**IEEE QP Wiki**

Engineer. Design. Innovate. Mentor. Showcase

# NodeMCU Wifi

## Prerequisites

Arduino IDE:

**https://www.arduino.cc/en/main/software**

ESP8266 Arduino Core:

Note: In order to install the ESP8266 Arduino Core you also need to install Python 2.7 as well!

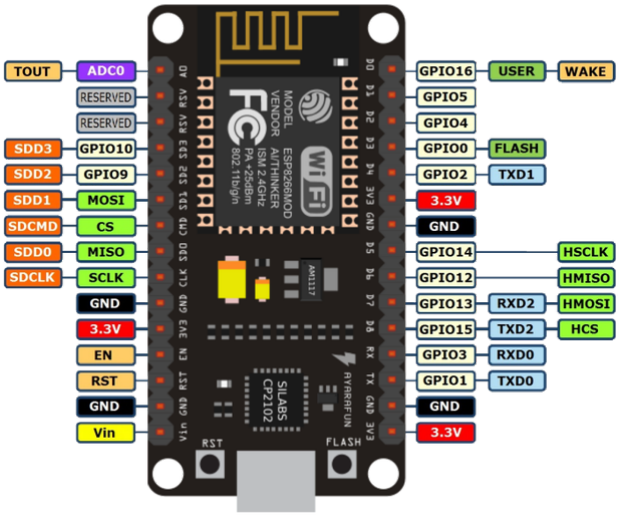
**https://arduino-esp8266.readthedocs.io/en/latest/installing.html**

WifiManager Arduino Library:

**https://github.com/tzapu/WiFiManager#quick-start**

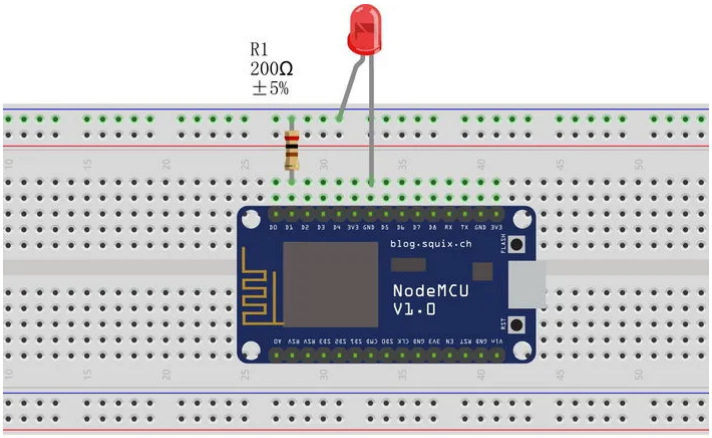
## Using the NodeMCU as a normal Arduino

One of the greatest advantages of the NodeMCU is that you can use it as a standard Arduino if you wanted to! The NodeMCU is capable of using many different devices, and can easily interface with many electronics, including motors and sensors. You can code the NodeMCU directly with LUA, but it's great to use the libraries that are available within the Arduino libraries. Right now were are just going to load the blink example to show how to load programs onto your NodeMCU!




Note in the pinout above, the pin labeled D1 is actually GPIO5. Therefore, in your Arduino code, you should use Pin 5 if you want to reference D1, use Pin 12 if you want to reference D6, etc.

Set up the circuit as shown below and make sure that




In order to select the NodeMCU, go to Tools > Board > and Select NodeMCU 1.0 (ESP-12E Module)

In order to test whether you have correctly installed all prerequisite software, upload a blank sketch as shown below to the device. This should take a while to upload, but if you are greeted with a 100% at the bottom of your IDE, meaning that you are ready to upload!

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Now, upload the code below and you should see your LED flashing on and off!

