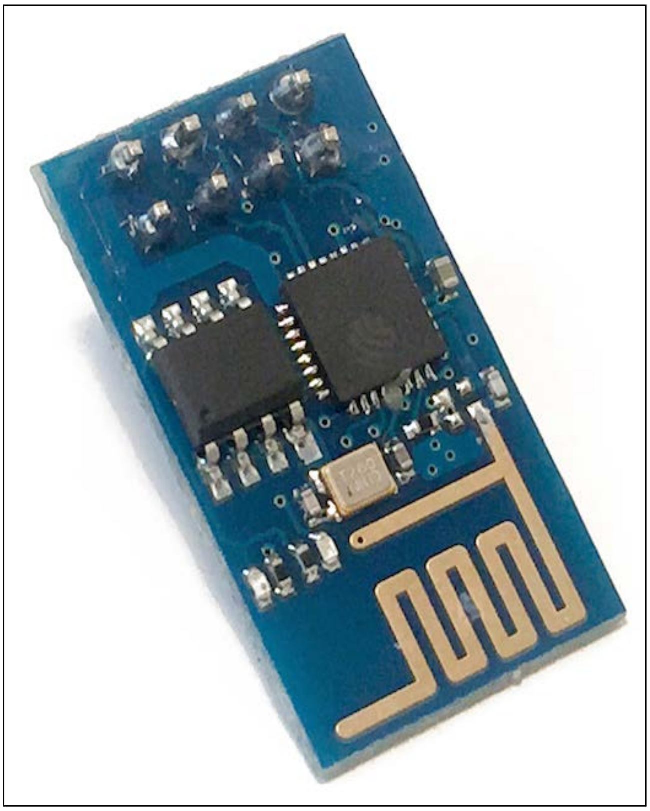
In this chapter, we are going to start by setting up the ESP8266 chip. We will learn how to choose the right module for your project and get all the additional hardware you need to use the chip. We will also see how to connect the ESP8266 to your computer, so you can program it using a USB cable.

Then, we are going to see how to configure and upload code to the ESP8266 chip. For that, we will be using the Arduino IDE. This makes using the ESP8266 much easier, as we will be using a well-known interface and language to configure the chip. We will also be able to use most of the already existing Arduino libraries for our projects. Let's start!

How to choose your ESP8266 module

We are first going to see how to choose the right ESP8266 module for your project. There are many modules available in the market and it is quite easy to get lost with all the choices available.

The first one that you have probably heard of is the small ESP8266 Serial Wireless Transceiver module:



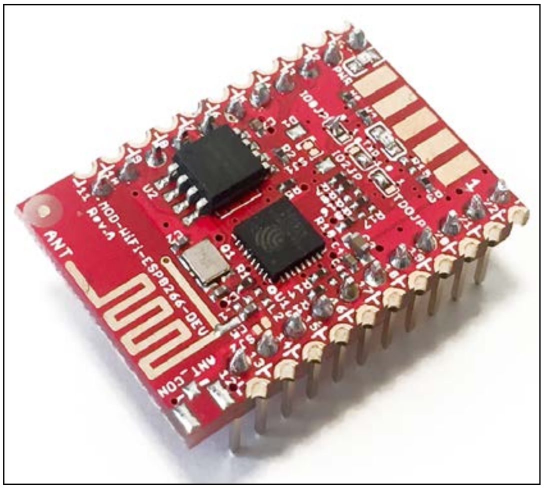
This module is the most famous one, as it is really small and only costs $5. However, the number of accessible GPIO pins (input/output pins) is quite limited. It is also difficult to plug it into a standard breadboard.

If you choose this module, there are some projects in this book that you might not be able to do. For example, you won't be able to do the projects using analog sensors, as the analog input pin is not accessible.

You can find more information about this module at:

`https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf`

But there are many other modules on the market that give you access to all the pins of the ESP8266. For example, I really like the ESP8266 Olimex module, which is also cheap (around $10):

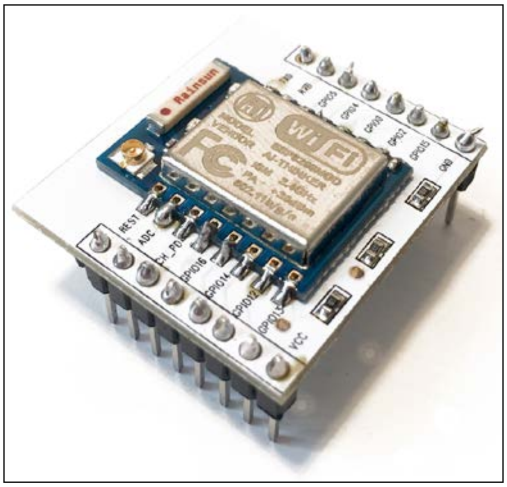


This module can easily be mounted on a breadboard and you can easily access all the pins of the ESP8266. This is the one I will use for most of this book and therefore I also recommend that you use a similar module.

You can find additional details about this module at:

`https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware`

One other choice is to use a board based on the ESP-12, which is a version of the ESP8266 made to be integrated on PCBs. This version also gives you access to all the pins of the ESP8266. It is relatively easy to find breakout boards for this chip. For example, this is a board that I bought on Tindie:



You can find more information about this module on:

`http://www.seeedstudio.com/wiki/images/7/7d/ESP-12E_brief_spec.pdf`

You can also get your hands on the Adafruit ESP8266 breakout board, which also integrates the ESP-12:

`http://www.adafruit.com/product/2471`

Another solution is to use the NodeMCU development kit, which is similar to the Olimex board but also has an integrated USB-to-Serial converter, as well as an onboard power supply. It is easier to use, but was hard to find at the time this book was written. You can get more information on the NodeMCU website:

`http://nodemcu.com/index_en.html`

Note that with the NodeMCU module, you will have to translate the pins from the module to the pins defined in the ESP8266 Arduino IDE, which we are going to use. You will find the correspondence between pins here:

`https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en#new_gpio_map`

Hardware requirements

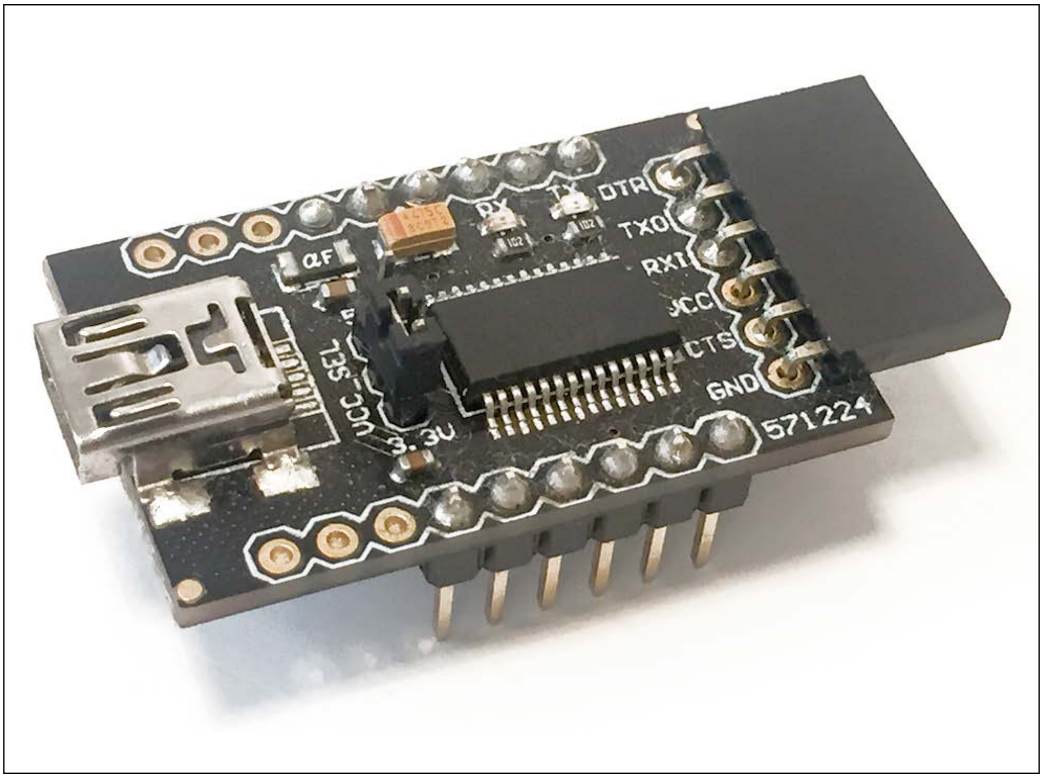
Let's now take a look at the things we need to make the ESP8266 chip work. It is usually, but incorrectly, assumed that you just need this little chip and nothing else to make it work, but we are going to see that it is not true.

