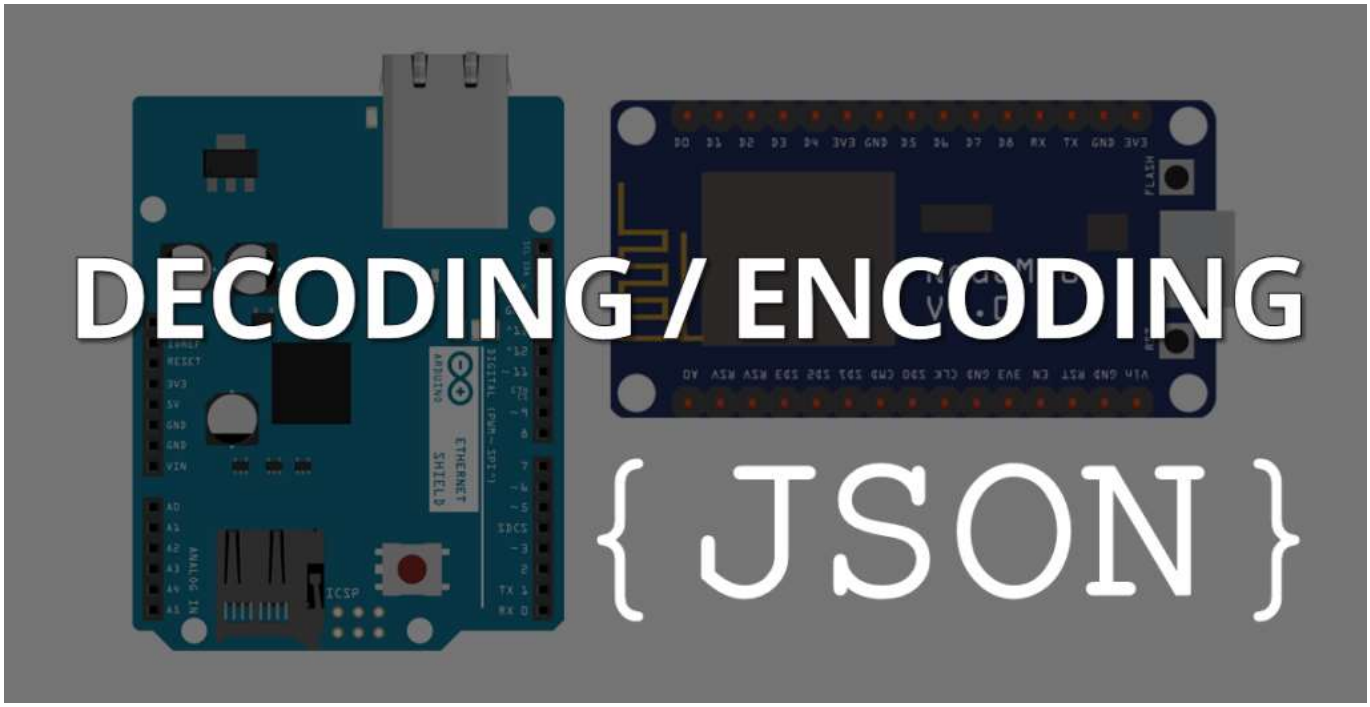
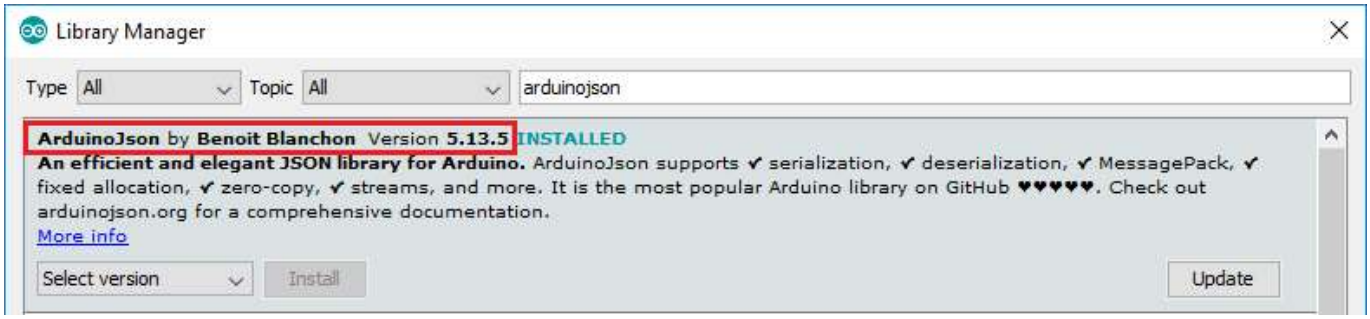
Decoding and Encoding JSON with Arduino or ESP8266

In this blog post you're going to learn how to decode (parse a JSON string) and encode (generate a JSON string) with the ArduinoJson library using the Arduino with the Ethernet shield. This guide also works with the ESP8266 and ESP32 Wi-Fi modules with small changes.



Important: this tutorial is only compatible with the ArduinoJSON library 5.13.5.



What is JSON?

JSON stands for **J**ava**S**cript **O**bject **N**otation. JSON is a lightweight text-based open standard design for exchanging data.

JSON is primarily used for serializing and transmitting structured data over network connection – transmit data between a server and a client. It is often used in services like APIs (Application Programming Interfaces) and web services that provide public data.

JSON syntax basics

In JSON, data is structured in a specific way. JSON uses symbols like **{ } , : ” ” [ ]** and it has the following syntax:

- Data is represented in key/value pairs
- The colon (:) assigns a value to key
- key/value pairs are separated with commas (,)
- Curly brackets hold objects ({ })
- Square brackets hold arrays ([ ])

For example, to represent data in JSON, the key/value pairs come as follows:

```
{"key1":"value1", "key2":"value2", "key3":"value3"}
```

JSON examples

In a real world example, you may want structure data about a user:

```
{"name":"Rui", "country": "Portugal", "age":24}
```

Or in a IoT project, you may want to structure data from your sensors:

```
{"temperature":27.23, "humidity":62.05, "pressure":1013.25}
```

In JSON, the values can be another JSON object (sports) or an array (pets) . For example:

```
{
  "name": "Rui",
  "sports": {
    "outdoor": "hiking",
    "indoor": "swimming"
  },
  "pets": [
    "Max",
    "Dique"
  ]
}
```

Here we are structuring data about a user and we have several keys: “**name”**, “**sports”** and “**pets”.**

The name has the value Rui assigned. Rui may practice different sports relating to where they are practiced. So, we create another JSON object to save Rui’s favorite sports. This JSON object is the value of the “**sports**” key.

The “**pets”** key has an array that contains Rui’s pets’ names and it has the values “**Max**” and “**Dique**” inside.

Most APIs return data in JSON and most values are JSON objects themselves. The following example shows the data provided by a weather API.

```
{ ⊟
  "coord":{ ⊟
     "lon":-8.61,
     "lat":41.15
  },
  "weather":[ ⊟
    { ⊟
      "id":803,
      "main":"Clouds",
      "description":"broken clouds",
```

```
      "icon":"04d"
    }
  ],
  "base":"stations",
  "main":{ ⊟
    "temp":288.15,
    "pressure":1020,
    "humidity":93,
    "temp_min":288.15,
    "temp_max":288.15
  },
  (...)
}
```


This API provides a lot of information. For example, the first lines store the coordinates with the longitude and latitude.

