**Node.js File System (FS) Operations**

**1. Creating Files and Folders**

**1.1 Create/Write to a File**

To create or write to a file, use fs.writeFile() or fs.writeFileSync() for synchronous operations.

* **Async version (fs.writeFile())**:

const fs = require('fs');

// Create or overwrite a file (async)

fs.writeFile('example.txt', 'Hello, World!', (err) => {

if (err) throw err;

console.log('File created and written to!');

});

* **Sync version (fs.writeFileSync())**:

const fs = require('fs');

// Create or overwrite a file (sync)

fs.writeFileSync('example.txt', 'Hello, World!');

console.log('File created and written to!');

**1.2 Create Multiple Files**

To create multiple files, use a loop:

const fs = require('fs');

const files = ['file1.txt', 'file2.txt', 'file3.txt'];

files.forEach((file) => {

fs.writeFile(file, 'Content', (err) => {

if (err) throw err;

console.log(`${file} created!`);

});

});

**1.3 Create Folders/Directories**

To create folders, use fs.mkdir() for asynchronous operations.

* **Async version (fs.mkdir())**:

const fs = require('fs');

// Create a folder

fs.mkdir('my\_folder', (err) => {

if (err) throw err;

console.log('Folder created!');

});

* **Sync version (fs.mkdirSync())**:

const fs = require('fs');

// Create a folder synchronously

fs.mkdirSync('my\_folder');

console.log('Folder created!');

**1.4 Create Nested Folders/Directories Recursively**

To create nested directories (e.g., parent\_folder/subfolder1/subfolder2), use { recursive: true }.

* **Async version (fs.mkdir() with recursion)**:

const fs = require('fs');

// Create nested folders recursively

fs.mkdir('parent\_folder/subfolder1/subfolder2', { recursive: true }, (err) => {

if (err) throw err;

console.log('Nested folders created!');

});

* **Sync version (fs.mkdirSync() with recursion)**:

const fs = require('fs');

// Create nested folders synchronously

fs.mkdirSync('parent\_folder/subfolder1/subfolder2', { recursive: true });

console.log('Nested folders created!');

**2. Reading Files and Folders**

**2.1 Read a File**

To read a file, use fs.readFile() or fs.readFileSync().

* **Async version (fs.readFile())**:

const fs = require('fs');

// Read a file asynchronously

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log('File content:', data);

});

* **Sync version (fs.readFileSync())**:

const fs = require('fs');

// Read a file synchronously

const data = fs.readFileSync('example.txt', 'utf8');

console.log('File content:', data);

**2.2 Read a Directory**

To read the contents of a directory, use fs.readdir() or fs.readdirSync().

* **Async version (fs.readdir())**:

const fs = require('fs');

// Read contents of a directory

fs.readdir('my\_folder', (err, files) => {

if (err) throw err;

console.log('Files in directory:', files);

});

* **Sync version (fs.readdirSync())**:

const fs = require('fs');

// Read contents of a directory synchronously

const files = fs.readdirSync('my\_folder');

console.log('Files in directory:', files);

**2.3 Get Stats for a File or Directory**

To get stats like file size, last modified time, etc., use fs.stat().

* **Async version (fs.stat())**:

const fs = require('fs');

// Get stats for a file or folder

fs.stat('example.txt', (err, stats) => {

if (err) throw err;

console.log('File stats:', stats);

});

* **Sync version (fs.statSync())**:

const fs = require('fs');

// Get stats synchronously

const stats = fs.statSync('example.txt');

console.log('File stats:', stats);

**3. Updating Files**

**3.1 Overwrite a File**

To overwrite a file, you can use fs.writeFile() or fs.writeFileSync().

const fs = require('fs');

// Overwrite the content of a file

fs.writeFile('example.txt', 'Updated content!', (err) => {

if (err) throw err;

console.log('File updated!');

});

**3.2 Append to a File**

To append content to a file, use fs.appendFile() or fs.appendFileSync().

* **Async version (fs.appendFile())**:

const fs = require('fs');

// Append to a file

fs.appendFile('example.txt', '\nAppended text', (err) => {

if (err) throw err;

console.log('Text appended!');

});

* **Sync version (fs.appendFileSync())**:

const fs = require('fs');

// Append to a file synchronously

fs.appendFileSync('example.txt', '\nAppended text');

console.log('Text appended!');

**4. Deleting Files and Folders**

**4.1 Delete a File**

To delete a file, use fs.unlink() or fs.unlinkSync().