First, you will need some way to program the ESP8266. You can use an Arduino board for that, but for me the really great thing about the ESP8266 is that it can function completely autonomously, using the onboard processor.

So to program the chip, I will use a USB FTDI programmer.

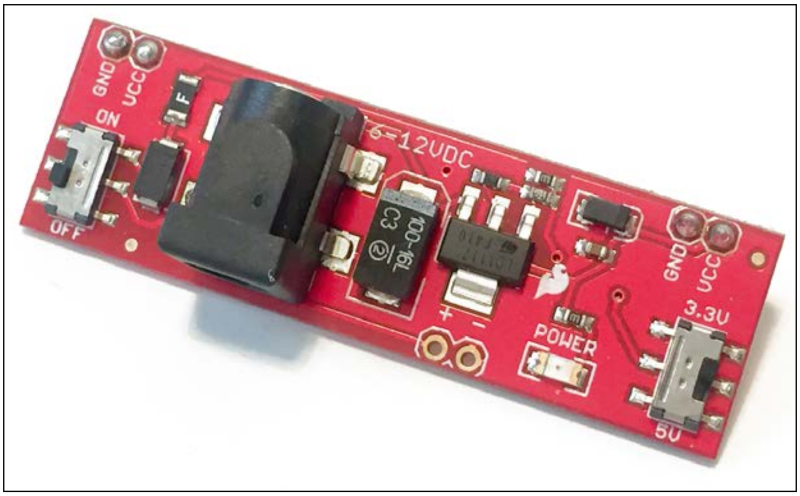
> Note that it has to be compatible with the logic level of the ESP8266 chip, so 3.3V.

I have used a module that can be switched between 3.3V and 5V:



You will also need a dedicated power supply to power the chip. This is a point that is often forgotten and leads to a lot of issues. If you are, for example, trying to power the ESP8266 chip from the 3.3V coming from the FTDI board or from an Arduino board, it simply won't work correctly.

Therefore, for most ESP8266 modules, you need a dedicated power supply that can deliver at least 300 mA to be safe. Some boards have an integrated micro-USB port and a voltage regulator that can provide the required current to the ESP8266, but that's not the case with the board we will use in this first chapter. I used a breadboard power supply that can deliver up to 500 mA at 3.3V:

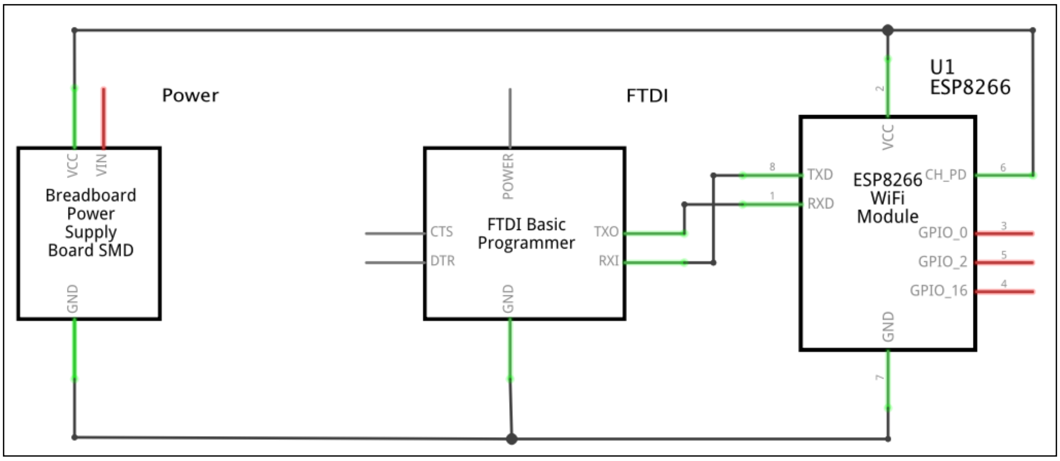


This is a list of all the components that you will need to use the ESP8266 chip:

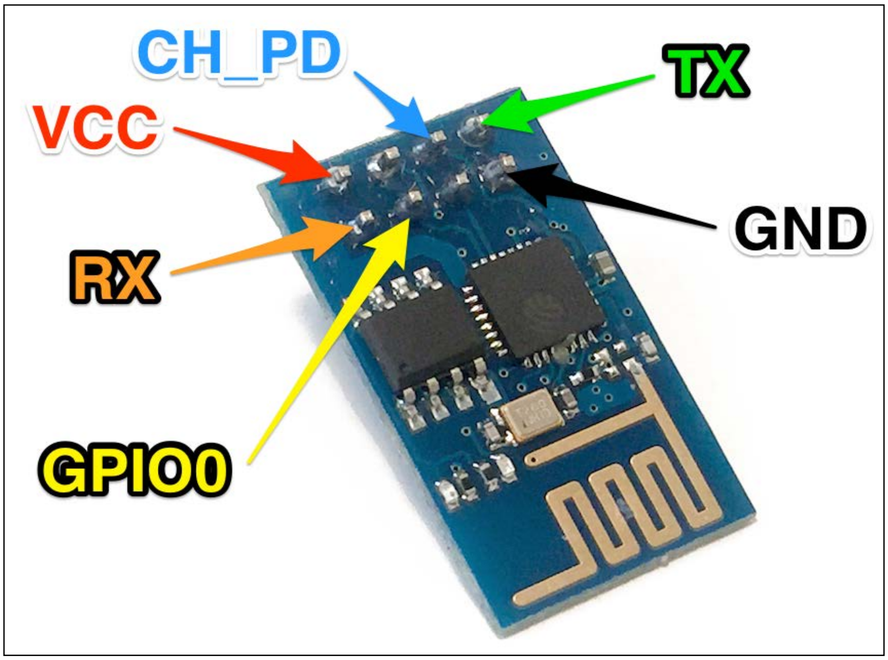
- ESP8266 Olimex module (`https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware`)
- Breadboard 3.3V power supply (`https://www.sparkfun.com/products/13032`)
- 3.3V FTDI USB module (`https://www.sparkfun.com/products/9873`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/12795`)