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital1 (GPIO5) as an output.
  pinMode(5, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(5, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(5, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

# Connecting to the Internet with WifiManager

As nice as it seems, most people would definitely not use the NodeMCU as just a simple Arduino, the strength of the NodeMCU is the ability to communicate via internet. Let's first connect to the internet. The easiest way to do this is to use the

What this program below does is that it sets up the NodeMCU to be used as a web server to host a small HTML site that contains a button to control an LED attached to the NodeMCU

Copy and paste the full code below into the Arduino IDE. Setting up internet is a one time ordeal to connect it to a single wifi connection. In order to set up this portion, uncomment wifiManager.resetSettings(); and then run the code. Make sure to comment this function back in the code so you don't have reconnect to the internet every time you turn on your device.

```
// Uncomment and run it once, if you want to erase all the stored inform
//wifiManager.resetSettings();
```

If you are doing this at the workshop or in a public place, make sure to change the SSID of the connection portal so you don't get your "AuttoConnectAP" mixed up with anyone else's. Rename

```
wifiManager.autoConnect("AutoConnectAP");
```

to

```
wifiManager.autoConnect("anything_you_want");
```

After you edit the code below with your changes, upload this code to your NodeMCU in order to

```
#include <ESP8266WiFi.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>         // https://github.com/tzapu/WiFiManager

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output5State = "off";

// Assign output variables to GPIO pins
const int output5 = 5;

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

  // Initialize the output variables as outputs
  pinMode(output5, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output5, LOW);

  // WiFiManager
  // Local intialization. Once its business is done, there is no need to k
  WiFiManager wifiManager;

  // Uncomment and run it once, if you want to erase all the stored inform
  //wifiManager.resetSettings();

  // set custom ip for portal
  //wifiManager.setAPConfig(IPAddress(10,0,1,1), IPAddress(10,0,1,1), IPAd

  // fetches ssid and pass from eeprom and tries to connect
  // if it does not connect it starts an access point with the specified n
  // here  "AutoConnectAP"
  // and goes into a blocking loop awaiting configuration
  wifiManager.autoConnect("AutoConnectAP"); //CHANGE THIS
  // or use this for auto generated name ESP + ChipID
  //wifiManager.autoConnect();

  // if you get here you have connected to the WiFi
  Serial.println("Connected.");

  server.begin();
```

```
  server.begin();
}

void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the
    String currentLine = "";                // make a String to hold incom
    while (client.connected()) {            // loop while the client's con

      if (client.available()) {             // if there's bytes to read fr
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial mon
        header += c;
        if (c == 'n') {                    // if the byte is a newline cha
          // if the current line is blank, you got two newline characters
          // that's the end of the client HTTP request, so send a response
