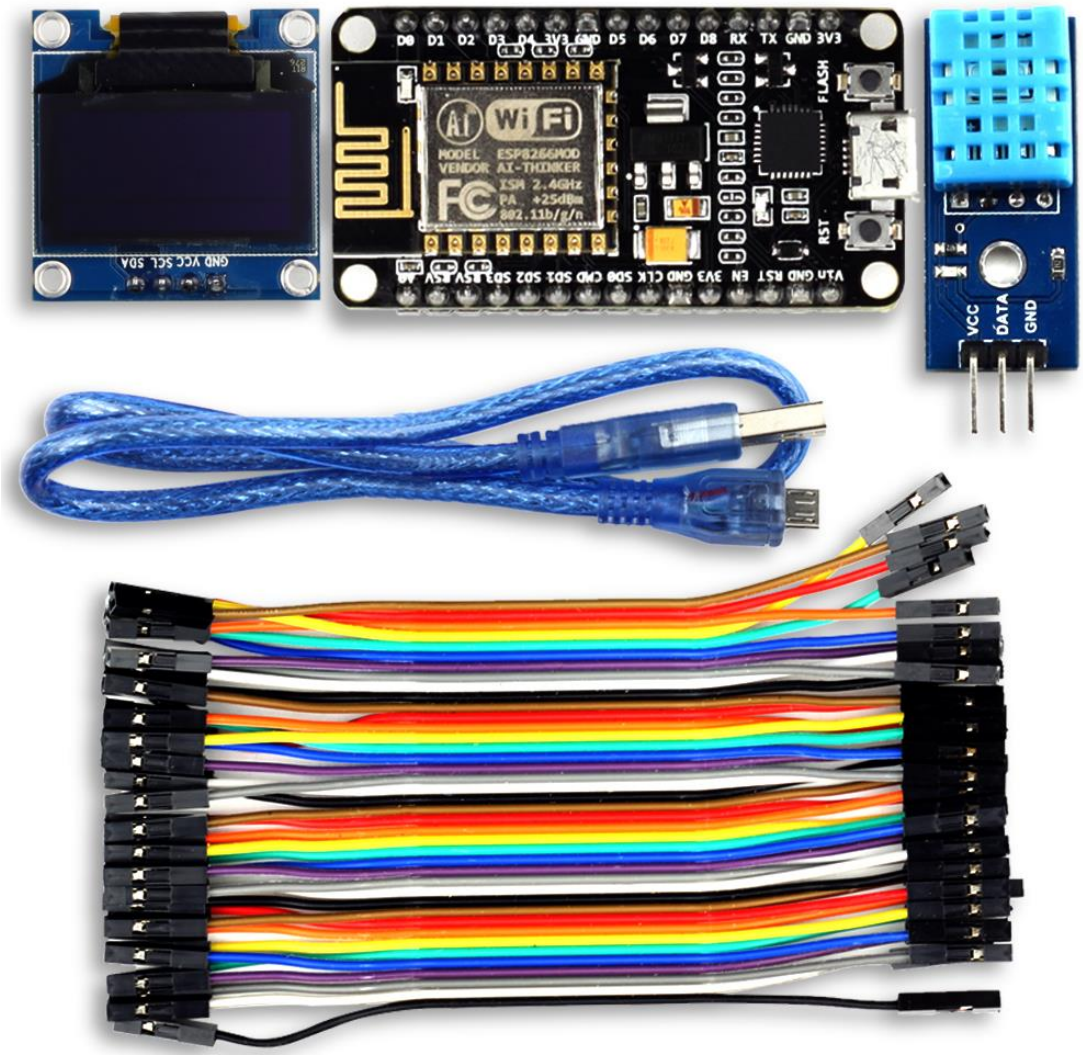
ESP8266 Weather Station

User Guide

V 1.0 Mar 2017



Contents

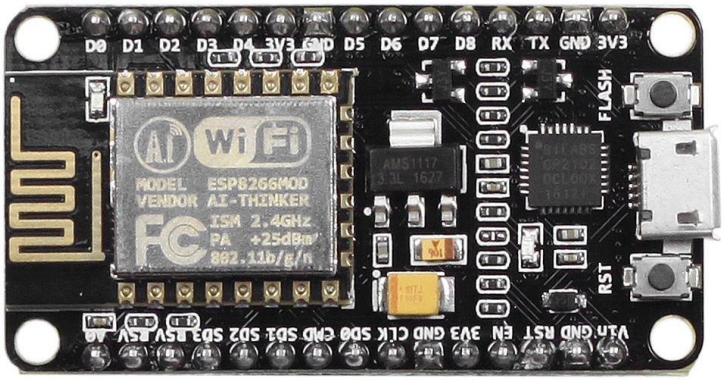
1.Introduction

The ESP8266 WeatherStation is one easy way to get started with the ESP8266 and IoT. The included guide helps you step-by-step to setup an internet connected weather station which shows current and forecasted weather information.

The Uctronics 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.

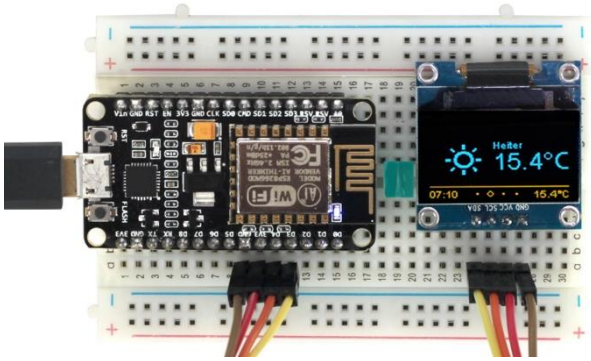
2.Assembly

A. ESP8266 Module



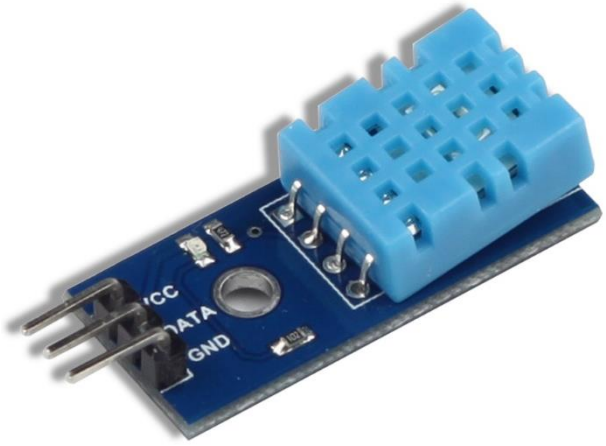
There are many different modules available based on ESP8266s, they differ in a number of aspects such as the quantity of available GPIO pins or if they can be programmed easily without need of an additional Serial-to-USB converter. If you are a beginer I suggest you use a developer-friendly module like the NodeMCU V1.0 or the Wemos D1 mini. They come with a USB connector and have the maximum number 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.

B. 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 0.96" displays. Displays with SPI interface will consume more of your free GPIO pins.

C. DHT11 Humidity & Temperature Sensor



This DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.

