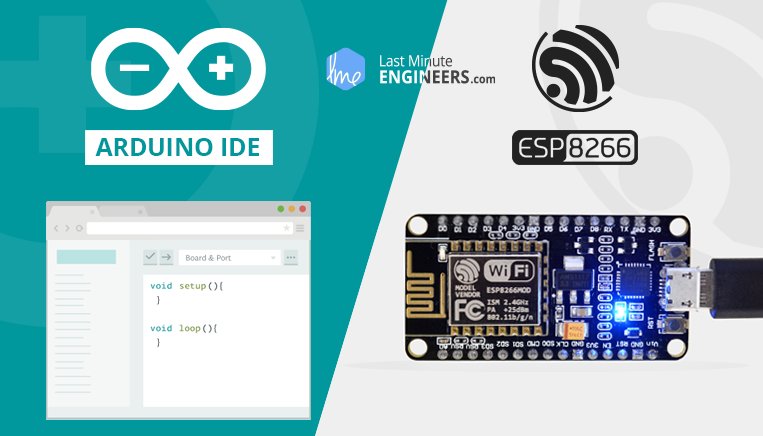
Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE



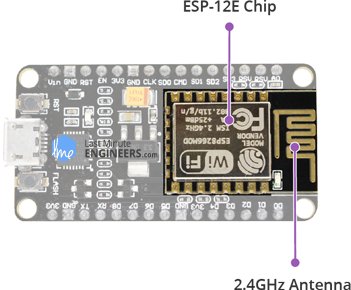
The Internet of Things (IoT) has been a trending field in the world of technology. It has changed the way we work. Physical objects and the digital world are connected now more than ever. Keeping this in mind, [Espressif Systems](https://www.espressif.com/) (A Shanghai-based Semiconductor Company) has released an adorable, bite-sized WiFi enabled microcontroller – ESP8266, at an unbelievable price! For less than $3, it can monitor and control things from anywhere in the world – perfect for just about any IoT project.

ESP-12E Module

The development board equips the ESP-12E module containing ESP8266 chip having Tensilica Xtensa® 32-bit LX106 RISC microprocessor which operates at 80 to 160 MHz adjustable clock frequency and supports RTOS.

ESP-12E Chip

* Tensilica Xtensa® 32-bit LX106
* 80 to 160 MHz Clock Freq.
* 128kB internal RAM
* 4MB external flash
* 802.11b/g/n Wi-Fi transceiver



There’s also 128 KB RAM and 4MB of Flash memory (for program and data storage) just enough to cope with the large strings that make up web pages, JSON/XML data, and everything we throw at IoT devices nowadays.

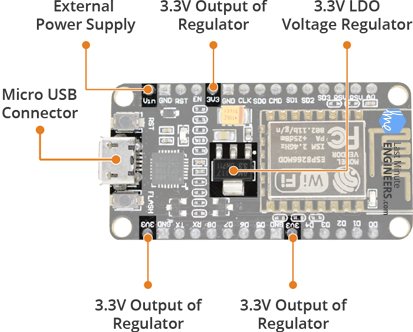
The ESP8266 Integrates 802.11b/g/n HT40 Wi-Fi transceiver, so it can not only connect to a WiFi network and interact with the Internet, but it can also set up a network of its own, allowing other devices to connect directly to it. This makes the ESP8266 NodeMCU even more versatile.

Power Requirement

As the operating voltage range of ESP8266 is 3V to 3.6V, the board comes with a LDO voltage regulator to keep the voltage steady at 3.3V. It can reliably supply up to 600mA, which should be more than enough when ESP8266 pulls as much as 80mA during RF transmissions. The output of the regulator is also broken out to one of the sides of the board and labeled as 3V3. This pin can be used to supply power to external components.

Power Requirement

* Operating Voltage: 2.5V to 3.6V
* On-board 3.3V 600mA regulator
* 80mA Operating Current
* 20 µA during Sleep Mode



Power to the ESP8266 NodeMCU is supplied via the on-board MicroB USB connector. Alternatively, if you have a regulated 5V voltage source, the VIN pin can be used to directly supply the ESP8266 and its peripherals.

Warning:

The ESP8266 requires a 3.3V power supply and 3.3V logic levels for communication. The GPIO pins are not 5V-tolerant! If you want to interface the board with 5V (or higher) components, you’ll need to do some level shifting.

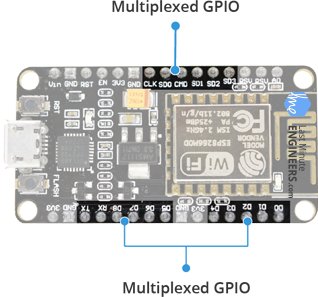
Peripherals and I/O

The ESP8266 NodeMCU has total 17 GPIO pins broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

* ADC channel – A 10-bit ADC channel.
* UART interface – UART interface is used to load code serially.
* PWM outputs – PWM pins for dimming LEDs or controlling motors.
* SPI, I2C & I2S interface – SPI and I2C interface to hook up all sorts of sensors and peripherals.
* I2S interface – I2S interface if you want to add sound to your project.

Multiplexed I/Os

* 1 ADC channels
* 2 UART interfaces
* 4 PWM outputs
* SPI, I2C & I2S interface



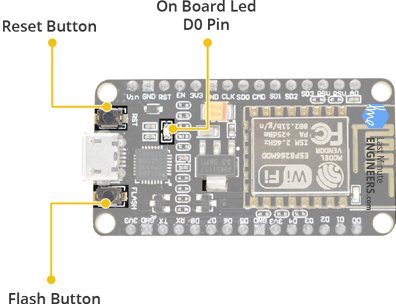
Thanks to the ESP8266’s pin multiplexing feature (Multiple peripherals multiplexed on a single GPIO pin). Meaning a single GPIO pin can act as PWM/UART/SPI.

On-board Switches & LED Indicator

The ESP8266 NodeMCU features two buttons. One marked as RST located on the top left corner is the Reset button, used of course to reset the ESP8266 chip. The other FLASH button on the bottom left corner is the download button used while upgrading firmware.

Switches & Indicators

* RST – Reset the ESP8266 chip
* FLASH – Download new programs
* Blue LED – User Programmable



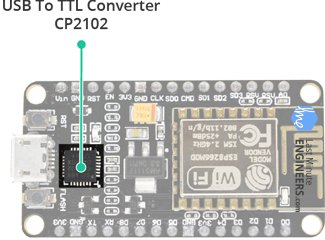
The board also has a LED indicator which is user programmable and is connected to the D0 pin of the board.

Serial Communication

The board includes CP2102 USB-to-UART Bridge Controller from [Silicon Labs](http://www.silabs.com/), which converts USB signal to serial and allows your computer to program and communicate with the ESP8266 chip.

Serial Communication

* CP2102 USB-to-UART converter
* 4.5 Mbps communication speed
* Flow Control support

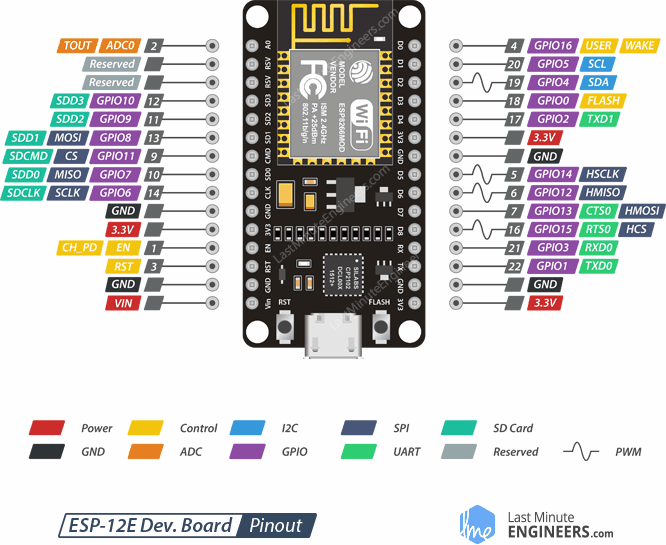


If you have an older version of CP2102 driver installed on your PC, we recommend upgrading now.

[CP2102 Driver](https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)

ESP8266 NodeMCU Pinout

The ESP8266 NodeMCU has total 30 pins that interface it to the outside world. The connections are as follows:



For the sake of simplicity, we will make groups of pins with similar functionalities.

Power Pins There are four power pins viz. one VIN pin & three 3.3V pins. The VIN pin can be used to directly supply the ESP8266 and its peripherals, if you have a regulated 5V voltage source. The 3.3V pins are the output of an on-board voltage regulator. These pins can be used to supply power to external components.

GND is a ground pin of ESP8266 NodeMCU development board.

I2C Pins are used to hook up all sorts of I2C sensors and peripherals in your project. Both I2C Master and I2C Slave are supported. I2C interface functionality can be realized programmatically, and the clock frequency is 100 kHz at a maximum. It should be noted that I2C clock frequency should be higher than the slowest clock frequency of the slave device.

GPIO Pins ESP8266 NodeMCU has 17 GPIO pins which can be assigned to various functions such as I2C, I2S, UART, PWM, IR Remote Control, LED Light and Button programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, it can also be set to edge-trigger or level-trigger to generate CPU interrupts.

ADC Channel The NodeMCU is embedded with a 10-bit precision SAR ADC. The two functions can be implemented using ADC viz. Testing power supply voltage of VDD3P3 pin and testing input voltage of TOUT pin. However, they cannot be implemented at the same time.

UART Pins ESP8266 NodeMCU has 2 UART interfaces, i.e. UART0 and UART1, which provide asynchronous communication (RS232 and RS485), and can communicate at up to 4.5 Mbps. UART0 (TXD0, RXD0, RST0 & CTS0 pins) can be used for communication. It supports fluid control. However, UART1 (TXD1 pin) features only data transmit signal so, it is usually used for printing log.

SPI Pins ESP8266 features two SPIs (SPI and HSPI) in slave and master modes. These SPIs also support the following general-purpose SPI features:

* 4 timing modes of the SPI format transfer
* Up to 80 MHz and the divided clocks of 80 MHz
* Up to 64-Byte FIFO
* SDIO Pins ESP8266 features Secure Digital Input/Output Interface (SDIO) which is used to directly interface SD cards. 4-bit 25 MHz SDIO v1.1 and 4-bit 50 MHz SDIO v2.0 are supported.
* PWM Pins The board has 4 channels of Pulse Width Modulation (PWM). The PWM output can be implemented programmatically and used for driving digital motors and LEDs. PWM frequency range is adjustable from 1000 μs to 10000 μs, i.e., between 100 Hz and 1 kHz.
* Control Pins are used to control ESP8266. These pins include Chip Enable pin (EN), Reset pin (RST) and WAKE pin.
* EN pin – The ESP8266 chip is enabled when EN pin is pulled HIGH. When pulled LOW the chip works at minimum power.
* RST pin – RST pin is used to reset the ESP8266 chip.
* WAKE pin – Wake pin is used to wake the chip from deep-sleep.

ESP8266 Development Platforms

Now, let’s move on to the interesting stuff!