Hardware configuration

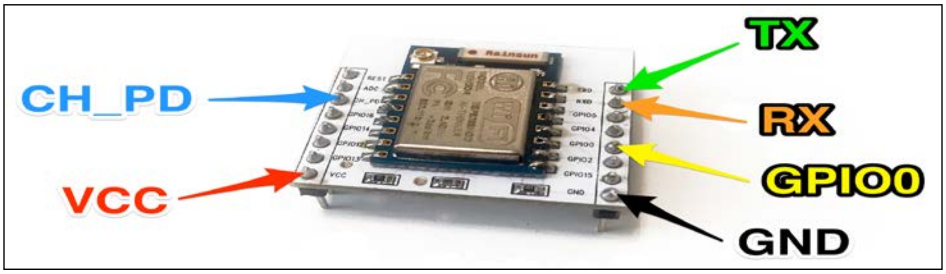
We are now going to take a look at the way to configure the hardware for the first use of your ESP8266 board. This is how we connect the different components:



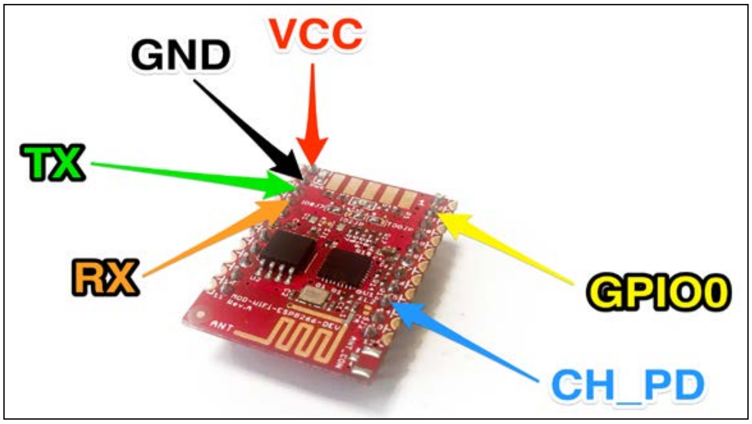
Depending on the board you are using, the pins can have different names. Therefore, I created pictures to help you out with each module. These are the pins you will need on the small ESP board:



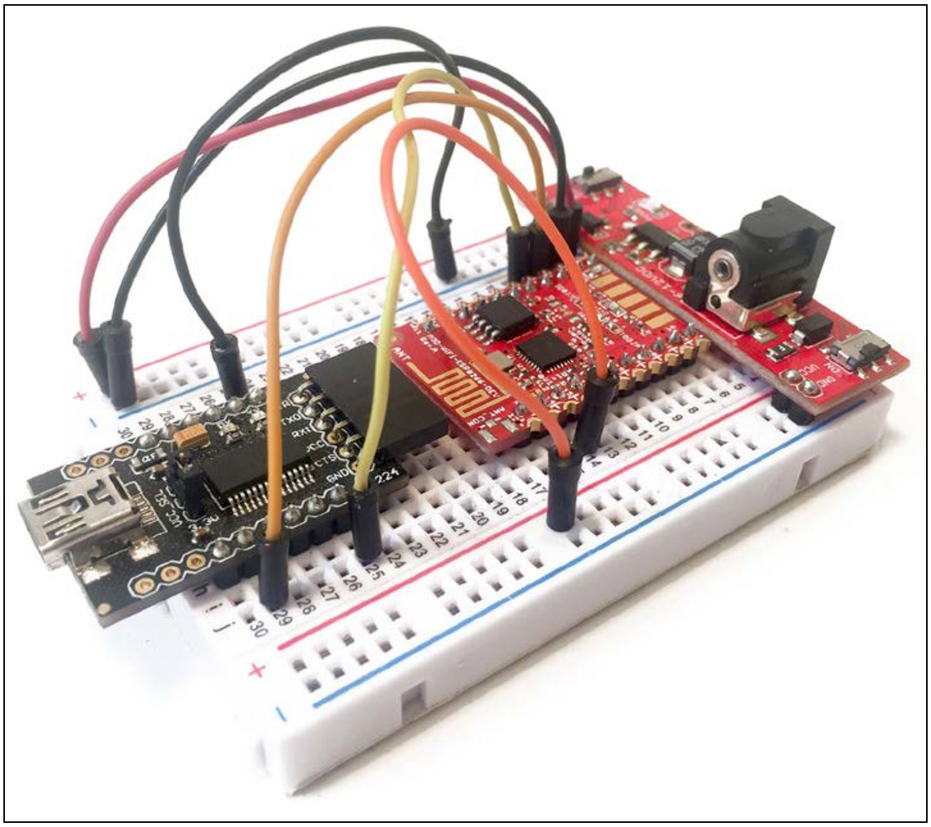
This is the same for the ESP-12 board mounted on a breadboard adapter:



Finally, this is the picture for the Olimex board:



This is what the Olimex board will look like at the end:



Make sure that you connect everything according to the schematics or you won't be able to continue.

[Also, make sure that all the switches of your components (FTDI module and power supply) are set to 3.3V, or it will damage your chip.]

Also, connect one wire to the GPIO 0 pin of the ESP8266. Don't connect it to anything else for now, but you will need it later to put the chip in programming mode.

Installing the Arduino IDE for the ESP8266

Now that we have completely set up the hardware for the ESP8266, we are ready to configure it using the Arduino IDE.

The most basic way to use the ESP8266 module is to use serial commands, as the chip is basically a Wi-Fi/Serial transceiver. However, this is not convenient and this is not what I recommend doing.

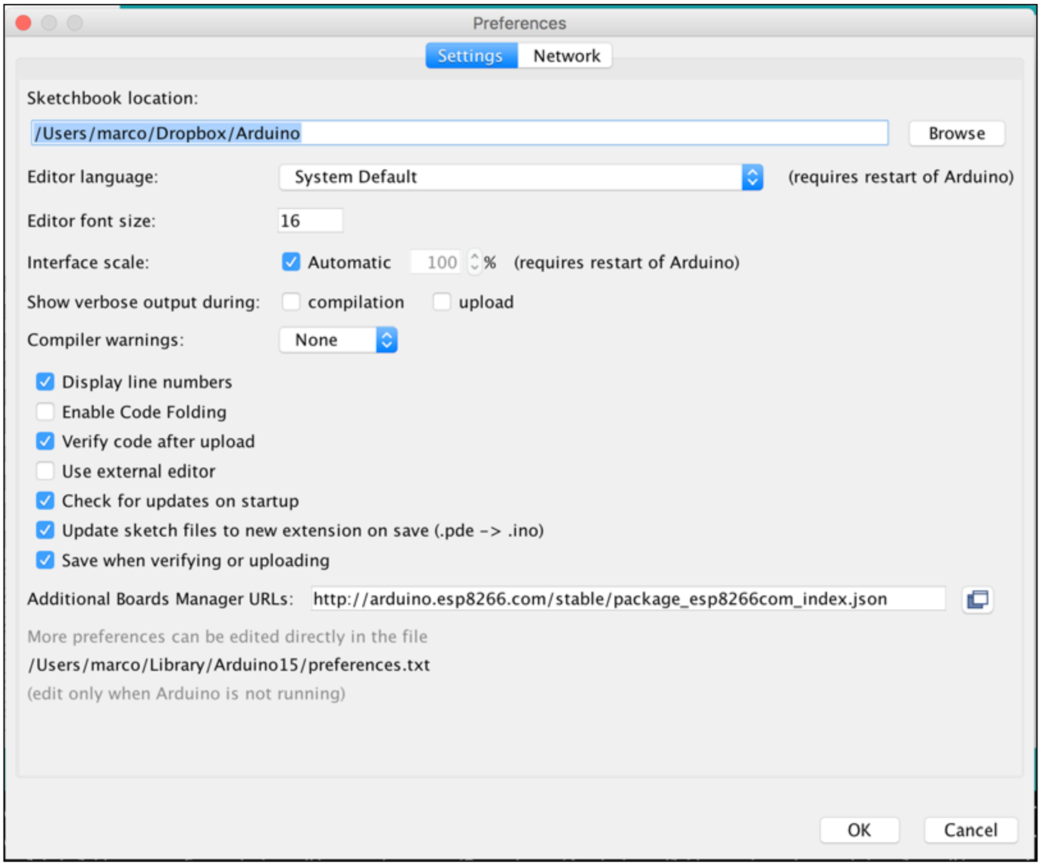
What I recommend is simply using the Arduino IDE, which you will need to install on your computer. This makes it very convenient to use the ESP8266 chip, as we will be using the well-known Arduino IDE, so this is the method that we will use in the entire book.