Arduino with Ethernet shield

The examples in this post use an Arduino with an Ethernet shield. Just mount the shield onto your Arduino board and connect it to your network with an RJ45 cable to establish an Internet connection (as shown in the figure below).

Note: the examples provided in this tutorial also work with the ESP8266 and ESP32 with small changes.

Preparing the Arduino IDE

The easiest way to decode and encode JSON strings with the Arduino IDE is using the ArduinoJson library 5.13.5 which was designed to be the most intuitive JSON library, with the smallest footprint and most efficiently memory management for Arduino.

It has been written with Arduino in mind, but it isn't linked to Arduino libraries so you can use this library in any other C++ project. There's also a documentation website for the library with examples and with the API reference.

Features

- JSON decoding (comments are supported)
- JSON encoding (with optional indentation)

- Elegant API, very easy to use
- Fixed memory allocation (zero malloc)
- No data duplication (zero copy)
- Portable (written in C++98)
- Self-contained (no external dependency)
- Small footprint
- Header-only library
- MIT License

Compatible with

- Arduino boards: Uno, Due, Mini, Micro, Yun…
- ESP8266, ESP32 and WeMos boards
- Teensy, RedBearLab boards, Intel Edison and Galileo
- PlatformIO, Particle and Energia

Installing the ArduinoJson library

For this project you need to install the ArduinoJson library in your Arduino IDE:

1. Click here to download the ArduinoJson version 5.13.5. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get ArduinoJson-master folder
3. Rename your folder from ~~ArduinoJson-master~~ to **ArduinoJson**
4. Move the ArduinoJson folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Decoding JSON – Parse JSON string

Let’s start by decoding/parsing the next JSON string:

```
{"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
```

Import the ArduinoJson library:

```
#include <ArduinoJson.h>
```

Arduino JSON uses a preallocated memory pool to store the JsonObject tree, this is done by the StaticJsonBuffer. You can use ArduinoJson Assistant to compute the exact buffer size, but for this example 200 is enough.

```
StaticJsonBuffer<200> jsonBuffer;
```

Create a char array called **json[]** to store a sample JSON string:

```
char json[] = "{\"sensor\":\"gps\",\"time\":1351824120,\"data\":[48
```

Use the function **parseObject()** to decode/parse the JSON string to a JsonObject called **root**.

```
JsonObject& root = jsonBuffer.parseObject(json);
```

To check if the decoding/parsing was successful, you can call **root.success()**:

```
if(!root.success()) {
  Serial.println("parseObject() failed");
  return false;
}
```

The result can be false for three reasons:

- the JSON string has invalid syntax;
- the JSON string doesn't represent an object;

- the StaticJsonBuffer is too small – use ArduinoJson Assistant to compute the buffer size.

Now that the object or array is in memory, you can extract the data easily. The simplest way is to use the JsonObject **root**:

```
const char* sensor = root["sensor"];
long time = root["time"];
double latitude = root["data"][0];
double longitude = root["data"][1];
```

You can use the decoded variables sensor, time, latitude or longitude in your code logic.

OpenWeatherMap API

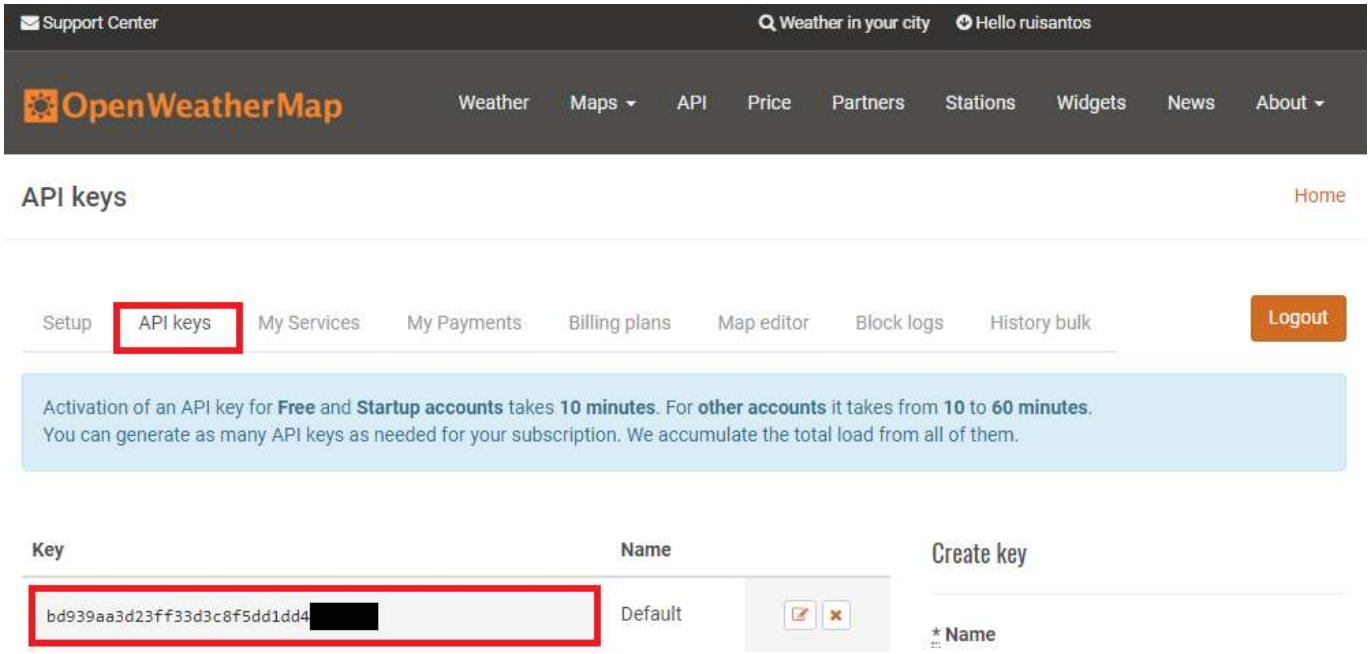
For a real example using an Arduino with an Ethernet shield, we're going to use a free API from OpenWeatherMap to request the day's weather forecast for your chosen location.

Learning to use APIs is a great skill because it allows you access to a wide variety of constantly-changing information, such as the current stock price, the currency exchange rate, the latest news, traffic updates, and much more.

Using the API

OpenWeatherMap's free plan provides everything you need for thins example. To use the API you need an API key, known as the APIID. To get an APIID:

1. Open a browser and go to OpenWeatherMap
2. Press the **Sign up** button and create a free account
3. Once your account is created, you'll be presented with a dashboard that contains several tabs (see figure below)
4. Select the **API Keys** tab and copy your unique **Key**



This is a unique key you need to pull information from the site. Copy and paste this key somewhere, you'll need it in a moment.