There are a variety of development platforms that can be equipped to program the ESP8266. You can go with [Espruino](https://www.espruino.com/) – JavaScript SDK and firmware closely emulating Node.js, or use [Mongoose OS](https://mongoose-os.com/) – An operating system for IoT devices (recommended platform by Espressif Systems and Google Cloud IoT) or use a software development kit (SDK) provided by Espressif or one of the platforms listed on [WiKiPedia](https://en.wikipedia.org/wiki/ESP8266#SDKs).

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

This ESP8266 add-on for Arduino is based on the amazing work by [Ivan Grokhotkov](https://github.com/igrr) and the rest of the ESP8266 community. Check out the [ESP8266 Arduino GitHub repository](https://github.com/esp8266/Arduino) for more information.

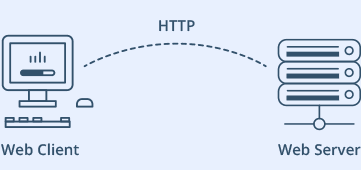
# Create A Simple ESP8266 NodeMCU Web Server In Arduino IDE



Over the past few years, the ESP8266 has been a growing star among IoT or WiFi-related projects. It’s an extremely cost-effective WiFi module that – with a little extra effort – can be programmed to **build a standalone web server**. How cool is that!

### What is a Web server and how it works?

Web server is a place which stores, processes and delivers web pages to Web clients. Web client is nothing but a web browser on our laptops and smartphones. The communication between client and server takes place using a special protocol called Hypertext Transfer Protocol (HTTP).



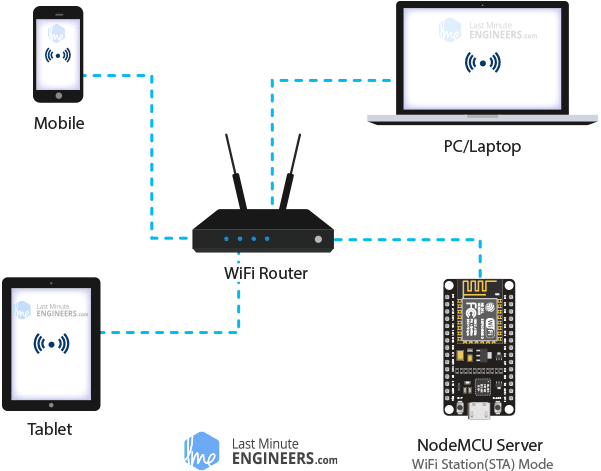
In this protocol, a client initiates communication by making a request for a specific web page using HTTP and the server responds with the content of that web page or an error message if unable to do so (like famous 404 Error). Pages delivered by a server are mostly HTML documents.

## ESP8266 Operating Modes

One of the greatest features ESP8266 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices to connect directly to it and access web pages. This is possible because ESP8266 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. This provides possibility of building [mesh networks](https://en.wikipedia.org/wiki/Mesh_networking).

### Station (STA) Mode

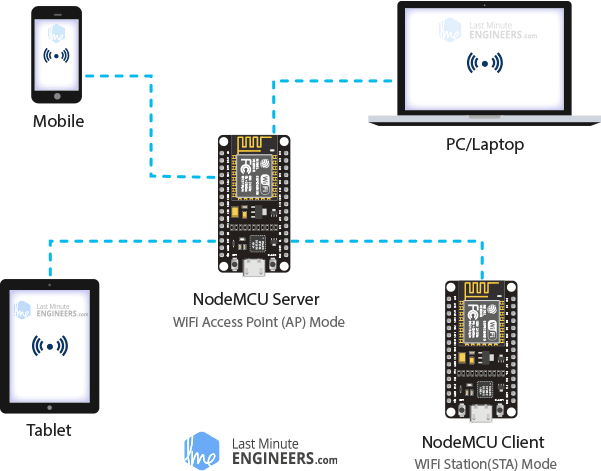
The ESP8266 that connects to an existing WiFi network (one created by your wireless router) is called **Station** (STA)



In STA mode ESP8266 gets IP from wireless router to which it is connected. With this IP address, it can set up a web server and **deliver web pages to all connected devices under existing WiFi network**.

### Soft Access Point (AP) Mode

The ESP8266 that creates its own WiFi network and acts as a hub (Just like WiFi router) for one or more stations is called **Access Point** (AP). Unlike WiFi router, it does not have interface to a wired network. So, such mode of operation is called **Soft Access Point** (soft-AP). Also the maximum number of stations that can connect to it is limited to five.



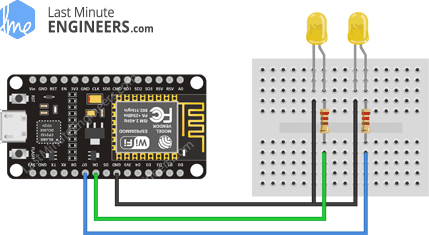
In AP mode ESP8266 creates a new WiFi network and sets SSID (Name of the network) and IP address to it. With this IP address, it can **deliver web pages to all connected devices under its own network**.

## Wiring – Connecting LEDs to ESP8266 NodeMCU

Now that we know the basics of how web server works, and in which modes ESP8266 can create a web server, it’s time to connect some LEDs to ESP8266 NodeMCU that we want to control over WiFi.

Start by placing the NodeMCU on to your breadboard, ensuring each side of the board is on a separate side of the breadboard. Next, connect two LEDs to digital GPIO D6 and D7 through a 220Ω current limiting resistor.

When you’re done you should have something that looks similar to the illustration shown below.

Wiring LEDs to ESP8266 NodeMCU

## Concept Behind Controlling Things From ESP8266 Web Server

So, you might be thinking, “How am I going to control things from a web server that merely processes and delivers web pages?” Well, then you need to understand what’s going on behind the scene.

When you type a URL in a web browser and hit ENTER, the browser sends a HTTP request (a.k.a. GET request) to a web server. It’s a job of web server to handle this request by doing something. You might have figured it out by now that we are going to control things by accessing a specific URL. For example, suppose we entered a URL like http://192.168.1.1/**ledon** in a browser. The browser then sends a HTTP request to ESP8266 to handle this request. When ESP8266 reads this request, it knows that user wants to turn the LED ON. So, it turns the LED ON and sends a dynamic webpage to a browser showing **LED status : ON** As easy as Pie!

## ESP8266 as HTTP Server using WiFi Access Point (AP) mode

Now let’s move on to the interesting stuff!

As the heading suggests, this example demonstrates how to turn the ESP8266 into an access point (AP), and serve up web pages to any connected client. To start with, plug your ESP8266 NodeMCU into your computer and Try the sketch out; and then we will dissect it in some detail.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

/\* Put your SSID & Password \*/

const char\* ssid = "NodeMCU"; // Enter SSID here

const char\* password = "12345678"; //Enter Password here

/\* Put IP Address details \*/

IPAddress local\_ip(192,168,1,1);

IPAddress gateway(192,168,1,1);

IPAddress subnet(255,255,255,0);

ESP8266WebServer server(80);

uint8\_t LED1pin = D7;

bool LED1status = LOW;

uint8\_t LED2pin = D6;

bool LED2status = LOW;

void setup() {

Serial.begin(115200);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);

WiFi.softAP(ssid, password);

WiFi.softAPConfig(local\_ip, gateway, subnet);

delay(100);

server.on("/", handle\_OnConnect);

server.on("/led1on", handle\_led1on);

server.on("/led1off", handle\_led1off);

server.on("/led2on", handle\_led2on);

server.on("/led2off", handle\_led2off);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

if(LED1status)

{digitalWrite(LED1pin, HIGH);}

else

{digitalWrite(LED1pin, LOW);}

if(LED2status)

{digitalWrite(LED2pin, HIGH);}

else

{digitalWrite(LED2pin, LOW);}

}

void handle\_OnConnect() {

LED1status = LOW;

LED2status = LOW;

Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,LED2status));

}

void handle\_led1on() {

LED1status = HIGH;

Serial.println("GPIO7 Status: ON");

server.send(200, "text/html", SendHTML(true,LED2status));

}

void handle\_led1off() {

LED1status = LOW;

Serial.println("GPIO7 Status: OFF");

server.send(200, "text/html", SendHTML(false,LED2status));

}

void handle\_led2on() {

LED2status = HIGH;

Serial.println("GPIO6 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));

}

void handle\_led2off() {

LED2status = LOW;

Serial.println("GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,false));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>LED Control</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

ptr +=".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr +=".button-on {background-color: #1abc9c;}\n";

ptr +=".button-on:active {background-color: #16a085;}\n";

ptr +=".button-off {background-color: #34495e;}\n";

ptr +=".button-off:active {background-color: #2c3e50;}\n";

ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<h1>ESP8266 Web Server</h1>\n";

ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

if(led1stat)

{ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}

else

{ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}

if(led2stat)

{ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}

else

{ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}

ptr +="</body>\n";

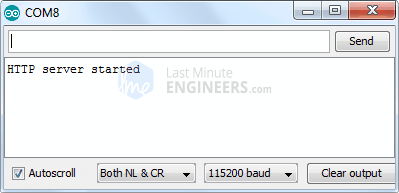
ptr +="</html>\n";

return ptr;

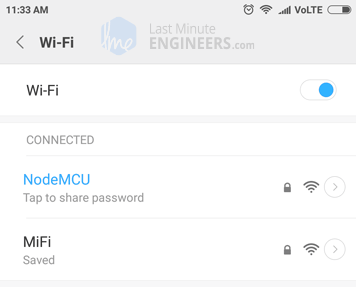
}

## Accessing the Web Server in AP mode

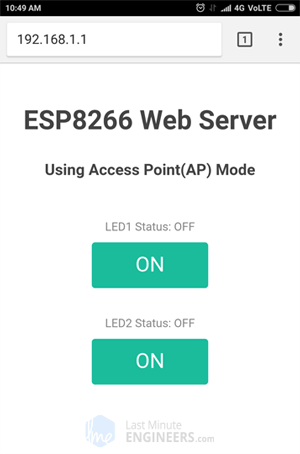
After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on ESP8266. If everything is OK, it will show **HTTP server started** message.

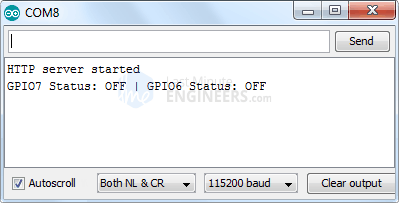


Next, find any device that you can connect to a WiFi network – phone, laptop, etc. And look for a network called **NodeMCU**. Join the network with password **123456789**.

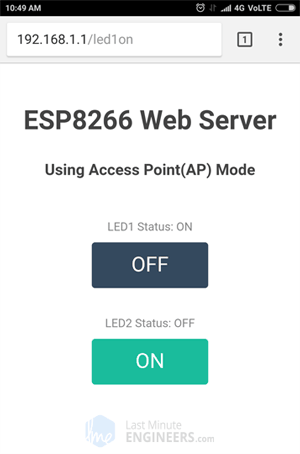


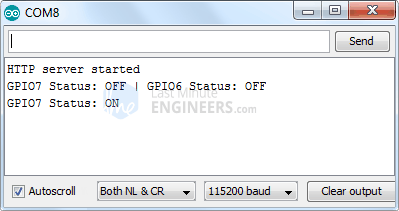
After connecting to your NodeMCU AP network, load up a browser and point it to 192.168.1.1 The NodeMCU should serve up a web page showing current status of LEDs and two buttons to control them. If take a look at the serial monitor at the same time, you can see status of NodeMCU’s GPIO pins.