We are now going to configure your ESP8266 chip using the Arduino IDE. This is a great way to use the chip, as you will be able to program it using the well-known Arduino IDE and also re-use several existing Arduino libraries.

If this is not done yet, install the latest version of the Arduino IDE. You can get it from `http://www.arduino.cc/en/main/software`.

Now, you need to take a follow steps to be able to configure the ESP8266 with the Arduino IDE:

1. Start the **Arduino IDE** and open the **Preferences** window.
2. Enter the following URL into the **Additional Board Manager** URLs field: `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
3. Open **Boards Manager** from the **Tools | Board** menu and install the **esp8266** platform as shown here:



Sketchbook location:
/Users/marco/Dropbox/Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 16

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
✓ Check for updates on startup
✓ Update sketch files to new extension on save (.pde -> .ino)
✓ Save when verifying or uploading

Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json

More preferences can be edited directly in the file
/Users/marco/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

Connecting your module to your Wi-Fi network

Now, we are going to check whether the ESP8266 and the Arduino IDE are working correctly, and connect your chip to your local Wi-Fi network.

To do so, let's perform the following steps:

1. First, we need to write the code and then upload it to the board. The code is simple; we just want to connect to the local Wi-Fi network and print the IP address of the board. This is the code to connect to the network:

```
// Import required libraries
#include <ESP8266WiFi.h>

// WiFi parameters
```

```
constchar* ssid = "your_wifi_name";
constchar* password = "your_wifi_password";

void setup(void)
{
// Start Serial
Serial.begin(115200);
// Connect to WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected");
// Print the IP address
Serial.println(WiFi.localIP());
}

void loop() {

}
```

You can simply copy the lines of the preceding code and then paste them into the ESP8266 Arduino IDE that you downloaded earlier. Of course, put your own Wi-Fi name and password in the code. Save this file with a name of your choice.

2. Now, navigate to **Tools | Boards** and select **Generic ESP8266 Module**. Also, select the correct **Serial port** that corresponds to the FTDI converter that your are using.
3. After that, we need to put the board in the bootloader mode, so we can program it. To do so, connect the pin GPIO 0 to the ground, via the cable we plugged into GPIO 0. Then, power cycle the board by switching the power supply off and then on again.
4. Now, upload the code to the board and open the Serial monitor when this is done. Set the Serial monitor speed to **115200**. Now, disconnect the cable between GPIO 0 and GND and power cycle the board again. You should see the following message:

```
WiFi connected
192.168.1.103
```

If you can see this message and an IP, congratulations, your board is now connected to your Wi-Fi network! You are now ready to build your first projects using the ESP8266 chip.

Summary

In this first chapter of the book, we learned the fundamentals about the ESP8266. We first learned about all the different boards that are available for your ESP8266 projects. Then, we saw how to wire your ESP8266 modules. Finally, we saw how to install the Arduino IDE and to configure it for the ESP8266, and we ended the chapter by actually uploading a very simple sketch to the ESP8266.

In the next chapter, we are going to use the tools we set up and build some basic projects using the ESP8266 Wi-Fi chip.

2
First Projects with the ESP8266

Now that your ESP8266 chip is ready to be used and you can connect it to your Wi-Fi network, we can build some basic projects with it. This will help you understand the basics of the ESP8266.

We are going to see three projects in this chapter: how to control an LED, how to read data from a GPIO pin, and how to grab the contents from a web page. We will also see how to read data from a digital sensor.

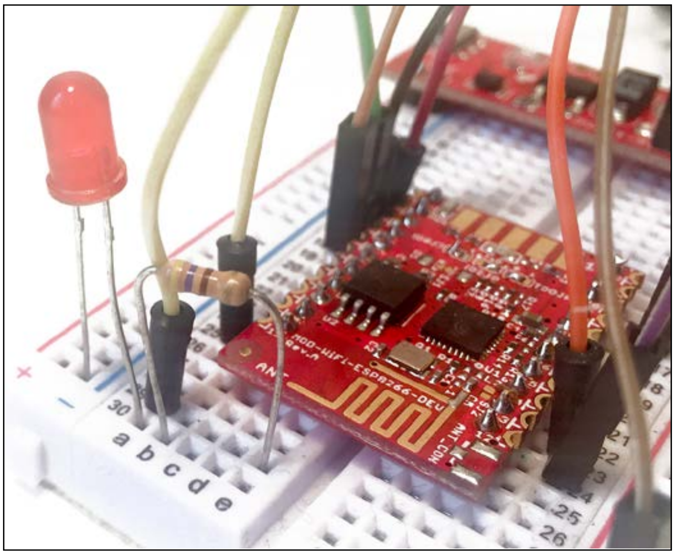
Controlling an LED

First, we are going to see how to control a simple LED. The GPIO pins of the ESP8266 can be configured to realize many functions: inputs, outputs, PWM outputs, and also SPI or I2C communications. This first project will teach you how to use the GPIO pins of the chip as outputs:

1. The first step is to add an LED to our project. These are the extra components you will need for this project:
    - 5 mm LED (`https://www.sparkfun.com/products/9590`)
    - 330 Ohm resistor to limit the current in the LED (`https://www.sparkfun.com/products/8377`)
2. The next step is to connect the LED with the resistor to the ESP8266 board. To do so, the first thing to do is to place the resistor on the breadboard.
3. Then, place the LED on the breadboard as well, connecting the longest pin of the LED (the anode) to one pin of the resistor.

4. Then, connect the other end of the resistor to GPIO pin 5 of the ESP8266, and the other end of the LED to the ground.

   This is what it should look like at the end:



5. We are now going to light up the LED by programming the ESP8266 chip, just as we did in the first chapter of the book by connecting it to the Wi-Fi network.

   This is the complete code for this section:

```
// Import required libraries
#include <ESP8266WiFi.h>

void setup() {

// Set GPIO 5 as output
pinMode(5, OUTPUT);

// Set GPIO 5 on a HIGH state
digitalWrite(5, HIGH);

}
void loop() {

}
```

This code simply sets the GPIO pin as an output, and then applies a HIGH state to it. The HIGH state means that the pin is active, and that positive voltage (3.3V) is applied to the pin. A LOW state would mean that the output is at 0V.