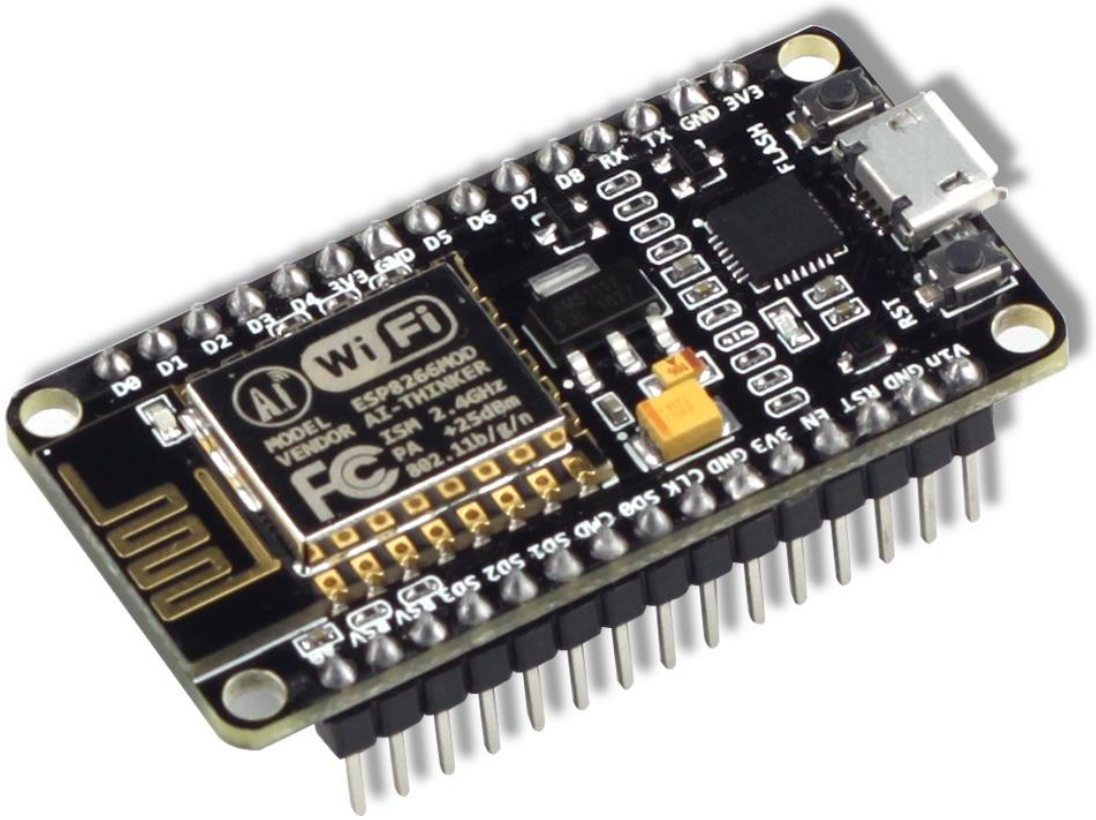
D. Wires & Cables

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, 7.87" (20cm) is usually enough.

To program the ESP8266 module you will also need a micro 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.

3.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.



A. 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. This ESP8266 development module has built in a high-quality Labs CP2102 Single-Chip USB to UART Bridge with Micro-USB connector that baud rates up to 921600. You can download and install the driver from here:

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

B. 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.

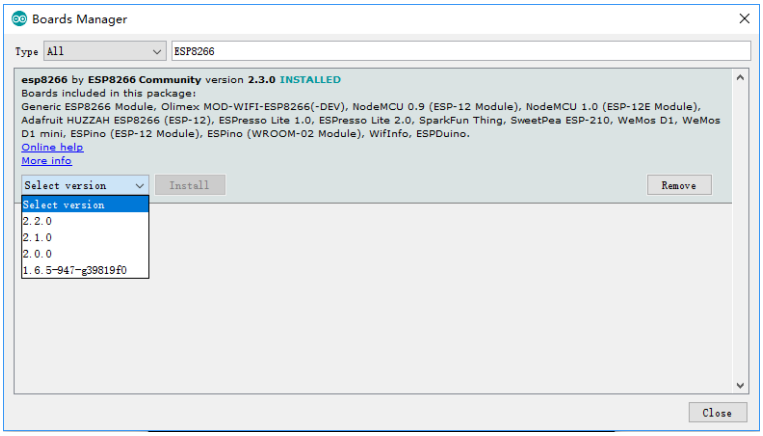
To install the Arduino IDE, go to https://www.arduino.cc/en/Main/Software and download the latest version matching your operating system:

- 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.

C. 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. The Arduino IDE has a wonderful feature, Board Manager, to save you the hassle of downloading many different files and copying them into obscure locations. 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:

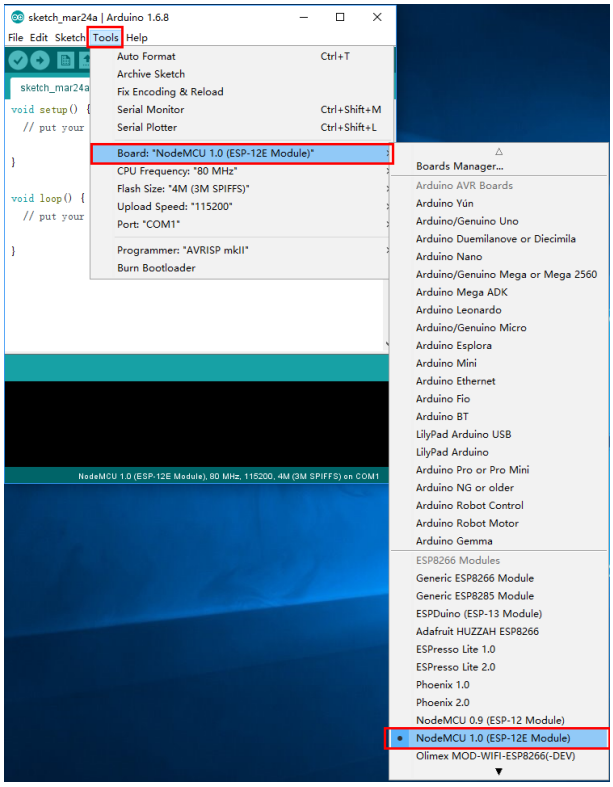
- Open the Arduino IDE
- Go to your preferences/settings and in the text box Additional Board Manager URLs enter this URL: http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Now go to Tools > Board: ··· > Boards Manager···, search for the ESP8266 board and click Install.
- Get a coffee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

D. 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.

E. Setting the Correct Port

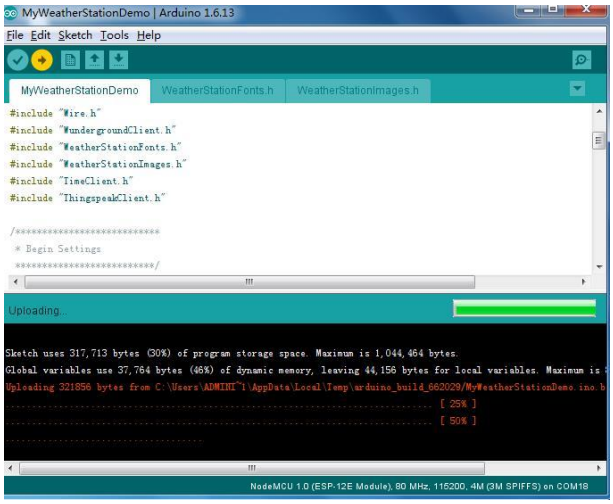
Serial interface: At the hardware level the ESP8266 is programmed through a serial interface. In short this is a very common communication interface which normally requires three lines: transmit (TX), receive (RX) and ground (GND). Both devices involved in the communication need to agree on the rate the characters are sent over the wire. This rate is usually measured in BAUD. 10 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 and the board is plugged into your computer 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 socket. If that doesn't help consider restarting your computer··· Make sure that you installed the driver as mentioned in the chapter about drivers.