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1
            // and a content-type so the client knows what's coming, then
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /5/on") >= 0) {
              Serial.println("GPIO 5 on");
              output5State = "on";
              digitalWrite(output5, HIGH);
            } else if (header.indexOf("GET /5/off") >= 0) {
              Serial.println("GPIO 5 off");
              output5State = "off";
              digitalWrite(output5, LOW);
            }

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name="viewport" content="width=dev
            client.println("<link rel="icon" href="data:,">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attr
            client.println("<style>html { font-family: Helvetica; display:
            client.println(".button { background-color: #195B6A; border: n
            client.println("text-decoration: none; font-size: 30px; margin
            client.println(".button2 {background-color: #77878A;}</style><

            // Web Page Heading
```

```
            // Web Page Heading
            client.println("<body><h1>IEEE Quarterly Projects ESP8266 Web

            // Display current state, and ON/OFF buttons for GPIO 5
            client.println("<p>LED - State " + output5State + "</p>");
            // If the output5State is off, it displays the ON button
            if (output5State=="off") {
              client.println("<p><a href="/5/on"><button class="button">ON
            } else {
              client.println("<p><a href="/5/off"><button class="button bu

            }


            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != 'r') {  // if you got anything else but a carriage
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

Make sure to have your Serial Monitor open for this, as you need to see what IP to connect to in order to finish

Connect to the ESP's wifi using your computer's network interface

In your serial monitor, it should also print an IP address. Open up your browser and type this IP address in your browser window, and press enter.

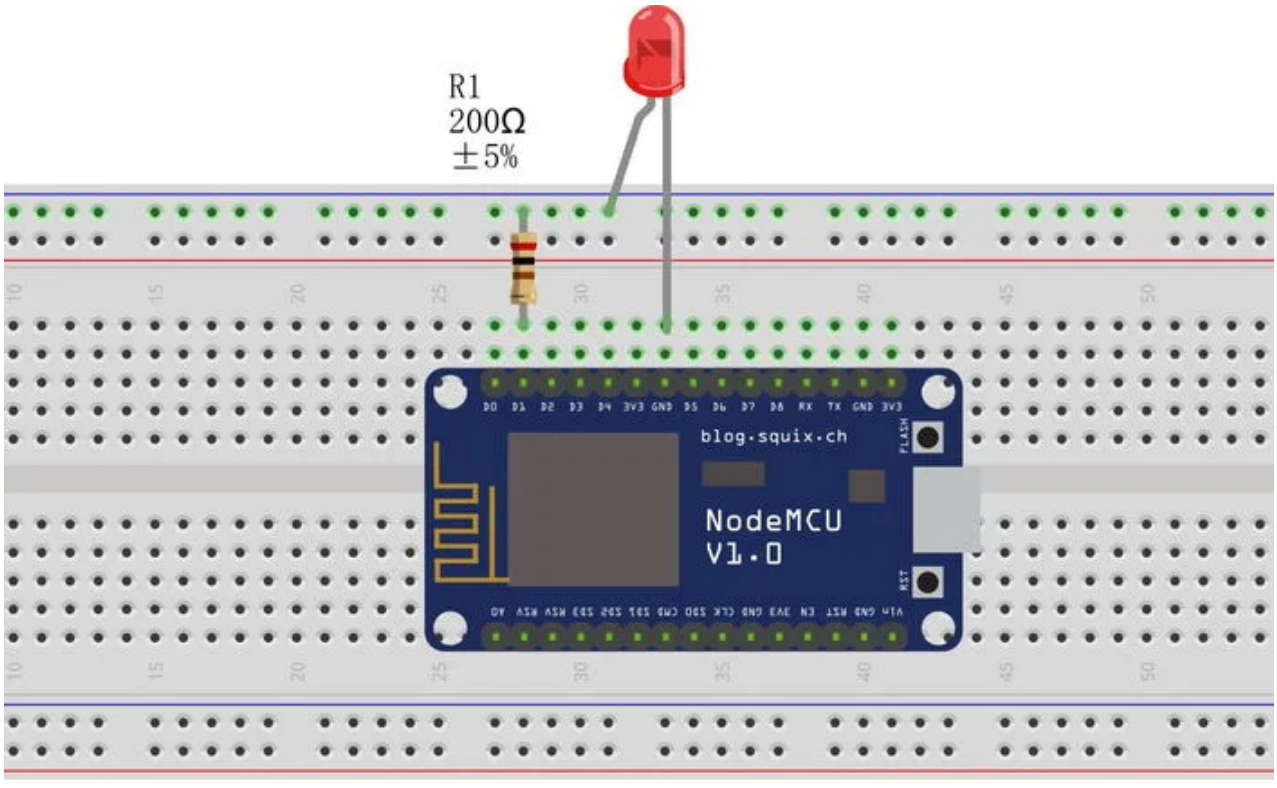
You should be greeted by this wonderful web interface to connect to a wifi network, which is much easier than fiddling around with SSID's and the password.

Go to Configure WiFi and select the Wifi connection that you want to connect to and type in the password. Click save and you should be good! Make sure to comment the wifiManager.resetSettings() function call afterwards! In this case I'm using my phone's hotspot because of finicky wifi!

# Start Simple: Controlling an LED

Connect to the same wifi network that your NodeMCU is connected to, this is a necessity. Make sure that you use your phone's hotspot or another access point that has a 2.4GHz connection.

Set up the ESP on a breadboard like the configuration below, with the long end of the LED (+) to D1, the short end (-) to the end of the resistor, and the other end of the resistor to GND.




Since you have already reuploaded the code, look at the Serial Monitor to see the IP address that you need to connect to and type it into your browser to connect it to website that the NodeMCU is hosting!

Once you have connected to this IP you should be met with a website that is hosted directly on the NodeMCU. Press the button in order to turn your LED on or off!

Just to give an explanation of the code:

These lines set up the server and set up the HTML along with the headers for the website.