6. You can now copy this code and paste it into the Arduino IDE.
7. Then, upload the code to the board, using the instructions from the previous chapter. You should immediately see that the LED lights up. You can shut it down again by using `digitalWrite(5, LOW)` in the code. You could also, for example, modify the code so the ESP8266 switches the LED on and off every second.

Reading data from a GPIO pin

In the second project in this chapter, we are going to read the state of a GPIO pin. For this, we will use the same pin as in the previous project. You can therefore remove the LED and the resistor that we used in the previous project.

Now, simply connect this pin (GPIO 5) of the board to the positive power supply on your breadboard with a wire, applying a 3.3V signal on this pin.

Reading data from a pin is really simple. This is the complete code for this part:

```
// Import required libraries
#include <ESP8266WiFi.h>

void setup(void)
{
// Start Serial (to display results on the Serial monitor)
Serial.begin(115200);

// Set GPIO 5 as input
pinMode(5, INPUT);}
void loop() {

// Read GPIO 5 and print it on Serial port
Serial.print("State of GPIO 5: ");
Serial.println(digitalRead(5));

// Wait 1 second
  delay(1000);
}
```

We simply set the pin as an input, read the value of this pin, and print it out every second. Copy and paste this code into the Arduino IDE, then upload it to the board using the instructions from the previous chapter.

This is the result you should get in the Serial monitor:

```
State of GPIO 5: 1
```

We can see that the returned value is 1 (digital state HIGH), which is what we expected, because we connected the pin to the positive power supply. As a test, you can also connect the pin to the ground, and the state should go to 0.

Grabbing the content from a web page

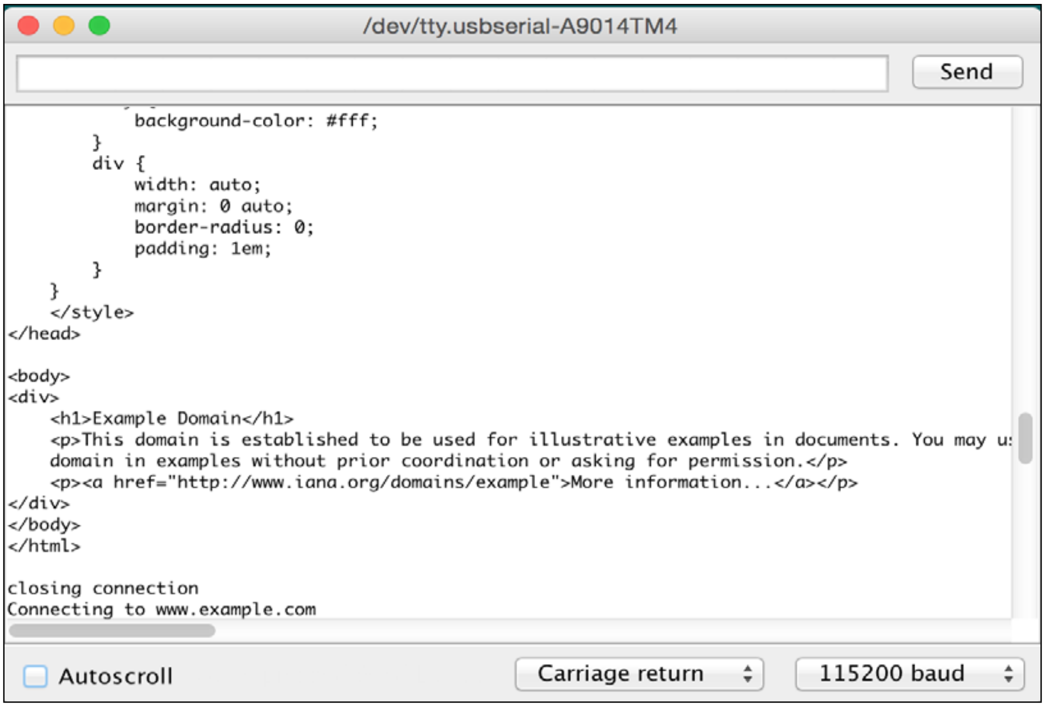
As the last project in this chapter, we are finally going to use the Wi-Fi connection of the chip to grab the content of a page. We will simply use the `www.example.com` page, as it's a basic page largely used for test purposes.

This is the complete code for this project:

```
// Import required libraries
#include <ESP8266WiFi.h>

// WiFi parameters
constchar* ssid = "your_wifi_network";
constchar* password = "your_wifi_password";

// Host
constchar* host = "www.example.com";

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

// We start by connecting to a WiFi network
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

int value = 0;

void loop() {

Serial.print("Connecting to ");
Serial.println(host);

// Use WiFiClient class to create TCP connections
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
  }

// This will send the request to the server
client.print(String("GET /") + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
  delay(10);

// 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");
  delay(5000);

}
```

The code is really basic: we first open a connection to the `example.com` website, and then send a GET request to grab the content of the page. Using the `while(client.available())` code, we also listen for incoming data, and print it all inside the Serial monitor.

You can now copy this code and paste it into the Arduino IDE. Then, upload it to the board using the instructions from *Chapter 1, Getting Started with the ESP8266*, in the section *Connecting Your Module to Your Wi-Fi Network*. This is what you should see in the Serial monitor:



```
            background-color: #fff;
        }
        div {
            width: auto;
            margin: 0 auto;
            border-radius: 0;
            padding: 1em;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in documents. You may u:
    domain in examples without prior coordination or asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>

closing connection
Connecting to www.example.com
```

This is basically the content of the page, in pure HTML code.

Reading data from a digital sensor

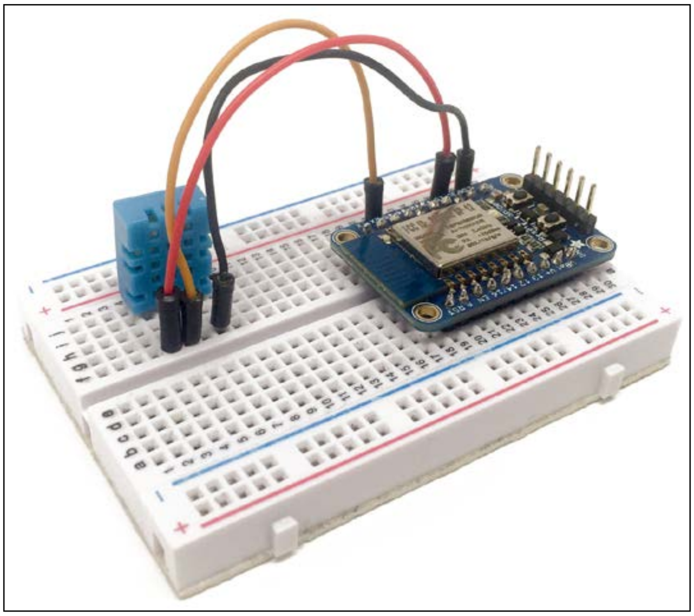
In this last section of this chapter, we are going to connect a digital sensor to our ESP8266 chip, and read data from it. As an example, we will use a DHT11 sensor, which can be used to get ambient temperature and humidity.

You will need to get this component for this section, the DHT11 sensor (`https://www.adafruit.com/products/386`)

Let's now connect this sensor to your ESP8266:

1. First, place the sensor on the breadboard. Then, connect the first pin of the sensor to VCC, the second pin to pin 5 of the ESP8266, and the fourth pin of the sensor to GND.