To pull information on weather in your chosen location, enter the following URL with the sections in curly brackets replaced with your chosen location information and your unique API key:

```
http://api.openweathermap.org/data/2.5/weather?q={your city},{your
```

Replace **{your city}** with the city you want data for, **{your country code}** with the country code for that city, and **{your API key}** with your unique API key we found previously. For example, our API URL for the town of Porto in Portugal, after replacing with the details, would be:

```
http://api.openweathermap.org/data/2.5/weather?q=Porto,PT&APPID=801
```

Note: more information on using the API to get weather information is available here.

Copy your URL into your browser and it should give you a bunch of information that corresponds to your local weather information.

In our case, it returns the weather in Porto, Portugal on the day of writing:

```
{
  "coord": {
    "lon": -8.61,
    "lat": 41.15
  },
  "weather": [
    {
      "id": 701,
      "main": "Mist",
      "description": "mist",
      "icon": "50d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 290.86,
    "pressure": 1014,
    "humidity": 88,
    "temp_min": 290.15,
    "temp_max": 292.15
  },
  (...)
}
```

Making an API Request with Arduino

Now that you have a URL that returns your local weather data. You can automate this task and access that data in your Arduino or ESP8266 projects. Here's the full script that you need to upload to your Arduino with Ethernet shield to return the temperature in Kelvin and humidity:

```
  strcpy(clientData->humidity, root["main"]["humidity"]);
  // It's not mandatory to make a copy, you could just use the po
  // Since, they are pointing inside the "content" buffer, so you
  // sure it's still in memory when you read the string

  return true;
}

// Print the data extracted from the JSON
void printclientData(const struct clientData* clientData) {
  Serial.print("Temp = ");
  Serial.println(clientData->temp);
  Serial.print("Humidity = ");
  Serial.println(clientData->humidity);
}

// Close the connection with the HTTP server
void disconnect() {
  Serial.println("Disconnect");
  client.stop();
}

// Pause for a 1 minute
void wait() {
  Serial.println("Wait 60 seconds");
  delay(60000);
}
```

View raw code

Note: make sure your replace the resource variable with your unique OpenWeatherMap URL resource:

```
const char* resource = "REPLACE_WITH_YOUR_URL_RESOURCE";
```

Modifying the code for your project

In this example, the Arduino performs an HTTP GET request to a desired service (in this case the OpenWeatherMap API), but you could change it to request any other web service. We won't explain the Arduino code line by line.

For this project it's important that you understand what you need to change in the Arduino code to decode/parse any JSON response. Follow these next three steps.

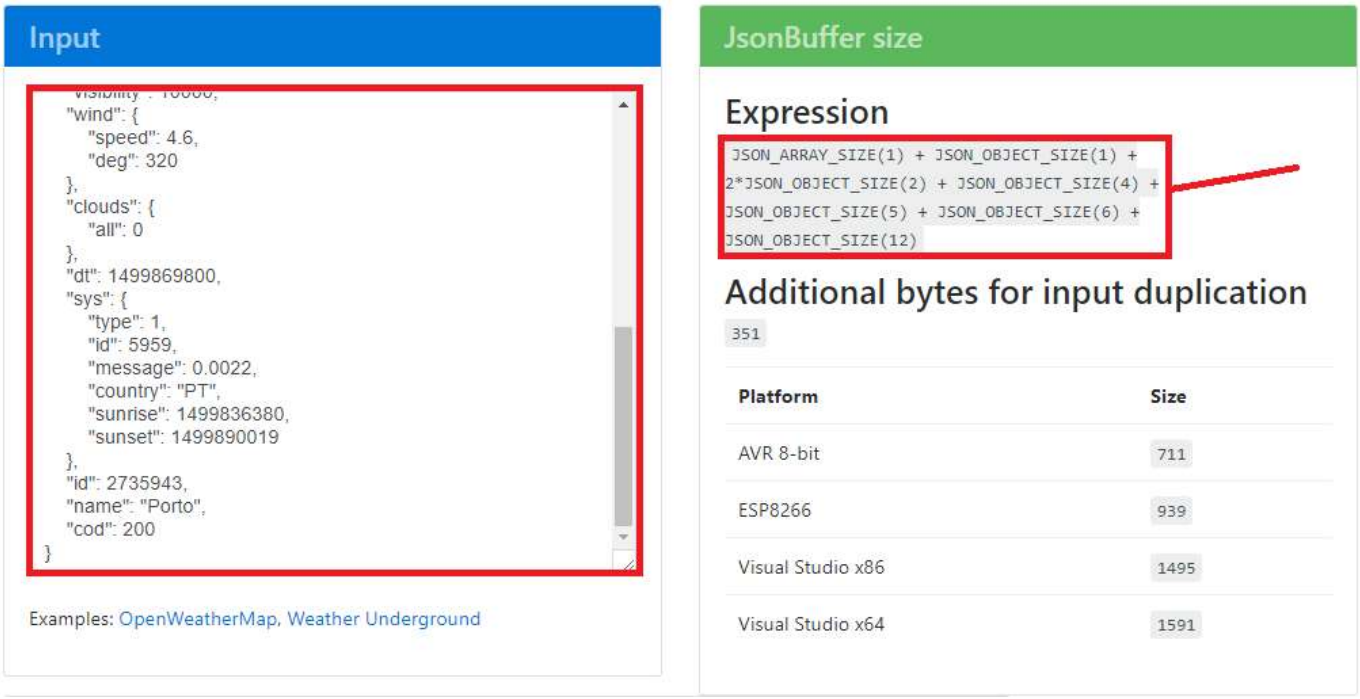
STEP #1 – struct

Create a data structure that can store the information that you'll want to extract from the API. In this case, we want to store the temperature and humidity in char arrays:

```
struct clientData {
  char temp[8];
  char humidity[8];
};
```

STEP #2 – JsonBuffer size

Go to the ArduinoJson Assistant and copy the full OpenWeatherMap API response to the Input field.



Copy the **Expression** generated (see the preceding figure), in my case:

```
JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(1) + 2*JSON_OBJECT_SIZE(2) +
```

You'll need to edit the **readReponseContent()** function with your the generated **JsonBuffer** size from ArduinoJson Assistant to allocate the appropriate memory for decoding the JSON response from an API:

```
bool readReponseContent(struct clientData* clientData) {
  const size_t bufferSize = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(1
    2*JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(4) + JSON_OBJECT_SIZE(
    JSON_OBJECT_SIZE(6) + JSON_OBJECT_SIZE(12) + 390;

  DynamicJsonBuffer jsonBuffer(bufferSize);
  JsonObject& root = jsonBuffer.parseObject(client);
```

Still inside the **readReponseContent()** function you need to copy the variables that you need for your project to your struct data:

```
strcpy(clientData->temp, root["main"]["temp"]);
strcpy(clientData->humidity, root["main"]["humidity"]);
```

