Last Minute ENGINEERS

ESP8266 Over The Air (OTA) Programming In Arduino IDE

A fantastic feature of any WiFi-enabled microcontroller like ESP8266 NodeMCU is the ability to update its firmware wirelessly. This is known as Over-The-Air (OTA) programming.

What is OTA programming in ESP8266?

The OTA programming allows updating/uploading a new program to ESP8266 using Wi-Fi instead of requiring the user to connect the ESP8266 to a computer via USB to perform the update.

Last Minute ENGINEERS

maintenance.

One important feature of OTA is that one central location can send an update to **multiple ESPs** sharing same network.

The only disadvantage is that you have to add an extra code for OTA with every sketch you upload, so that you're able to use OTA in the next update.

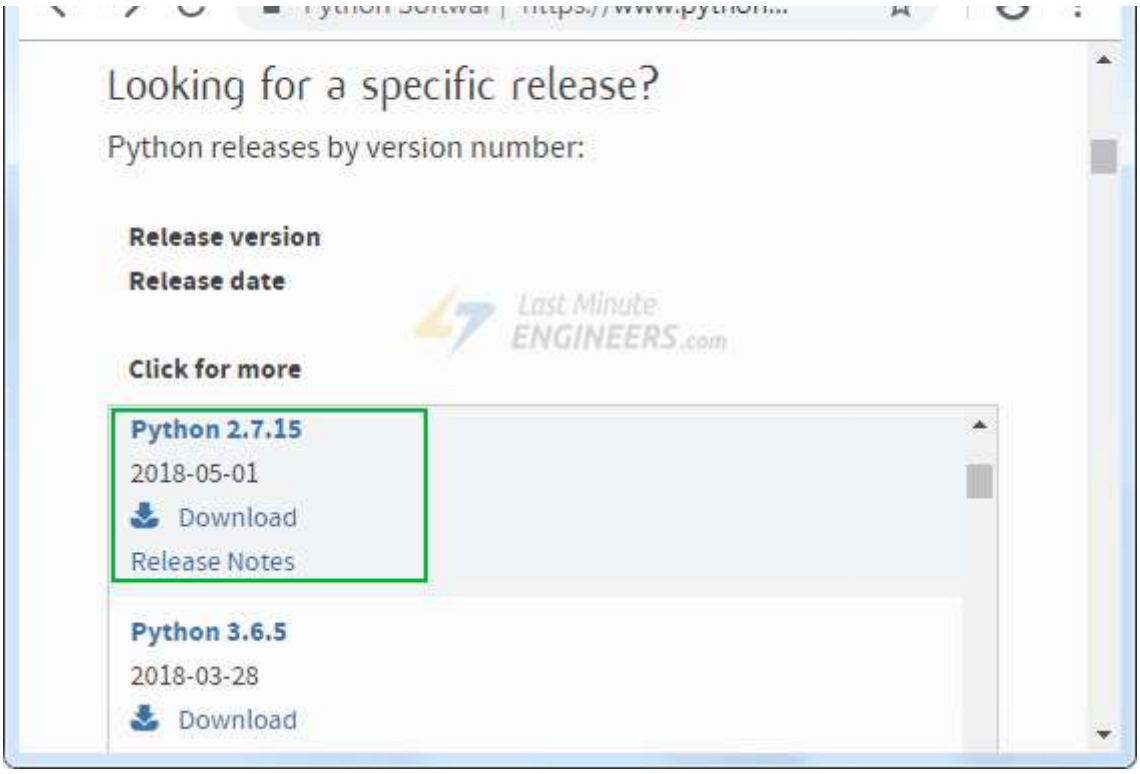
3 Simple Steps To Use OTA with ESP8266

1. **Install Python 2.7.x series** The first step is to install Python 2.7.x series in your computer.
2. **Upload Basic OTA Firmware Serially** Upload the sketch containing OTA firmware serially. It's a mandatory step, so that you're able to do the next updates/uploads over-the-air.
3. **Upload New Sketch Over-The-Air** Now, you can upload new sketches to the ESP8266 from Arduino IDE over-the-air.

