

[< Previous](#)

[Next >](#)

Export Module in Node.js

Here, you will learn how to expose different types as a module using `module.exports`.

The `module.exports` is a special object which is included in every JavaScript file in the Node.js application by default. The `module` is a variable that represents the current module, and `exports` is an object that will be exposed as a module. So, whatever you assign to `module.exports` will be exposed as a module.

Let's see how to expose different types as a module using `module.exports`.

Export Literals

As mentioned above, `exports` is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in `Message.js`.

Message.js	Copy
<pre>module.exports = 'Hello world';</pre>	

Now, import this message module and use it as shown below.

app.js	Copy
<pre>var msg = require('./Message.js'); console.log(msg);</pre>	

Run the above example and see the result, as shown below.

```
C:\> node app.js  
Hello World
```

Note:

You must specify `./` as a path of root folder to import a local module. However, you do not need to specify the path to import Node.js core modules or NPM modules in the `require()` function.

NOW PLAYING

SETUP AND INTRODUCTION TO NGRX

TWO TYPES OF PROGRAMMING CAREER

Too old to learn...
Are you too old to learn programming or coding at the age of... [Watch Video](#)

1/4 NgRx...
NgRx is quite tricky to setup for production projects,... [Watch Video](#)

Wordpress vs...
Wordpress or custom website, two ways of choosing... [Watch Video](#)

Export Object

The `exports` is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in `Message.js` file.

Message.js	Copy
<pre>exports.SimpleMessage = 'Hello world'; //or module.exports.SimpleMessage = 'Hello world';</pre>	

In the above example, we have attached a property `SimpleMessage` to the `exports` object. Now, import and use this module, as shown below.

app.js	Copy
<pre>var msg = require('./Messages.js'); console.log(msg.SimpleMessage);</pre>	

In the above example, the `require()` function will return an object `{ SimpleMessage : 'Hello World' }` and assign it to the `msg` variable. So, now you can use `msg.SimpleMessage`.

Run the above example by writing `node app.js` in the command prompt and see the output as shown below.

```
C:\> node app.js
Hello World
```

In the same way as above, you can expose an object with function. The following example exposes an object with the `log` function as a module.

Log.js	Copy
<pre>module.exports.log = function (msg) { console.log(msg); };</pre>	

The above module will expose an object- { log : function(msg){ console.log(msg); } } . Use the above module as shown below.

app.js	Copy
<pre>var msg = require('./Log.js'); msg.log('Hello World');</pre>	

Run and see the output in command prompt as shown below.

```
C:\> node app.js  
Hello World
```

You can also attach an object to module.exports , as shown below.

data.js	Copy
<pre>module.exports = { firstName: 'James', lastName: 'Bond' }</pre>	

app.js	Copy
<pre>var person = require('./data.js'); console.log(person.firstName + ' ' + person.lastName);</pre>	

Run the above example and see the result, as shown below.

```
C:\> node app.js  
James Bond
```

Export Function

You can attach an anonymous function to exports object as shown below.

Log.js	Copy

```
module.exports = function (msg) {  
    console.log(msg);  
};
```

Now, you can use the above module, as shown below.

app.js

 Copy

```
var msg = require('./Log.js');  
  
msg('Hello World');
```

The `msg` variable becomes a function expression in the above example. So, you can invoke the function using parenthesis `()`. Run the above example and see the output as shown below.

```
C:\> node app.js  
Hello World
```

Export Function as a Class

In JavaScript, a function can be treated like a class. The following example exposes a function that can be used like a class.

Person.js

 Copy

```
module.exports = function (firstName, lastName) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.fullName = function () {  
        return this.firstName + ' ' + this.lastName;  
    }  
}
```

The above module can be used, as shown below.

app.js

 Copy

```
var person = require('./Person.js');  
  
var person1 = new person('James', 'Bond');  
  
console.log(person1.fullName());
```

As you can see, we have created a `person` object using the `new` keyword. Run the above example, as shown below.

```
C:\> node app.js
James Bond
```

In this way, you can export and import a local module created in a separate file under root folder.

Node.js also allows you to create modules in sub folders. Let's see how to load module from sub folders.

Load Module from the Separate Folder

Use the full path of a module file where you have exported it using `module.exports` . For example, if the `log` module in the `log.js` is stored under the `utility` folder under the root folder of your application, then import it, as shown below.

app.js	Copy
<pre>var log = require('./utility/log.js');</pre>	

In the above example, `.` is for the root folder, and then specify the exact path of your module file. Node.js also allows us to specify the path to the folder without specifying the file name. For example, you can specify only the `utility` folder without specifying `log.js` , as shown below.

app.js	Copy
<pre>var log = require('./utility');</pre>	

In the above example, Node.js will search for a package definition file called `package.json` inside the `utility` folder. This is because Node assumes that this folder is a package and will try to look for a package definition. The `package.json` file should be in a module directory. The `package.json` under `utility` folder specifies the file name using the `main` key, as shown below.

./utility/package.json	Copy
<pre>{ "name" : "log", "main" : "./log.js" }</pre>	

Now, Node.js will find the `log.js` file using the `main` entry in `package.json` and import it.

Note:

If the `package.json` file does not exist, then it will look for `index.js` file as a module file by default.

Want to check how much you know Node.js?

Start Node.js Test

f ▶ Share

🐦 ▶ Tweet

in ▶ Share

📞 ▶ Whatsapp

< Previous

Next >

- C#
- Object Oriented C#
- ASP.NET Core
- ASP.NET MVC
- LINQ
- Inversion of Control
- Web API

Database Tutorials

- SQL
- SQL Server
- PostgreSQL
- MongoDB

JavaScript Tutorials

- JavaScript
- TypeScript
- jQuery
- Angular 11
- Node.js
- D3.js
- Sass

Programming Tutorials

- Python
- Go lang

HTTPS (SSL)

TutorialsTeacher.com

TutorialsTeacher.com is optimized for learning web technologies step by step. Examples might be simplified to improve reading and basic understanding. While using this site, you agree to have read and accepted our terms of use and privacy policy.

 [Contact Us](#)

E-mail list

Subscribe to TutorialsTeacher email list and get latest updates, tips & tricks on C#, .Net, JavaScript, jQuery, AngularJS, Node.js to your inbox.

Email address

We respect your privacy.

© 2023 TutorialsTeacher.com. All Rights Reserved.