Node File I/O

What is File I/O?

- Applications store data in many ways (Memory for the short term, databases, servers, files etc. for long term)
- Files are the most reliable & easy long term storage
- Most every language has a way of writing and reading files

Node's `fs` Library

- It's a core Node module that deals with the **F** ile **S** ystem
- Allows us to do all of the file actions we know how to do manually: Create, open, move and remove files and folders
- By default, performs all file operations asynchronously, using Node style callbacks

Reading Files

```
// Require the `fs` library
const fs = require("fs");

// The first argument is the path to the file*
// The second (optional) argument is the file encoding type**
// The third argument is the callback
fs.readFile("./helloworld.txt", "utf-8", function(err, data) {
    if (err) {
        console.error("Couldn't read helloworld.txt");
        console.error(err);
        return;
    }

    console.log(data);
});
```

Like with* **`require`*, the paths can be relative or absolute*

***Providing this ensures we get text back, rather than a file buffer*

Reading Files (Buffer)

- If we don't provide an encoding type (2nd arg) we get a buffer in the callback instead of a string
- This is meant for binary / octet files, rather than text files
- These are things like images, movies, music etc.
- You can read more about Buffers here

```
// Read in a jpeg image
rs.readFile("./file.jpeg", function(err, data) {
    console.log(data); // <Buffer 68 65 6c 6c 6f ...>
});
```

Writing Files

- If we want to store data, we'll need to write it to a new file
- The callback comes with no data, only potentially an error

```
const fs = require("fs");
const myObj = { property: "value" };

// Write this object to a file as JSON
fs.writeFile("./my-obj.json", JSON.stringify(myObj), function(err) {
    if (err) {
        console.error("Oh no, couldn't save my-obj.json!");
        console.error(err);
        return;
    }

    console.log("Saved object to my-obj.json, nice work");
});
```

Removing Files

- Removed files don't go in the trash can, *they're gone for good*
- This callback also doesn't receive any extra data, just an error

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

fs.unlink("./badfile.log", function(err) {
    if (err) {
        console.error("Couldn't remove badfile.log");
        console.error(err);
        return;
    }

    console.log("We rid ourselves of that nasty badfile.log, huzzah!");
});
```

Getting File Info

- Sometimes we want meta-information about files
- Things like size, last edited, creation date etc.
- All available information can be found here

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

fs.stat("./puppy.jpg", function(err, stats) {
    if (err) {
        console.error("Unable to get file stats");
        console.error(err);
        return;
    }

    console.log("File is" + stats.size + " bytes big");
    console.log("File was last edited " + stats.mtime);
});
```

Directory Operations

- Mostly same behavior as files operations
- Familiar naming from the terminal, `mkdir` and `rmdir`

```
const fs = require("fs");
const path = "/path/to/newfolder";

// Create a new directory
fs.mkdir(path, function(err) {
    console.log("Now you see me...");

    // Then remove it for kicks
    fs.rmdir(path, function(err) {
        console.log("And now you don't!");
    });
});
```

Possible Errors with File Handling

- Trying to read a file that does not exist will cause an error

```
js
fs.readFile("/does/not/exist.txt", function(err, data) {
// "Error: ENOENT: no such file or directory, open '/does/not/exist.txt'"
// ENOENT is C shorthand for "Error No ENTry"
console.log(err);
});
```

- Trying to read or write to a file without proper permissions will also cause an error

```
js
fs.writeFile("./helloworld.txt", "overwriting hello world!", function(err) {
// "Error: EACCES: permission denied, open './helloworld.txt'"
// Some files can be marked as read-only, unchangeable by programs
console.log(err);
});
```

Reading All Files in a Folder

- Sometimes we want to do something to multiple files in a directory
- But we don't want to pay the "cost" of opening each file, we just want to get their names

```
const fs = require("fs");
fs.readdir("./relative/directory", function(err, files){
    if (err) {
        console.error("Unable to read directory");
        console.error(err);
        return;
    }

    if (!files.length) {
        console.error("Directory is empty!");
        return;
    }

    // Print each file's name
    console.log("All files in directory:");
    for (let i = 0; i < files.length; i++) {
        console.log(files[i]);
    }
});
```

Synchronous File I/O

- As noted, by default, all of these functions are asynchronous
- However, many come with a *Sync( ) version that runs synchronously
- Instead of an "errback", you have to use try / catch for errors
- This may look convenient, but it can be *significantly* slower than async
- So for upcoming file I/O assignments, use the async versions of functions to practice

```
// Adds your signature to a file
function signaturize(path, name) {
    const content = fs.readFileSync(path, "utf-8");
    fs.writeFileSync(path, content + "\n\nWritten by " + name);
}
```

Exercise: File Sorter

- Make a function that, given a path to a directory will:
    - Get a list of all the files in the directory
    - Print them out in alphabetical order
- Once you have them printed alphabetically, we'll want to:
    - Get the stats of all of the files in the directory
    - Print them out by size order