```
WiFiClient client = server.available();   // Listen for incoming clients

if (client) {                             // If a new client connects,
  Serial.println("New Client.");          // print a message out in the
  String currentLine = "";                // make a String to hold incom
  while (client.connected()) {            // loop while the client's con
    if (client.available()) {             // if there's bytes to read fr
      char c = client.read();             // read a byte, then
      Serial.write(c);                    // print it out the serial mon
      header += c;
      if (c == 'n') {                    // if the byte is a newline cha
        // if the current line is blank, you got two newline characters
        // that's the end of the client HTTP request, so send a response
        if (currentLine.length() == 0) {
          // HTTP headers always start with a response code (e.g. HTTP/1
          // and a content-type so the client knows what's coming, then
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println("Connection: close");
          client.println();
```

These lines set up the style for the entire website in addition to the button objects!

```
client.println("<style>html { font-family: Helvetica; display: inline-bloc
client.println(".button { background-color: #195B6A; border: none; color:
client.println("text-decoration: none; font-size: 30px; margin: 2px; curso
client.println(".button2 {background-color: #77878A;}</style></head>");
```

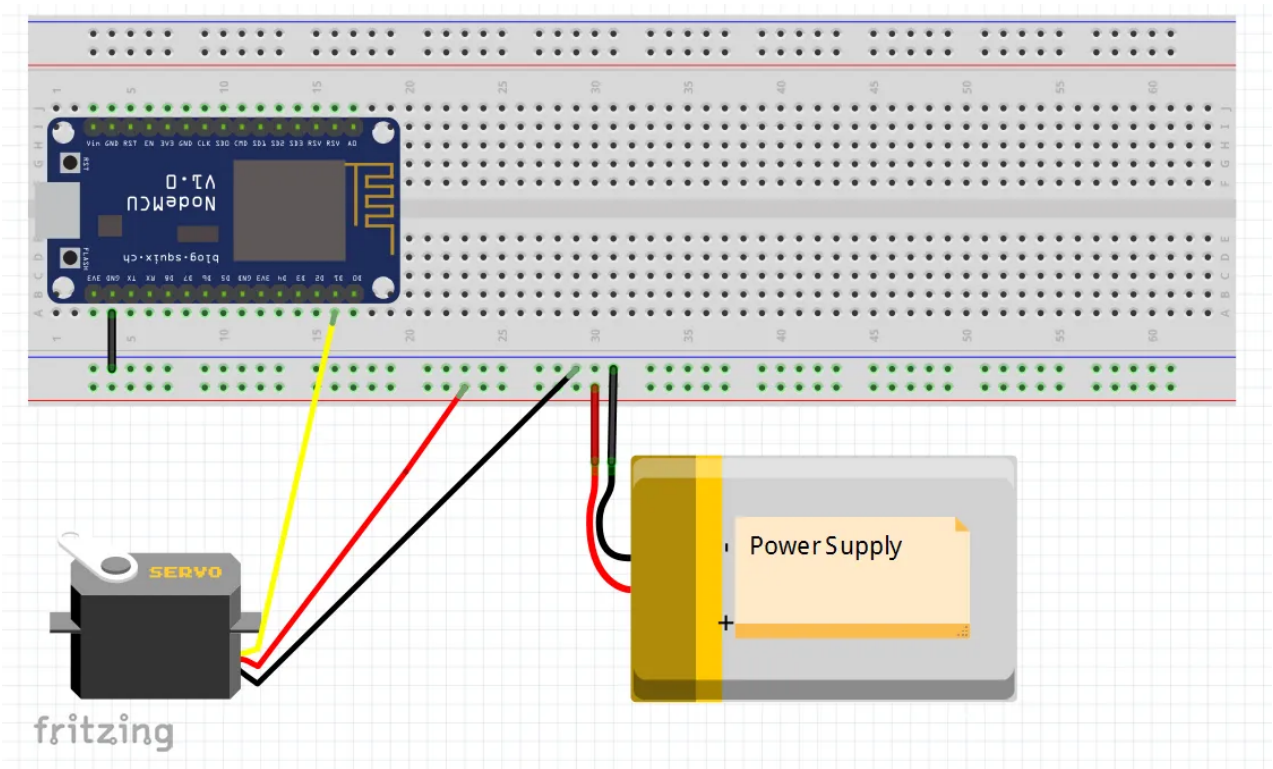
Below are the if statements for when the button is pressed, changing the state of the LED and the buttons styling