Now, click the button to turn LED1 ON while keeping an eye on the URL. Once you click the button, the ESP8266 receives a request for **/led1on** URL. It then turns the LED1 ON and serves a web page with status of LED updated. It also prints the status of GPIO pin on the serial monitor.





You can test LED2 button and check that it works in a similar way.

Now, let’s take a closer look at the code to see how it works, so that you are able to modify it to fulfill your needs.

## Detailed Code Explanation

The sketch starts by including [ESP8266WiFi.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFi.h) library. This library provides ESP8266 specific WiFi methods we are calling to connect to network. Following that we also include the [ESP8266WebServer.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WebServer/src/ESP8266WebServer.h) library, which has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

As we are setting the ESP8266 NodeMCU in Access Point (AP) mode, it will create a WiFi network. Hence, we need to set its SSID, Password, IP address, IP subnet mask and IP gateway.

/\* Put your SSID & Password \*/

const char\* ssid = "NodeMCU"; // Enter SSID here

const char\* password = "12345678"; //Enter Password here

/\* Put IP Address details \*/

IPAddress local\_ip(192,168,1,1);

IPAddress gateway(192,168,1,1);

IPAddress subnet(255,255,255,0);

Next, we declare an object of ESP8266WebServer library, so we can access its functions. The constructor of this object takes [port](https://en.wikipedia.org/wiki/Port_(computer_networking)) (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

// declare an object of ESP8266WebServer library

ESP8266WebServer server(80);

Next, we declare the NodeMCU’s GPIO pins to which LEDs are connected and their initial state.

uint8\_t LED1pin = D7;

bool LED1status = LOW;

uint8\_t LED2pin = D6;

bool LED2status = LOW;

### Inside Setup() Function

We configure our HTTP server before actually running it. First of all, we open a serial connection for debugging purpose and set GPIO ports to OUTPUT.

Serial.begin(115200);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);

Then, we set up a soft access point to establish a Wi-Fi network by proving SSID, Password, IP address, IP subnet mask and IP gateway.

WiFi.softAP(ssid, password);

WiFi.softAPConfig(local\_ip, gateway, subnet);

delay(100);

In order to handle incoming HTTP requests, we need to specify which code to execute when a particular URL is hit. To do so, we use [on](https://github.com/esp8266/Arduino/blob/4897e0006b5b0123a2fa31f67b14a3fff65ce561/libraries/ESP8266WebServer/src/ESP8266WebServer.cpp#L135) method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

For example, the first line of below code snippet indicates that when a server receives an HTTP request on the root (**/**) path, it will trigger the handle\_OnConnect() function. Note that the URL specified is a relative path.

Likewise, we need to specify 4 more URLs to handle two states of 2 LEDs.

server.on("/", handle\_OnConnect);

server.on("/led1on", handle\_led1on);

server.on("/led1off", handle\_led1off);

server.on("/led2on", handle\_led2on);

server.on("/led2off", handle\_led2off);

We haven’t specified what the server should do if the client requests any URL other than specified with server.on() . It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use server.onNotFound() to tell it that it should execute it when it receives a request for a URI that wasn’t specified with server.on

server.onNotFound(handle\_NotFound);

Now, to start our server, we call the begin method on the server object.

server.begin();

Serial.println("HTTP server started");

### Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the handleClient() method on the server object. We also change the state of LED as per the request.

void loop() {

server.handleClient();

if(LED1status)

{digitalWrite(LED1pin, HIGH);}

else

{digitalWrite(LED1pin, LOW);}

if(LED2status)

{digitalWrite(LED2pin, HIGH);}

else

{digitalWrite(LED2pin, LOW);}

}

Next, we need to create a function we attached to root (/) URL with server.on. Remember? At the start of this function, we set the status of both the LEDs to LOW (Initial state of LEDs) and print it on serial monitor. In order to respond to the HTTP request, we use the [send](https://github.com/esp8266/Arduino/blob/4897e0006b5b0123a2fa31f67b14a3fff65ce561/libraries/ESP8266WebServer/src/ESP8266WebServer.cpp#L284) method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code **200** (one of the [HTTP status codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)), which corresponds to the **OK** response. Then, we are specifying the content type as “text/html“, and finally we are calling SendHTML() custom function which creates a dynamic HTML page containing status of LEDs.

void handle\_OnConnect()

{

LED1status = LOW;

LED2status = LOW;

Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,LED2status));

}

Likewise, we need to create four functions to handle LED ON/OFF requests and 404 Error page.

void handle\_led1on() {

LED1status = HIGH;

Serial.println("GPIO7 Status: ON");

server.send(200, "text/html", SendHTML(true,LED2status));

}

void handle\_led1off() {

LED1status = LOW;

Serial.println("GPIO7 Status: OFF");

server.send(200, "text/html", SendHTML(false,LED2status));

}

void handle\_led2on() {

LED2status = HIGH;

Serial.println("GPIO6 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));

}

void handle\_led2off() {

LED2status = LOW;

Serial.println("GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,false));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

### Displaying the HTML Web Page

SendHTML() function is responsible for generating a web page whenever the ESP8266 web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the server.send() function we discussed earlier. The function takes status of LEDs as a parameter to dynamically generate the HTML content.

The first text you should always send is the [**<!DOCTYPE>** declaration](https://www.w3schools.com/tags/tag_doctype.asp) that indicates that we’re sending HTML code.

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

Next, the [<meta> viewport element](https://www.w3schools.com/css/css_rwd_viewport.asp) makes the web page responsive in any web browser. While title tag sets the title of the page.

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>LED Control</title>\n";

### Styling the Web Page

Next, we have some CSS to style the buttons and the web page appearance. We choose the Helvetica font, define the content to be displayed as an inline-block and aligned at the center.

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

Following code then sets color, font and margin around the body, H1, H3 and p tags.

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

Some styling is applied to the buttons as well with properties like color, size, margin, etc. The ON and OFF button has different background color while [:active selector](https://www.w3schools.com/cssref/sel_active.asp) for buttons ensure button click effect

ptr +=".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr +=".button-on {background-color: #1abc9c;}\n";

ptr +=".button-on:active {background-color: #16a085;}\n";

ptr +=".button-off {background-color: #34495e;}\n";

ptr +=".button-off:active {background-color: #2c3e50;}\n";

### Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

ptr +="<h1>ESP8266 Web Server</h1>\n";

ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

### Displaying the Buttons and Corresponding State

To dynamically generate the buttons and LED status, we use if statement. So, depending upon the status of the GPIO pins, ON/OFF button is displayed.

if(led1stat)

{ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}

else

{ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}

if(led2stat)

{ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}

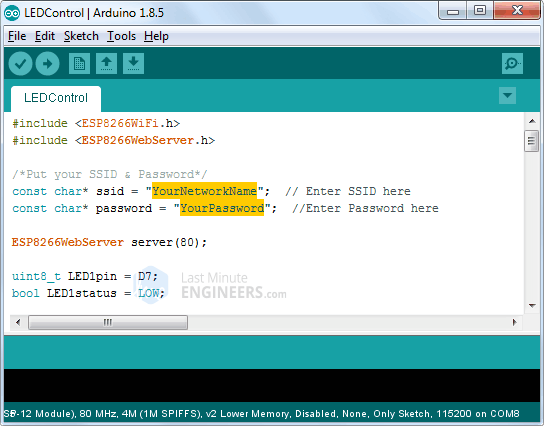
else

{ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}

## ESP8266 as HTTP Server using WiFi Station (STA) mode

Now let’s move on to our next example which demonstrates how to turn the ESP8266 into Station (STA) mode, and serve up web pages to any connected client under existing network.

Before you head for uploading the sketch, you need to make some changes to make it work for you. You need to modify the following two variables with your network credentials, so that ESP8266 can establish a connection with existing network.



Once you are done, go ahead and try the sketch out.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);

uint8\_t LED1pin = D7;

bool LED1status = LOW;

uint8\_t LED2pin = D6;

bool LED2status = LOW;

void setup() {

Serial.begin(115200);

delay(100);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle\_OnConnect);

server.on("/led1on", handle\_led1on);

server.on("/led1off", handle\_led1off);

server.on("/led2on", handle\_led2on);

server.on("/led2off", handle\_led2off);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

if(LED1status)

{digitalWrite(LED1pin, HIGH);}

else

{digitalWrite(LED1pin, LOW);}

if(LED2status)

{digitalWrite(LED2pin, HIGH);}

else

{digitalWrite(LED2pin, LOW);}

}

void handle\_OnConnect() {

LED1status = LOW;

LED2status = LOW;

Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,LED2status));

}

void handle\_led1on() {

LED1status = HIGH;

Serial.println("GPIO7 Status: ON");

server.send(200, "text/html", SendHTML(true,LED2status));

}

void handle\_led1off() {

LED1status = LOW;

Serial.println("GPIO7 Status: OFF");

server.send(200, "text/html", SendHTML(false,LED2status));

}

void handle\_led2on() {

LED2status = HIGH;

Serial.println("GPIO6 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));

}

void handle\_led2off() {

LED2status = LOW;

Serial.println("GPIO6 Status: OFF");

server.send(200, "text/html", SendHTML(LED1status,false));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(uint8\_t led1stat,uint8\_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>LED Control</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

ptr +=".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr +=".button-on {background-color: #1abc9c;}\n";

ptr +=".button-on:active {background-color: #16a085;}\n";

ptr +=".button-off {background-color: #34495e;}\n";

ptr +=".button-off:active {background-color: #2c3e50;}\n";

ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<h1>ESP8266 Web Server</h1>\n";

ptr +="<h3>Using Station(STA) Mode</h3>\n";

if(led1stat)

{ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}

else

{ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}

if(led2stat)

{ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}

else

{ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}

ptr +="</body>\n";

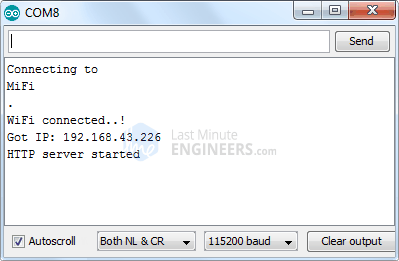
ptr +="</html>\n";

return ptr;

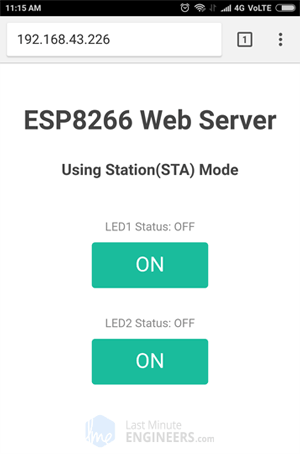
}

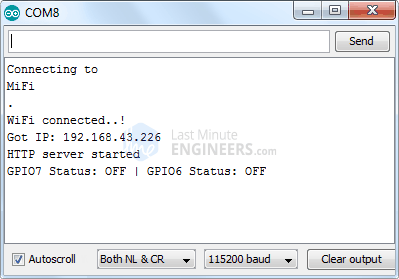
## Accessing the Web Server in STA mode

After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on ESP8266. If everything is OK, it will output the dynamic IP address obtained from your router and show **HTTP server started** message.