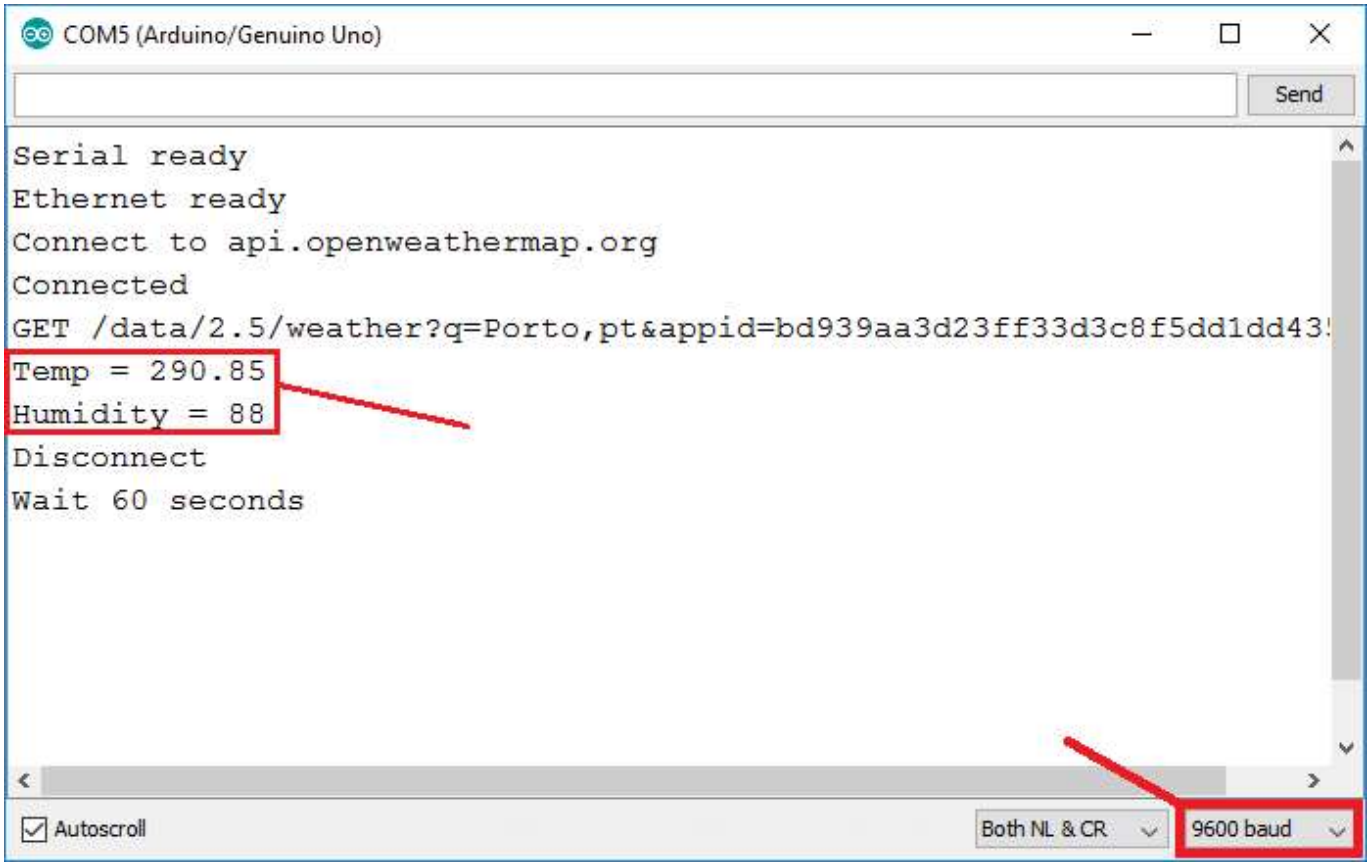
STEP #3 – accessing the decoded data

Then, you can easily access the decoded JSON data in your Arduino code and do something with it. In this example we're simply printing the temperature in Kelvin and humidity in the Arduino IDE serial monitor:

```
void printclientData(const struct clientData* clientData) {
  Serial.print("Temp = ");
  Serial.println(clientData->temp);
  Serial.print("Humidity = ");
  Serial.println(clientData->humidity);
}
```

Demonstration

Open the Arduino IDE serial monitor at a baud rate of 9600 and you'll see the temperature in Kelvin and the humidity in percentage being printed in the Serial monitor every 60 seconds.



You can access the rest of the information in the OpenWeatherMap API response, but for demonstration purposes we only decoded the temperature and humidity.

Encoding JSON – Generate JSON string

Let's learn how to encode/generate the next JSON string:

```
{"sensor":"gps","time":1351824120,"data":[48.756080,2.302038]}
```

You can read the docs about enconding here.

Import the ArduinoJson library:

```
#include <ArduinoJson.h>
```

Arduino JSON uses a preallocated memory pool to store the object tree, this is done by the StaticJsonBuffer. You can use ArduinoJson Assistant to compute the exact

buffer size, but for this example 200 is enough.

```
StaticJsonBuffer<200> jsonBuffer;
```

Create a JsonObject called root that will hold your data. Then, assign the values gps and 1351824120 to the **sensor** and **time** keys, respectively:

```
JsonObject& root = jsonBuffer.createObject();
root["sensor"] = "gps";
root["time"] = 1351824120;
```

Then, to hold an array inside a **data** key, you do the following:

```
JsonArray& data = root.createNestedArray("data");
data.add(48.756080);
data.add(2.302038);
```

It is very likely that you'll need to print the generated JSON in your Serial monitor for debugging purposes, to do that:

```
root.printTo(Serial);
```

After having your information encoded in a JSON string you could post it to another device or web service as shown in the next example.

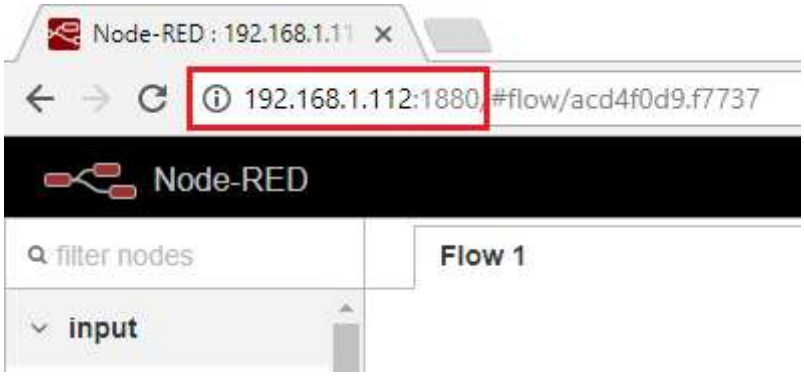
Encoding example with Arduino and Node-RED

For this example you need to Node-RED or a similar software that can receive HTTP POST requests. You can install Node-RED on your computer, but I recommend running Node-RED on a Raspberry Pi.