   This is what it will look like at the end:



> Note that here I've used another ESP8266 board, the Adafruit ESP8266 breakout board. I will use this board in several chapters of this book.

   We will also use the aREST framework in this example, so it's easy for you to access the measurements remotely. aREST is a complete framework to control your ESP8266 boards remotely (including from the cloud), and we are going to use it several times in the book. You can find more information about it at the following URL: `http://arest.io/`.

2. Let's now configure the board. The code is too long to be inserted here, but I will detail the most important part of it now.

> Note that you can find the code on the GitHub repository of the book: `https://github.com/openhomeautomation/iot-esp8266-packt`.

3. It starts by including the required libraries:

   ```
   #include "ESP8266WiFi.h"
   #include <aREST.h>
   #include "DHT.h"
   ```

4. To install those libraries, simply look for them inside the Arduino IDE library manager. Next, we need to set the pin that the DHT sensor is connected to:

   ```
   #define DHTPIN 5
   #define DHTTYPE DHT11
   ```

5. After that, we declare an instance of the DHT sensor:

   ```
   DHT dht(DHTPIN, DHTTYPE, 15);
   ```

6. As earlier, you will need to insert your own Wi-Fi name and password into the code:

   ```
   const char* ssid = "wifi-name";
   const char* password = "wifi-pass";
   ```

7. We also define two variables that will hold the measurements of the sensor:

   ```
   float temperature;
   float humidity;
   ```

8. In the `setup()` function of the sketch, we initialize the sensor:

   ```
   dht.begin();
   ```

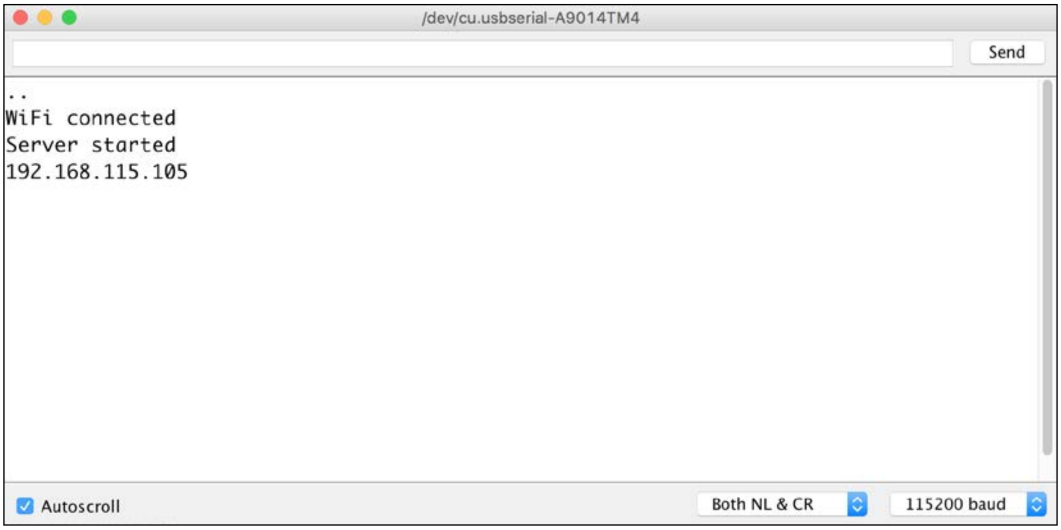
9. Still in the `setup()` function, we expose the variables to the aREST API, so we can access them remotely via Wi-Fi:

   ```
   rest.variable("temperature",&temperature);
   rest.variable("humidity",&humidity);
   ```

10. Finally, in the `loop()` function, we make the measurements from the sensor:

    ```
    humidity = dht.readHumidity();
    temperature = dht.readTemperature();
    ```

11. It's now time to test the project! Simply grab all the code and put it inside the Arduino IDE. Also make sure to install the aREST Arduino library using the Arduino library manager.
12. Now, put the ESP8266 board in bootloader mode, and upload the code to the board. After that, reset the board, and open the Serial monitor. You should see the IP address of the board being displayed:



```
..
WiFi connected
Server started
192.168.115.105
```

13. Now, we can access the measurements from the sensor remotely. Simply go to your favorite web browser, and type:

    ```
    192.168.115.105/temperature
    ```

    You should immediately get the answer from the board, with the temperature being displayed:

    ```
    {
      "temperature": 25.00,
    "id": "1",
    "name": "esp8266",
    "connected": true
    }
    ```

    You can of course do the same with humidity.

> Note that here we used the aREST API, which we will use in several other chapters in this book. You can learn more about it at `http://arest.io/`.

Congratulations, you just completed your very first projects using the ESP8266 chip! Feel free to experiment with what you learned in this chapter, and start learning more about how to configure your ESP8266 chip.

Summary

In this chapter, we realized our first basic projects using the ESP8266 Wi-Fi chip. We first learned how to control a simple output, by controlling the state of an LED. Then, we saw how to read the state of a digital pin on the chip. Finally, we learned how to read data from a digital sensor, and actually grab this data using the aREST framework which we will use in several chapters of this book.

In the next chapter, we are going to go right into the main topic of the book, and build our first Internet of Things project using the ESP8266.

3
Cloud Data Logging with the ESP8266

In this chapter, we are going to use the ESP8266 to automatically log temperature and humidity measurements in the cloud, and display these measurements inside an online dashboard.

By following this project, you will be able to build a small and cheap measurement platform that logs data in the cloud. Of course, this can be applied to several types of sensor, such as motion detectors. Let's dive in!

Hardware and software requirements

For this project, you will need the following hardware:

- Of course you need an ESP8266 chip. You can, for example, use an Olimex ESP8266 module.
- You will also need a temperature sensor. I used a DHT11 sensor, which is very easy to use and will allow us to measure the ambient temperature and humidity.
- You will also need a 3.3V FTDI USB module to program the ESP8266 chip. Finally, you will also need some jumper wires and a breadboard.

This is a list of all the components that will be used in this chapter, along with the sources where you can purchase them:

- ESP8266 Olimex module (`https://www.olimex.com/Products/IoT/MOD-WIFI-ESP8266-DEV/open-source-hardware`)
- Breadboard 3.3V power supply (`https://www.sparkfun.com/products/13032`)
- 3.3V FTDI USB module (`https://www.sparkfun.com/products/9873`)
- DHT11 sensor (`https://www.adafruit.com/products/386`)
- Breadboard (`https://www.sparkfun.com/products/12002`)
- Jumper wires (`https://www.sparkfun.com/products/9194`)

On the software side, you will need:

- The latest version of the Arduino IDE, which you can get from: `http://www.arduino.cc/en/Main/Software`.