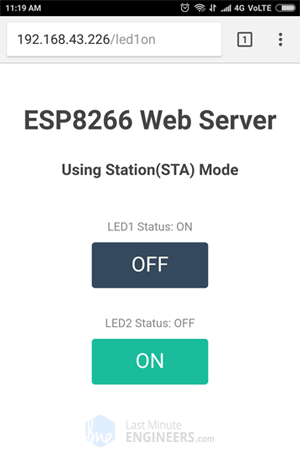


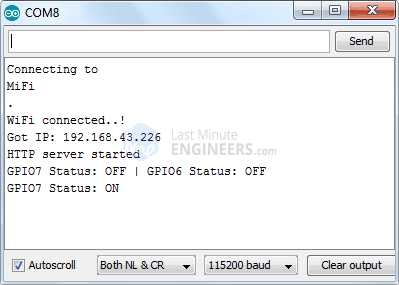
Next, load up a browser and point it to the IP address shown on the serial monitor. The NodeMCU should serve up a web page showing current status of LEDs and two buttons to control them. If take a look at the serial monitor at the same time, you can see status of NodeMCU’s GPIO pins.





Now, click the button to turn LED1 ON while keeping an eye on the URL. Once you click the button, the ESP8266 receives a request for **/led1on** URL. It then turns the LED1 ON and serves a web page with status of LED updated. It also prints the status of GPIO pin on the serial monitor.





You can test LED2 button and check that it works in a similar way.

## Code Explanation

If you observe this code with the previous code, the only difference is that we are not setting the soft Access Point, Instead we are joining existing network using WiFi.begin() function.

//connect to your local wi-fi network

WiFi.begin(ssid, password);

While the ESP8266 tries to connect to the network, we can check the connectivity status with WiFi.status() function

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Just for your information, this function returns the following statuses:

* **WL\_CONNECTED**: assigned when connected to a Wi-Fi network
* **WL\_NO\_SHIELD**: assigned when no Wi-Fi shield is present
* **WL\_IDLE\_STATUS**: a temporary status assigned when WiFi.begin() is called and remains active until the number of attempts expires (resulting in WL\_CONNECT\_FAILED) or a connection is established (resulting in WL\_CONNECTED)
* **WL\_NO\_SSID\_AVAIL**: assigned when no SSID are available
* **WL\_SCAN\_COMPLETED**: assigned when the scan networks is completed
* **WL\_CONNECT\_FAILED**: assigned when the connection fails for all the attempts
* **WL\_CONNECTION\_LOST**: assigned when the connection is lost
* **WL\_DISCONNECTED**: assigned when disconnected from a network

Once the ESP8266 is connected to the network, the sketch prints the IP address assigned to ESP8266 by displaying WiFi.localIP() value on serial monitor.

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

The only difference between AP & STA mode is one creates the network and other joins the existing network. So, rest of the code for handling HTTP requests and serving web page in STA mode is same as that of AP mode explained above. This includes:

* Declaring NodeMCU’s GPIO pins to which LEDs are connected
* Defining multiple server.on() methods to handle incoming HTTP requests
* Defining server.onNotFound() method to handle HTTP 404 error
* Creating custom functions that are executed when specific URL is hit
* Creating HTML page
* Styling the web page
* Creating buttons and displaying their status

# Interface DHT11 DHT22 w/ ESP8266 NodeMCU Using Web Server



Have you ever wanted to have sensors scattered all around your house and garden reporting their temperature regularly to a central server? Then, this IoT project might be the solid launching point for you! This project uses ESP8266 NodeMCU as the control device that easily connects to existing WiFi network & creates a Web Server. When any connected device accesses this web server, ESP8266 NodeMCU reads in temperature & relative humidity from the DHT11, DHT22/AM2302 sensor & sends it to the web browser of that device with a nice interface. Excited? Let’s get started!

It may look intimidating, but there are a few concepts you should be familiar with, before venturing further into this tutorial. If any of the concepts below sound foreign to you, consider reading (at least skimming) through that tutorial first:

[](https://lastminuteengineers.com/dht11-dht22-arduino-tutorial/)

[How DHT11 DHT22 Sensors Work & Interface With Arduino](https://lastminuteengineers.com/dht11-dht22-arduino-tutorial/)

Give your next Arduino project the ability to sense the world around it with the inexpensive DHT11 or DHT22 Digital Temperature & Humidity Sensor from...

[](https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/)

[Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE](https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/)

The Internet of Things (IoT) has been a trending field in the world of technology. It has changed the way we work. Physical objects and...

[](https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/)

[Create A Simple ESP8266 NodeMCU Web Server In Arduino IDE](https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/)

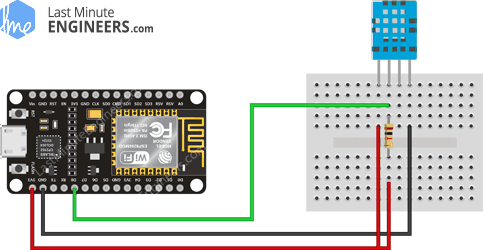
Over the past few years, the ESP8266 has been a growing star among IoT or WiFi-related projects. It’s an extremely cost-effective WiFi module that -...

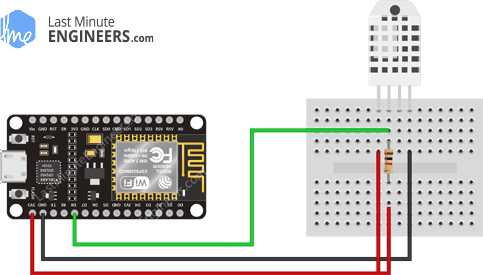
## Wiring – Connecting DHT11, DHT22/AM2302 sensor to ESP8266 NodeMCU

Connecting DHT11/DHT22/AM2302 sensor to ESP8266 NodeMCU is fairly simple. Start by placing the NodeMCU on to your breadboard, ensuring each side of the board is on a separate side of the breadboard.

Now place the sensor on to your breadboard besides NodeMCU. Connect VCC pin on the sensor to the 3.3V pin on the NodeMCU and ground to ground. Also connect Data pin on the sensor to D8 pin of the ESP8266 NodeMCU. Finally, we need to place a pull-up resistor of 10KΩ between VCC and data line to keep it HIGH for proper communication between sensor and NodeMCU. If you happen to have a breakout board of the sensor, you need not add any external pull-up. It comes with a built-in pull-up resistor.

When you’re done you should have something that looks similar to the illustration shown below.

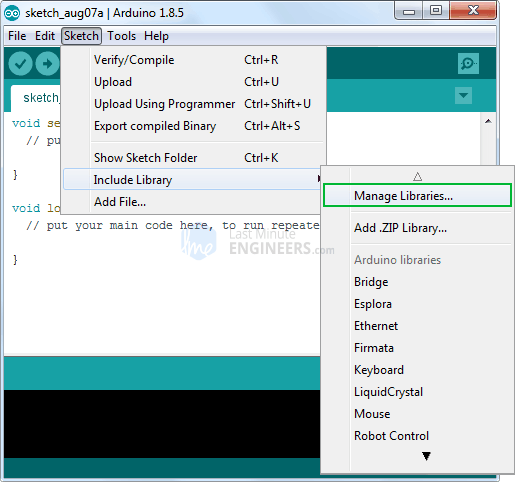
Wiring DHT11 Temperature Humidity Sensor with ESP8266 NodeMCU

Wiring DHT22 Temperature Humidity Sensor with ESP8266 NodeMCU

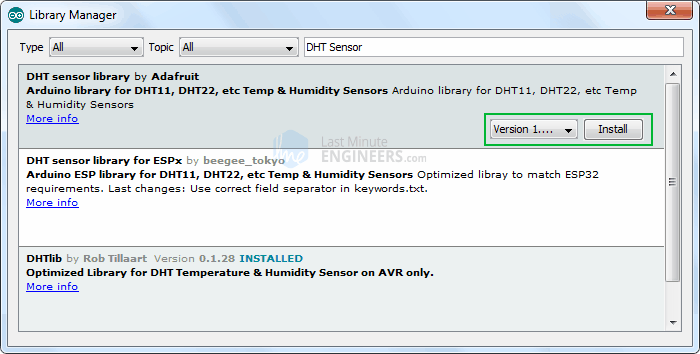
## Installing DHT Sensor Library

Communicating with DHT11, DHT22/AM2302 sensors is a bunch of work, as they have their own single wire protocol for data transfer. And this protocol requires precise timing. Fortunately, we don’t have to worry much about this because we are going to use the DHT library from Adafruit which takes care of almost everything. The library is so powerful that it runs on both Arduino and ESP architecture.

To install the library navigate to the **Sketch > Include Library > Manage Libraries…** Wait for Library Manager to download libraries index and update list of installed libraries.



Filter your search by typing ‘**DHT sensor**’. There should be a couple entries. Look for **DHT sensor library** by **Adafruit**. Click on that entry, and then select Install.



The DHT sensor library uses the [Adafruit Sensor support backend](https://github.com/adafruit/Adafruit_Sensor). So, search the library manager for **Adafruit Unified Sensor** and install that too (you may have to scroll a bit)

## Create ESP8266 NodeMCU Web Server using WiFi Station (STA) mode

Now let’s move on to the interesting stuff!

As the heading suggests, we are going to configure our ESP8266 NodeMCU into Station (STA) mode, and create a web server to serve up web pages to any connected client under existing network.

Before you head for uploading the sketch, you need to **make one change** to make it work for you. You need to modify the following two variables with your network credentials, so that ESP8266 NodeMCU can establish a connection with existing network.

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

Once you are done, go ahead and try the sketch out and then we will dissect it in some detail.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include "DHT.h"

// Uncomment one of the lines below for whatever DHT sensor type you're using!

//#define DHTTYPE DHT11 // DHT 11

//#define DHTTYPE DHT21 // DHT 21 (AM2301)

#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);

// DHT Sensor

uint8\_t DHTPin = D8;

// Initialize DHT sensor.

DHT dht(DHTPin, DHTTYPE);

float Temperature;

float Humidity;

void setup() {

Serial.begin(115200);

delay(100);

pinMode(DHTPin, INPUT);

dht.begin();

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle\_OnConnect);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

}

void handle\_OnConnect() {

Temperature = dht.readTemperature(); // Gets the values of the temperature

Humidity = dht.readHumidity(); // Gets the values of the humidity

server.send(200, "text/html", SendHTML(Temperature,Humidity));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(float Temperaturestat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 NodeMCU Weather Report</h1>\n";

ptr +="<p>Temperature: ";

ptr +=(int)Temperaturestat;

ptr +="°C</p>";

ptr +="<p>Humidity: ";

ptr +=(int)Humiditystat;

ptr +="%</p>";

ptr +="</div>\n";

ptr +="</body>\n";

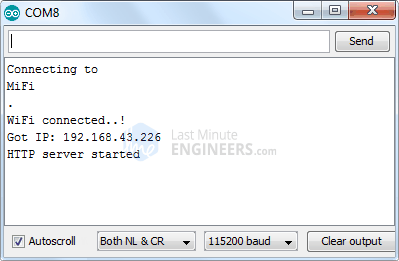
ptr +="</html>\n";

return ptr;

}

## Accessing the Web Server

After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on the NodeMCU. If everything is OK, it will output the dynamic IP address obtained from your router and show **HTTP server started** message.