Creating the flow

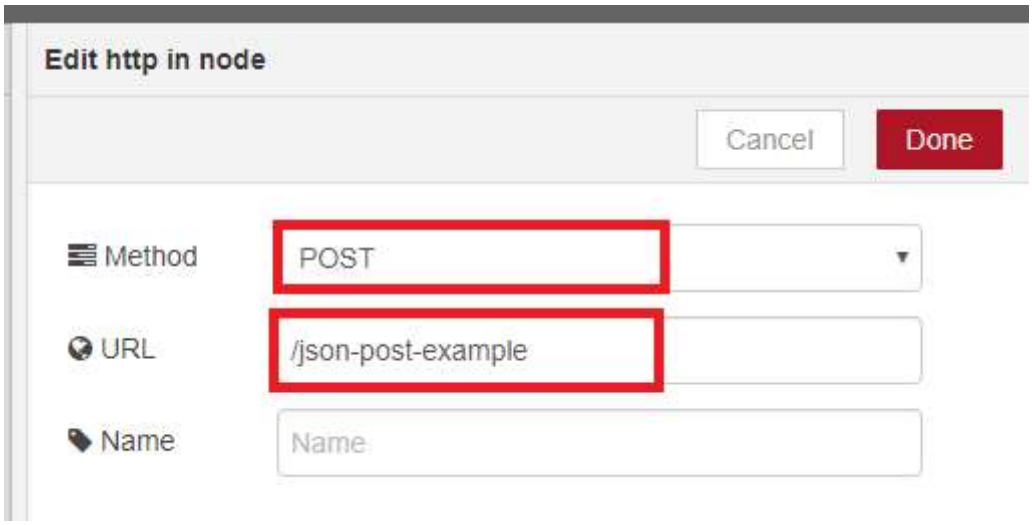
In this flow, you're going to receive an HTTP POST request and print the receive data in the **Debug** window. Follow these next 6 steps to create your flow:

1) Open the Node-RED software in your browser

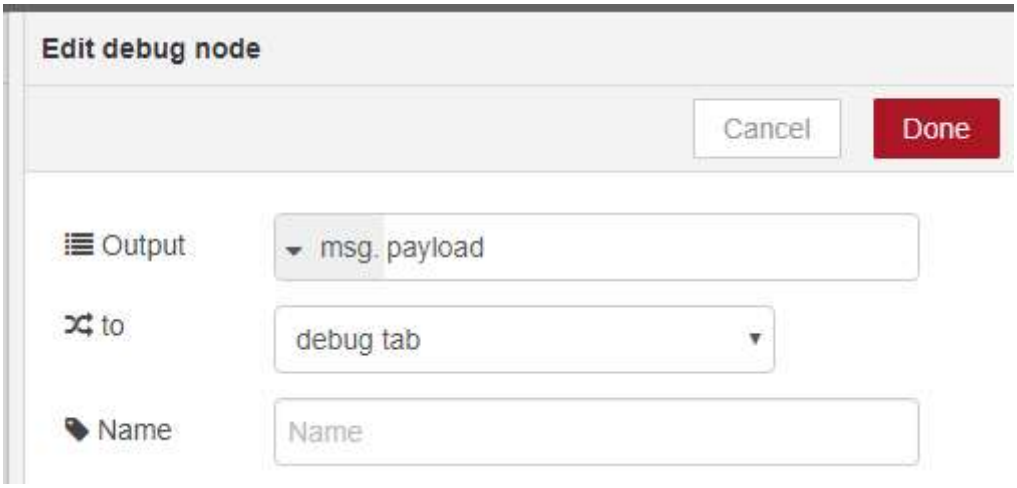


2) Drag an **HTTP input** node and a **debug** node

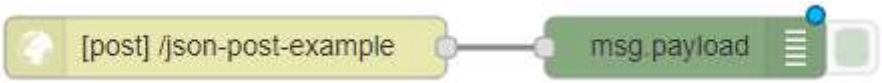
3) Edit the **HTTP input** by adding the **POST** method and the **/json-post-example** URL



4) You can leave the default settings for the **debug** node



5) Connect your nodes



6) To save your application, you need to click the deploy button on the top right corner

Your application is saved and ready.

Send JSON data with Arduino

After having Node-RED prepared to receive POST requests at the /json-post-example URL, you can use the next code example on an Arduino with an Ethernet shield to send data to it.

```
/*
 * Rui Santos
 * Complete Project Details https://randomnerdtutorials.com
 * Based on the Arduino Ethernet Web Client Example
 * and on the sketch "Sample Arduino Json Web Client" of the Ardu
 */

#include <ArduinoJson.h>
```

```
#include <Ethernet.h>
#include <SPI.h>

EthernetClient client;

// Replace with your Raspberry Pi IP address
const char* server = "REPLACE_WITH_YOUR_RASPBERRY_PI_IP_ADDRESS";

// Replace with your server port number frequently port 80 - with
int portNumber = 1880;

// Replace with your unique URL resource
const char* resource = "/json-post-example";

const unsigned long HTTP_TIMEOUT = 10000;  // max respone time fr
const size_t MAX_CONTENT_SIZE = 512;       // max size of the HTT
```

View raw code

Note: make sure your replace the server variable with your Raspberry Pi IP address:

```
const char* server = "REPLACE_WITH_YOUR_RASPBERRY_PI_IP_ADDRESS";
```

Modifying the code for your project

In this example, the Arduino performs an HTTP POST request to Node-RED, but you could change it to make request another web service or server. We won't explain the Arduino code line by line. For this project it's important that you understand what you need to change in the Arduino code to encode/generate a JSON request.

sendRequest() function

For this project you can modify the ***sendRequest()*** function with your own JSON data structure:

```
bool sendRequest(const char* host, const char* resource) {
```

Start by reserving memory space for your JSON data. You can use ArduinoJson Assistant to compute the exact buffer size, but for this example 200 is enough.

```
StaticJsonBuffer<200> jsonBuffer;
```

Create a JsonObject called root that will hold your data and assign the values to your keys (in this example we have the **sensor** key):

```
JsonObject& root = jsonBuffer.createObject();
root["sensor"] = "dht11";
```

To hold data inside an array, you do the following:

```
JsonObject& data = root.createNestedObject("data");
data.set("temperature", "30.1");
data.set("humidity", "70.1");
```

Print the generated JSON string in the Arduino IDE serial monitor for debugging purposes:

```
root.printTo(Serial);
```

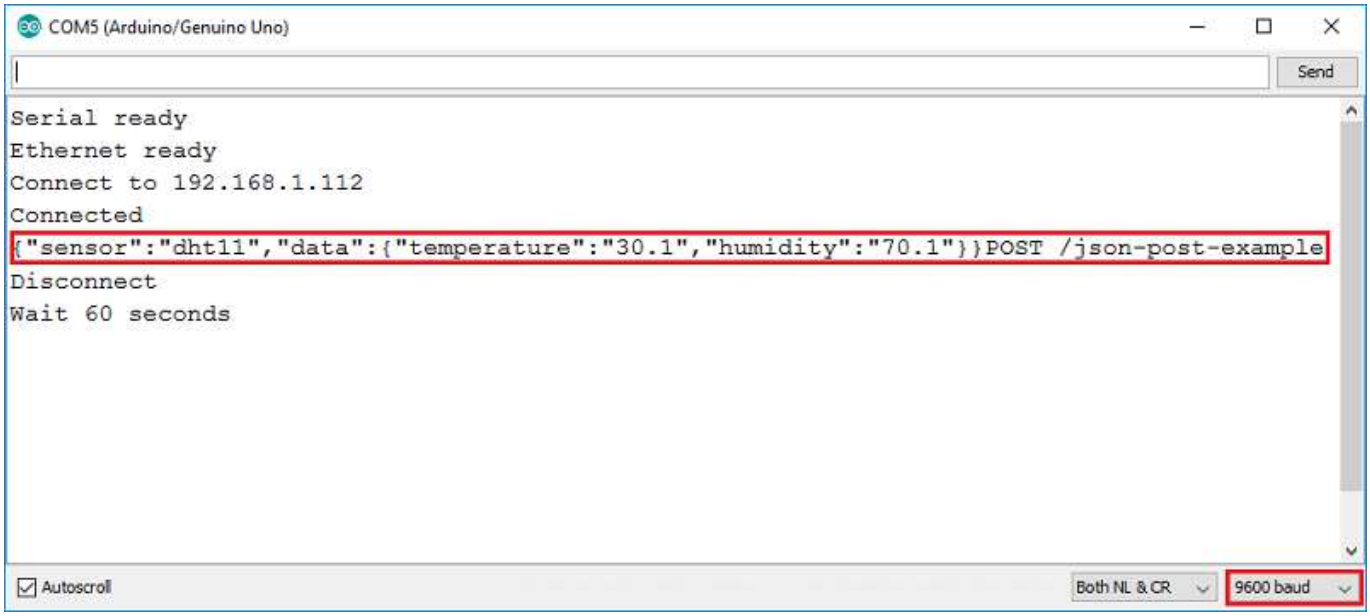
The rest of the sendRequest() function is the POST request.

```
  client.print("POST ");
  client.print(resource);
  client.println(" HTTP/1.1");
  client.print("Host: ");
  client.println(host);
  client.println("Connection: close\r\nContent-Type: application/json
  client.print("Content-Length: ");
  client.print(root.measureLength());
  client.print("\r\n");
  client.println();
  root.printTo(client);
```