F. Testing the Setup: WiFi Scanner

Thanks for bearing with the complicated preparation until getting 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 andupload the binary to the ESP. It might look something like this:



Wifi Scanner Output

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 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:

Serial Console of the Wifi Scanner

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

G. 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.

4.ESP8266 Programming Basics

In this chapter, we will have a look at the building blocks of an Arduino sketch. This will help you to understand and modify the Weather Station which we will build in the next chapter. If you just want to get the WeatherStation running you can skip this chapter.

A. Preparation

In this chapter, we will work with exercises which you can download from GitHub. They contain several Arduino projects for the ESP8266. For an exercise open the related project in your Arduino IDE and try to solve the given task. If you get stuck or want to see an alternative solution open the project which ends with "_Solution":

- Exercise_04_01: contains the first exercise in chapter 4
- Exercise_04_01_Solution: contains a possible solution

Now download the Zip file from GitHub and extract it in a place you will find it later. There is a green "Clone or download" button which lets you download a Zip file:
https://github.com/squix78/esp8266-getting-started

B. The Arduino Sketch

The Arduino platform was built with the beginner in mind. Compared to a normal C program the Arduino IDE hides a few things from you to simplify the setup. First of all you do not have to create a makefile to build your code into an executable binary. The Arduino IDE also includes a default header file for you: #include "Arduino.h". This contains all definitions needed for a regular Arduino program.

Another important change compared to a regular C/C++ program are the two default functions setup() and loop(). The first will be only called once during startup, while the loop() method will be called repeatedly. On a normal Arduino hardware (ATmega chip) you can theoretically write code and never leave the loop() method again. The ESP8266 is a bit different here. If your operations run for too much time a so-called watchdog will reset the ESP8266. You can prevent this by allowing the controller to do important operations while you are still in the main loop. Calling yield() or delay(ms) will do this.

C. Hello World: The serial console

Every self-respecting programming tutorial starts with a “Hello World” program. And I don’t want to break with this tradition here. A Hello-World program usually does no more than printing these two words somewhere on the screen. But we are programming a microcontroller which does not have a screen yet. So where can we display the text? We will use the Serial object to do that. While you are developing a program on the ESP8266, the microcontroller is connected to the computer that the Arduino IDE is running on. We use this connection to write a new binary onto the flash memory of the ESP8266. And while our program is running we can also use it to write messages from the ESP8266 back to our computer.

Using the Serial object is fairly easy. You have to initialize it first:

```
Serial.begin(115200);
```

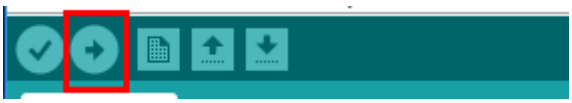
This tells the Serial object that you want to communicate with a baud rate of 115200. Remember to set the same transfer rate later in the serial console on your computer. Both partners in the communication need to have the same speed settings or you will just see garbage. If you want to send a message from your program to your computer you just do this:

```
Serial.print("Hello ");
Serial.println("World");
```

Please have a look at the little difference between the first and the second line. The first uses a method called print() and the second println(). The only difference is that the latter adds a line break to the output.

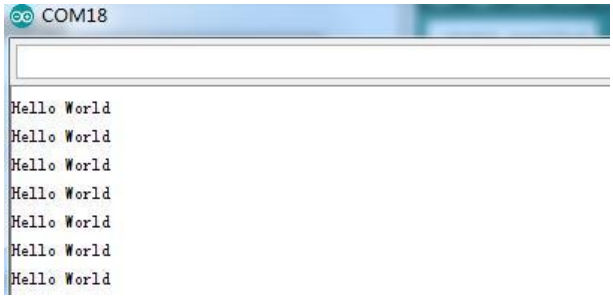
Exercise 04.01: Hello world!

Now it is time to write our first program. Open the project Exercise_04_01 in your Arduino IDE and fill in the required code to print “1. Hello World”, “2. Hello World” etc. Remember that you only need to initialize the Serial object once, while you’ll have to print “<number>. Hello world” as long as the code runs. Once you are happy with your solution upload the sketch to your Arduino by clicking



If that was successful open the serial console by clicking on the magnifying glass:

Now your output should look something like this:



If you want to learn more about the Serial object you can find more in the following link:

http://esp8266.github.io/Arduino/versions/2.3.0/doc/reference.html#serial

The exercise contains another important built-in function:

delay(1000);

This instructs the processor to wait 1000 milliseconds (1 second) before continuing with the execution. As mentioned earlier with this command it also gives the processor time to handle other tasks, such as receiving or sending network packets over WiFi. In this context a call to yield() does the same as delay(0).

D. Input/Output: GPIO pins

Now that we can talk to our microcontroller over the serial line it is time to interact with the real world. Our ESP8266 is equipped with several so-called General Purpose Input Output or in short GPIO pins. They can be used for many different applications such as sensing and generating digital signals of the 3.3 Volt range. This is important if you plan to use an external component with your ESP8266: hardware designed for older Arduinos often uses the 5V (CMOS) range. Using such a device without a logic level shifter might destroy your ESP8266.

Using a GPIO pin is quite easy: first you tell the microcontroller if you want to read or write from the pin. Then you do it. Here is the code for reading:

```
pinMode(PIN, INPUT);
int state = digitalRead(PIN);
```

Unless you want to change the mode of a pin you only need to call pinMode() once. Please note that depending on the pin you can also use INPUT_PULLUP or INPUT_PULLDOWN. Writing to a pin is not much different:

```
pinMode(PIN, OUTPUT);
digitalWrite(PIN, HIGH); // or
digitalWrite(PIN, LOW);
```

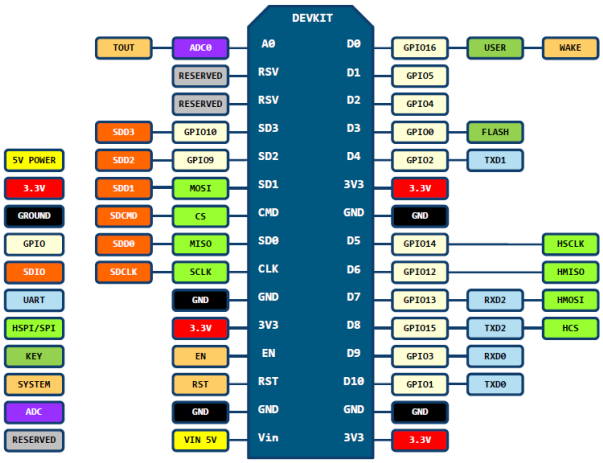
The second statement will show a HIGH level on PIN which will be 3.3V. The third statement will set the pin to LOW which is 0V.

