Node.js is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine. It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications. Node.js is open source, completely free, and used by thousands of developers around the world.

Node.js is an open source server framework.

Node.js allows you to run JavaScript on the server.

Prerequisites

understanding of other web technologies such as HTML, CSS, AJAX, etc.

* Node.js is an open source server framework
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server
* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database
* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

Node.js Modules:

## **What is a Module in Node.js?**

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

## **Built-in Modules**

## **Include Modules**

To include a module, use the require() function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server:

http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.end('Hello World!');  
}).listen(8080);

## **Create Your Own Modules**

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

### **Example**

Create a module that returns the current date and time:

exports.myDateTime = function () {  
    return Date();  
};

Use the exports keyword to make properties and methods available outside the module file.

Save the code above in a file called "myfirstmodule.js"

## **Include Your Own Module**

Now you can include and use the module in any of your Node.js files.

### **Example**

Use the module "myfirstmodule" in a Node.js file:

var http = require('http');  
**var dt = require('./myfirstmodule');**  
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write("The date and time are currently: " + **dt.myDateTime()**);  
    res.end();  
}).listen(8080);

[Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_module)

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

Save the code above in a file called "demo\_module.js", and initiate the file:

Initiate demo\_module.js:

C:\Users\Your Name>node demo\_module.js

If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)

# Node.js Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

Here is a list of the built-in modules of Node.js version 6.10.3:

|  |  |
| --- | --- |
| **Module** | **Description** |
| [assert](https://www.w3schools.com/nodejs/ref_assert.asp) | Provides a set of assertion tests |
| [buffer](https://www.w3schools.com/nodejs/ref_buffer.asp) | To handle binary data |
| child\_process | To run a child process |
| [cluster](https://www.w3schools.com/nodejs/ref_cluster.asp) | To split a single Node process into multiple processes |
| [crypto](https://www.w3schools.com/nodejs/ref_crypto.asp) | To handle OpenSSL cryptographic functions |
| [dgram](https://www.w3schools.com/nodejs/ref_dgram.asp) | Provides implementation of UDP datagram sockets |
| [dns](https://www.w3schools.com/nodejs/ref_dns.asp) | To do DNS lookups and name resolution functions |
| domain | Deprecated. To handle unhandled errors |
| [events](https://www.w3schools.com/nodejs/ref_events.asp) | To handle events |
| [fs](https://www.w3schools.com/nodejs/ref_fs.asp) | To handle the file system |
| [http](https://www.w3schools.com/nodejs/ref_http.asp) | To make Node.js act as an HTTP server |
| [https](https://www.w3schools.com/nodejs/ref_https.asp) | To make Node.js act as an HTTPS server. |
| [net](https://www.w3schools.com/nodejs/ref_net.asp) | To create servers and clients |
| [os](https://www.w3schools.com/nodejs/ref_os.asp) | Provides information about the operation system |
| [path](https://www.w3schools.com/nodejs/ref_path.asp) | To handle file paths |
| punycode | Deprecated. A character encoding scheme |
| [querystring](https://www.w3schools.com/nodejs/ref_querystring.asp) | To handle URL query strings |
| [readline](https://www.w3schools.com/nodejs/ref_readline.asp) | To handle readable streams one line at the time |
| [stream](https://www.w3schools.com/nodejs/ref_stream.asp) | To handle streaming data |
| [string\_decoder](https://www.w3schools.com/nodejs/ref_string_decoder.asp) | To decode buffer objects into strings |
| [timers](https://www.w3schools.com/nodejs/ref_timers.asp) | To execute a function after a given number of milliseconds |
| [tls](https://www.w3schools.com/nodejs/ref_tls.asp) | To implement TLS and SSL protocols |
| tty | Provides classes used by a text terminal |
| [url](https://www.w3schools.com/nodejs/ref_url.asp) | To parse URL strings |
| [util](https://www.w3schools.com/nodejs/ref_util.asp) | To access utility functions |
| v8 | To access information about V8 (the JavaScript engine) |
| [vm](https://www.w3schools.com/nodejs/ref_vm.asp) | To compile JavaScript code in a virtual machine |
| [zlib](https://www.w3schools.com/nodejs/ref_zlib.asp) | To compress or decompress files |

# HTTP Module

### **Example**

Create a server that listens on port 8080 of your computer.

When port 8080 get accessed, write "Hello World!" back as a response:

var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.write('Hello World!');  
  res.end();  
}).listen(8080);

## **Definition and Usage**

The HTTP module provides a way of making Node.js transfer data over HTTP (Hyper Text Transfer Protocol).

## **Syntax**

The syntax for including the HTTP module in your application:

var http = require('http');

## **HTTP Properties and Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| createClient() | Deprecated. Creates a HTTP client |
| [createServer()](https://www.w3schools.com/nodejs/met_http_createserver.asp) | Creates an HTTP server |
| get() | Sets the method to GET, and returns an object containing the user's request |
| globalAgent | Returns the HTTP Agent |
| request() | Returns an object containing the user's request |

# Node.js File System Module

## **Node.js as a File Server**

The Node.js file system module allow you to work with the file system on your computer.

To include the File System module, use the require() method:

var fs = require('fs');

Common use for the File System module:

* Read files
* Create files
* Update files
* Delete files
* Rename files

## **Read Files**

The fs.readFile() method is used to read files on your computer.

Assume we have the following HTML file (located in the same folder as Node.js):

demofile1.html

<html>  
<body>  
<h1>My Header</h1>  
<p>My paragraph.</p>  
</body>  
</html>

Create a Node.js file that reads the HTML file, and return the content:

### **Example**

var http = require('http');  
var fs = require('fs');  
http.createServer(function (req, res) {  
**fs.readFile('demofile1.html', function(err, data) {**    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(data);  
    res.end();  
  });  
}).listen(8080);

Save the code above in a file called "demo\_readfile.js", and initiate the file:

Initiate demo\_readfile.js:

C:\Users\Your Name>node demo\_readfile.js

If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)

## **Create Files**

The File System module has methods for creating new files:

* fs.appendFile()
* fs.open()
* fs.writeFile()

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

### **Example**

Create a new file using the appendFile() method:

var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

### **Example**

Create a new, empty file using the open() method:

var fs = require('fs');  
  
fs.open('mynewfile2.txt', 'w', function (err, file) {  
  if (err) throw err;  
  console.log('Saved!');  
});

The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

### **Example**

Create a new file using the writeFile() method:

var fs = require('fs');  
  
fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

## **Update Files**

The File System module has methods for updating files:

* fs.appendFile()
* fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file:

### **Example**

Append "This is my text." to the end of the file "mynewfile1.txt":

var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {  
  if (err) throw err;  
  console.log('Updated!');  
});

The fs.writeFile() method replaces the specified file and content:

### **Example**

Replace the content of the file "mynewfile3.txt":

var fs = require('fs');  
  
fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {  
  if (err) throw err;  
  console.log('Replaced!');  
});

## **Delete Files**

To delete a file with the File System module,  use the fs.unlink() method.

The fs.unlink() method deletes the specified file:

### **Example**

Delete "mynewfile2.txt":

var fs = require('fs');  
  
fs.unlink('mynewfile2.txt', function (err) {  
  if (err) throw err;  
  console.log('File deleted!');  
});

## **Rename Files**

To rename a file with the File System module,  use the fs.rename() method.

The fs.rename() method renames the specified file:

### **Example**

Rename "mynewfile1.txt" to "myrenamedfile.txt":

var fs = require('fs');  
  
fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {  
  if (err) throw err;  
  console.log('File Renamed!');  
});