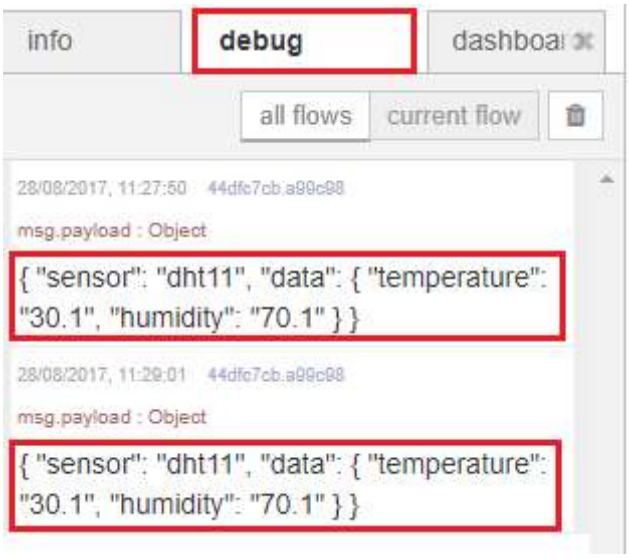
Note that you can use the **root.measureLength()** to determine the length of your generated JSON. The root.printTo(client) send the JSON data to the Ethernet client.

Demonstration

Open the Arduino IDE serial monitor at a baud rate of 9600 and you'll see the JSON object printed in the serial monitor every 60 seconds.



In the Node-RED **debug** tab you'll see the same JSON object being receive every 60 seconds:



Finally, you create functions in Node-RED that do something useful with the received data, but for demonstration purposes we're just printing the sample data.

Wrapping up

In this tutorial we've shown you several examples on how to decode and encode JSON data. You can follow these basic steps to build more advanced projects that require exchanging data between devices.

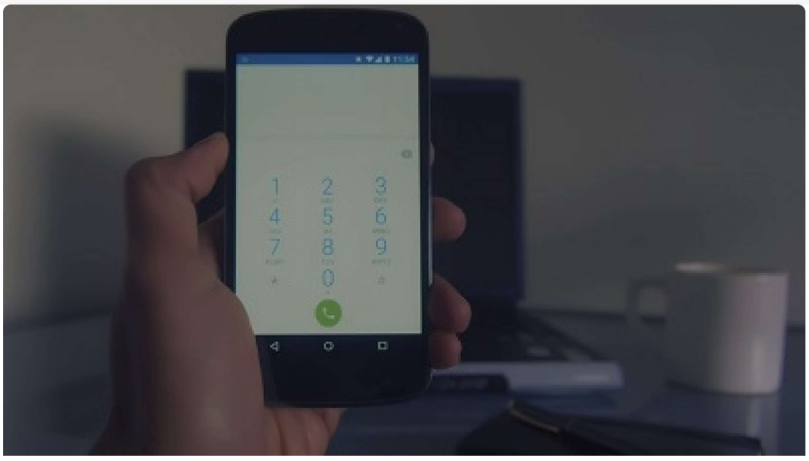
We hope you've found this tutorial useful.

If you liked this project and Home Automation make sure you check our course: Build a Home Automation System for $100.

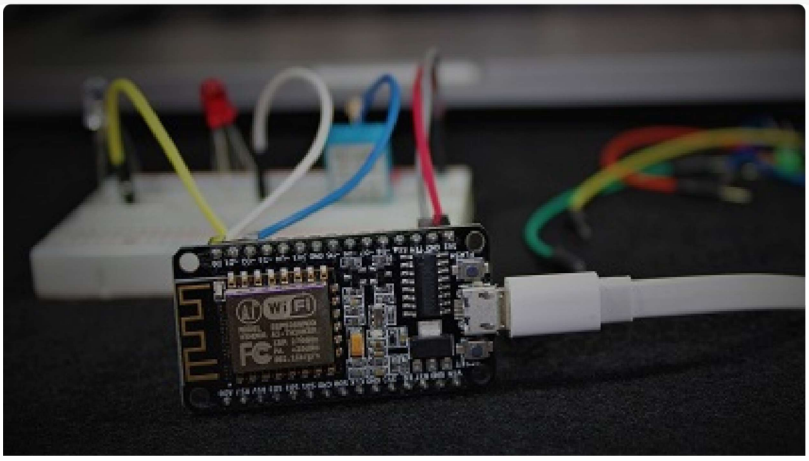
[eBook] Build Web Servers with ESP32 and ESP8266 (2nd Edition)

Build Web Server projects with the ESP32 and ESP8266 boards to control outputs and monitor sensors remotely. Learn HTML, CSS, JavaScript and client-server communication protocols **DOWNLOAD »**

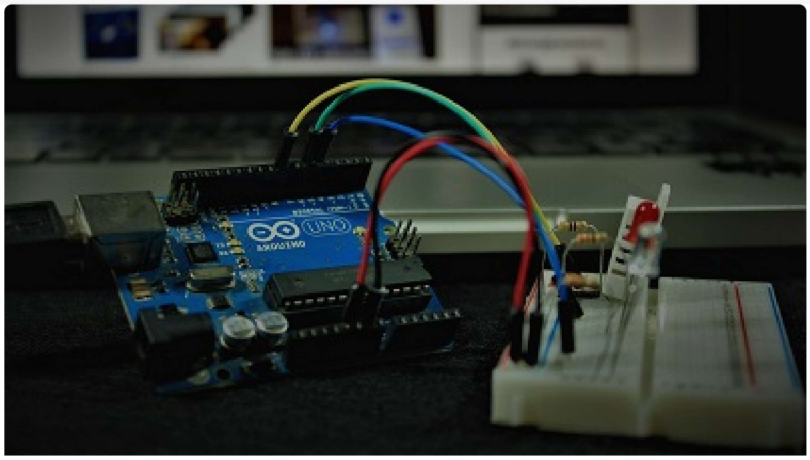
Recommended Resources



Build a Home Automation System from Scratch » With Raspberry Pi, ESP8266, Arduino, and Node-RED.



