NODE JS Documents

**Node.js Module Types**

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

**core modules** are compiled into its binary distribution and load automatically when Node.js process starts.

**Syntax** – var/const module = require('module\_name');

The following table lists some of the important core modules in Node.js.

| Core Module | Description |
| --- | --- |
| [http](https://nodejs.org/api/http.html) | http module includes classes, methods and events to create Node.js http server. |
| [url](https://nodejs.org/api/url.html) | url module includes methods for URL resolution and parsing. |
| [querystring](https://nodejs.org/api/querystring.html) | querystring module includes methods to deal with query string. |
| [path](https://nodejs.org/api/path.html) | path module includes methods to deal with file paths. |
| [fs](https://nodejs.org/api/fs.html) | fs module includes classes, methods, and events to work with file I/O. |
| [util](https://nodejs.org/api/util.html) | util module includes utility functions useful for programmers. |

**Local modules** are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders.

Ex - Let's write simple logging module which logs the information, warning or error to the console.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

var log = {

info: function (info) {

console.log ('Info: ' + info);

},

warning:function (warning) {

console.log('Warning: ' + warning);

},

error:function (error) {

console.log('Error: ' + error);

}

};

module.exports = log

The following example demonstrates how to use the above logging module contained in Log.js.

var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');

# **Export Module in Node.js**

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.

Ex –

module.exports = 'Hello world';

**Export Object -** exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';

**Export Function -** module.exports = function (msg) {

console.log(msg);

};

## Export Function as a Class –

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.

Var/const person = require('./Person.js');

var/const person1 = new person('James', 'Bond');

console.log(person1.fullName());

include separate file under a folder

var log = require('./utility/log.js');

you can specify only the utility folder without specifying log.js, as shown below.

var log = require('./utility');

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”

{

"name”: "log",

"main”: "./log.js",

“name1”: “Server”,

“main1”: “server.js”

}

 Note:

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

# **Node.js Web Server**

Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously.

## Create Node.js Web Server - The following example is a simple Node.js web server contained in server.js file

## “./server.js”

var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) { // 2 - creating server

//handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')

## Note - In the above example, we import the http module using require() function. The http module is a core module of Node.js, so no need to install it using NPM. The next step is to call createServer() method of http and specify callback function with request and response parameter. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

## Handle HTTP Request –

## The http.createServer() method includes [request](https://nodejs.org/api/http.html#http_http_incomingmessage) and [response](https://nodejs.org/api/http.html#http_class_http_serverresponse) parameters which is supplied by Node.js. The request object can be used to get information about the current HTTP request e.g., url, request header, and data. The response object can be used to send a response for a current HTTP request.

## The following example demonstrates handling HTTP request and response in Node.js.

Var/const http = require('http'); // Import Node.js core module

Var/const server = http.createServer(function (req, res) { //create web server

if (req.url == '/') { //check the URL of the current request

// set response header

res.writeHead(200, { 'Content-Type': 'text/html' });

// set response content

res.write('<html><body><p>This is home Page.</p></body></html>');

res.end();

}

else if (req.url == "/student") {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write('<html><body><p>This is student Page.</p></body></html>');

res.end();

}

else if (req.url == "/admin") {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write('<html><body><p>This is admin Page.</p></body></html>');

res.end();

}

else

res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')

## Note - In the above example, req.url is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using writeHead() method and then writes a string as a response body using write() method. Finally, Node.js web server sends the response using end() method.

## Sending JSON Response

var http = require('http');

var server = http.createServer(function (req, res) {

if (req.url == '/data') { //check the URL of the current request

res.writeHead(200, { 'Content-Type': 'application/json' });

res.write(JSON.stringify({ message: "Hello World"}));

res.end();

}

});

server.listen(5000);

console.log('Node.js web server at port 5000 is running..')

# **Node.js File System**

Node.js includes **fs** module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.

**Reading File** - fs.readFile(fileName [,options], callback)

Parameter Description:

* filename: Full path and name of the file as a string.
* options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
* callback: A function with two parameters err and fd. This will get called when readFile operation completes.

The following example demonstrates reading existing TestFile.txt asynchronously.

Ex –

var fs = require('fs');

fs.readFile('TestFile.txt', function (err, data) {

if (err) throw err;

console.log(data);

});

Use fs.readFileSync() method to read file synchronously as shown below.

var fs = require('fs');

var data = fs.readFileSync('dummyfile.txt', 'utf8');

console.log(data);

**Writing File** - fs.writeFile(filename, data[, options], callback)

Parameter Description:

* filename: Full path and name of the file as a string.
* Data: The content to be written in a file.
* options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
* callback: A function with two parameters err and fd. This will get called when write operation completes.