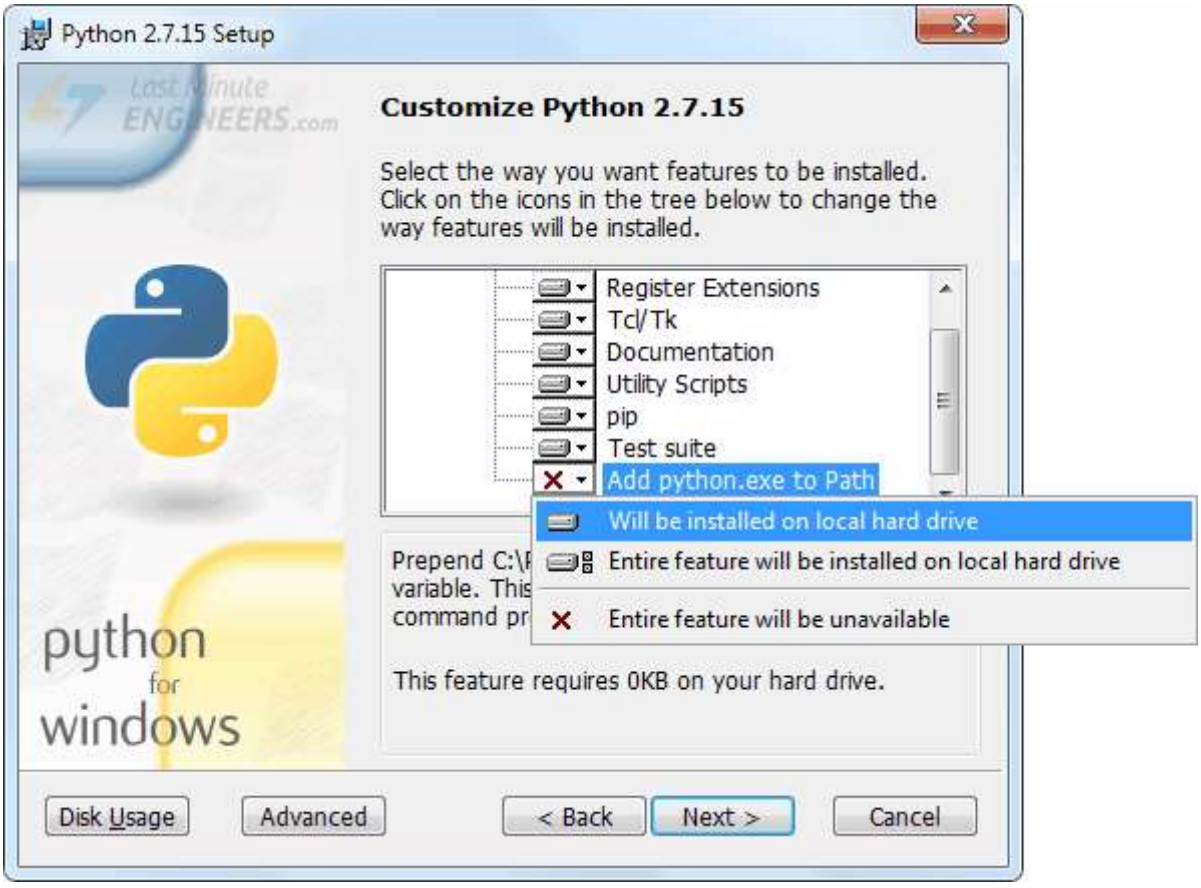
Step 1: Install Python 2.7.x series

In order to use OTA functionality, you need to install the Python 2.7.x version, if not already installed on your machine.

Go to Python's official website and download 2.7.x (specific release) for Windows (MSI installer)



Open the installer and follow the installation wizard.