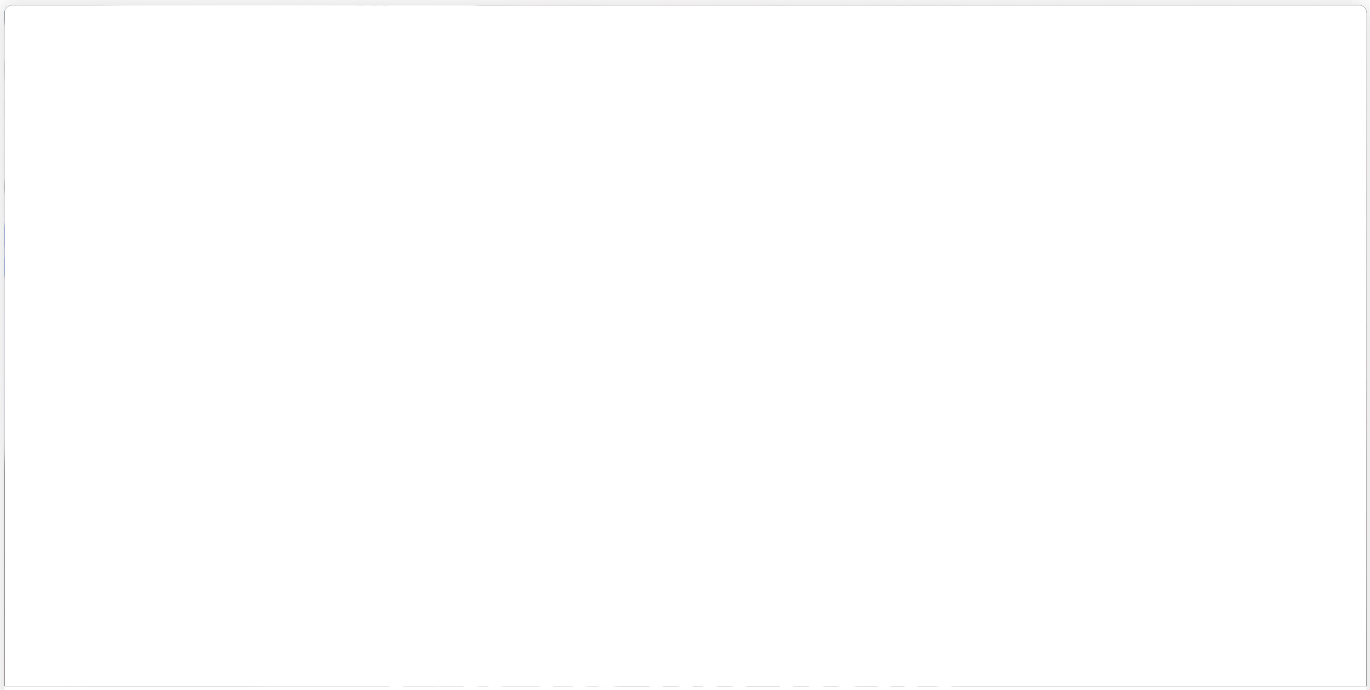
Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Arduino Step-by-Step Projects » Build 25 Arduino projects with our course, even with no prior experience!

What to Read Next…

ESP32 with HC-SR04 Ultrasonic Sensor with Arduino IDE



ESP8266 RGB Color Picker

Enjoyed this project? Stay updated by subscribing our newsletter!

Your Email Address

SUBSCRIBE

32 thoughts on “Decoding and Encoding JSON with Arduino or ESP8266”

Anupam Majumdar

August 30, 2017 at 2:54 am

Hi,
Again an awesome post ! I have been following your post for nearly a year and in almost posts I find a new flavour. Thanks for your good work specially with esp8266.
Will keep in touch. You can drop a mail from your personal email id.
Thank you, Anupam Majumdar, India

Reply

Rui Santos

August 31, 2017 at 12:08 pm

You’re welcome!
Thanks for reading

Reply

Rodolfo

March 1, 2022 at 10:26 am

I have a ESP32, and get error of migration from 5 to 6, this not work, help ?

Reply

Peter Rhodin

August 30, 2017 at 5:49 am

Very very good tutorial regarding JSON. Thanks a lot!!

Reply

Rui Santos

August 31, 2017 at 12:07 pm

You're welcome Peter! I finally posted it, I hope it was helpful. I've tried to come up with examples that you could take and use in your own project regardless of the API.
If you want to use the ESP8266, you simply need to use the HTTP GET or POST examples with WiFi client, but the process of encoding and decoding JSON is the same!

Regards,
Rui

Reply

Abhishek Deokar

January 17, 2018 at 5:24 am

Can you share the esp8266 code as well?

Reply

Sara Santos

January 17, 2018 at 9:47 am

Hi.
The json library works the same way for Arduino and ESP8266.
The code provided refers to an Arduino with an Ethernet shield.
To make it work with ESP8266, you need to modify the part in which you establish an Ethernet connection. The rest of the code works the same.
I hope this helps.
Thanks

Reply

mur

September 6, 2017 at 8:17 am

very good tutorial ; worked on first attempt ; I converted it into ESP8266 thank you very much

Reply

Rui Santos

September 9, 2017 at 11:16 am

You're welcome!
Thanks for trying it 🙂

Reply

Benoit Blanchon

October 25, 2017 at 7:11 am

Thanks for this awesome tutorial ?
I'm glad to see that you made a good use of ArduinoJson and shared your experience.

However, I think the easiest way to install the library is to use the Arduino Library Manager. That way, you don't need to download the zip file.
See: https://bblanchon.github.io/ArduinoJson/doc/installation/

Again thanks for sharing this and keep up with these excellent tutorials.

Reply

Sara Santos

October 26, 2017 at 2:25 pm

Hi Benoit.
Yes, the library manager is now the easiest way to install libraries.
However, some of the libraries are not available through that method.
With this method, we can guarantee that everyone gets the latest version.
But yes, ArduinoJson is in Arduino's Library Manager and they can automatically install it from there.
Thank you for your support.
Regards
Sara 🙂

Reply

Mark
May 27, 2020 at 2:55 am

Agree

Reply

ted
July 14, 2018 at 10:21 pm

Merci beaucoup

Reply

Matt Weber

December 15, 2018 at 1:06 pm

This was absolutely the most helpful tutorial I have found on this topic. Thank you so much for posting it! I'm trying to adapt this to work with the Darksky.net API, because the weather data they provide is more useful to me. However, I'm getting the "Json parsing failed" message, and I can't figure out why. I'd like to see the text that it is receiving and trying to parse. Can you tell me how to do that? Are there common mistakes that lead to the "parsing failed" message? I'm a noob, obviously. Trying to finish a project for Xmas.