Next, load up a browser and point it to the IP address shown on the serial monitor. The ESP8266 NodeMCU should serve up a web page showing temperature and relative humidity.



## Detailed Code Explanation

The sketch starts by including **ESP8266WiFi.h** library. This library provides ESP8266 NodeMCU specific WiFi methods we are calling to connect to network. Following that we also include the **ESP8266WebServer.h** library, which has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details. Finally we include **DHT.h** library.

#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

#include "DHT.h"

Next, we need to define the type of DHT sensor we are using. Uncomment one of the lines below accordingly!

//#define DHTTYPE DHT11 // DHT 11

//#define DHTTYPE DHT21 // DHT 21 (AM2301)

#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321

As we are configuring ESP8266 NodeMCU in Station (STA) mode, it will join existing WiFi network. Hence, we need to provide it with your network’s SSID & Password.

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

Next, we declare an object of **ESP8266WebServer** library, so we can access its functions. The constructor of this object takes [port](https://en.wikipedia.org/wiki/Port_(computer_networking)) (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

// declare an object of WebServer library

ESP8266WebServer server(80);

Next, we need to define the ESP8266 NodeMCU’s pin number to which our sensor’s Data pin is connected and create a DHT object. So, that we can access special functions related to the DHT library.

// DHT Sensor

uint8\_t DHTPin = D8;

// Initialize DHT sensor.

DHT dht(DHTPin, DHTTYPE);

Two float variables viz. Temperature & Humidity are declared to store respective values.

float Temperature;

float Humidity;

### Inside Setup() Function

Inside Setup() Function we configure our HTTP server before actually running it. First of all, we open a serial connection for debugging purpose and set GPIO ports to INPUT. We also need to initialize the DHT object using begin() function.

Serial.begin(115200);

delay(100);

pinMode(DHTPin, INPUT);

dht.begin();

Now, we need to join the WiFi network using WiFi.begin() function. The function takes SSID (Network Name) and password as a parameter.

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

While the ESP8266 NodeMCU tries to connect to the network, we can check the connectivity status with WiFi.status() function.

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Once the ESP8266 NodeMCU is connected to the network, the sketch prints the IP address assigned to ESP8266 NodeMCU by displaying WiFi.localIP() value on serial monitor.

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

In order to handle incoming HTTP requests, we need to specify which code to execute when a URL is hit. To do so, we use **on** method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

The code below indicates that when a server receives an HTTP request on the root (**/**) path, it will trigger the handle\_OnConnect function. Note that the URL specified is a relative path.

server.on("/", handle\_OnConnect);

We haven’t specified what the server should do if the client requests any URL other than specified with **server.on**. It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use server.onNotFound to tell it that it should execute it when it receives a request for a URL that wasn’t specified with server.on

server.onNotFound(handle\_NotFound);

Now, to start our server, we call the begin method on the server object.

server.begin();

Serial.println("HTTP server started");

### Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the handleClient() method on the server object.

server.handleClient();

Next, we need to create a function we attached to root (/) URL with server.on. Remember? At the start of this function, we get the values of temperature and humidity from the sensor. In order to respond to the HTTP request, we use the **send** method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code **200** (one of the [HTTP status codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)), which corresponds to the **OK** response. Then, we are specifying the content type as “text/html“, and finally we are calling SendHTML() custom function which creates a dynamic HTML page containing values of temperature and humidity.

void handle\_OnConnect() {

Temperature = dht.readTemperature(); // Gets the values of the temperature

Humidity = dht.readHumidity(); // Gets the values of the humidity

server.send(200, "text/html", SendHTML(Temperature,Humidity));

}

Likewise, we need to create a function to handle 404 Error page.

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

### Displaying the HTML Web Page

SendHTML() function is responsible for generating a web page whenever the ESP8266 NodeMCU web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the server.send() function we discussed earlier. The function takes values of temperature and humidity as a parameter to dynamically generate the HTML content.

The first text you should always send is the <!DOCTYPE> declaration that indicates that we’re sending HTML code.

String SendHTML(float Temperaturestat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

Next, the <meta> viewport element makes the web page responsive in any web browser, while title tag sets the title of the page.

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

### Styling the Web Page

Next, we have some CSS to style the web page appearance. We choose the Helvetica font, define the content to be displayed as an inline-block and aligned at the center.

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

Following code then sets color, font and margin around the body, H1 and p tags.

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

### Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Report</h1>\n";

### Displaying Temperature and Humidity on Web Page

To dynamically display values of Temperature & humidity, we put those values in paragraph tag. These values are converted to integer by type casting. To display degree symbol, we use HTML entity **&deg;** .

ptr +="<p>Temperature: ";

ptr +=(int)Temperaturestat;

ptr +="°C</p>";

ptr +="<p>Humidity: ";

ptr +=(int)Humiditystat;

ptr +="%</p>";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

## Styling Web Page to Look More Professional

Programmers like us are often intimidated by design – but a little effort can make your web page look more attractive and professional. Below screenshot will give you a basic idea of what we are going to do.



Pretty amazing, Right? Without further ado, let’s apply some style to our previous HTML page. To start with, copy-paste below code to replace **SendHTML()** function from the sketch above. Try the new sketch out and then we will do its detailed breakdown.

String SendHTML(float TempCstat,float TempFstat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<link href=\"https://fonts.googleapis.com/css?family=Open+Sans:300,400,600\" rel=\"stylesheet\">\n";

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #333333;}\n";

ptr +="body{margin-top: 50px;}\n";

ptr +="h1 {margin: 50px auto 30px;}\n";

ptr +=".side-by-side{display: inline-block;vertical-align: middle;position: relative;}\n";

ptr +=".humidity-icon{background-color: #3498db;width: 30px;height: 30px;border-radius: 50%;line-height: 36px;}\n";

ptr +=".humidity-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".humidity{font-weight: 300;font-size: 60px;color: #3498db;}\n";

ptr +=".temperature-icon{background-color: #f39c12;width: 30px;height: 30px;border-radius: 50%;line-height: 40px;}\n";

ptr +=".temperature-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".temperature{font-weight: 300;font-size: 60px;color: #f39c12;}\n";

ptr +=".superscript{font-size: 17px;font-weight: 600;position: absolute;right: -20px;top: 15px;}\n";

ptr +=".data{padding: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Report</h1>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side temperature-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_1\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n";

ptr +="width=\"9.915px\" height=\"22px\" viewBox=\"0 0 9.915 22\" enable-background=\"new 0 0 9.915 22\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M3.498,0.53c0.377-0.331,0.877-0.501,1.374-0.527C5.697-0.04,6.522,0.421,6.924,1.142\n";

ptr +="c0.237,0.399,0.315,0.871,0.311,1.33C7.229,5.856,7.245,9.24,7.227,12.625c1.019,0.539,1.855,1.424,2.301,2.491\n";

ptr +="c0.491,1.163,0.518,2.514,0.062,3.693c-0.414,1.102-1.24,2.038-2.276,2.594c-1.056,0.583-2.331,0.743-3.501,0.463\n";

ptr +="c-1.417-0.323-2.659-1.314-3.3-2.617C0.014,18.26-0.115,17.104,0.1,16.022c0.296-1.443,1.274-2.717,2.58-3.394\n";

ptr +="c0.013-3.44,0-6.881,0.007-10.322C2.674,1.634,2.974,0.955,3.498,0.53z\"/>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side temperature-text\">Temperature</div>\n";

ptr +="<div class=\"side-by-side temperature\">";

ptr +=(int)TempCstat;

ptr +="<span class=\"superscript\">°C</span></div>\n";

ptr +="</div>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side humidity-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_2\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n\"; width=\"12px\" height=\"17.955px\" viewBox=\"0 0 13 17.955\" enable-background=\"new 0 0 13 17.955\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M1.819,6.217C3.139,4.064,6.5,0,6.5,0s3.363,4.064,4.681,6.217c1.793,2.926,2.133,5.05,1.571,7.057\n";

ptr +="c-0.438,1.574-2.264,4.681-6.252,4.681c-3.988,0-5.813-3.107-6.252-4.681C-0.313,11.267,0.026,9.143,1.819,6.217\"></path>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side humidity-text\">Humidity</div>\n";

ptr +="<div class=\"side-by-side humidity\">";

ptr +=(int)Humiditystat;

ptr +="<span class=\"superscript\">%</span></div>\n";

ptr +="</div>\n";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

We already know that <!DOCTYPE> declaration tells the browser we’re sending HTML code and the <meta> viewport element makes the web page responsive. The only addition here is we are going to use [google fonts](https://fonts.google.com/). Google has hundreds of web fonts and are free for commercial and personal use. Perfect!

We will use Google commissioned [Open Sans](https://fonts.google.com/specimen/Open+Sans) web font for our web page. Google font is embedded using link tag in <head> of your HTML document. We have selected 300 (Light), 400 (Regular) & 600 (bold) font weight for our page. You can select as many as you want but keep in mind that selecting unnecessary font weights hamper page load time. You can also add italic style by just adding i character at the end of font weight e.g. 400i will embed italic style.

Note that you cannot see Google font, without active internet connection on the device, you are going to access this page. Google fonts are loaded on the fly.

String SendHTML(float TempCstat,float TempFstat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<link href=\"https://fonts.googleapis.com/css?family=Open+Sans:300,400,600\" rel=\"stylesheet\">\n";

Next, we are going to apply ‘Open Sans’ font for our whole HTML. We also need to specify sans-serif as our fallback font, to ensure maximum compatibility between browsers/operating systems. If the browser does not support the first font, it tries the next font.

ptr +="<title>ESP8266 Weather Report</title>\n";

ptr +="<style>html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #333333;}\n";

ptr +="body{margin-top: 50px;}\n";

ptr +="h1 {margin: 50px auto 30px;}\n";

Next, we need to apply CSS for icons, titles and actual values of Humidity & Temperature. All these three things are made inline and aligned vertical. Background of icons is made circle using 50% border radius and made 30px height and width.

ptr +=".side-by-side{display: inline-block;vertical-align: middle;position: relative;}\n";

ptr +=".humidity-icon{background-color: #3498db;width: 30px;height: 30px;border-radius: 50%;line-height: 36px;}\n";

ptr +=".humidity-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".humidity{font-weight: 300;font-size: 60px;color: #3498db;}\n";

ptr +=".temperature-icon{background-color: #f39c12;width: 30px;height: 30px;border-radius: 50%;line-height: 40px;}\n";

ptr +=".temperature-text{font-weight: 600;padding-left: 15px;font-size: 19px;width: 160px;text-align: left;}\n";

ptr +=".temperature{font-weight: 300;font-size: 60px;color: #f39c12;}\n";

ptr +=".superscript{font-size: 17px;font-weight: 600;position: absolute;right: -20px;top: 15px;}\n";

ptr +=".data{padding: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

Next, we will show temperature readings with this  nice little icon.