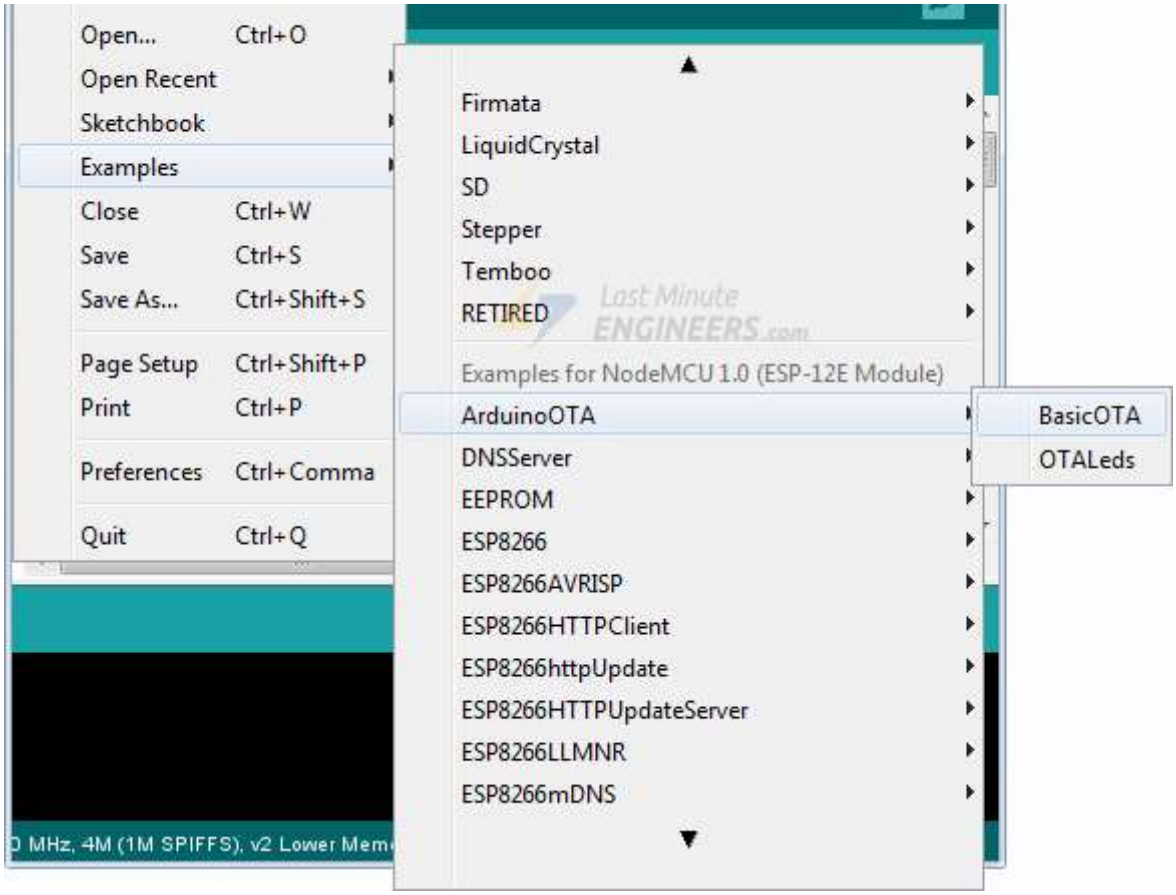


Step 2: Upload OTA Routine Serially

The factory image in ESP8266 doesn't have an OTA Upgrade capability. So, you need to load the OTA firmware on the ESP8266 through serial interface first.

It's a mandatory step to initially update the firmware, so that you're able to do the next updates/uploads over-the-air.

The ESP8266 add-on for the Arduino IDE comes with a OTA library & BasicOTA example. You can access it through **File** > **Examples** > **ArduinoOTA** > **BasicOTA**.