The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

var fs = require('fs');

fs.writeFile('test.txt', 'Hello World!', function (err) {

if (err)

console.log(err);

else

console.log ('Write operation complete.');

});

## In the same way, use fs.appendFile() method to append the content to an existing file.

var fs = require('fs');

fs.appendFile('test.txt', 'Hello World!', function (err) {

if (err)

console.log(err);

else

console.log ('Append operation complete.');

});

**Open File** – fs.open(path, flags[, mode], callback)

## Alternatively, you can open a file for reading or writing using fs.open() method.

### **Flags**

The following table lists all the flags which can be used in read/write operation.

| Flag | Description |
| --- | --- |
| r | Open file for reading. An exception occurs if the file does not exist. |
| r+ | Open file for reading and writing. An exception occurs if the file does not exist. |
| rs | Open file for reading in synchronous mode. |
| rs+ | Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' about using this with caution. |
| w | Open file for writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx | Like 'w' but fails if path exists. |
| w+ | Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists). |
| wx+ | Like 'w+' but fails if path exists. |
| a | Open file for appending. The file is created if it does not exist. |
| ax | Like 'a' but fails if path exists. |
| a+ | Open file for reading and appending. The file is created if it does not exist. |
| ax+ | Like 'a+' but fails if path exists. |

The following example opens an existing file and reads its content.

var fs = require('fs');

fs.open('TestFile.txt', 'r', function (err, fd) {

if (err) {

return console.error(err);

}

var buffr = new Buffer(1024);

fs.read(fd, buffr, 0, buffr.length, 0, function (err, bytes) {

if (err) throw err;

// Print only read bytes to avoid junk.

if (bytes > 0) {

console.log(buffr.slice(0, bytes).toString());

}

// Close the opened file.

fs.close(fd, function (err) {

if (err) throw err;

});

});

});

**Delete File** - fs.unlink(path, callback);

**EX –**

var fs = require('fs');

fs.unlink('test.txt', function () {

console.log('write operation complete.');

});

## Important method of fs module

| Method | Description |
| --- | --- |
| fs.readFile(fileName [,options], callback) | Reads existing file. |
| fs.writeFile(filename, data[, options], callback) | Writes to the file. If file exists then overwrite the content otherwise creates new file. |
| fs.open(path, flags[, mode], callback) | Opens file for reading or writing. |
| fs.rename(oldPath, newPath, callback) | Renames an existing file. |
| fs.chown(path, uid, gid, callback) | Asynchronous chown. |
| fs.stat(path, callback) | Returns fs.stat object which includes important file statistics. |
| fs.link(srcpath, dstpath, callback) | Links file asynchronously. |
| fs.symlink(destination, path[, type], callback) | Symlink asynchronously. |
| fs.rmdir(path, callback) | Renames an existing directory. |
| fs.mkdir(path[, mode], callback) | Creates a new directory. |
| fs.readdir(path, callback) | Reads the content of the specified directory. |
| fs.utimes(path, atime, mtime, callback) | Changes the timestamp of the file. |
| fs.exists(path, callback) | Determines whether the specified file exists or not. |
| fs.access(path[, mode], callback) | Tests a user's permissions for the specified file. |
| fs.appendFile(file, data[, options], callback) | Appends new content to the existing file. |

Visit Node documentation for more information on [fs module](https://nodejs.org/api/fs.html#fs_fs_readfile_file_options_callback)

## Debug Node.js Application

You can debug Node.js application using various tools including following:

1. Core Node.js debugger
2. Node Inspector
3. Built-in debugger in IDEs

## Core Node.js Debugger

Syntax –

var fs = require('fs');

fs.readFile('test.txt', 'utf8', function (err, data) {

debugger;

if (err) throw err;

console.log(data);

});

## Note - Now, to debug the above application, run the following command.

## node debug app.js

The following table lists important debugging commands:

| Command | Description |
| --- | --- |
| next | Stop at the next statement. |
| cont | Continue execute and stop at the debugger statement if any. |
| step | Step in function. |
| out | Step out of function. |
| watch | Add the expression or variable into watch. |
| watcher | See the value of all expressions and variables added into watch. |
| Pause | Pause running code. |

**Node.js EventEmitter –**

Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

**Syntax –**

// get the reference of EventEmitter class of events module

var events = require('events');

//create an object of EventEmitter class by using above reference

var em = new events.EventEmitter();

//Subscribe for FirstEvent

em.on('FirstEvent', function (data) {

console.log('First subscriber: ' + data);

});

// Raising FirstEvent

em.emit('FirstEvent', 'This is my first Node.js event emitter example.');

The following table lists all the important methods of EventEmitter class.

| EventEmitter Methods | Description |
| --- | --- |
| [emitter.addListener(event, listener)](https://nodejs.org/api/events.html#events_emitter_addlistener_event_listener) | Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. |
| [emitter.on(event, listener)](https://nodejs.org/api/events.html#events_emitter_on_event_listener) | Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener() |
| [emitter.once(event, listener)](https://nodejs.org/api/events.html#events_emitter_once_event_listener) | Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed. |
| [emitter.removeListener(event, listener)](https://nodejs.org/api/events.html#events_emitter_removelistener_event_listener) | Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener. |
| [emitter.removeAllListeners([event])](https://nodejs.org/api/events.html#events_emitter_removealllisteners_event) | Removes all listeners, or those of the specified event. |
| [emitter.setMaxListeners(n)](https://nodejs.org/api/events.html#events_emitter_setmaxlisteners_n) | By default EventEmitters will print a warning if more than 10 listeners are added for a particular event. |
| [emitter.getMaxListeners()](https://nodejs.org/api/events.html#events_emitter_getmaxlisteners) | Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners. |
| [emitter.listeners(event)](https://nodejs.org/api/events.html#events_emitter_listeners_event) | Returns a copy of the array of listeners for the specified event. |
| [emitter.emit(event[, arg1][, arg2][, ...])](https://nodejs.org/api/events.html#events_emitter_emit_event_arg1_arg2) | Raise the specified events with the supplied arguments. |
| [emitter.listenerCount(type)](https://nodejs.org/api/events.html#events_emitter_listenercount_type) | Returns the number of listeners listening to the type of event. |