* **Async version (fs.unlink())**:

const fs = require('fs');

// Delete a file

fs.unlink('example.txt', (err) => {

if (err) throw err;

console.log('File deleted!');

});

* **Sync version (fs.unlinkSync())**:

const fs = require('fs');

// Delete a file synchronously

fs.unlinkSync('example.txt');

console.log('File deleted!');

**4.2 Delete a Folder**

To delete an empty folder, use fs.rmdir() or fs.rmdirSync().

* **Async version (fs.rmdir())**:

const fs = require('fs');

// Delete an empty folder

fs.rmdir('my\_folder', (err) => {

if (err) throw err;

console.log('Folder deleted!');

});

* **Sync version (fs.rmdirSync())**:

const fs = require('fs');

// Delete an empty folder synchronously

fs.rmdirSync('my\_folder');

console.log('Folder deleted!');

**4.3 Delete Folder and Contents Recursively**

To delete a folder and its contents recursively, use fs.rm() (from Node.js v16+).

* **Async version (fs.rm() for recursive deletion)**:

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

// Delete folder and contents recursively

async function deleteFolderRecursive(path) {

try {

await fs.rm(path, { recursive: true, force: true });

console.log(`${path} and all contents deleted!`);

} catch (err) {

console.error('Error deleting folder', err);

}

}

deleteFolderRecursive('parent\_folder');

* **Recursive deletion manually** (for older versions or custom logic):

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

async function deleteFolderRecursive(path) {

try {

const files = await fs.readdir(path);

for (let file of files) {

const currentPath = `${path}/${file}`;

const stat = await fs.stat(currentPath);

if (stat.isDirectory()) {

// Recursively delete subfolders

await deleteFolderRecursive(currentPath);

} else {

// Delete files

await fs.unlink(currentPath);

}

}

// Delete the empty folder

await fs.rmdir(path);

console.log(`${path} deleted!`);

} catch (err) {

console.error('Error deleting folder', err);

}

}

deleteFolderRecursive('parent\_folder');

**5. Notes on Recursion for Folders**

* **Creating Nested Folders**: The { recursive: true } option helps create multiple levels of nested directories without errors.
* **Deleting Nested Folders**: To delete folders recursively, you need to loop through the directory’s contents, delete each file/folder, and then remove the parent folder once it’s empty.
* **Error Handling**: Always handle errors (e.g., file not found, permission issues) with proper try-catch blocks for async operations or callbacks for async FS functions.

**1. Checking if a File or Folder Exists**

Node.js doesn't have a direct fs.exists() method anymore (it has been deprecated). However, you can use other methods like fs.stat() or fs.access().

**1.1 Using fs.stat()**

The fs.stat() method checks whether a file or folder exists and retrieves details like its stats (size, type, permissions, etc.). You can handle the error if the file/folder doesn't exist.

* **Async Version (fs.stat())**:

const fs = require('fs');

fs.stat('example.txt', (err, stats) => {

if (err) {

if (err.code === 'ENOENT') {

console.log('File or directory does not exist!');

} else {

console.error('Some other error:', err);

}

} else {

console.log('File or directory exists!');

console.log('Stats:', stats);

}

});

* **Sync Version (fs.statSync())**:

const fs = require('fs');

try {

const stats = fs.statSync('example.txt');

console.log('File or directory exists!');

console.log('Stats:', stats);

} catch (err) {

if (err.code === 'ENOENT') {

console.log('File or directory does not exist!');

} else {

console.error('Some other error:', err);

}

}

**1.2 Using fs.access()**

The fs.access() method checks if a file or directory is accessible (i.e., if it exists and if you have permission to access it). This method doesn't return file stats, it just checks access.

* **Async Version (fs.access())**:

const fs = require('fs');

fs.access('example.txt', fs.constants.F\_OK, (err) => {

if (err) {

console.log('File or directory does not exist!');

} else {

console.log('File or directory exists!');

}

});

* **Sync Version (fs.accessSync())**:

const fs = require('fs');

try {

fs.accessSync('example.txt', fs.constants.F\_OK);

console.log('File or directory exists!');

} catch (err) {

console.log('File or directory does not exist!');

}

**2. Explanation of Methods**

* **fs.stat() / fs.statSync()**: Provides file stats (size, creation date, permissions). If the file doesn't exist, it throws an error.
* **fs.access() / fs.accessSync()**: Checks if the file or directory exists and if it's accessible. It's faster because it doesn't return file stats, only checks if the file is there and can be accessed.
  + fs.constants.F\_OK: This constant is used to check if the file or directory exists.

**3. Recap**

* Use fs.stat() if you need file details and want to check existence.
* Use fs.access() if you just want to check if a file or directory is there and accessible.