What values for PIN can you use? If you are using a generic ESP8266 module (not a NodeMCU) your pins will be labeled GPIO0, GPIO1, etc. To use pin GPIO0 you would write digitalWrite(0, HIGH); If you are using a NodeMCU things get a little bit more complicated. The original creators of the NodeMCU LUA firmware and the development module of the same name had the idea to give the pins different names. They are called D0, D1, etc. That by itself would not be confusing yet but they are not using the same digits, e.g. GPIO1 is not equal to D1. Here is a table to map the pins:

Raw Module Name	NodeMCU & Wemos Name	Raw Module Name	NodeMCU & Wemos Name
GPIO0	D3	GPIO9	D11
GPIO1	D10	GPIO10	D12
GPIO2	D4	GPIO11	N/A
GPIO3	D9	GPIO12	D6
GPIO4	D2	GPIO13	D7

GPIO5	D1	GPIO14	D5
GPIO6	N/A	GPIO15	D8
GPIO7	N/A	GPIO16	D0
GPIO8	N/A		

PIN DEFINITION



NodeMCU DevKit Pin Map https://github.com/nodemcu/nodemcu-devkit-v1.0

D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Exercise 04.02: Can't touch this!?

In this exercise, you are going to read from the button on your NodeMCU labelled with FLASH. This button is connected to the D3 pin. Put the D3 pin to input mode in the setup() method and read from the pin repeatedly in the loop() and write the results to the console.

Exercise 04.03: Led it shine!

Let's control an LED! Your NodeMCU has one built in and it is connected to D0. Like in the previous example set the pin to the correct mode and then toggle it once per second.

Exercise 04.04: Every breath you take! (Bonus for Experts)

Pure blinking is boring. We want our little ESP8266 to live! Control the intensity of the LED and make it look as if the ESP8266 was breathing. Note: if the LED was on another pin than D0 we would use pulse width modulation (PWM) and the writeAnalog(PIN, INTENSITY) method. But this feature is not available on D0 so you will have to do this in your code.

E. Interrupts

Depending on your age you might remember interrupts from your PC. They were always important to get your sound card to play beautiful music. The ESP8266 can also be controlled by interrupts. In the previous exercises we were checking regularly for the state of a GPIO pin. This is fine if you are not doing anything else in the main loop. But you might miss a change in a state if it is very short, and that is were the interrupts can help. With an interrupt handler you can tell the processor that you are interested in a specific type of change and a given pin. This is how it works:

```
void buttonPressed() {
...
}

void setup() {
pinMode(PIN, INPUT);
attachInterrupt(digitalPinToInterrupt(PIN), buttonPressed, CHANGE);
}
```

buttonPressed() is a method without parameters that gets called when there is a change on PIN. Instead of CHANGE you can also use RISING which triggers the callback when the pin changes from LOW to HIGH, or FALLING for a change in the opposite direction. Please do not execute long tasks in the callback method. The ESP's watchdog will reset the processor if calling the interrupt takes too much time. You should not do much more than changing a flag.

Exercise 04.05: I don't want to miss a thing!

In this exercise we will setup an interrupt and turn an LED on and off with every press to the button. See what happens when you use different interrupt modes like RISING, FALLING and CHANGE.

F. Measuring analog signals

So far we can read and write the digital states HIGHand LOW, but what if we want to deal with analog signals? The ESP has one Analog To Digital Converter (ADC) which can be used to measure voltage in the range 0 - 1V. To do that use the following command:

```
unsigned value = analogRead(A0);
```

You can also use the ADC to measure the input voltage without any additional wiring. You have to instruct the processor that you want to measure the supply voltage rather than the value on A0 with a special command outside the setup() and loop() method. Here is an example:

```
ADC_MODE(ADC_VCC);

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

void loop() {
Serial.println(ESP.getVcc());
delay(500);
}
```

G. WiFi

The last few chapters were all about built-in functions of the Arduino/ESP8266 platform. Now we will start using libraries which are part of the platform and are already installed. So how can we use the WiFi module of the ESP8266? First of all you need to know that the ESP8266 can operate as a WiFi client (like a smartphone or laptop) and/or as an access point (like a WiFi router or extender). You can set this mode with:

WiFi.mode(m);

where m must be one of the following modes: WIFI_AP (access point), WIFI_STA (client), WIFI_AP_STA(AP and client) or WIFI_OFF.

Now let's connect to your access point:

WiFi.begin(WIFI_SSID, WIFI_PWD);

This will connect you to an access point given its SSID and the password. Please note that this call is not blocking. This means that the code will immediately proceed to the next instruction whether the ESP successfully connects to the access point or not.

H. HTTP

By connecting to the internet you can exchange data between your ESP8266 and the network. Let's look at how we can load content from a web server using the Hyper Text Transfer Protocol (HTTP). This protocol is the foundation of the World Wide Web.

```
#include <ESP8266WiFi.h>

char* ssid = "SSID";
const char* password = "PASSW0RD";

const char* host = "www.squix.org";

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

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

  WiFi.begin(ssid, password);

  // Wait until WiFi is connected
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

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

    void loop() {
    delay(5000);
```

```
30
31    Serial.print("connecting to ");
32    Serial.println(host);
33
34    // Use WiFiClient class to create TCP connections
35    WiFiClient client;
36    const int httpPort = 80;
37    if (!client.connect(host, httpPort)) {
38      Serial.println("connection failed");
39      return;
40  }
41
42    // We now create a URI for the request
43    String url = "/guide/";
44
45    Serial.print("Requesting URL: ");
46    Serial.println(url);
47
48    // This will send the request to the server
49    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
50                 "Host: " + host + "\r\n" +
51                 "Connection: close\r\n\r\n");
52
53    unsigned long timeout = millis();
54    while (client.available() == 0) {
55      if (millis() - timeout > 5000) {
56        Serial.println(">>> Client Timeout !");
57        client.stop();
58        return;
59      }
60  }
61
62    // Read all the lines of the reply from server and print them to Serial
63    while(client.available()){
```

```
64      String line = client.readStringUntil('\r');
65      Serial.print(line);
66    }
67
68  }
```

How does this work? First we define the SSID and password of the WiFi access point we want to connect to. Please note that there are better ways to do that. The WiFiManager (https://github.com/tzapu/WiFiManager) for instance starts the ESP8266 as an access point if it cannot connect to any SSID. You then use your smartphone to configure the WiFi credentials and there is no need to hard code these into your firmware. But for the sake of simplicity let's ignore this here.

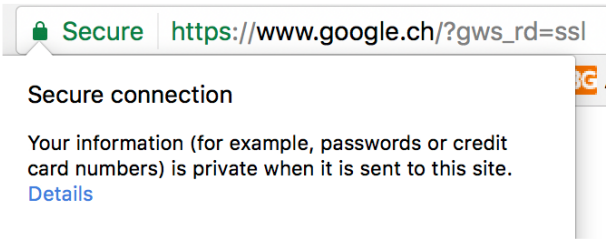
On *line 14* we start connecting to the defined access point and wait until the connection is established. After all there is no point to send requests to a server if the network connection is not confirmed yet.