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

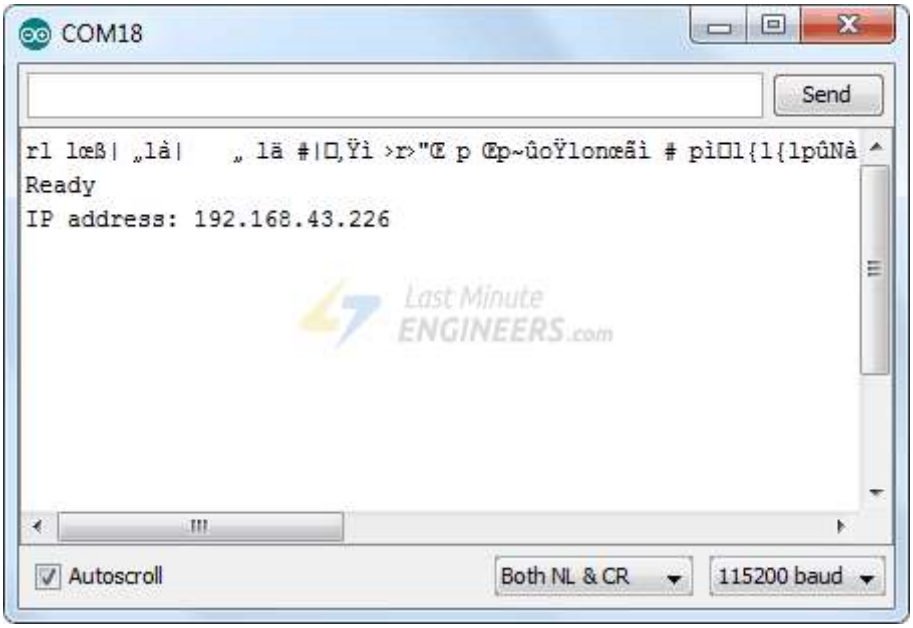
```
const char* ssid = ".........";
const char* password = ".........";
```

Once you are done, go ahead and upload the sketch.

```
    // NOTE: if updating SPIFFS this would be the place to unmount
SPIFFS using SPIFFS.end()
    Serial.println("Start updating " + type);
  });
```

```
  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());
`
```

Now, open the Serial Monitor at a baud rate of 115200. And press the RST button on ESP8266. If everything is OK, it will output the **dynamic IP address** obtained from your router. Note it down.



Remember! you need to add the code for OTA in every sketch you upload. Otherwise, you'll loose OTA capability and will not be able to do next uploads over-the-air. So, it's recommended to modify the above code to include your new code.

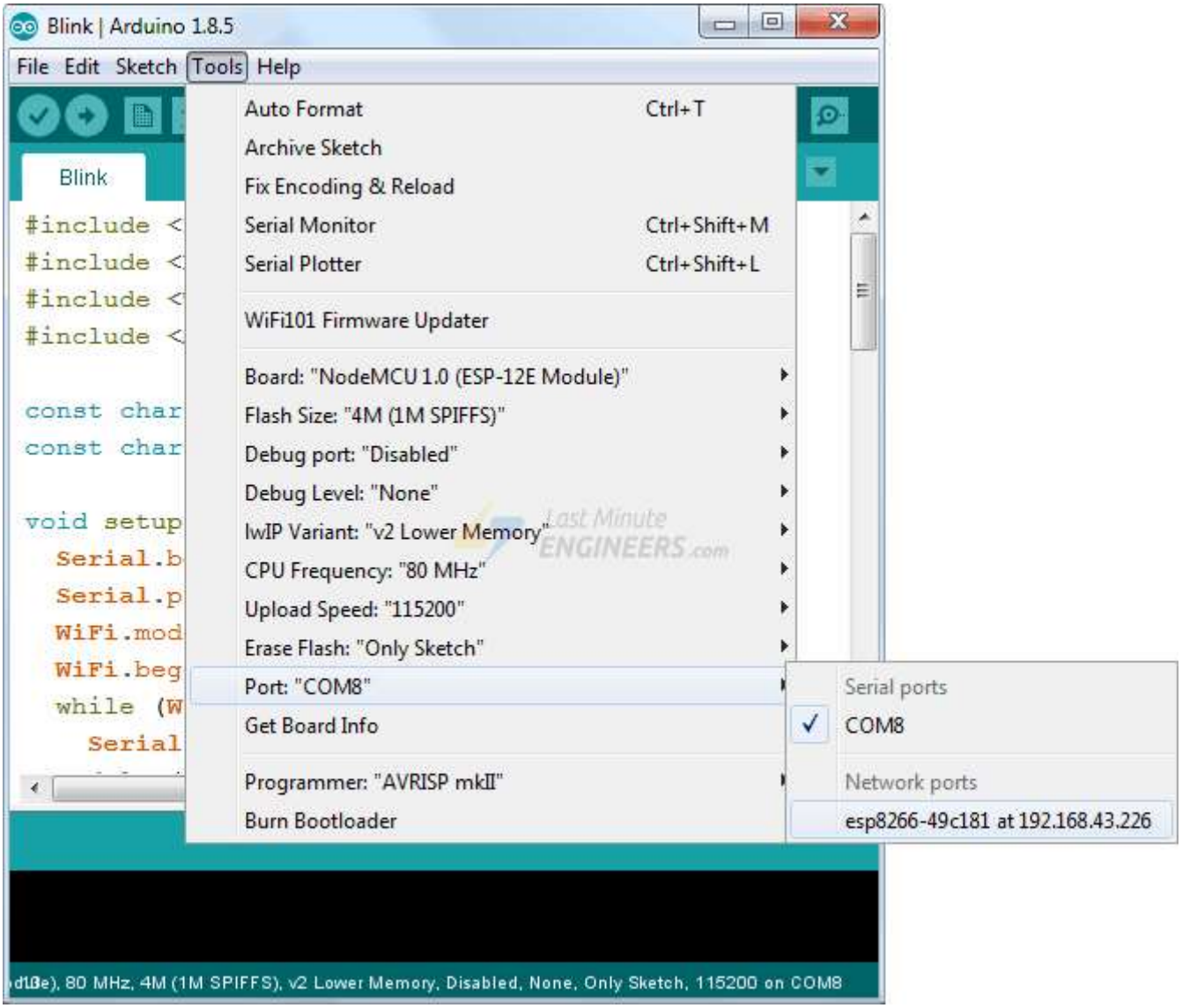
As an example we will include a **simple Blink** sketch in the Basic OTA code. Remember to modify the SSID and password variables with your network credentials.