The temperature icon is actually a [Scalable Vector Graphics](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics) (SVG) defined in <svg> tag. Creating SVG doesn’t require any special programming skills. You can use [Google SVG Editor](https://svg-edit.github.io/svgedit/releases/latest/editor/svg-editor.html) for creating graphics for your page. After the icon, we are going to show the actual value of temperature from the sensor.

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 NodeMCU Weather Report</h1>\n";

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side temperature-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_1\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n";

ptr +="width=\"9.915px\" height=\"22px\" viewBox=\"0 0 9.915 22\" enable-background=\"new 0 0 9.915 22\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M3.498,0.53c0.377-0.331,0.877-0.501,1.374-0.527C5.697-0.04,6.522,0.421,6.924,1.142\n";

ptr +="c0.237,0.399,0.315,0.871,0.311,1.33C7.229,5.856,7.245,9.24,7.227,12.625c1.019,0.539,1.855,1.424,2.301,2.491\n";

ptr +="c0.491,1.163,0.518,2.514,0.062,3.693c-0.414,1.102-1.24,2.038-2.276,2.594c-1.056,0.583-2.331,0.743-3.501,0.463\n";

ptr +="c-1.417-0.323-2.659-1.314-3.3-2.617C0.014,18.26-0.115,17.104,0.1,16.022c0.296-1.443,1.274-2.717,2.58-3.394\n";

ptr +="c0.013-3.44,0-6.881,0.007-10.322C2.674,1.634,2.974,0.955,3.498,0.53z\"/>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side temperature-text\">Temperature</div>\n";

ptr +="<div class=\"side-by-side temperature\">";

ptr +=(int)TempCstat;

ptr +="<span class=\"superscript\">°C</span></div>\n";

ptr +="</div>\n";

Next, we will show humidity readings with this  icon.

Again it’s a SVG. After printing humidity values, we are going to end all the open tags like body and html.

ptr +="<div class=\"data\">\n";

ptr +="<div class=\"side-by-side humidity-icon\">\n";

ptr +="<svg version=\"1.1\" id=\"Layer\_2\" xmlns=\"http://www.w3.org/2000/svg\" xmlns:xlink=\"http://www.w3.org/1999/xlink\" x=\"0px\" y=\"0px\"\n\"; width=\"12px\" height=\"17.955px\" viewBox=\"0 0 13 17.955\" enable-background=\"new 0 0 13 17.955\" xml:space=\"preserve\">\n";

ptr +="<path fill=\"#FFFFFF\" d=\"M1.819,6.217C3.139,4.064,6.5,0,6.5,0s3.363,4.064,4.681,6.217c1.793,2.926,2.133,5.05,1.571,7.057\n";

ptr +="c-0.438,1.574-2.264,4.681-6.252,4.681c-3.988,0-5.813-3.107-6.252-4.681C-0.313,11.267,0.026,9.143,1.819,6.217\"></path>\n";

ptr +="</svg>\n";

ptr +="</div>\n";

ptr +="<div class=\"side-by-side humidity-text\">Humidity</div>\n";

ptr +="<div class=\"side-by-side humidity\">";

ptr +=(int)Humiditystat;

ptr +="<span class=\"superscript\">%</span></div>\n";

ptr +="</div>\n";

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;

}

## Improvement to the Code – Auto Page Refresh

One of the improvements you can do with our code is refreshing the page automatically in order to update the sensor value.

With the addition of a single meta tag into your HTML document, you can instruct the browser to automatically reload the page at a provided interval.

<meta http-equiv="refresh" content="2" >

Place this code in the the <head> tag of your document, this meta tag will instruct the browser to refresh every two seconds. Pretty nifty!

## Dynamically load Sensor Data with AJAX

Refreshing a web page isn’t too practical if you have a heavy web page. A better method is to use [Asynchronous Javascript And Xml](https://en.wikipedia.org/wiki/Ajax_(programming)) (**AJAX**) so that we can request data from the server asynchronously (in the background) without refreshing the page.

The [XMLHttpRequest](https://en.wikipedia.org/wiki/XMLHttpRequest) object within JavaScript is commonly used to execute AJAX on webpages. It performs the silent GET request on the server and updates the element on the page. AJAX is not a new technology, or different language, just existing technologies used in new ways. Besides this, AJAX also makes it possible to

* Request data from a server after the page has loaded
* Receive data from a server after the page has loaded
* Send data to a server in the background

Here is the AJAX script that we’ll be using. Place this script just before you close </head> tag.

ptr +="<script>\n";

ptr +="setInterval(loadDoc,200);\n";

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.getElementById(\"webpage\").innerHTML =this.responseText}\n";

ptr +="};\n";

ptr +="xhttp.open(\"GET\", \"/\", true);\n";

ptr +="xhttp.send();\n";

ptr +="}\n";

ptr +="</script>\n";

The script starts with <script> tag, as AJAX script is nothing but a javascript so, we need to write it in <script> tag. In order for this function to be repeatedly called, we will be using the javascript setInterval() function. It takes two parameters – a function to be executed and time interval (in milliseconds) on how often to execute the function.

ptr +="<script>\n";

ptr +="setInterval(loadDoc,200);\n";

The heart of this script is a loadDoc() function. Inside this function, an XMLHttpRequest() object is created. This object is used to request data from a web server.

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

The xhttp.onreadystatechange() function is called every time the readyState changes. The readyState property holds the status of the XMLHttpRequest. It has one of the following values.

* 0: request not initialized
* 1: server connection established
* 2: request received
* 3: processing request
* 4: request finished and response is ready

The status property holds the status of the XMLHttpRequest object. It has one of the following values.

* 200: “OK”
* 403: “Forbidden”
* 404: “Page not found”

When readyState is 4 and status is 200, the response is ready. Now, the content of element with id **webpage** (div holding values of temperature & humidity) is updated.

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.getElementById(\"webpage\").innerHTML =this.responseText}\n";

ptr +="};\n";

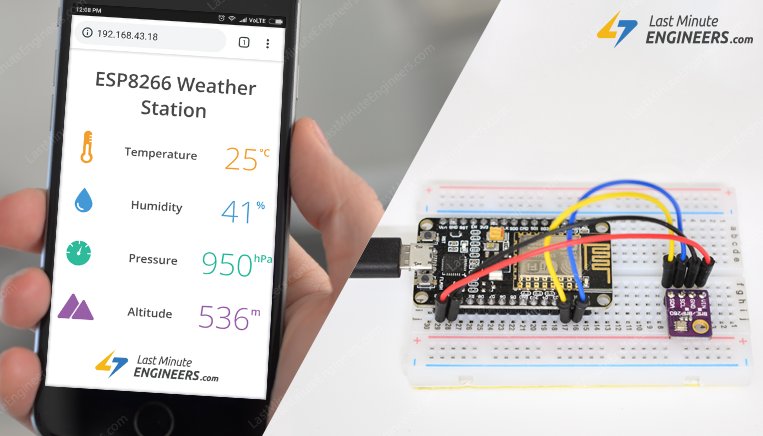
The HTTP request is then initiated via the open() and send() functions.

ptr +="xhttp.open(\"GET\", \"/\", true);\n";

ptr +="xhttp.send();\n";

ptr +="}\n";

# Create A Simple ESP8266 Weather Station With BME280



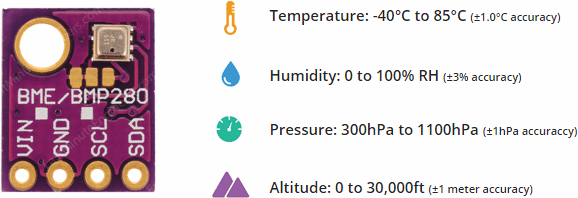
Don’t let the smartphone weather apps or commercial weather stations(that feeds you with data from stations based miles away) ruin your outdoor plans. With this IoT project you can be your own weatherman!

This project uses ESP8266 NodeMCU as the control device that easily connects to existing WiFi network & creates a Web Server. When any connected device accesses this web server, ESP8266 reads in temperature, humidity, barometric pressure & altitude from BME280 & sends it to the web browser of that device with a nice interface. Excited? Let’s get started!

## BME280 Temperature, Humidity and Pressure Sensor

First, let’s take a quick look at the BME280 module.

BME280 is the next-generation digital temperature, humidity and pressure sensor manufactured by Bosch. It’s a successor to sensors like BMP180, BMP085 or BMP183.



The operating voltage of the BME280 module is from **3.3V to 5V** – Perfect for interfacing with 3.3V microcontrollers like ESP8266.

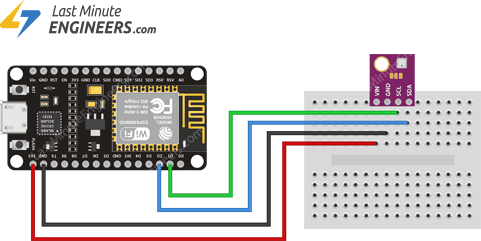
The module features a simple two-wire **I2C interface** for communication. The default I2C address of the BME280 module is **0x76** and can be changed to 0x77 easily with [this procedure](https://lastminuteengineers.com/bme280-arduino-tutorial/#procedure-to-change-i2c-address).

## Wiring BME280 Sensor to ESP8266 NodeMCU

Connections are fairly simple. Start by connecting **VIN** pin to the 3.3V output on the ESP8266 NodeMCU and connect **GND**to ground.

Next, Connect the **SCL** pin to the I2C clock **D1** pin on your ESP8266 and connect the **SDA** pin to the I2C data **D2**pin on your ESP8266.

The following diagram shows you how to wire everything.

Wiring ESP8266 & BME280 Temperature Humidity Pressure Sensor

## Preparing the Arduino IDE

There’s an add-on for the Arduino IDE that allows you to program the ESP8266 NodeMCU using the Arduino IDE. Follow below tutorial to prepare your Arduino IDE to work with the ESP8266, if you haven’t already.

[](https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/)

[Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE](https://lastminuteengineers.com/esp8266-nodemcu-arduino-tutorial/)

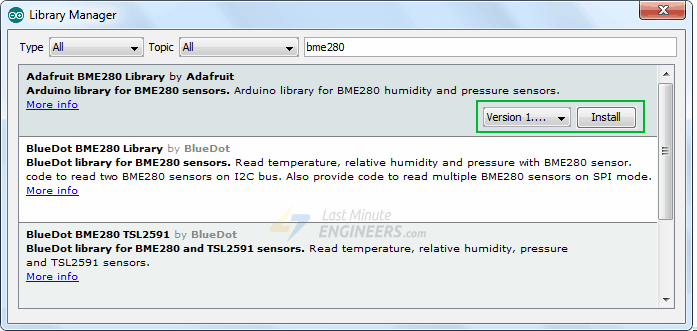
The Internet of Things (IoT) has been a trending field in the world of technology. It has changed the way we work. Physical objects and...

