

<b>Name of Student: Pushkar Sane</b>		
<b>Roll Number: 45</b>		<b>Lab Assignment Number: 4</b>
<b>Title of Lab Assignment: Using file handling to demonstrate all basic file operations (create, write, read, delete).</b>		
<b>DOP: 09-10-2023</b>		<b>DOS: 10-10-2023</b>
<b>CO Mapped:</b> <b>CO1</b>	<b>PO Mapped:</b> <b>PO3, PO5, PSO1, PSO2</b>	<b>Signature:</b>

## **Practical No. 5**

### **Aim:**

Using file handling to demonstrate all basic file operations (Create, write, read, delete).

### **Theory:**

#### **File Handling in Node.js:**

1. The most important functionalities provided by programming languages are Reading and Writing files from computers. Node.js provides the functionality to read and write files from the computer.
2. Reading and Writing the file in Node.js is done by using one of the coolest Node.js modules called fs module, it is one of the most well-known built-in Node.js modules out there.
3. The file can be read and written in node.js in both Synchronous and Asynchronous ways.
4. A Synchronous method is a code-blocking method which means the given method will block the execution of code until its execution is finished (i.e. Complete file is read or written).
5. On the other hand, an Asynchronous method has a callback function that is executed on completion of the execution of the given method and thus allows code to run during the completion of its execution.

#### **Creating a file:**

To create a new file in Node.js, you can use the `fs.writeFile()` method. This method allows you to specify the filename, the content to be written to the file, and a callback function to handle any potential errors. Here's an example:

#### **Example:**

```
const fs = require('fs');
fs.writeFile('sample.txt', 'This is a sample file created in Node.js.', (err) => {
  if (err) throw err;
  console.log('File created successfully!');
});
```

**Writing to a File:**

To add data to an existing file or append to its content, you can use the `fs.appendFile()` method. This method allows you to append data to a file without overwriting its existing content:

Example:

```
const fs = require('fs');
fs.appendFile('sample.txt', '\nThis is additional text.', (err) => {
  if (err) throw err;
  console.log('Data appended to the file!');
});
```

**Reading from a File:**

To read data from a file in Node.js, you can use the `fs.readFile()` method. This method reads the content of a file and provides it as a callback parameter for further processing:

Example:

```
const fs = require('fs');
fs.readFile('sample.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File content:\n', data);
});
```

**Deleting a File:**

To delete a file in Node.js, you can use the `fs.unlink()` method. This method takes the filename as an argument and deletes the file:

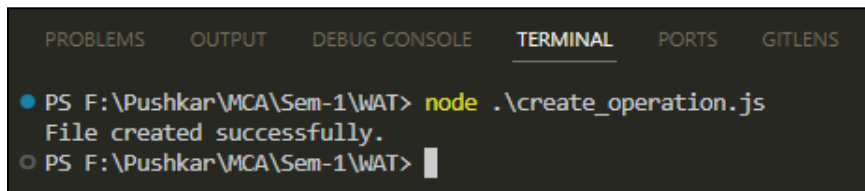
Example:

```
const fs = require('fs');
fs.unlink('sample.txt', (err) => {
  if (err) throw err;
  console.log('File deleted successfully!');
});
```

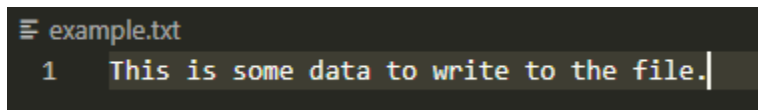
When working with file operations in Node.js, it's important to handle errors gracefully, as various issues like file permission problems or file not found errors can occur. The provided code examples include error handling to ensure robust file handling in your Node.js applications.

**Code:****1. Creating a File (Write Operation):**

```
const fs = require('fs');  
// Data to be written to the file  
const dataToWrite = 'This is some data to write to the file.';  
// Specify the filename and content  
fs.writeFile('example.txt', dataToWrite, (err) => {  
  if (err) {  
    console.error('Error writing to the file:', err);  
  } else {  
    console.log('Hello World in this file from Sweetie ');  
  }  
});
```

**Output:**

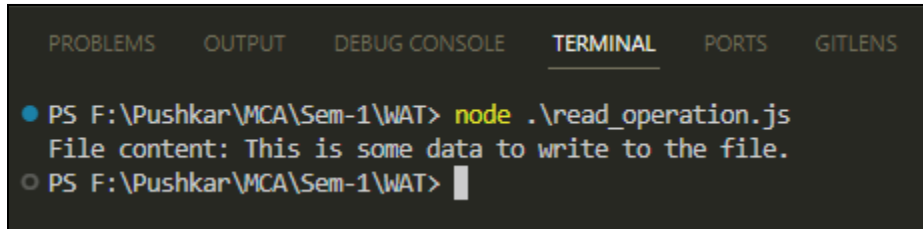
The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and GITLENS. The TERMINAL tab is active. It shows a PowerShell prompt at 'PS F:\Pushkar\MCA\Sem-1\WAT>' where the command 'node .\create\_operation.js' has been executed. The output is 'File created successfully.' followed by another PowerShell prompt.