Reply

Rui Santos

December 17, 2018 at 3:57 pm

To be honest I'm not sure, it can be many different projects that are causing that issue… Did you try my exact example? Did it work for you? Only after that I would start modifying the project…

Reply

Ghareisa

February 21, 2019 at 8:43 pm

THANK YOU!! AMAZING TUTORIAL!

Reply

Roberto

April 1, 2019 at 7:58 pm

Hi, the JSON Encode exemple link is off. Can you upload it again ?

Reply

Sara Santos

April 3, 2019 at 1:42 pm

Hi Roberto.
That link worked for the oldest library version.
The new one is in this link:
github.com/bblanchon/ArduinoJson/blob/6.x/examples/JsonGeneratorExample/JsonGeneratorExample.ino
I have to update the links.
Thanks for noticing.
Regards,
Sara

Reply

kabilesh

April 2, 2019 at 5:33 am

can you do the same for esp8266

Reply

Sara Santos

April 3, 2019 at 1:45 pm

This works with Arduino and ESP8266.
Regards,
Sara

Reply

fabricio de sousa

May 10, 2019 at 1:52 pm

Muito obrigado por compartilhar um pouco de seu conhecimento, me ajudou bastante.
Tudo de bom para você!

Reply

Frank O'Donnell

August 30, 2019 at 9:27 pm

Thank you,
For running a WEMOS using DeepSleep mode on battery , can you delete anything in the Loop, move it to SETUP?

Reply

Sara Santos
September 1, 2019 at 9:52 am

Yes.
You can do that.
Regards,
Sara

Reply

lokito
November 19, 2019 at 12:32 pm

Thank you for the great work, Benoit!

As nowadays most of the web services run over secure connections (HTTPS), could you share how to use ArduinoJson (I am trying to use it with the WiFiClientSecure.h library on a NodeMCU ESP8266 V3)

Thank you!

Reply

lokito

November 19, 2019 at 12:35 pm

…And many thanks to Rui for creating this tutorial – I messed around with the authors! ;(

Reply

Oleg

January 31, 2020 at 9:12 am

Hi Sara! I have an ESP 8266 + BME 280 (server) and a combi board Arduino Mega + ESP 8266, which is connected to a 3.2-inch display. I am trying to start a data transfer scheme on an display. In the combi board, Arduino Mega and ESP 8266 transmit data on Serieal3 (Mega). With the help of your tutorials, I managed to transfer data from the server to the ESP 8266 on the combi board, as well as output to Serial3 data in the following format: HTTP Response code: 200

```
{
  “Temp2”: 31.13,
  “Hum2”: 6.009766,
  “Pres2”: 733.5735
}
```

Parse JSON string is now required to display the variables Temp2, Hum2, Pres2. But how to do that? I can’t get Parse JSON string from Serial3. I can’t find the answer anywhere.

And another question: how to reduce the number of decimal places in json?

In Serial the data has 1-2 decimal places, and in json it is too much.
Thank you! And thanks again for your work!

Reply

Octavian

January 30, 2022 at 6:21 am

Thank you for this straight forward tutorial. I was able to setup a Wemos to send data to thingspeak without issues. But I need to send data to our server from the Institute where I work, and I need to know if I can use the GET request instead POST? It is the same?

Reply

Bob Pearn

March 11, 2022 at 7:07 pm

Hello Rui, Sara,
I love your tutorials!
I have made an ESP32 LED matrix and it currently displays (the day, date and year), (OpenWeather API for up to 5 different cities), (a user message) and (special events). Using ESPAsyncWebServer I have several webpages where I can change many of the ESP settings. I have some of these setting stored in preferences but most are kept in text files. These setting are all loaded/updated is a reboot or web-post.
As I progress, I need a method to be able to edit my event list. Currently it is an array hard coded in the sketch. So to add/delete or edit an event I have to

recompile and OTA update every time. I believe my solution is to use a JSON file. In fact I could put all my settings into one file.
I need some guidance, I have made an eventlist.json, and using a webserver, I can open and display all the events in the array. I haven't tried using the ESP yet. What I am having trouble with is how to put all the data into a form I can edit, then update the JSON file.
Any direction you can point me in will be appreciated.

Reply

Sara Santos
March 13, 2022 at 12:01 am

Hi,
Take a look at the library documentation, it may help:
https://arduinojson.org/v6/doc/
Regards,
Sara

Reply

Lakshaga Jyothi M
May 16, 2022 at 6:03 pm

Hi,
Your tutorial is great. I have my Arduino UNO connected to NodeMCU serially, when i try to serialise json data from Arduino to NodeMCU with hard coded sensor values. And again parse data at NodeMCU. It works well. I tried to capture real values inside the Json in Arduino UNO from the sensors fixed ex. Temperature and Humidity values that are not hard coded. It goes quite

well. But when I try to parse the same values in NodeMCU. I could not get the sensor values instead i get the hard coded values inserted in the json structure of parsing program.

Reply

Jaden

October 17, 2022 at 5:39 am

Great tutorial.
thanks!

Could you recommand other project that use API?
I want to know how to use this skill in project.

Reply

Tunturi

March 2, 2023 at 8:29 pm

Very useful indeed. Would be fantastic to see a arduniojson 6 version also. My rudimentary c++ knowledge did not allow me to do the upgrade myself.

Reply

Leave a Comment

Name *

Email *

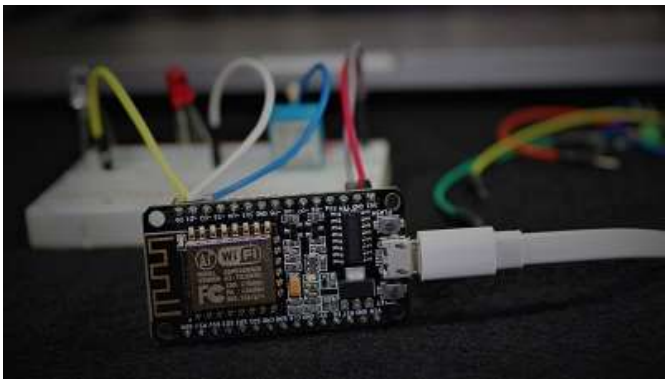
Website

Notify me of follow-up comments by email.

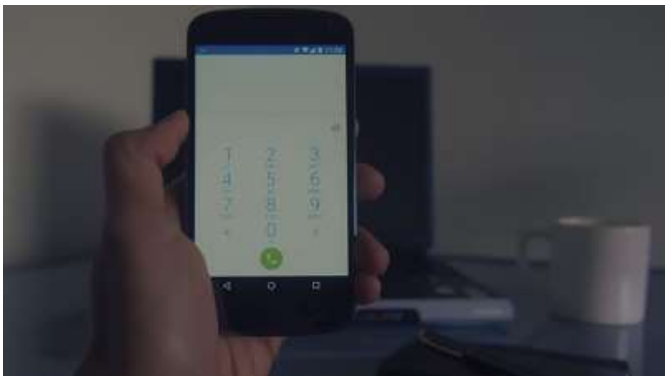
Notify me of new posts by email.

Post Comment

Visit Maker Advisor – Tools and Gear for makers, hobbyists and DIYers »



Home Automation using ESP8266 eBook and video course » Build IoT and home automation projects.



Build Web Servers with ESP32 and ESP8266 » boards to control outputs and monitor sensors remotely.