Now let's follow this procedure to add the ESP8266 board to the Arduino IDE:

1. Start the Arduino IDE and open the **Preferences** window.
2. Enter the following URL into the **Additional Board Manager URLs** field:

   ```
   http://arduino.esp8266.com/package_esp8266com_index.json
   ```

3. Open `Boards Manager` by navigating to the **Tools** | **Board** menu, and install the **esp8266 platform**.
4. You will also need the DHT library, which you can get from:

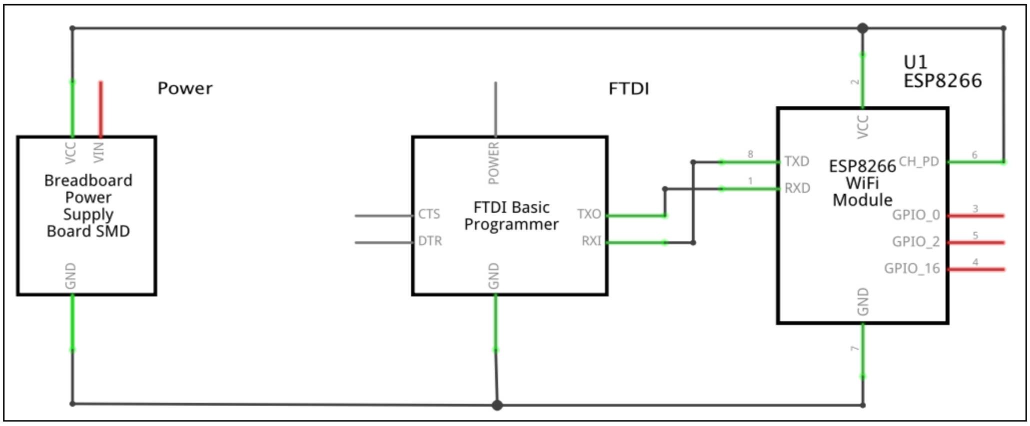
   ```
   https://github.com/adafruit/DHT-sensor-library
   ```

To install an Arduino library:

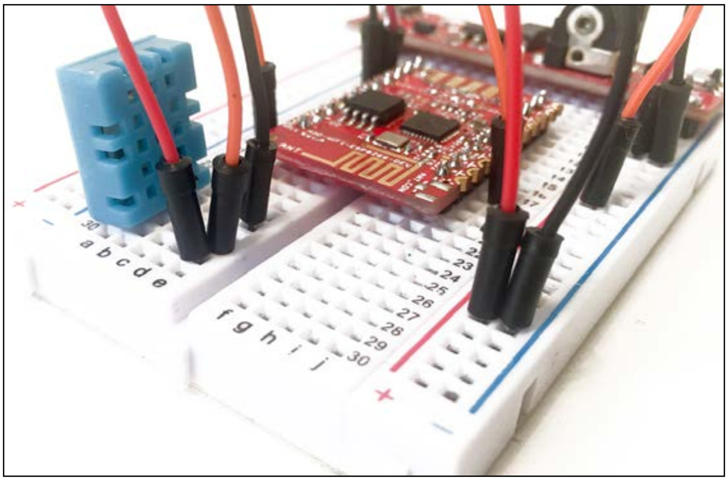
1. First, download the library from the GitHub repository.
2. Then, go into the Arduino IDE, and navigate to **Sketch** | **Include Library** | **Add .ZIP Library**.
3. Finally, select the file that you just downloaded.

Hardware configuration

We are first going to see how to configure the hardware to use the ESP8266 board. This is how to connect the different components:



1. Basically, you need to connect your breadboard power supply VCC and GND to the ESP8266 VCC and GND. Also, connect the GND pin of the FTDI converter board to the ESP8266 GND.
2. Then, connect TX from the FTDI board to RX of the ESP8266 board, and then RX to TX.
3. Finally, connect the CH_PD (or CHIP_EN) pin of the ESP8266 board to VCC.
4. Once this is done, simply put the DHT11 sensor on the breadboard.
5. Then, connect the left pin to VCC (red power rail), the right pin to GND (blue power rail), and the pin next to VCC to the GPIO pin 5 on your ESP8266 chip. This is the final result, not showing the USB-to-Serial FTDI cables:



Make sure that you've connected everything according to the schematics, or you won't be able to continue. Also make sure that all the switches of your components (FTDI module and power supply) are set to 3.3V, or it will damage your chip.

6. Also, connect one wire to GPIO pin 0 of the ESP8266. Don't connect it to anything else for now, but you will need it later to put the chip in programming mode.

Testing the sensor

We are now going to use the sensor. Again, remember that we are using the Arduino IDE, so we can code just like we would do using an Arduino board. Here, we will simply print the value of the temperature inside the Serial monitor of the Arduino IDE. If it has not been done yet, install the Adafruit DHT sensor library using the Arduino IDE library manager.

This is the complete code for this part:

```
// Libraries
#include "DHT.h"

// Pin
#define DHTPIN 5

// Use DHT11 sensor
#define DHTTYPE DHT11

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE, 15);

void setup() {

// Start Serial
Serial.begin(115200);

// Init DHT
dht.begin();
}

void loop() {
```

```
// Reading temperature and humidity
float h = dht.readHumidity();
// Read temperature as Celsius
float t = dht.readTemperature();

// Display data
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.println(" *C ");

// Wait 2 seconds between measurements.
delay(2000);

}
```

Let's see the details of the code. You can see that all the measurement part is contained inside the `loop()` function, which makes the code inside it repeat every 2 seconds.

Then, we read data from the DHT11 sensor, and print the value of the temperature and humidity on the Serial port.

> Note that the complete code can also be found inside the GitHub repository for the project:
>
> `https://github.com/openhomeautomation/iot-esp8266`

Now let's start with the steps to test the sensor:

1. Paste the previous code into the Arduino IDE. Then, go to **Tools | Boards**, and select **Generic ESP8266 Module**. Also select the correct Serial port that corresponds to the FTDI converter you are using.
2. After that, we need to put the board in bootloader mode, so we can program it. To do so, connect GPIO pin 0 to the ground, via the cable we plugged into GPIO 0 before.
3. Then, power cycle the board by switching the power supply off and then on again.

4. Now, upload the code to the board and open the Serial monitor when this is done. Also, set the Serial monitor speed to 115200.
5. Now, disconnect the cable between GPIO 0 and GND, and power cycle the board again.

You should immediately see the temperature and humidity readings inside the Serial monitor. My sensor was reading around 24 degrees Celsius when I tested it, which is a realistic value.

Logging data to Dweet.io

We are now going to see how to log the temperature and humidity measurements in the cloud. We will use the Dweet.io cloud service here, which is very convenient for logging data online:

`http://dweet.io/`

As the code for this part is very long, we will only see the important parts here. You can of course get all the code from the GitHub repository for this project at `https://github.com/openhomeautomation/iot-esp8266`.

Again all the measurements are done inside the `loop()` function of the sketch, which makes the code repeat every 10 seconds, using a `delay()` function.

Inside this loop, we connect to the Dweet.io server with:

```
WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
Serial.println("connection failed");
return;
}
```

Then, we read the data from the sensor with:

```
int h = dht.readHumidity();
int t = dht.readTemperature();
```

After that, we send data to the Dweet.io server with:

```
client.print(String("GET /dweet/for/myesp8266?temperature=") +
  String(t) + "&humidity=" + String(h) + " HTTP/1.1\r\n" +
"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
```

You might want to replace `myesp8266` here, which is your device name on Dweet.io. Use a complicated name (just like a password) to make sure you are creating a unique device on Dweet.io.

We also print any data received on the serial port with:

```
while(client.available()){
  String line = client.readStringUntil('\r');
Serial.print(line);
}
```

Note that you also need to modify the code to insert your own Wi-Fi network name and password. You can now upload the code to the ESP8266 board, using the instructions that we saw earlier, and open the Serial monitor.