The screenshot shows a text editor window with the filename 'example.txt'. The content of the file is '1 This is some data to write to the file.' with the cursor at the end of the line.

**2. Reading a File (Read Operation):**

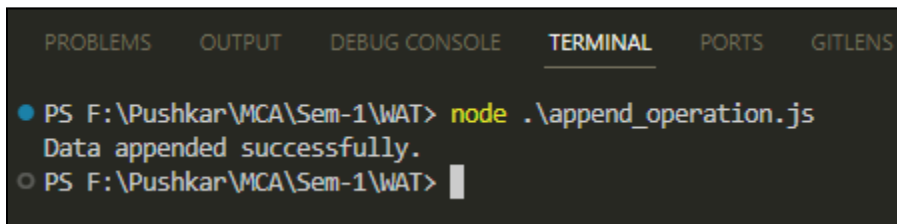
```
const fs = require('fs');  
fs.readFile('example.txt', 'utf8', (err, data) => {  
  if (err) {  
    console.error('Error reading the file:', err);  
  } else {  
    console.log('File content:', data);  
  }  
});
```

**Output:**

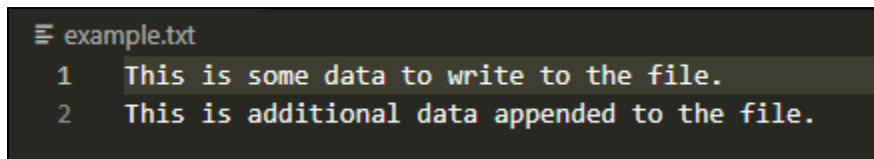
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\read_operation.js
  File content: This is some data to write to the file.
○ PS F:\Pushkar\MCA\Sem-1\WAT> 
```

**3. Appending to a File (Write Operation):**

```
const fs = require("fs");
const newDataToAppend = "\nThis is additional data appended to the file.";
fs.appendFile("example.txt", newDataToAppend, (err) => {
  if (err) {
    console.error("Error appending to the file:", err);
  } else {
    console.log("Data appended successfully.");
  }
});
```

**Output:**

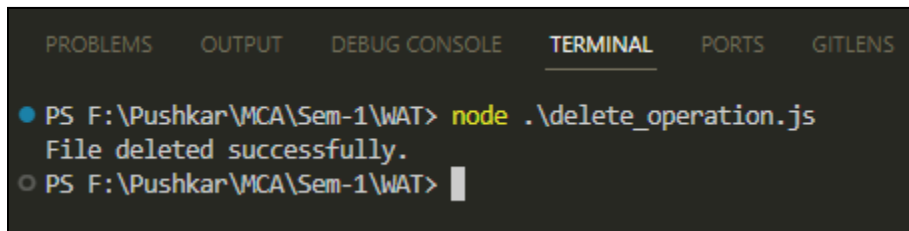
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\append_operation.js
  Data appended successfully.
○ PS F:\Pushkar\MCA\Sem-1\WAT> 
```



```
example.txt
1 This is some data to write to the file.
2 This is additional data appended to the file.
```

**4. Deleting a File (Delete Operation):**

```
const fs = require("fs");
fs.unlink("example.txt", (err) => {
  if (err) {
    console.error("Error deleting the file:", err);
  } else {
    console.log("File deleted successfully.");
  }
});
```

**Output:**A screenshot of a Visual Studio Code terminal window. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is active), PORTS, and GITLENS. The terminal shows a command prompt where the user has run 'node .\delete\_operation.js'. The output of the script is 'File deleted successfully.' followed by a new command prompt line.

```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\delete_operation.js
File deleted successfully.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

**Conclusion:**

By showcasing basic file operations in Node.js, including file creation, writing, reading, and deletion, we've illustrated the fundamental capabilities of file handling in a programming environment. These operations form the building blocks for managing data persistence and interaction with the file system within Node.js applications. Mastery of these operations is essential for effective file management and data manipulation in various software solutions.