Line 49 sends the request to the server. The command GET /guide/ HTTP/1.1\r\n might look strange to you. This is how your browser talks to the web server. GET is the command for the webserver, /guide/ is the resource on the server we want to get and HTTP/1.1 is the protocol that we are using. If you are interested how this works in detail have a look at this Wikipedia article: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

On *line 63* we print out the response line by line as long as there is text coming in. Sadly this is quite complicated. Especially if we want to add encryption in the form of SSL to the connection. This protects your data and makes sure that you are talking to the right server. With the following command we can verify that the host matches the given SHA1 fingerprint.

```
1  if (client.verify(fingerprint, host)) {
2    Serial.println("certificate matches");
3  } else {
4    Serial.println("certificate doesn't match");
5    return;
6  }
```

How can you find this fingerprint? Your browser can help you with this. I will show it with Chrome. First open the page you need the fingerprint for, in my case www.google.ch. Then click on the little lock symbol and then on Details:



A click on View Certificate will bring up the detail window about Google's certificate:

Scroll down to the bottom of the window and copy the value behind SHA1. This is the fingerprint to verify that you are actually talking to www.google.ch.

Exercise 04.06: Better safe than sorry!

In this exercise we will start with the same program as I included earlier in this chapter. But now you are going to change the code to receive the search site from google on a secure channel. Complete the following tasks:

- Change the host from www.squix.org to www.google.ch
- Get the SHA1 fingerprint for www.google.ch
- Add a check that this fingerprint matches www.google.ch

5.The ESP8266 WeatherStation

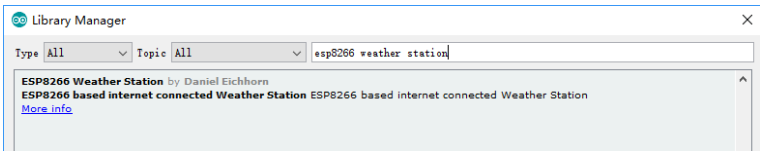
In this chapter we will get the WeatherStation to run. We will install several libraries used for setting up access to the internet, for reading and parsing the data from the service providing your local weather forecasts, as well as a library to display the data on the OLED display. Then we will adjust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

A. Installing the libraries

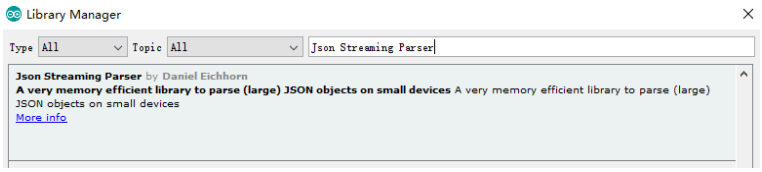
Libraries: If you are new to programming you might ask what libraries are. When we develop programs we use libraries to not have to invent the wheel over and over again. Libraries contain functionality that might be used in different places without creating copies of code which is hard to maintain. So 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 the program gets from the weather forecast service. And the third is needed to use the beautiful OLED display.

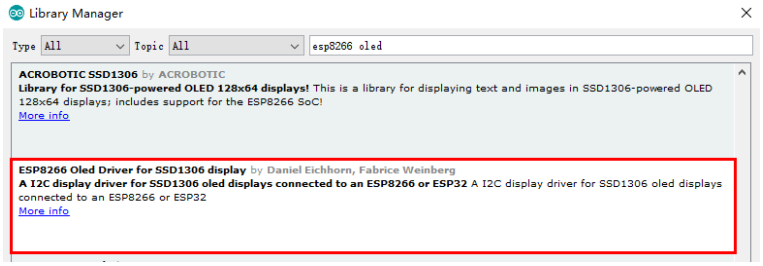
Go to Sketch > Include Library··· > Manage Libraries··· and install the following three libraries. Make sure that you always have the latest version of the libraries installed. Users have reported many issues which could be reduced by simply updating the library. Also make sure that you only have one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



ESP8266 Weather Station Library



Json Streaming Parser Library



SSD1306 OLED Library

B. 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.

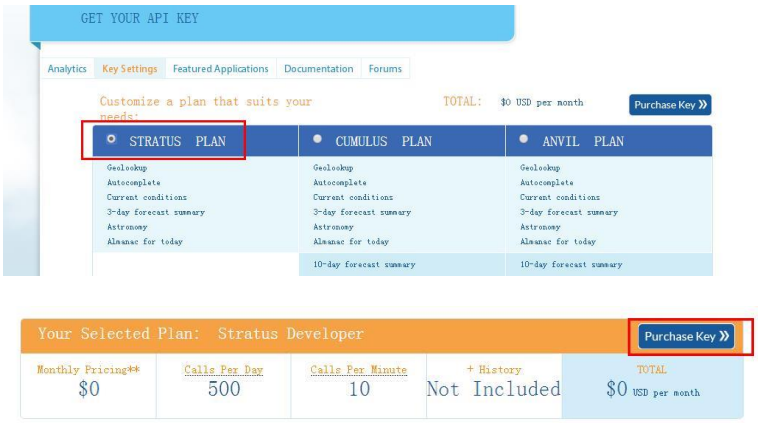
C. Getting the Wunderground API Key

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:



Wunderground API Page

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

D. Configuring the Weather Station

Earlier when you chose the WeatherStation example you created a copy of the code included in thelibrary. 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 goodvalue, 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-hour offset, thanks to the user who lives in such a time zone and made me implement that. (Ignorance is 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/APIKEY/conditions/q/CA/San_Francisco.json and replace APIKEY with yours and CA and San_Francisco with your state or country and city.

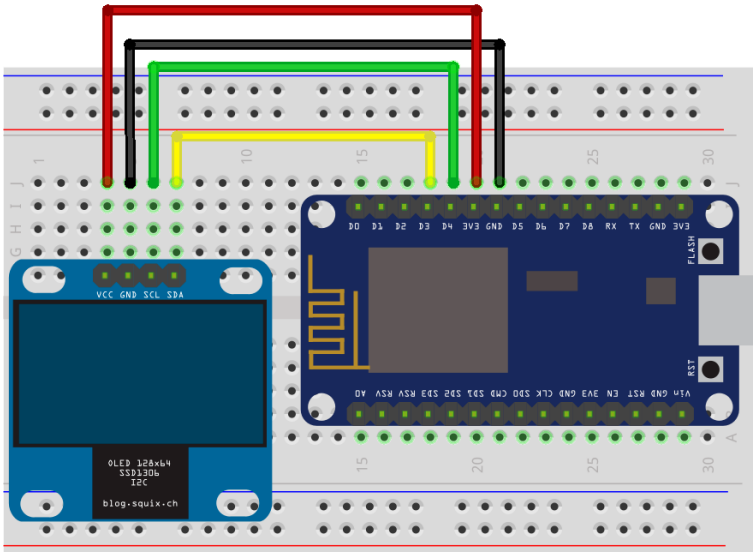
For the moment ignore the ThingSpeak settings. You might use it in a future project (have a look at the "More Projects" chapter). Now we are almost ready to get the weather station running on the ESP8266 for the first time. But we need to wire the display to the NodeMCU first.

E. 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 (colored) 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.