Changes in the Basic OTA program are highlighted in Blue.

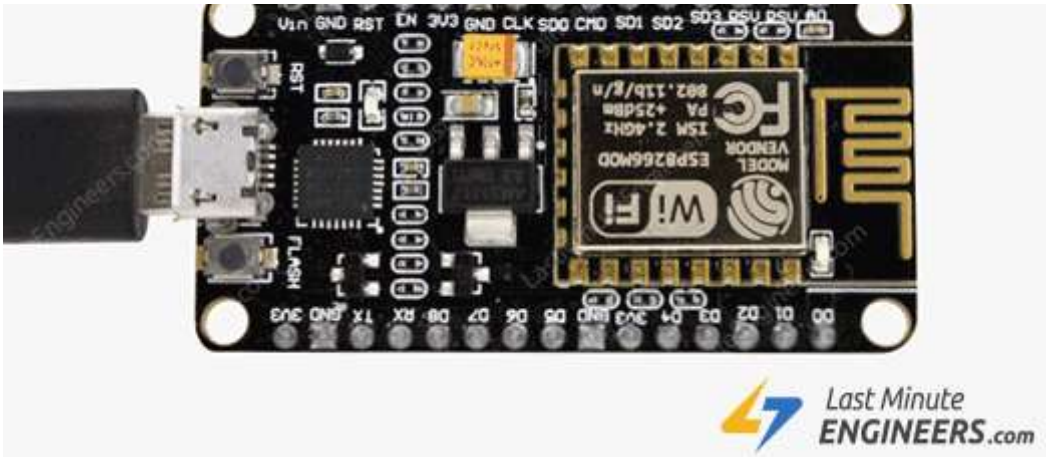
```
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    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();

//loop to blink without delay
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
  // save the last time you blinked the LED
```

In above program, we have not used delay() for blinking an LED, because ESP8266 pauses your program during the delay(). If next OTA request is generated while Arduino is paused waiting for the delay() to pass, your program will miss that request.

Once you copy above sketch to your Arduino IDE, go to **Tools** > **Port** option and you should see something like this: **esp8266-xxxxxx at your_esp_ip_address** If you can't find it, you may need to restart your IDE.



Select the port and click **Upload** button. Within a few seconds, the new sketch will be uploaded. And you should see the on-board LED blinking.



Share