Tutorials References Exercises Sign In
CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C

Node.js File System Module

❮ Previous Next ❯

Introduction to Node.js File System

The Node.js File System module (fs) provides a comprehensive set of methods for working with the file system on your computer.

It allows you to perform file I/O operations in both synchronous and asynchronous ways.

Note: The File System module is a core Node.js module, so no installation is required.

Importing the File System Module

You can import the File System module using CommonJS `require()` or ES modules `import` syntax:

CommonJS (Default in Node.js)

```
const fs = require('fs');
```

ES Modules (Node.js 14+ with "type": "module" in package.json)

Promise-based API

Node.js provides promise-based versions of the File System API in the `fs/promises` namespace, which is recommended for modern applications:

```
// Using promises (Node.js 10.0.0+)
const fs = require('fs').promises;

// Or with destructuring
const { readFile, writeFile } = require('fs').promises;

// Or with ES modules
// import { readFile, writeFile } from 'fs/promises';
```

Common Use Cases

File Operations

- Read and write files
- Create and delete files
- Append to files
- Rename and move files
- Change file permissions

Directory Operations

- Create and remove directories
- List directory contents
- Watch for file changes
- Get file/directory stats
- Check file existence

Advanced Features

- File streams
- File descriptors
- Symbolic links
- File watching
- Working with file permissions

Tutorials References Exercises Sign In
CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C

Performance Tip: For large files, consider using streams (`fs.createReadStream` and `fs.createWriteStream`) to avoid high memory usage.

Reading Files

Node.js provides several methods to read files, including both callback-based and promise-based approaches.

The most common method is `fs.readFile()` .

Note: Always handle errors when working with file operations to prevent your application from crashing.

Reading Files with Callbacks

Here's how to read a file using the traditional callback pattern:

myfile.txt

```
This is the content of myfile.txt
```

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

Example: Reading a file with callbacks

```
const fs = require('fs');

// Read file asynchronously with callback
fs.readFile('myfile.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
```

```
// For binary data (like images), omit the encoding
fs.readFile('image.png', (err, data) => {
  if (err) throw err;
  // data is a Buffer containing the file content
  console.log('Image size:', data.length, 'bytes');
});
```

Run example »

Reading Files with Promises (Modern Approach)

Using `fs.promises` or `util.promisify` for cleaner async/await syntax:

Example: Reading a file with async/await

```
// Using fs.promises (Node.js 10.0.0+)
const fs = require('fs').promises;

async function readFileExample() {
  try {
    const data = await fs.readFile('myfile.txt', 'utf8');
    console.log('File content:', data);
  } catch (err) {
    console.error('Error reading file:', err);
  }
}

readFileExample();

// Or with util.promisify (Node.js 8.0.0+)
const { promisify } = require('util');
const readFileAsync = promisify(require('fs').readFile);

async function readWithPromisify() {
  try {
    const data = await readFileAsync('myfile.txt', 'utf8');
    console.log(data);
```

```
readWithPromisify();
```

Run example »

Reading Files Synchronously

For simple scripts, you can use synchronous methods, but avoid them in production servers as they block the event loop:

Example: Reading a file synchronously

```
const fs = require('fs');

try {
  // Read file synchronously
  const data = fs.readFileSync('myfile.txt', 'utf8');
  console.log('File content:', data);
} catch (err) {
  console.error('Error reading file:', err);
}
```

Best Practice: Always specify the character encoding (like 'utf8') when reading text files to get a string instead of a Buffer.

Creating and Writing Files

Node.js provides several methods for creating and writing to files.

Creates a new file or overwrites an existing file with the specified content:

Example: Writing to a file

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

async function writeFileExample() {
  try {
    // Write text to a file
    await fs.writeFile('myfile.txt', 'Hello, World!', 'utf8');

    // Write JSON data
    const data = { name: 'John', age: 30, city: 'New York' };
    await fs.writeFile('data.json', JSON.stringify(data, null, 2),
'utf8');

    console.log('Files created successfully');
  } catch (err) {
    console.error('Error writing files:', err);
  }
}

writeFileExample();
```

Run example »

2. Using `fs.appendFile()`

Appends content to a file, creating the file if it doesn't exist:

Example: Appending to a file

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

async function appendToFile() {
  try {
    // Append a timestamped log entry
```

```
    console.log('Log entry added');
  } catch (err) {
    console.error('Error appending to file:', err);
  }
}

appendToFile();
```

Run example »

3. Using File Handles

For more control over file operations, you can use file handles:

Example: Using file handles

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

async function writeWithFileHandle() {
  let fileHandle;

  try {
    // Open a file for writing (creates if doesn't exist)
    fileHandle = await fs.open('output.txt', 'w');

