As you have already seen how to write a program in nodeJS. Now you will learn how to work with files in nodeJS using file system module

**Need for File system module**

If a web server is programmed using NodeJS, then one of the important things is to be able to read or write the content of a file as in a web application, these file operations are a frequent occurrence.

Node.js comes with the file system module **'fs'.**

**What is an fs module?**

It is a module which provides a wrapper for performing standard file operations.

**How to use File system module?**

Use the require() method, to include the File System module

1. const fs = require('fs');

**Note:** The methodrequire() invokes the particular JavaScript module into the file. So, if you want to load a local, relative JavaScript module into your node application, you can use the require()method.

Some of the file operations that we will be discussing are:

* Writing data to a file
* Reading data from a file
* Updating content in a file

We will now begin our file system operations by understanding how to write data to a file.

The File System module has the following methods for creating new file and writing data to that file:

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

The methodfs.writeFile() will **overwrite** the content if the content already exists.

If the file does not exist, then a new file will be created with the specified name and content.

**Syntax:**

1. fs.writeFile(file\_path, data, callback)

**file\_path**: Placeholder to give the file path in which you are going to write the data.

**data**: For data/content that is to be written to the file.

**callback**: The callback method, that is to be executed whenwriteFile() function is called. This callback will be executed in both success as well as failure scenario.

Let us now see how to write data to a file.

# Highlights:

* Usage of fs.writeFile() method
* To 'write' data to a file

**Step 1:** Create a file **fileSystem.js** and paste the below code in the file

1. const fs = require('fs');
2. let fileWrite = () => {
3. let str = "Hey, there.. ";
4. fs.writeFile('./log.txt', str + " At: " + new Date().getFullYear(), (err) => {
5. if (!err)
6. console.log('Data Written!');
7. else
8. throw err;
9. })
10. }
11. fileWrite();

**Step 2:** Run the above code using the node command '**node fileSystem**' to see the output.

**Step 3:**Observe in the explorer section of your VS code IDE. You can observe that a new file named **log.txt** has been automatically created logging the string content and date.

https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_32692787699721360000_shared/web-hosted/assets/fileread.PNG

**Step 4:** Provide different values in the data/content part of the fs.writeFile() function and observe the '**log.txt**' file.

**Step 5:** You might have observed that the log file is logging only the recent content.The earlier content will be replaced with the new data.

The fs.writeFile() method has overwritten the previous content. To resolve this, we will use fs.appendFile().

# ProblemStatement:

The fs.appendFile() first checks if the file exists or not. If the file does not exist, then it creates a new file with the content, else it appends the given content to the existing file.

# Syntax:

1. fs.appendFile(file\_path, data, callback);

**file\_path :** Placeholder to give the file path in which you are going to append the data.

**data :** For data which has to be appended to the file.

**callback :**The callback method, that will be implemented, when appendFile() function is called.

The fs.readFile() method is used to read the content from a given file.

**Syntax:**

1. fs.readFile(file\_path, encoding, callback);

**file\_path:** Path where the file with data/content resides, with respect to the root folder.

**encoding:** encoding is an optional parameter that specifies the type of encoding to read the file. Possible encodings are 'ascii', 'utf8', and 'base64'.

**callback:** The callback method, that will be executed, when writeFile() function is called. The first parameter of the callback function is error object and the second parameter is the content read from the given file.

**Example:**

1. const fs = require('fs');
3. fs.readFile('demo.txt', 'utf8', (err, content) => {
4. console.log(content);
5. });

Next, we'll see a demo on reading data from a file.

# Highlights:

* Usage of fs.readFile() method
* To read the content of a file

**Step 1:** Create a JavaScript file 'fileSystem.js' and add the below code:

1. const fs = require('fs');
3. fs.readFile('demo.txt', 'utf8', (err, content) => {
4. if(!err)
5. console.log(content);
6. else
7. throw err;
8. });
10. console.log('After calling readFile');

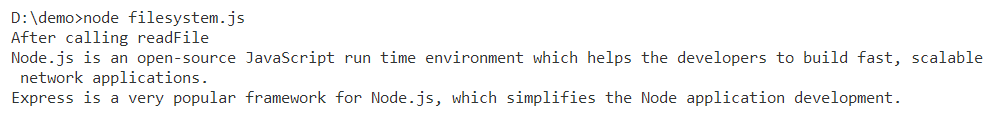
**Step 2:** Create a file **demo.txt** and add the following text.

1. Node.js is an open-source JavaScript run time environment which helps the developers to build fast, scalable network applications.
2. Express is a very popular framework for Node.js, which simplifies the Node application development.

**Step 3:** Run the file 'fileSystem.js' using node command

https://academy.onwingspan.com/common-content-store/Shared/Shared/Public/lex_32692787699721360000_shared/web-hosted/assets/nodefs.PNG

**Step 4:** Observe the output in the console.



Task:

**Problem Statement:**

Create a text file **names.txt** and add the following data to it.

1. Oliver,Jack,Harry,Jacob,Charlie,Thomas

Now create a JavaScript file **user.js** and copy paste the below code

1. const fs = require("fs");
2. let logUser = (username) => {
3. *//Your implementation goes here*
4. };
5. logUser("Sam");

Whenever a function call happens, you should check whether the username already exists in the text file or not. If exists display a message saying "User already exists". If not, add the user to the text file as a comma separated. After adding the username to the text file, display a success message saying "<<username>> has been added successfully".

**Note:** Perform case insensitive comparison

**Sample Input and Expected Output:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sam | Sam has been added successfully |
| jack | User already exists |

Next we will be learn how to modularize the program in a node application.