```
// Display current state, and ON/OFF buttons for GPIO 5
            client.println("<p>LED - State " + output5State + "</p>");
            // If the output5State is off, it displays the ON button
            if (output5State=="off") {
              client.println("<p><a href="/5/on"><button class="button">ON
            } else {
              client.println("<p><a href="/5/off"><button class="button bu
            }
```

# Using Motors

In the code above, you can see that we used digital pins in order to turn an LED on or off. This is used by using the digitalWrite() function, which will write values of high (3.3V) or low (0V) to the digital pins. You can also use these digital pins to control motors as well! We are using the Servo library for this, which is built into the Arduino IDE.

For most servos you have to use a external power supply such as the power supply below, but for smaller servos such as the SG90 (which you are using in this workshop, you can power it with the Vin port on the NodeMCU, since this port is 5V! (Try not to do this though, as most servos and motors need much more current than what that the NodeMCU can supply!



Here is the code that is changed in order to get your servo running:

```
#include <Servo.h>
#include <ESP8266WiFi.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>         // https://github.com/tzapu/WiFiManager

// Set web server port number to 80
WiFiServer server(80);
Servo servod1;
// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String servoPinState = "0";

// Assign output variables to GPIO pins
const int servoPin= 5;
void setup() {
  Serial.begin(115200);
  int pos = 0;
  // Initialize the output variables as outputs
  servod1.attach(servoPin);
  // WiFiManager
  // Local intialization. Once its business is done, there is no need to k
  WiFiManager wifiManager;

  // Uncomment and run it once, if you want to erase all the stored inform
  //wifiManager.resetSettings();

  // set custom ip for portal
  //wifiManager.setAPConfig(IPAddress(10,0,1,1), IPAddress(10,0,1,1), IPAd

  // fetches ssid and pass from eeprom and tries to connect
  // if it does not connect it starts an access point with the specified n
  // here  "AutoConnectAP"
  // and goes into a blocking loop awaiting configuration
  wifiManager.autoConnect("AutoConnectAP");
  // or use this for auto generated name ESP + ChipID
  //wifiManager.autoConnect();

  // if you get here you have connected to the WiFi
  Serial.println("Connected.");

  server.begin();
}

void loop(){
```

```
void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the
    String currentLine = "";                // make a String to hold incom
    while (client.connected()) {            // loop while the client's con
      if (client.available()) {             // if there's bytes to read fr
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial mon

        header += c;
        if (c == 'n') {                    // if the byte is a newline cha
          // if the current line is blank, you got two newline characters
          // that's the end of the client HTTP request, so send a response
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1
            // and a content-type so the client knows what's coming, then
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /5/on") >= 0) {
              Serial.println("Motor Turning to 180 Degrees");
              servoPinState = "on";
              servod1.write(180);

            } else if (header.indexOf("GET /5/off") >= 0) {
              Serial.println("Motor turning back to 0 degrees");
              servoPinState = "off";
              servod1.write(0);

            }
            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name="viewport" content="width=dev
            client.println("<link rel="icon" href="data:,">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attr
            client.println("<style>html { font-family: Helvetica; display:
            client.println(".button { background-color: #195B6A; border: n
            client.println("text-decoration: none; font-size: 30px; margin
            client.println(".button2 {background-color: #77878A;}</style><

            // Web Page Heading
            client.println("<body><h1>IEEE Quarterly Projects ESP8266 Web
```

```
            // Display current state, and ON/OFF buttons for GPIO 5
            client.println("<p>Motor - Degrees " + servoPinState + "</p>")
            // If the servoPinState is off, it displays the ON button
            if (servoPinState=="off") {
              client.println("<p><a href="/5/on"><button class="button">ON
            } else {
              client.println("<p><a href="/5/off"><button class="button bu
            }


            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != 'r') {  // if you got anything else but a carriage
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

Go to the IP given by the serial monitor as given and you should be met with the same site as given below!

**Share this:**

Press This Twitter Facebook

Like

Be the first to like this.

Edit

IEEE QP Wiki, Powered by WordPress.com.