    // Write content to the file
    await fileHandle.write('First line\n');
    await fileHandle.write('Second line\n');
    await fileHandle.write('Third line\n');

    console.log('Content written successfully');
  } catch (err) {
    console.error('Error writing to file:', err);
  } finally {
    // Always close the file handle
    if (fileHandle) {
      await fileHandle.close();
    }
```

Run example »

4. Using Streams for Large Files

For writing large amounts of data, use streams to avoid high memory usage:

Example: Writing large files with streams

```
const fs = require('fs');
const { pipeline } = require('stream/promises');
const { Readable } = require('stream');

async function writeLargeFile() {
  // Create a readable stream (could be from HTTP request, etc.)
  const data = Array(1000).fill().map((_, i) => `Line ${i + 1}:
${'x'.repeat(100)}\n`);
  const readable = Readable.from(data);

  // Create a writable stream to a file
  const writable = fs.createWriteStream('large-file.txt');

  try {
    // Pipe the data from readable to writable
    await pipeline(readable, writable);
    console.log('Large file written successfully');
  } catch (err) {
    console.error('Error writing file:', err);
  }
}

writeLargeFile();
```

Run example »

- `'w+'` - Open for reading and writing (file is created or truncated)
- `'a'` - Open for appending (file is created if it doesn't exist)
- `'ax'` - Like 'a' but fails if the path exists
- `'r+'` - Open for reading and writing (file must exist)

Deleting Files and Directories

Node.js provides several methods to delete files and directories.

Here's how to handle different deletion scenarios:

1. Deleting a Single File

Use `fs.unlink()` to delete a file:

Example: Deleting a file

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

async function deleteFile() {
  const filePath = 'file-to-delete.txt';

  try {
    // Check if file exists before deleting
    await fs.access(filePath);

    // Delete the file
    await fs.unlink(filePath);
    console.log('File deleted successfully');
  } catch (err) {
    if (err.code === 'ENOENT') {
      console.log('File does not exist');
    } else {
      console.error('Error deleting file:', err);
    }
  }
```

Tutorials References Exercises Sign In
CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C

Run example »

2. Deleting Multiple Files

To delete multiple files, you can use Promise.all() with fs.unlink():

Example: Deleting multiple files

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

async function deleteFiles() {
  const filesToDelete = [
    'temp1.txt',
    'temp2.txt',
    'temp3.txt'
  ];

  try {
    // Delete all files in parallel
    await Promise.all(
      filesToDelete.map(file =>
        fs.unlink(file).catch(err => {
          if (err.code !== 'ENOENT') {
            console.error(`Error deleting ${file}:`, err);
          }
        })
      )
    );

    console.log('Files deleted successfully');
  } catch (err) {
    console.error('Error during file deletion:', err);
  }
}

deleteFiles();
```

Tutorials References Exercises Sign In
CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C

3. Deleting Directories

To delete directories, you have several options depending on your needs:

Example: Deleting directories

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

async function deleteDirectory(dirPath) {
  try {
    // Check if the directory exists
    const stats = await fs.stat(dirPath);

    if (!stats.isDirectory()) {
      console.log('Path is not a directory');
      return;
    }

    // For Node.js 14.14.0+ (recommended)
    await fs.rm(dirPath, { recursive: true, force: true });

    // For older Node.js versions (deprecated but still works)
    // await fs.rmdir(dirPath, { recursive: true });

    console.log('Directory deleted successfully');
  } catch (err) {
    if (err.code === 'ENOENT') {
      console.log('Directory does not exist');
    } else {
      console.error('Error deleting directory:', err);
    }
  }
}

// Usage
deleteDirectory('directory-to-delete');
```

Run example »

Example: Emptying a directory

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

async function emptyDirectory(dirPath) {
  try {
    // Read the directory
    const files = await fs.readdir(dirPath, { withFileTypes: true });

    // Delete all files and directories in parallel
    await Promise.all(
      files.map(file => {
        const fullPath = path.join(dirPath, file.name);
        return file.isDirectory()
          ? fs.rm(fullPath, { recursive: true, force: true })
          : fs.unlink(fullPath);
      })
    );

    console.log('Directory emptied successfully');
  } catch (err) {
    console.error('Error emptying directory:', err);
  }
}