Wiring the Weather Station

Please note: there are versions of the OLED display with swapped GND and VCC pins. Becareful to connect them according to the printed labels, not (necessarily) 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 name 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++.

F. 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 now be trying 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 of what else you can build with the WeatherStation hardware.

G. 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.

6.The WeatherStation Code Explained

In this chapter we will have a look at the building blocks of the WeatherStation. This project is a relatively complex piece of code and I hope to improve this chapter over time with new details.

A. The JSON Streaming Parser

You might not know it but the most important puzzle piece for the WeatherStation project is a thing called a streaming parser. What is a streaming parser? You are most certainly using parsers every day. A parser is a piece of code that analyses an input (text, document) by reading in its content. To do that the parser has knowledge about the structure of the text, sometimes called a syntax. The syntax is like the grammar of your natural language. A web browser you are using to read news uses an HTML parser to understand the tags that are downloaded from the webserver and then put into a visualization with formatted text, pictures and links.

So now that we roughly understand what a parser is the next question would be what is a streaming parser? With a modern smartphone or desktop computer we often don't need streaming parsers anymore, we use document object model parsers (DOM) parsers instead. A DOM parser creates a tree-like structure of the document it parses, keeps this structure in memory and makes it available for the code that does something meaningful with it. DOM parsers are very easy to use, fast and convenient. But this convenience comes at the price of memory requirements. The DOM parser needs a lot of memory, since it keeps the whole document in memory until it is no longer used. If you have a lot of RAM and your documents are not that big this is perfectly fine. But if the documents are big compared to the available (heap) memory you might run into a serious problem.

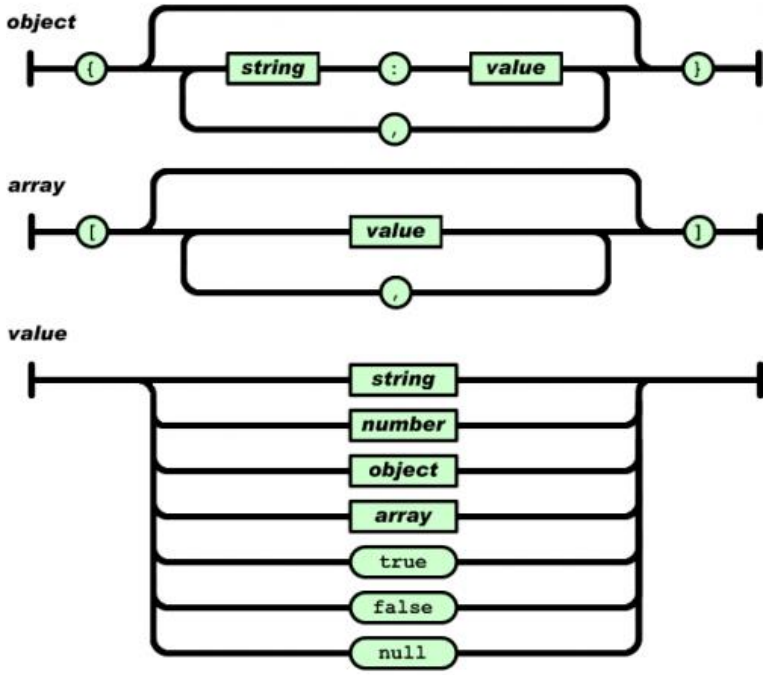
Imagine the parser to be something like a water meter and we are comparing now two different types of meters. A water meter which works like a DOM parser needs a bucket and measures the amount of water by filling the bucket and then measuring the weight of the water in the bucket. If there is a lot of water then the bucket must be big. A water meter which works like a streaming parser measures the water while it flows through and doesn't care what happens to the water afterwards. The bucket in this analogy is the heap or working memory of your microcontroller, the water is the stream of bits and bytes that you receive, either from the file system or from a remote server. And the parser does not just measure the amount of

bits and bytes but also tries to understand the content. The streaming parser doesn't care how big the document (or the amount of water) is, it just takes out what it needs from the stream. Streaming parsers are also referred to as event-based parsers since they react to certain events in the data stream. DOM parsers are referred to as tree-based parsers since they build a full representation of the document in the tree-like structure. In an HTML tree the html element would be the root of the tree, and the body tag a fork in that tree.

B. The Grammar

The following image describes the grammar of a valid JSON object in a very concise way. It means that a JSON object knows three basic types:

- object
- array and
- value



JSON grammar

Objects always start and end with curly brackets. They can be either empty (line to the top) or contain string/value pairs, separated by a colon. These pairs can be repeated by adding a comma between them. Arrays start and end with square brackets. They can be either empty or contain a value. At this point we don't know yet what a value is. Values in an array can be repeated and must be separated by a comma. Values were already used for the two previous definitions and here lies the power of this kind of grammar: because a value can contain a simple text, a number, an object (yes, the object we defined before!!!), an array (also defined before), booleans or a null value.

This is so powerful because we are reusing the definitions and we are nesting them within each other: an object can contain a value; a value can contain an array or an object. And finally, an array can contain a value, repeatedly! Isn't this beautiful?

C. The JSON Streaming Parser Library

Why would we want to use a streaming parser on the ESP8266? Embedded devices usually have very limited resources available. One scarce resource is the heap memory. Many of the REST APIs I am using in my projects provide big response objects, but we are usually just interested in a small fraction of it. As mentioned earlier, a tree-based parser would load the whole document into memory and make it available once the document stream has ended. And this would just crash the ESP8266 pretty quickly; it does not have the resources to keep 200kb on the heap.

This made me port a PHP JSON parser over to C++ and make it available as a library, mostly to be used in my own projects. Let's have a look at the header file of the JsonListener:

```
1  class JsonListener {
2  public:
3  virtual void whitespace(char c) = 0;
4  virtual void startDocument() = 0;
5  virtual void key(String key) = 0;
6  virtual void value(String value) = 0;
7  virtual void endArray() = 0;
8  virtual void endObject() = 0;
9  virtual void endDocument() = 0;
10 virtual void startArray() = 0;
11 virtual void startObject() = 0;
12 };
```

The methods here are callback methods which will get invoked if the respective event happens while parsing the document. Let's start with an example. For the JSON object {"name": "Eichhorn"} we get the following invocations:

- startDocument(): we start receiving a json document
- startObject(): the json object starts with "{"
- key("name"): the parser detected a key object which contains "name"
- value("Eichhorn"): the parser detected a value containing "Eichhorn"
- endObject(): the object ends with "}"
- endDocument(): the stream of data ends and so does the document

I often just implement (AKA "write code") for the key() and the value() methods. In the key() method I store the value of the key parameter. Then later in the value() method I check what

the last key was I had seen and then I store the value in the appropriate variable. For the example from before I would do

```
1  void ExampleListener::key(String key) {
2  currentKey_ = key;
3  }
4
5  void ExampleListener::value(String key) {
6  if (currentKey_ == "name") {
7  name_ = value;
8  } else if (currentKey_ == "city") {
9  city_ = value;
10  }
11  }
```

In the stream of the object {“name”: “Eichhorn”} we will first get a call to the method key() with the value “name” which we store in currentKey_. Next the parser will detect a value and call our value() method with the value “Eichhorn”. The parser can now make the connection (or create a context) that after the key “name” the value “Eichhorn” should be stored in the member variable name_.

