Node.js provides multiple methods for handling files, primarily through the built-in fs (file system) module. Here's an overview of common file-handling tasks and examples:

1. Reading a File

- **Using fs.readFile**: Reads the entire content of a file asynchronously.

javascript
Copy code

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

- **Using fs.createReadStream**: Reads a file in chunks, which is useful for large files to prevent memory overload.

javascript
Copy code

```
const fs = require('fs');
const readStream = fs.createReadStream('example.txt', 'utf8');
readStream.on('data', (chunk) => {
    console.log('Received chunk:', chunk);
});
readStream.on('end', () => {
    console.log('File read completed');
});
readStream.on('error', (err) => {
    console.error('Error reading file:', err);
});
```

2. Writing to a File

- **Using** `fs.writeFile`: Writes data to a file asynchronously. If the file does not exist, it will create it; if it exists, it will overwrite it.

javascript
Copy code

```
const fs = require('fs');
fs.writeFile('output.txt', 'Hello, Node.js!', (err) => {
    if (err) {
        console.error('Error writing to file:', err);
        return;
    }
    console.log('File written successfully');
});
```

- **Using** `fs.createWriteStream`: Writes data in chunks. This is efficient for large data writes or streams.

javascript
Copy code

```
const fs = require('fs');
const writeStream = fs.createWriteStream('output.txt');
writeStream.write('Hello, ');
writeStream.write('Node.js!\n');
writeStream.end();
writeStream.on('finish', () => {
    console.log('File written successfully');
});
writeStream.on('error', (err) => {
    console.error('Error writing to file:', err);
});
```

3. Appending Data to a File

- **Using** `fs.appendFile`: Adds data to the end of an existing file. If the file does not exist, it will create it.

javascript

Copy code

```
const fs = require('fs');
fs.appendFile('output.txt', 'This is an appended line.\n', (err) => {
    if (err) {
        console.error('Error appending to file:', err);
        return;
    }
    console.log('Data appended to file');
});
```

4. Deleting a File

- **Using** fs.unlink: Deletes a specified file.

javascript
Copy code

```
const fs = require('fs');
fs.unlink('output.txt', (err) => {
    if (err) {
        console.error('Error deleting file:', err);
        return;
    }
    console.log('File deleted successfully');
});
```

5. Renaming a File

- **Using** fs.rename: Renames or moves a file.

javascript
Copy code

```
const fs = require('fs');
fs.rename('output.txt', 'newOutput.txt', (err) => {
    if (err) {
        console.error('Error renaming file:', err);
        return;
    }
    console.log('File renamed successfully');
});
```

6. Checking File Existence

- **Using** `fs.existsSync`: Checks if a file exists synchronously.

javascript
Copy code

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

if (fs.existsSync('example.txt')) {
    console.log('File exists');
} else {
    console.log('File does not exist');
}
```

7. Opening a File Descriptor

- **Using** `fs.open`: Opens a file and returns a file descriptor for lower-level operations.

javascript
Copy code

```
fs.open('example.txt', 'r', (err, fd) => {
    if (err) {
        console.error('Error opening file:', err);
        return;
    }
    console.log('File descriptor:', fd);
    fs.close(fd, (err) => {
        if (err) console.error('Error closing file:', err);
    });
});
```

8. Reading a File Asynchronously in Chunks

This approach is useful when you want to read a file progressively, especially if the file size is large.

javascript
Copy code

```javascript
const fs = require('fs');
fs.open('example.txt', 'r', (err, fd) => {
    if (err) {
        console.error('Error opening file:', err);
        return;
    }
    const buffer = Buffer.alloc(1024); // Allocate a buffer of 1 KB
    fs.read(fd, buffer, 0, buffer.length, 0, (err, bytesRead, buffer) => {
        if (err) {
            console.error('Error reading file:', err);
            return;
        }
        console.log('Bytes read:', bytesRead);
        console.log('File content:', buffer.toString('utf8', 0, bytesRead));
        fs.close(fd, (err) => {
            if (err) console.error('Error closing file:', err);
        });
    });
});
```

Summary

Task	Method	Description
Read File	fs.readFile	Reads the file content asynchronously
Read in Chunks	fs.createReadStream	Reads large files in chunks
Write File	fs.writeFile	Writes data, creating or overwriting a file
Write in Chunks	fs.createWriteStream	Writes data in chunks
Append File	fs.appendFile	Adds data to the end of an existing file
Delete File	fs.unlink	Deletes the file

Rename File	fs.rename	Renames or moves the file
Check Existence	fs.existsSync	Checks if a file exists synchronously
Open File	fs.open	Opens a file for low-level operations

These examples demonstrate core file handling tasks in Node.js, making it easy to manage files in applications.