// Usage
emptyDirectory('directory-to-empty');
```

Run example »

Security Note: Be extremely careful with file deletion, especially when using recursive options or wildcards. Always validate and sanitize file paths to prevent directory traversal attacks.

It's a versatile method for file system operations that involve changing file paths.

1. Basic File Renaming

To rename a file in the same directory:

Example: Renaming a file

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

async function renameFile() {
  const oldPath = 'old-name.txt';
  const newPath = 'new-name.txt';

  try {
    // Check if source file exists
    await fs.access(oldPath);

    // Check if destination file already exists
    try {
      await fs.access(newPath);
      console.log('Destination file already exists');
      return;
    } catch (err) {
      // Destination doesn't exist, safe to proceed
    }

    // Perform the rename
    await fs.rename(oldPath, newPath);
    console.log('File renamed successfully');
  } catch (err) {
    if (err.code === 'ENOENT') {
      console.log('Source file does not exist');
    } else {
      console.error('Error renaming file:', err);
    }
  }
}
```

2. Moving Files Between Directories

You can use `fs.rename()` to move files between directories:

Example: Moving a file to a different directory

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

async function moveFile() {
  const sourceFile = 'source/file.txt';
  const targetDir = 'destination';
  const targetFile = path.join(targetDir, 'file.txt');

  try {
    // Ensure source file exists
    await fs.access(sourceFile);

    // Create target directory if it doesn't exist
    await fs.mkdir(targetDir, { recursive: true });

    // Move the file
    await fs.rename(sourceFile, targetFile);

    console.log('File moved successfully');
  } catch (err) {
    if (err.code === 'ENOENT') {
      console.log('Source file does not exist');
    } else if (err.code === 'EXDEV') {
      console.log('Cross-device move detected, using copy+delete
fallback');
      await moveAcrossDevices(sourceFile, targetFile);
    } else {
      console.error('Error moving file:', err);
    }
  }
}
```

```
    // Copy the file
    await fs.copyFile(source, target);

    // Delete the original
    await fs.unlink(source);

    console.log('File moved across devices successfully');
  } catch (err) {
    // Clean up if something went wrong
    try { await fs.unlink(target); } catch (e) {}
    throw err;
  }
}

// Usage
moveFile();
```

Run example »

3. Batch Renaming Files

To rename multiple files matching a pattern:

Example: Batch renaming files

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

async function batchRename() {
  const directory = 'images';
  const pattern = /^image(\d+)\.jpg$/;

  try {
    // Read directory contents
    const files = await fs.readdir(directory);

    // Process each file
    for (const file of files) {
```

```
        const oldPath = path.join(directory, file);
        const newPath = path.join(directory, newName);

        // Skip if the new name is the same as the old name
        if (oldPath !== newPath) {
          await fs.rename(oldPath, newPath);
          console.log(`Renamed: ${file} - ${newName}`);
        }
      }
    }

    console.log('Batch rename completed');
  } catch (err) {
    console.error('Error during batch rename:', err);
  }
}

batchRename();
```

Run example »

4. Atomic Rename Operations

For critical operations, use a temporary file to ensure atomicity:

Example: Atomic file update

```
const fs = require('fs').promises;
const path = require('path');
const os = require('os');

async function updateFileAtomic(filePath, newContent) {
  const tempPath = path.join(
    os.tmpdir(),
    `temp-${Date.now()}-${Math.random().toString(36).substr(2, 9)}`
  );

  try {
```

```
    const stats = await fs.stat(tempPath);
    if (stats.size === 0) {
      throw new Error('Temporary file is empty');
    }

    // 3. Rename (atomic on most systems)
    await fs.rename(tempPath, filePath);

    console.log('File updated atomically');
  } catch (err) {
    // Clean up temp file if it exists
    try { await fs.unlink(tempPath); } catch (e) {}

    console.error('Atomic update failed:', err);
    throw err;
  }
}

// Usage
updateFileAtomic('important-config.json', JSON.stringify({ key:
'value' }, null, 2));
```

Cross-Platform Note: The `fs.rename()` operation is atomic on Unix-like systems but may not be on Windows.

For cross-platform atomic operations, consider using a temporary file approach as shown in the example above.

Exercise ?

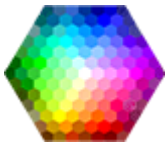
Drag and drop the correct word to complete the sentence.

Node.js provides both and asynchronous methods for file

Submit Answer »

❮ Previous Sign in to track progress Next ❯

COLOR PICKER



PLUS SPACES

GET CERTIFIED FOR TEACHERS

FOR BUSINESS CONTACT US

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
AngularJS Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples

Get Certified

HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate

FORUM ABOUT ACADEMY

W3Schools is optimized for learning and training. Examples might be simplified to improve reading and learning.
Tutorials, references, and examples are constantly reviewed to avoid errors, but we cannot warrant full correctness
of all content. While using W3Schools, you agree to have read and accepted our terms of use, cookie and privacy policy.

Copyright 1999-2025 by Refsnes Data. All Rights Reserved. W3Schools is Powered by W3.CSS.