### Installing Library For BME280

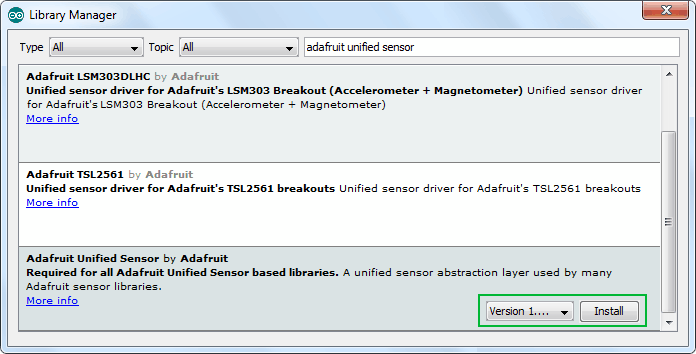
Communicating with a BME280 module is a bunch of work. Fortunately, [Adafruit BME280 Library](https://github.com/adafruit/Adafruit_BME280_Library) was written to hide away all the complexities so that we can issue simple commands to read the temperature, relative humidity & barometric pressure data.

To install the library navigate to the **Arduino IDE > Sketch > Include Library > Manage Libraries…** Wait for Library Manager to download libraries index and update list of installed libraries.

Filter your search by typing ‘**bme280**’. There should be a couple entries. Look for **Adafruit BME280 Library** by **Adafruit**. Click on that entry, and then select Install.



The BME280 sensor library uses the [Adafruit Sensor support backend](https://github.com/adafruit/Adafruit_Sensor). So, search the library manager for **Adafruit Unified Sensor** and install that too (you may have to scroll a bit)



## Displaying Temperature, Humidity, Pressure & Altitude On ESP8266 Web Server

Now, we are going to configure our ESP8266 into Station (STA) mode, and create a web server to serve up web pages to any connected client under existing network.

If you want to learn about creating a web server with ESP8266 in AP/STA mode, check this tutorial out.

[](https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/)

[Create A Simple ESP8266 NodeMCU Web Server In Arduino IDE](https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/)

Over the past few years, the ESP8266 has been a growing star among IoT or WiFi-related projects. It’s an extremely cost-effective WiFi module that -...

Before you head for uploading the sketch, you need to **make one change** to make it work for you. You need to modify the following two variables with your network credentials, so that ESP8266 can establish a connection with existing network.

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

Once you are done, go ahead and try the sketch out.

#include <ESP8266WebServer.h>

#include <Wire.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

#define SEALEVELPRESSURE\_HPA (1013.25)

Adafruit\_BME280 bme;

float temperature, humidity, pressure, altitude;

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);

void setup() {

Serial.begin(115200);

delay(100);

bme.begin(0x76);

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle\_OnConnect);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

server.handleClient();

}

void handle\_OnConnect() {

temperature = bme.readTemperature();

humidity = bme.readHumidity();

pressure = bme.readPressure() / 100.0F;

altitude = bme.readAltitude(SEALEVELPRESSURE\_HPA);

server.send(200, "text/html", SendHTML(temperature,humidity,pressure,altitude));

}

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

String SendHTML(float temperature,float humidity,float pressure,float altitude){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Station</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Station</h1>\n";

ptr +="<p>Temperature: ";

ptr +=temperature;

ptr +="&deg;C</p>";

ptr +="<p>Humidity: ";

ptr +=humidity;

ptr +="%</p>";

ptr +="<p>Pressure: ";

ptr +=pressure;

ptr +="hPa</p>";

ptr +="<p>Altitude: ";

ptr +=altitude;

ptr +="m</p>";

ptr +="</div>\n";

ptr +="</body>\n";

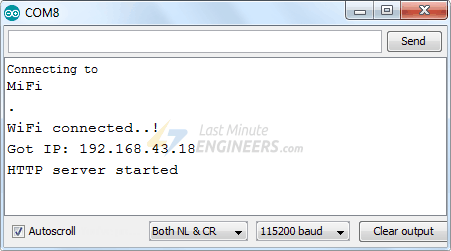
ptr +="</html>\n";

return ptr;

}

## Accessing the Web Server

After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the EN button on NodeMCU. If everything is OK, it will output the dynamic IP address obtained from your router and show **HTTP server started** message.



Next, load up a browser and point it to the IP address shown on the serial monitor. The ESP8266 should serve up a web page showing temperature, humidity, pressure and altitude from BME280.



## Detailed Code Explanation

The sketch starts by including following libraries.

* **ESP8266WebServer.h** library provides ESP8266 specific WiFi methods we are calling to connect to network. It also has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details.
* **Wire.h** library communicates with any I2C device not just BME280.
* **Adafruit\_BME280.h & Adafruit\_Sensor.h** libraries are hardware-specific libraries which handles lower-level functions.

#include <ESP8266WebServer.h>

#include <Wire.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

Next we create an object of the sensor and variables to store temperature, humidity, pressure and altitude.

#define SEALEVELPRESSURE\_HPA (1013.25)

Adafruit\_BME280 bme;

float temperature, humidity, pressure, altitude;

As we are configuring ESP8266 in Station (STA) mode, it will join existing WiFi network. Hence, we need to provide it with your network’s SSID & Password. Next we start web server at port 80.

/\*Put your SSID & Password\*/

const char\* ssid = "YourNetworkName"; // Enter SSID here

const char\* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);

### Inside Setup() Function

Inside Setup() Function we configure our HTTP server before actually running it.

First of all, we initialize serial communication with PC and initialize BME object using begin() function. It initializes I2C interface with given I2C Address(0x76) and checks if the chip ID is correct. It then resets the chip using soft-reset & waits for the sensor for calibration after wake-up.

Serial.begin(115200);

delay(100);

bme.begin(0x76);

Now, we need to join the WiFi network using WiFi.begin() function. The function takes SSID (Network Name) and password as a parameter.

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

While the ESP8266 tries to connect to the network, we can check the connectivity status with WiFi.status() function.

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

Once the ESP8266 is connected to the network, the sketch prints the IP address assigned to ESP8266 by displaying WiFi.localIP() value on serial monitor.

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

In order to handle incoming HTTP requests, we need to specify which code to execute when a URL is hit. To do so, we use **on** method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

The code below indicates that when a server receives an HTTP request on the root (**/**) path, it will trigger the handle\_OnConnect function. Note that the URL specified is a relative path.

server.on("/", handle\_OnConnect);

We haven’t specified what the server should do if the client requests any URL other than specified with server.on. It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use server.onNotFound to tell it that it should execute it when it receives a request for a URL that wasn’t specified with server.on

server.onNotFound(handle\_NotFound);

Now, to start our server, we call the begin method on the server object.

server.begin();

Serial.println("HTTP server started");

### Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the handleClient() method on the server object.

server.handleClient();

Next, we need to create a function we attached to root (/) URL with server.on Remember?

At the start of this function, we get the temperature, humidity, pressure & altitude readings from the sensor. In order to respond to the HTTP request, we use the **send** method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code **200** (one of the [HTTP status codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)), which corresponds to the **OK** response. Then, we are specifying the content type as “text/html“, and finally we are calling SendHTML() custom function which creates a dynamic HTML page containing temperature, humidity, pressure & altitude readings.

void handle\_OnConnect() {

temperature = bme.readTemperature();

humidity = bme.readHumidity();

pressure = bme.readPressure() / 100.0F;

altitude = bme.readAltitude(SEALEVELPRESSURE\_HPA);

server.send(200, "text/html", SendHTML(temperature,humidity,pressure,altitude));

}

Likewise, we need to create a function to handle 404 Error page.

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

### Displaying the HTML Web Page

SendHTML() function is responsible for generating a web page whenever the ESP8266 web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the server.send() function we discussed earlier. The function takes temperature, humidity, pressure & altitude readings as a parameter to dynamically generate the HTML content.

The first text you should always send is the <!DOCTYPE> declaration that indicates that we’re sending HTML code.

String SendHTML(float temperature,float humidity,float pressure,float altitude){

String ptr = "<!DOCTYPE html> <html>\n";

Next, the <meta> viewport element makes the web page responsive in any web browser, while title tag sets the title of the page.

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>ESP8266 Weather Station</title>\n";

### Styling the Web Page

Next, we have some CSS to style the web page appearance. We choose the Helvetica font, define the content to be displayed as an inline-block and aligned at the center.

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

Following code sets color, font and margin around the body, H1 and p tags.

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

### Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>ESP8266 Weather Station</h1>\n";

### Displaying Readings on Web Page

To dynamically display temperature, humidity, pressure & altitude readings, we put those values in paragraph tag. To display degree symbol, we use HTML entity **&deg;**

ptr +="<p>Temperature: ";

ptr +=temperature;

ptr +="&deg;C</p>";

ptr +="<p>Humidity: ";

ptr +=humidity;

ptr +="%</p>";

ptr +="<p>Pressure: ";

ptr +=pressure;

ptr +="hPa</p>";

ptr +="<p>Altitude: ";

ptr +=altitude;

ptr +="m</p>";

ptr +="</div>\n";

ptr +="</body>\n";

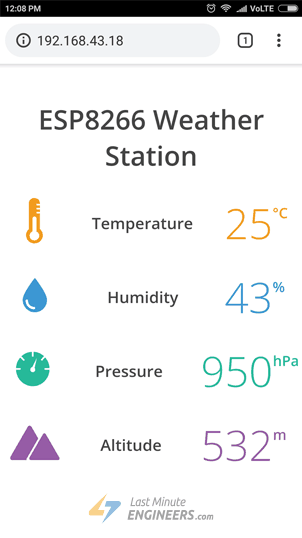
ptr +="</html>\n";

return ptr;

}

## Styling Web Page to Look More Professional

Programmers like us are often intimidated by design – but a little effort can make your web page look more attractive and professional. Below screenshot will give you a basic idea of what we are going to do.



Pretty amazing, Right? Without further ado, let’s apply some style to our previous HTML page. To start with, copy-paste below code to replace SendHTML() function from the sketch above.