If this example was too simple then have a look here: https://github.com/squix78/esp8266-weatherstation/blob/master/WundergroundClient.cpp This is the code which parses the responses from Wunderground for my WeatherStation.

D. Conclusion

For a document or object of the size we had in the example a streaming parser is usually an extreme overkill. It is complicated to use, requires you to write a lot of code and is memory-wise probably even worse than a tree parser. It is only recommended to implement a streaming parser if you have big objects or if you just don't know how big your object might be. In those cases a streaming parser will be a good friend, since it only requires memory for the objects you actually want to use from the whole big document. You can find my library here: https://github.com/squix78/json-streamingparser

7.Collecting and Displaying Local Data

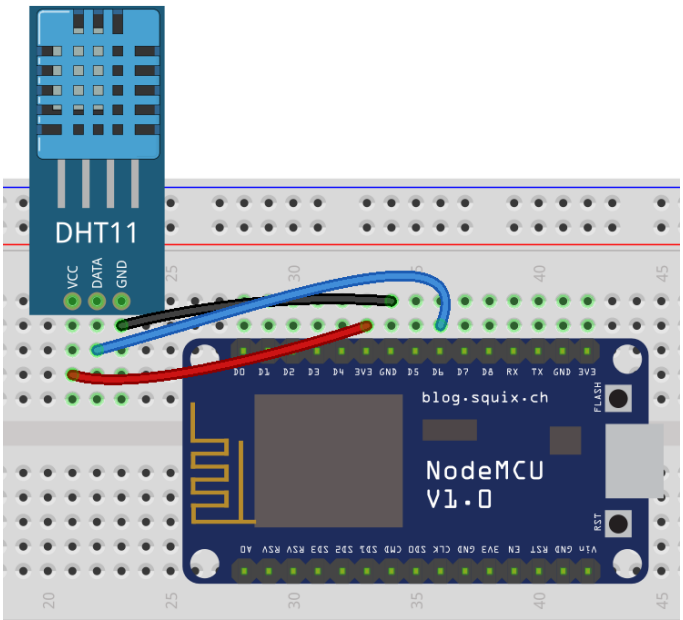
So far we have created a device which connects to the internet and uses an API to fetch weather data. But this is not really an Internet-of-Things application; after all we are just displaying data. In this chapter, we will change that.

A. The Climate Node Setup

This project will enable you to collect humidity and temperature in one room of your house or apartment and display the values in another room on the WeatherStation. In order to do so you will need additional hardware which is not included in the basic WeatherStation Starter Kit:

- An additional ESP8266 module, ideally a NodeMCU
- A DHT11 or DHT22 humidity and temperature breakout module

I will call this combo “The Climate Node”. Now use the female-to-female jumpers to connect them like this:

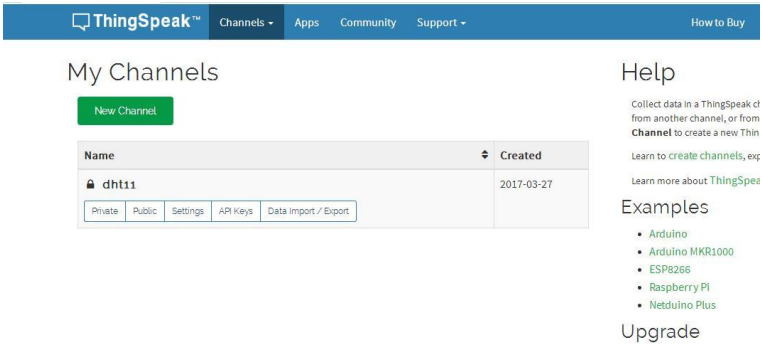


Wiring for the Climate Node

B. Thingspeak Setup

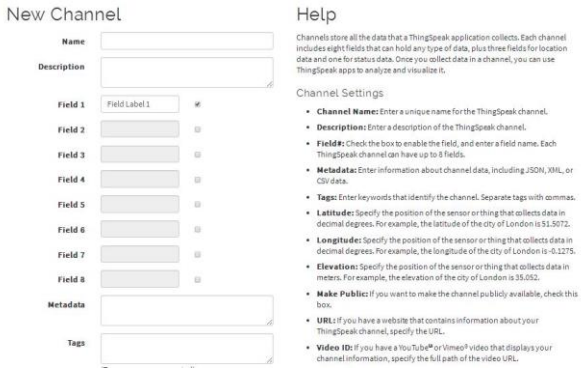
Thingspeak is a free cloud service which allows you to easily post (sensor) data, to visualize it and to retrieve it again using simple HTTP methods. I'd like to point out that you could also use Wunderground to store your climate data and it probably would be very easy to do as well. But I believe that for education purposes Thingspeak permits more degrees of freedom. After all you could also send readouts from a motion sensor and visualize this information on a chart. Thingspeak has some nice additional features which lets you program webhooks to trigger a push notification on your cell phone, etc.

So first of all you have to sign up for a (free) account on Thingspeak. Go to https://thingspeak.com/users/sign_up and create the account. After you completed that process, log in to your new account and go to My Channels:



Navigate to My Channel

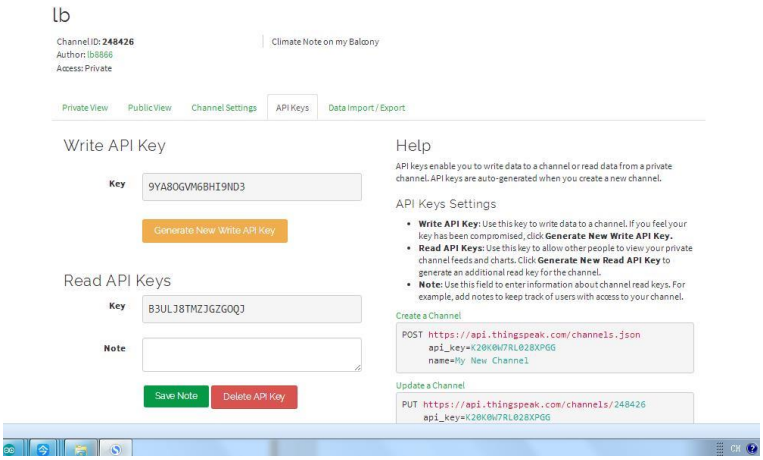
Then you click on the New Channel button, and fill out the form:



Fill out the New Channel Form

Explanation: The name just helps you to recognize your channel among many others that you might create over time. The important parts are the field names. These names will later show up in the chart and with this you are telling Thingspeak that the value you send later with the field1 attribute should be displayed as "Temperature".

Now navigate to the API Keys tab and note the two generated keys:



Thingspeak API keys

The first one will allow you to write to this channel in Thingspeak, and the second one will allow you to read from it. Treat them as secrets and with care. Others might be able to spam your channel or to "steal" your data. We will use these keys soon enough··· Also note the channel ID on top of the screen, in my case 76642.

C. Programming the Climate Node

Now we have all the ingredients to post the climate data to Thingspeak. We just have to program the ESP8266 accordingly. Go to: https://github.com/supprot/ArduCAM_esp8266-dht-thingspeak-logger.git and download the code as a Zip file (or optionally do a GitHub checkout).