You should see that every 10 seconds, the request is sent to the `Dweet.io server`, and you get the answer back:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 147
Date: Mon, 16 Mar 2015 10:03:37 GMT
Connection: keep-alive

{"this":"succeeded","by":"dweeting","the":"dweet",
"with":{"thing":"myesp8266","created":"2015-03-16T10:03:37.053Z",
"content":{"temperature":24, "humidity":39}
  }
}
```

If you can see the `succeeded` message, congratulations, you just logged data in the cloud with your ESP8266 chip!

Displaying data using Freeboard.io

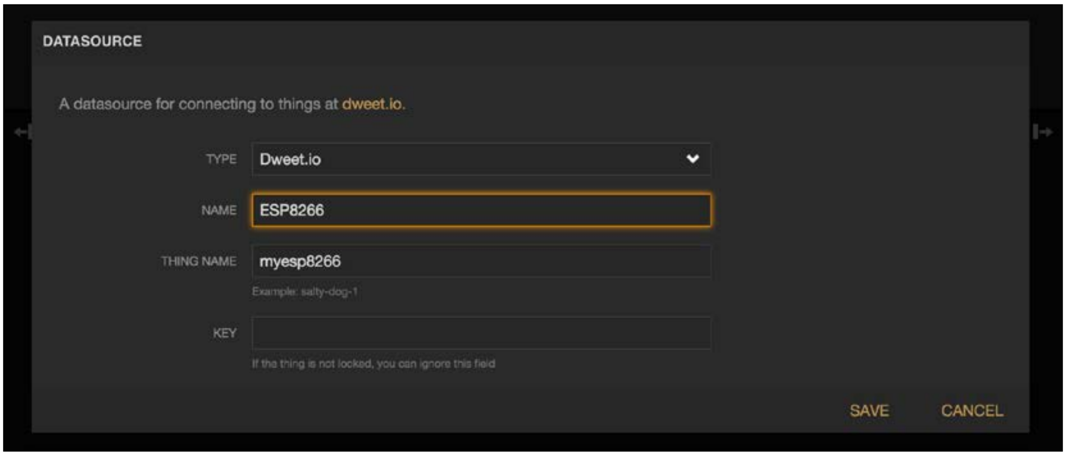
Now, we would like to actually display the recorded data inside a dashboard that can be accessed from anywhere in the world. For that, we are going to use a service that I love to use along with Dweet.io: Freeboard.io.

Let's get started with using Freeboard.io:

1. First, create an account there by going to:

   `https://www.freeboard.io/`

2. Then, create a new dashboard, and inside this dashboard, create a new `datasource`. Link this `datasource` to your **Dweet.io** thing that you defined in the ESP8266 code:

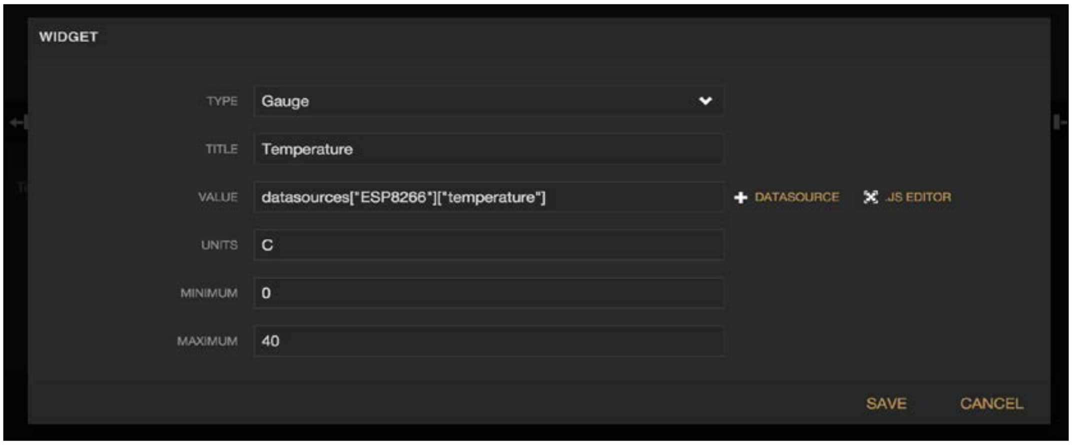


DATASOURCE

A datasource for connecting to things at dweet.io.

TYPE	Dweet.io
NAME	ESP8266
THING NAME	myesp8266
	Example: salty-dog-1
KEY	
	If the thing is not locked, you can ignore this field

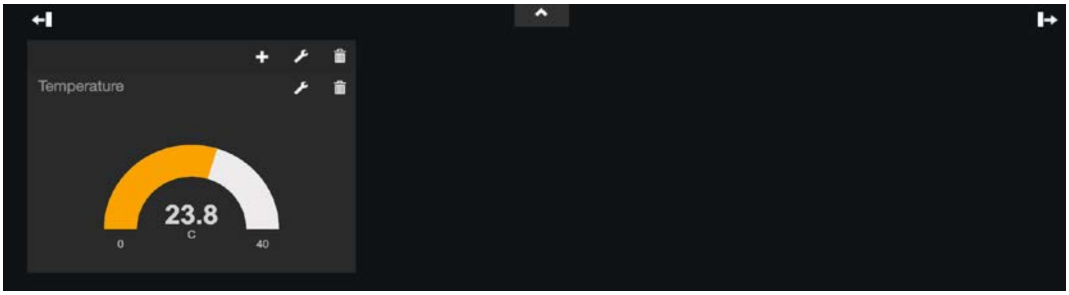
3. Now create a new Gauge widget that we will use to display the temperature. Give it a name, and link it to the temperature field of our datasource:



WIDGET

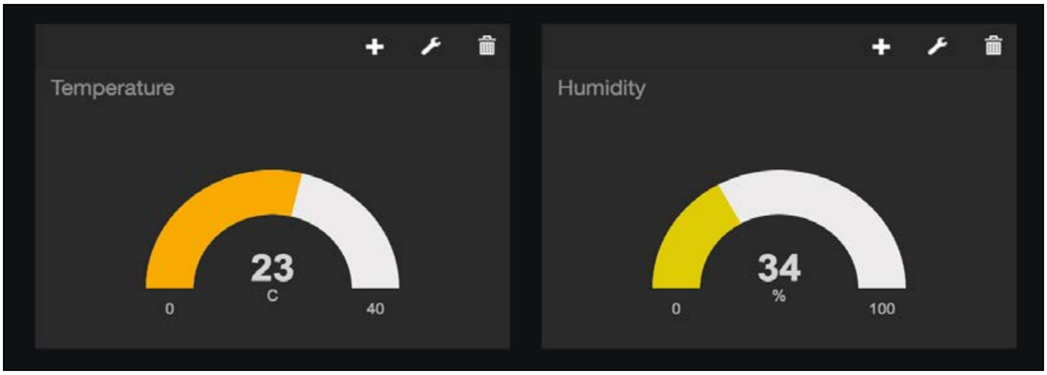
TYPE	Gauge
TITLE	Temperature
VALUE	datasources["ESP8266"]["temperature"]
UNITS	C
MINIMUM	0
MAXIMUM	40

This is the final result:



You should see that the temperature data coming from the ESP8266 is logged every 10 seconds and is immediately displayed inside the Freeboard.io panel. Congratulations, you have built a very small and cheap and temperature sensor that logs data in the cloud!

You can then do the same with the humidity measurements and also display them on this dashboard:


Temperature
23
C
0
40
Humidity
34
%
0
100