String SendHTML(float temperature,float humidity,float pressure,float altitude){

String ptr = "<!DOCTYPE html>";

ptr +="<html>";

ptr +="<head>";

ptr +="<title>ESP8266 Weather Station</title>";

ptr +="<meta name='viewport' content='width=device-width, initial-scale=1.0'>";

ptr +="<link href='https://fonts.googleapis.com/css?family=Open+Sans:300,400,600' rel='stylesheet'>";

ptr +="<style>";

ptr +="html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #444444;}";

ptr +="body{margin: 0px;} ";

ptr +="h1 {margin: 50px auto 30px;} ";

ptr +=".side-by-side{display: table-cell;vertical-align: middle;position: relative;}";

ptr +=".text{font-weight: 600;font-size: 19px;width: 200px;}";

ptr +=".reading{font-weight: 300;font-size: 50px;padding-right: 25px;}";

ptr +=".temperature .reading{color: #F29C1F;}";

ptr +=".humidity .reading{color: #3B97D3;}";

ptr +=".pressure .reading{color: #26B99A;}";

ptr +=".altitude .reading{color: #955BA5;}";

ptr +=".superscript{font-size: 17px;font-weight: 600;position: absolute;top: 10px;}";

ptr +=".data{padding: 10px;}";

ptr +=".container{display: table;margin: 0 auto;}";

ptr +=".icon{width:65px}";

ptr +="</style>";

ptr +="</head>";

ptr +="<body>";

ptr +="<h1>ESP8266 Weather Station</h1>";

ptr +="<div class='container'>";

ptr +="<div class='data temperature'>";

ptr +="<div class='side-by-side icon'>";

ptr +="<svg enable-background='new 0 0 19.438 54.003'height=54.003px id=Layer\_1 version=1.1 viewBox='0 0 19.438 54.003'width=19.438px x=0px xml:space=preserve xmlns=http://www.w3.org/2000/svg xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M11.976,8.82v-2h4.084V6.063C16.06,2.715,13.345,0,9.996,0H9.313C5.965,0,3.252,2.715,3.252,6.063v30.982";

ptr +="C1.261,38.825,0,41.403,0,44.286c0,5.367,4.351,9.718,9.719,9.718c5.368,0,9.719-4.351,9.719-9.718";

ptr +="c0-2.943-1.312-5.574-3.378-7.355V18.436h-3.914v-2h3.914v-2.808h-4.084v-2h4.084V8.82H11.976z M15.302,44.833";

ptr +="c0,3.083-2.5,5.583-5.583,5.583s-5.583-2.5-5.583-5.583c0-2.279,1.368-4.236,3.326-5.104V24.257C7.462,23.01,8.472,22,9.719,22";

ptr +="s2.257,1.01,2.257,2.257V39.73C13.934,40.597,15.302,42.554,15.302,44.833z'fill=#F29C21 /></g></svg>";

ptr +="</div>";

ptr +="<div class='side-by-side text'>Temperature</div>";

ptr +="<div class='side-by-side reading'>";

ptr +=(int)temperature;

ptr +="<span class='superscript'>&deg;C</span></div>";

ptr +="</div>";

ptr +="<div class='data humidity'>";

ptr +="<div class='side-by-side icon'>";

ptr +="<svg enable-background='new 0 0 29.235 40.64'height=40.64px id=Layer\_1 version=1.1 viewBox='0 0 29.235 40.64'width=29.235px x=0px xml:space=preserve xmlns=http://www.w3.org/2000/svg xmlns:xlink=http://www.w3.org/1999/xlink y=0px><path d='M14.618,0C14.618,0,0,17.95,0,26.022C0,34.096,6.544,40.64,14.618,40.64s14.617-6.544,14.617-14.617";

ptr +="C29.235,17.95,14.618,0,14.618,0z M13.667,37.135c-5.604,0-10.162-4.56-10.162-10.162c0-0.787,0.638-1.426,1.426-1.426";

ptr +="c0.787,0,1.425,0.639,1.425,1.426c0,4.031,3.28,7.312,7.311,7.312c0.787,0,1.425,0.638,1.425,1.425";

ptr +="C15.093,36.497,14.455,37.135,13.667,37.135z'fill=#3C97D3 /></svg>";

ptr +="</div>";

ptr +="<div class='side-by-side text'>Humidity</div>";

ptr +="<div class='side-by-side reading'>";

ptr +=(int)humidity;

ptr +="<span class='superscript'>%</span></div>";

ptr +="</div>";

ptr +="<div class='data pressure'>";

ptr +="<div class='side-by-side icon'>";

ptr +="<svg enable-background='new 0 0 40.542 40.541'height=40.541px id=Layer\_1 version=1.1 viewBox='0 0 40.542 40.541'width=40.542px x=0px xml:space=preserve xmlns=http://www.w3.org/2000/svg xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M34.313,20.271c0-0.552,0.447-1,1-1h5.178c-0.236-4.841-2.163-9.228-5.214-12.593l-3.425,3.424";

ptr +="c-0.195,0.195-0.451,0.293-0.707,0.293s-0.512-0.098-0.707-0.293c-0.391-0.391-0.391-1.023,0-1.414l3.425-3.424";

ptr +="c-3.375-3.059-7.776-4.987-12.634-5.215c0.015,0.067,0.041,0.13,0.041,0.202v4.687c0,0.552-0.447,1-1,1s-1-0.448-1-1V0.25";

ptr +="c0-0.071,0.026-0.134,0.041-0.202C14.39,0.279,9.936,2.256,6.544,5.385l3.576,3.577c0.391,0.391,0.391,1.024,0,1.414";

ptr +="c-0.195,0.195-0.451,0.293-0.707,0.293s-0.512-0.098-0.707-0.293L5.142,6.812c-2.98,3.348-4.858,7.682-5.092,12.459h4.804";

ptr +="c0.552,0,1,0.448,1,1s-0.448,1-1,1H0.05c0.525,10.728,9.362,19.271,20.22,19.271c10.857,0,19.696-8.543,20.22-19.271h-5.178";

ptr +="C34.76,21.271,34.313,20.823,34.313,20.271z M23.084,22.037c-0.559,1.561-2.274,2.372-3.833,1.814";

ptr +="c-1.561-0.557-2.373-2.272-1.815-3.833c0.372-1.041,1.263-1.737,2.277-1.928L25.2,7.202L22.497,19.05";

ptr +="C23.196,19.843,23.464,20.973,23.084,22.037z'fill=#26B999 /></g></svg>";

ptr +="</div>";

ptr +="<div class='side-by-side text'>Pressure</div>";

ptr +="<div class='side-by-side reading'>";

ptr +=(int)pressure;

ptr +="<span class='superscript'>hPa</span></div>";

ptr +="</div>";

ptr +="<div class='data altitude'>";

ptr +="<div class='side-by-side icon'>";

ptr +="<svg enable-background='new 0 0 58.422 40.639'height=40.639px id=Layer\_1 version=1.1 viewBox='0 0 58.422 40.639'width=58.422px x=0px xml:space=preserve xmlns=http://www.w3.org/2000/svg xmlns:xlink=http://www.w3.org/1999/xlink y=0px><g><path d='M58.203,37.754l0.007-0.004L42.09,9.935l-0.001,0.001c-0.356-0.543-0.969-0.902-1.667-0.902";

ptr +="c-0.655,0-1.231,0.32-1.595,0.808l-0.011-0.007l-0.039,0.067c-0.021,0.03-0.035,0.063-0.054,0.094L22.78,37.692l0.008,0.004";

ptr +="c-0.149,0.28-0.242,0.594-0.242,0.934c0,1.102,0.894,1.995,1.994,1.995v0.015h31.888c1.101,0,1.994-0.893,1.994-1.994";

ptr +="C58.422,38.323,58.339,38.024,58.203,37.754z'fill=#955BA5 /><path d='M19.704,38.674l-0.013-0.004l13.544-23.522L25.13,1.156l-0.002,0.001C24.671,0.459,23.885,0,22.985,0";

ptr +="c-0.84,0-1.582,0.41-2.051,1.038l-0.016-0.01L20.87,1.114c-0.025,0.039-0.046,0.082-0.068,0.124L0.299,36.851l0.013,0.004";

ptr +="C0.117,37.215,0,37.62,0,38.059c0,1.412,1.147,2.565,2.565,2.565v0.015h16.989c-0.091-0.256-0.149-0.526-0.149-0.813";

ptr +="C19.405,39.407,19.518,39.019,19.704,38.674z'fill=#955BA5 /></g></svg>";

ptr +="</div>";

ptr +="<div class='side-by-side text'>Altitude</div>";

ptr +="<div class='side-by-side reading'>";

ptr +=(int)altitude;

ptr +="<span class='superscript'>m</span></div>";

ptr +="</div>";

ptr +="</div>";

ptr +="</body>";

ptr +="</html>";

return ptr;

}

If you try to compare this function with the previous one, you’ll come to know that they are similar except these changes.

* We have used Google commissioned [Open Sans](https://fonts.google.com/specimen/Open+Sans) web font for our web page. Note that you cannot see Google font, without active internet connection on the device. Google fonts are loaded on the fly.

ptr +="<link href='https://fonts.googleapis.com/css?family=Open+Sans:300,400,600' rel='stylesheet'>";

* The icons used to display temperature, humidity, pressure & altitude readings are actually a [Scalable Vector Graphics](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics) (SVG) defined in <svg> tag. Creating SVG doesn’t require any special programming skills. You can use [Google SVG Editor](https://svg-edit.github.io/svgedit/releases/latest/editor/svg-editor.html) for creating graphics for your page. We have used these SVG icons.

## Improvement to the Code – Auto Page Refresh

One of the improvements you can do with our code is refreshing the page automatically in order to update the sensor value.

With the addition of a single meta tag into your HTML document, you can instruct the browser to automatically reload the page at a provided interval.

<meta http-equiv="refresh" content="2" >

Place this code in the the <head> tag of your document, this meta tag will instruct the browser to refresh every two seconds. Pretty nifty!

## Dynamically load Sensor Data with AJAX

Refreshing a web page isn’t too practical if you have a heavy web page. A better method is to use [Asynchronous Javascript And Xml](https://en.wikipedia.org/wiki/Ajax_(programming)) (**AJAX**) so that we can request data from the server asynchronously (in the background) without refreshing the page.

The [XMLHttpRequest](https://en.wikipedia.org/wiki/XMLHttpRequest) object within JavaScript is commonly used to execute AJAX on webpages. It performs the silent GET request on the server and updates the element on the page. AJAX is not a new technology, or different language, just existing technologies used in new ways. Besides this, AJAX also makes it possible to

* Request data from a server after the page has loaded
* Receive data from a server after the page has loaded
* Send data to a server in the background

Here is the AJAX script that we’ll be using. Place this script just before you close </head> tag.

ptr +="<script>\n";

ptr +="setInterval(loadDoc,1000);\n";

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.body.innerHTML =this.responseText}\n";

ptr +="};\n";

ptr +="xhttp.open(\"GET\", \"/\", true);\n";

ptr +="xhttp.send();\n";

ptr +="}\n";

ptr +="</script>\n";

The script starts with <script> tag. As AJAX script is nothing but a javascript, we need to write it in <script> tag. In order for this function to be repeatedly called, we will be using the javascript setInterval() function. It takes two parameters – a function to be executed and time interval (in milliseconds) on how often to execute the function.

ptr +="<script>\n";

ptr +="setInterval(loadDoc,1000);\n";

The heart of this script is a loadDoc() function. Inside this function, an XMLHttpRequest() object is created. This object is used to request data from a web server.

ptr +="function loadDoc() {\n";

ptr +="var xhttp = new XMLHttpRequest();\n";

The xhttp.onreadystatechange() function is called every time the readyState changes. The readyState property holds the status of the XMLHttpRequest. It has one of the following values.

* 0: request not initialized
* 1: server connection established
* 2: request received
* 3: processing request
* 4: request finished and response is ready

The status property holds the status of the XMLHttpRequest object. It has one of the following values.

* 200: “OK”
* 403: “Forbidden”
* 404: “Page not found”

When readyState is 4 and status is 200, the response is ready. Now, the content of **body** (holding temperature readings) is updated.

ptr +="xhttp.onreadystatechange = function() {\n";

ptr +="if (this.readyState == 4 && this.status == 200) {\n";

ptr +="document.body.innerHTML =this.responseText}\n";

ptr +="};\n";

The HTTP request is then initiated via the open() and send() functions.

ptr +="xhttp.open(\"GET\", \"/\", true);\n";

ptr +="xhttp.send();\n";

ptr +="}\n";