Now adapt the settings to your needs: In particular the Wifi settings and the Thingspeak API key have to be updated. Take the Write API key from the previous step. For testing you might also play with the update interval which is a number in seconds. Please be aware that the minimum update interval in Thingspeak is about 15 seconds. Pick a smaller interval and your updates will be ignored. Now flash your program to the Node MCU and your Climate Node should start logging. To check the results you can go back to Thingspeak and look at the charts:

The Charts of your Climate Node

D. Displaying the data on the WeatherStation

Now to the real easy part. Like a cook in the TV kitchen I have prepared this step a long time ago (possibly to your confusion;-)). The WeatherStationDemo that comes as an example with the WeatherStation library already contains everything needed to display your own Climate Node data!

Look for these lines in the demo and replace the read api key and the channel ID with the ones you got in the Thingspeak step:

```
1 //Thingspeak Settings
2 const String THINGSPEAK_CHANNEL_ID = "248426";
3 const String THINGSPEAK_API_READ_KEY = "B3ULJ8TMZJGZGOQJ";
```

If you didn't remove the climate node section in your weather station code you just need to flash your WeatherStation with the updated API key and channel ID and voilà: you just successfully sent the temperature and humidity data from the next room once around the world just to display it on a tiny OLED display. I know some people (including my wife) who wouldn't be impressed by that at all.

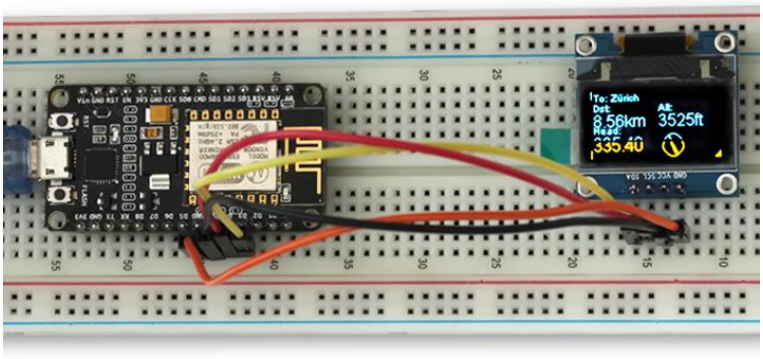
Explanation: This last step might have been a bit confusing, since the TV cook had prepared everything hours before the show actually started, so here's the summary in slow motion: the Weather Station library now comes with a class called ThingspeakClient which does all the legwork. You pass in the channel id and API key and it downloads a JSON object and picks out just the last data set, since we are currently interested only in that. Have a look at https://github.com/squix78/esp8266-weather-station/blob/master/ThingspeakClient.cpp to understand what happens and maybe adapt/extend it to your needs. Why not display the chart of the last 24 hours on the OLED display?

8.More Projects

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.

A. The ESP8266 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 the airspace defined by your parameters. I built this fun project because I see airplanes starting and landing from the nearby airport of Zurich. Since starting FlightRadar24 on my iPhone is not nearly as nerdy as building a dedicated device I went to work.



The ESP8266 PlaneSpotter in Action

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 lowcost repurposed DVB-T dongles which receive the ADS-B signal transmitted by aircraft. 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.

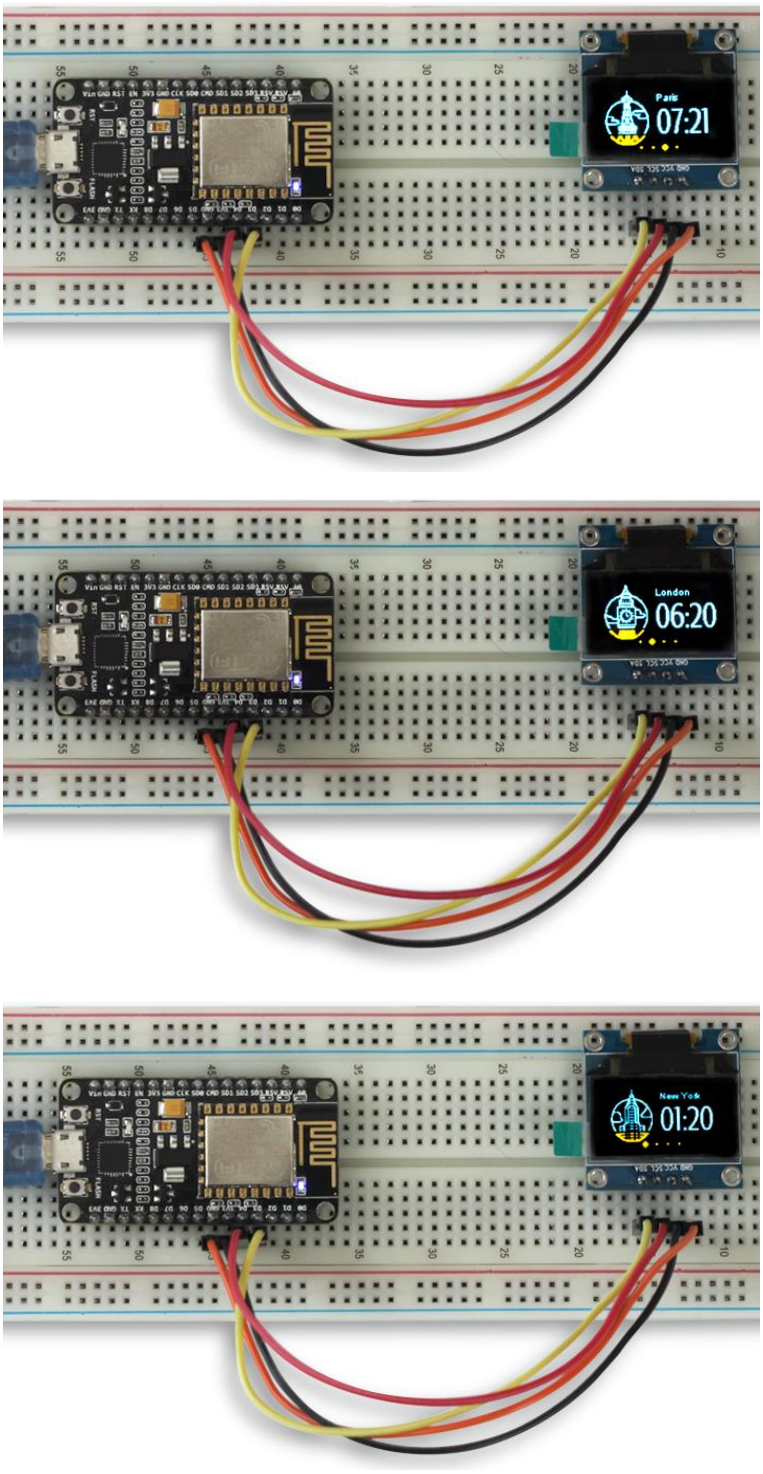
To build this project you can use the PlaneSpotterDemo which comes with the WeatherStation library: File > Examples > ESP8266 Weather Station > PlaneSpotterDemo

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

B. The ESP8266 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



The ESP8266 World Clock