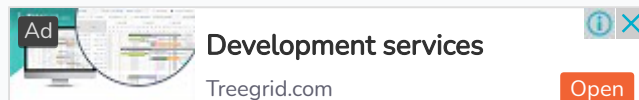



[Home](#) > [Articles](#)

Reading and Writing JSON Files with Node.js

Scott Robinson 

One of the best ways to exchange information between applications written in different languages is to use the [JSON](#) (JavaScript Object Notation) format. Thanks to its uniformity and simplicity, JSON has almost completely replaced XML as the standard data exchange format in software, particularly in web services.

Given the extensive use of JSON in software applications, and especially JavaScript-based applications, it is important to know how to read and write JSON data to a file in Node.js. In this article we'll explain how to perform these functions.

Reading a JSON File

Let's first see how we can [read a file](#) that has already been created. But before we do that we need to actually create the file. Open a new window in your favorite text editor and add the following text to it:

```
{  
  "name": "Sara",  
  "age": 23,  
  "gender": "Female",  
  "department": "History",  
  "car": "Honda"
```

```
}
```

Now save this file as "student.json" to your project directory.

To read the JSON data from the file we can use the Node.js `fs` module. There are two functions available in this module that we can use to read files from the file system:

`readFile` and `readFileSync`.

Although both of these functions perform similar tasks i.e. reading files from disk, the difference lies in the way these functions are actually executed, which we'll explain in more detail in the sections below.

Using `fs.readFileSync`

The `readFileSync` function reads data from a file in a synchronous manner. This function blocks the rest of the code from executing until all the data is read from a file. The function is particularly useful when your application has to load configuration settings before it can perform any other tasks.

To continue with our example, let's use this function to read the "student.json" file that we created earlier, using the `readFileSync` function. Add the following code to a '.js' file:

```
'use strict';

const fs = require('fs');

let rawdata = fs.readFileSync('student.json');
let student = JSON.parse(rawdata);
console.log(student);
```

In the above Node.js code we first load the `fs` module to our application. Next we use the `readFileSync` function and pass it the relative file path to the file that we want to read. If you print the object `rawdata` to the console, you will see raw data (in a `Buffer`) on the console screen:

```
<Buffer 7b 20 0a 20 20 20 20 22 6e 61 6d 65 22 3a 20 22 53 61 72 61 22 2c 0a 20
```

However, we want to read the file in its JSON format, not the raw hex data. This is where the `JSON.parse` function comes into play. This function handles parsing the raw data, converts it to ASCII text, and parses the actual JSON data in to a JavaScript object. Now, if you print the `student` object on the console, you will get the following output:

```
{ name: 'Sara',  
  age: 23,  
  gender: 'Female',  
  department: 'History',  
  car: 'Honda' }
```

As you can see, the JSON from our file was successfully loaded in to the `student` object.

Using `fs.readFile`

Another way you can read a JSON file in Node.js is using the `readFile` function. Unlike `readFileSync` function, the `readFile` function reads file data in an asynchronous manner. When a `readFile` function is called, the file reading process starts and immediately the control shifts to next line executing the remaining lines of code. Once

the file data has been loaded, this function will call the callback function provided to it. This way you aren't blocking code execution while waiting for the operating system to get back to you with data.

In our example, the `readFile` function takes two parameters: The path to the file that is to be read and the callback function that is to be called when the file is read completely. You can optionally also include a parameter with options, but we won't be covering those in this article.

Take a look at the following example to understand how to use the `readFile` function.

```
'use strict';

const fs = require('fs');

fs.readFile('student.json', (err, data) => {
  if (err) throw err;
  let student = JSON.parse(data);
  console.log(student);
});

console.log('This is after the read call');
```

The code above does exactly what our previous code snippet did (with an extra `console.log` call), but it does so asynchronously. Here are a few of the differences, which you may have noticed:

- `(err, data) => {}`: This is our callback function that is executed once the file is completely read
- `err`: Since we can't easily use try/catch with asynchronous code, the function instead gives us an `err` object if something goes wrong. It is `null` if there were no errors

You may have also noticed that we print a string to the console immediately after calling `readFile`. This is to show you the behavior of asynchronous code. When the above script is executed, you will see that this `console.log` executes *before* the `readFile` callback function executes. This is because `readFile` does not block code from executing while it reads data from the file system.

The output of the code will look like this:

```
This is after the read call
```

```
{ name: 'Sara',  
  age: 23,  
  gender: 'Female',  
  department: 'History',  
  car: 'Honda' }
```

As you can see, the last line of code in our file is actually the one that shows up first in the output.

Using `require`

Free eBook: Git Essentials

Check out our hands-on, practical guide to learning Git, with best-practices, industry-accepted standards, and included cheat sheet. Stop Googling Git commands and actually *learn* it!

[Download the eBook →](#)

Another approach is to use the global `require` method to read and parse JSON files. This is the same method you use to load Node modules, but it can also be used to load JSON.

Take a look at the following example.

```
'use strict';

let jsonData = require('./student.json');

console.log(jsonData);
```

It works exactly like the `readFileSync` code we showed above, but it is a globally available method that you can use anywhere, which has its advantages.

However there are a few drawbacks of `require` function:

- `Require` is synchronous function and is called only once, which means the calls receive a cached result. If the file is updated you can't re-read it using this method
- Your file must have `.json` extension, so it can't be as flexible. Without the proper extension `require` doesn't treat the file as JSON file.

Writing JSON to a File

Similar to the `readFile` and `readFileSync` functions, there are two functions for **writing data to files**: `writeFile` and `writeFileSync`. As the names suggest, the `writeFile` method writes data to a file in an asynchronous way while `writeFileSync` function writes data to a file in a synchronous manner.

We'll take a closer look in the following sections.

Using `fs.writeFileSync`

The `writeFileSync` function accepts 2-3 parameters: The path of the file to write data to, the data to write, and an optional parameter.

Note that if the file doesn't already exist, then a new file is created for you. Take a look at the following example:

```
'use strict';

const fs = require('fs');

let student = {
  name: 'Mike',
  age: 23,
  gender: 'Male',
  department: 'English',
  car: 'Honda'
};

let data = JSON.stringify(student);
fs.writeFileSync('student-2.json', data);
```

In the above example we are storing our JSON object `student` to a file named "student-2.json". Notice that here we have to use the `JSON.stringify` function before saving the data. Just like we needed to parse the data into JSON format when we read the JSON file, we need to "stringify" the data before we can store it in a string form in the file.

Execute the above code and open the "student-2.json" file. You should see following content in the file:

```
{"name":"Mike","age":23,"gender":"Male","department":"English","car":"Honda"}
```

Although this is the data that we wanted to write, the data is in the form of one line of string, which is difficult for us to read. If you'd like the serialized JSON to be human readable, then change the `JSON.stringify` function as follows:

```
let data = JSON.stringify(student, null, 2);
```

Here we are telling the method to add newlines and a couple of indentations to the serialized JSON. Now if you open the "student-2.json" file, you should see the text in following format.

```
{
  "name": "Mike",
```

```
"age": 23,  
"gender": "Male",  
"department": "English",  
"car": "Honda"  
}
```

Using fs.writeFile

As I mentioned earlier, the `writeFile` function executes in asynchronous manner, which means our code is not blocked while data is written to the file. And just like the asynchronous methods from before, we need to pass a callback to this function.

Let's write another file, "student-3.json", using the `writeFile` function.

```
'use strict';  
  
const fs = require('fs');  
  
let student = {  
  name: 'Mike',  
  age: 23,  
  gender: 'Male',  
  department: 'English',  
  car: 'Honda'  
};  
  
let data = JSON.stringify(student, null, 2);  
  
fs.writeFile('student-3.json', data, (err) => {  
  if (err) throw err;  
  console.log('Data written to file');  
});
```



```
console.log('This is after the write call');
```

The output of the above script will be:

```
This is after the write call  
Data written to file
```

And again, you can see that the last line of our code actually shows up first in the console since our callback hasn't been called yet. This ends up saving quite a bit of execution time if you have large amounts of data to write to your file, or if you have quite a few files to write to.

Learn More

Want to learn more about the fundamentals of Node.js? Personally, I'd recommend taking an online course like [Learn Node.js by Wes Bos](#). Not only will you learn the most up-to-date ES2017 syntax, but you'll get to build a full stack restaurant app. In my experience, building real-world apps like this is the fastest way to learn.

Conclusion

Throughout this article we showed how you can read and write JSON data from and to files, which is a very common and important task to know how to do as a web programmer.

There are a couple of methods in the `fs` module both for reading from and writing to JSON files. The `readFile` and `readFileSync` functions will read JSON data from the file in an asynchronous and synchronous manner, respectively. You can also use the global `require` method to handle reading/parsing JSON data from a file in a single line of code. However, `require` is synchronous and can only read JSON data from files with `'.json'` extension.

Similarly, the `writeFile` and `writeFileSync` functions from the `fs` module write JSON data to the file in an asynchronous and synchronous manner respectively.

[#how to](#)[#json](#)[#javascript](#)[#node](#)

Last Updated: March 2nd, 2019

Was this article helpful? ☆☆☆☆☆



You might also like...

- Using NVM to Install Node
- Read Files with Node.js
- How to Create C/C++ Addons in Node
- Testing Node.js Code with Mocha and Chai
- 6 Easy Ways to Speed Up Express

Improve your dev skills!

Get tutorials, guides, and dev jobs in your inbox.

Sign Up

No spam ever. Unsubscribe at any time. Read our [Privacy Policy](#).

Scott Robinson *Author*



Want a remote job?

 More Jobs

Jobs by [HireRemote.io](https://hireremote.io)

Prepping for an interview?

Improve your skills by solving one coding problem every day

Get the solutions the next morning via email

Practice on **actual problems** asked by top companies, like:

 [Daily Coding Problem](#)

Getting Started with AWS in Node.js

Build the foundation you'll need to provision, deploy, and run Node.js applications in the AWS cloud. Learn Lambda, EC2, S3, SQS, and more!

[Learn more →](#)



© 2013-2021 Stack Abuse. All rights reserved.

[Disclosure](#) | [Privacy](#) | [Terms](#)