Lesson 3: File system module, query String - https://nodejs.org/api/fs.html, https://nodejs.org/api/querystring.html

he fs module provides a lot of very useful functionality to access and interact with the file system.

There is no need to install it. Being part of the Node.js core, it can be used by simply requiring it:

const fs = require('fs')

Once you do so, you have access to all its methods, which include:

- fs.access(): check if the file exists and Node.js can access it with its permissions
- fs.appendFile(): append data to a file. If the file does not exist, it's created
- fs.chmod(): change the permissions of a file specified by the filename passed. Related: fs.lchmod(), fs.fchmod()
- fs.chown(): change the owner and group of a file specified by the filename passed. Related: fs.fchown(), fs.lchown()
- fs.close(): close a file descriptor
- fs.copyFile(): copies a file
- fs.createReadStream(): create a readable file stream
- fs.createWriteStream(): create a writable file stream
- fs.link(): create a new hard link to a file
- fs.mkdir(): create a new folder
- fs.mkdtemp(): create a temporary directory
- fs.open(): set the file mode
- fs.readdir(): read the contents of a directory
- fs.readFile(): read the content of a file. Related: fs.read()
- fs.readlink(): read the value of a symbolic link
- fs.realpath(): resolve relative file path pointers (., ..) to the full path
- fs.rename(): rename a file or folder
- fs.rmdir(): remove a folder
- fs.stat(): returns the status of the file identified by the filename passed. Related: fs.fstat(), fs.lstat()
- fs.symlink(): create a new symbolic link to a file
- fs.truncate(): truncate to the specified length the file identified by the filename passed. Related: fs.ftruncate()
- fs.unlink(): remove a file or a symbolic link
- fs.unwatchFile(): stop watching for changes on a file
- fs.utimes(): change the timestamp of the file identified by the filename passed. Related: fs.futimes()
- fs.watchFile(): start watching for changes on a file. Related: fs.watch()
- fs.writeFile(): write data to a file. Related: fs.write()

One peculiar thing about the fs module is that all the methods are asynchronous by default, but they can also work synchronously by appending Sync.

For example:

- fs.rename()
- fs.renameSync()
- fs.write()
- fs.writeSync()

This makes a huge difference in your application flow.

Common use for the File System module:

- Open and Read files
- Write to a file
- Update files
- Delete files
- Truncate files

Open a File

Syntax

Following is the syntax of the method to open a file in asynchronous mode −

```
fs.open(path, flags[, mode], callback)
```

Parameters

Here is the description of the parameters used −

- **path** − This is the string having file name including path.
- **flags** − Flags indicate the behavior of the file to be opened. All possible values have been mentioned below.
- **mode** − It sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.
- **callback** − This is the callback function which gets two arguments (err, fd).

Flags

Flags for read/write operations are −

Sr.No.	Flag & Description
1	**r** Open file for reading. An exception occurs if the file does not exist.
2	**r+** Open file for reading and writing. An exception occurs if the file does not exist.
3	**rs** Open file for reading in synchronous mode.
4	**rs+** Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.
5	**w** Open file for writing. The file is created (if it does not exist) or truncated (if it exists).

6	**wx** Like 'w' but fails if the path exists.
7	**w+** Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).
8	**wx+** Like 'w+' but fails if path exists.
9	**a** Open file for appending. The file is created if it does not exist.
10	**ax** Like 'a' but fails if the path exists.
11	**a+** Open file for reading and appending. The file is created if it does not exist.
12	**ax+** Like 'a+' but fails if the the path exists.

Writing a File

Syntax

Following is the syntax of one of the methods to write into a file −

```
fs.writeFile(filename, data[, options], callback)
```

This method will over-write the file if the file already exists. If you want to write into an existing file then you should use another method available.

Parameters

Here is the description of the parameters used −

- **path** − This is the string having the file name including path.
- **data** − This is the String or Buffer to be written into the file.
- **options** − The third parameter is an object which will hold {encoding, mode, flag}. By default. encoding is utf8, mode is octal value 0666. and flag is 'w'
- **callback** − This is the callback function which gets a single parameter err that returns an error in case of any writing error.

Reading a File

Syntax

Following is the syntax of one of the methods to read from a file −

```
fs.read(fd, buffer, offset, length, position, callback)
```

This method will use file descriptor to read the file. If you want to read the file directly using the file name, then you should use another method available.

Parameters

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by fs.open().
- **buffer** − This is the buffer that the data will be written to.
- **offset** − This is the offset in the buffer to start writing at.
- **length** − This is an integer specifying the number of bytes to read.
- **position** − This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.
- **callback** − This is the callback function which gets the three arguments, (err, bytesRead, buffer).

Closing a File

Syntax

Following is the syntax to close an opened file −

```
fs.close(fd, callback)
```

Parameters

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by file fs.open() method.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

Truncate a File

Syntax

Following is the syntax of the method to truncate an opened file −

```
fs.ftruncate(fd, len, callback)
```

Parameters

Here is the description of the parameters used −

- **fd** − This is the file descriptor returned by fs.open().
- **len** − This is the length of the file after which the file will be truncated.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

Delete a File

Syntax

Following is the syntax of the method to delete a file −

```
fs.unlink(path, callback)
```

Parameters

Here is the description of the parameters used −

- **path** − This is the file name including path.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

URL Module:

As nodejs.org suggests:

The URL module provides utilities for URL resolution and parsing. It can be accessed using:

```
1. var url = require('url');
```

Url module is one of the core modules that comes with node.js, which is used to parse the URL and its other properties.

By using URL module, it provides us with so many properties to work with.

These all are listed below:

Property	Description
.href	Provides us the complete url string
.host	Gives us host name and port number
.hostname	Hostname in the url
.path	Gives us path name of the url
.pathname	Provides host name , port and pathname
.port	Gives us port number specified in url
.auth	Authorization part of url
.protocol	Protocol used for the request
.search	Returns query string attached with url

As you can see in the above screen, there are various properties used for URL module.

Code Demonstrated in Class:

```
//////////////////////////////////////////////////////////////////
Opening a file:
//////////////////////////////////////////////////////////////////
var fs = require("fs");
```

```
// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, data) {
   if (err) {
      return console.error(err);
   }
   console.log(data);
   console.log("File opened successfully!");
});


////////////////////////////////////////////////////////////////////////
Reading from file:
////////////////////////////////////////////////////////////////////////
var fs = require('fs');

fs.readFile('input.txt', function (err, data) {
                    if (err) throw err;

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

var fs = require('fs');

var data = fs.readFileSync('data.txt', 'utf8');
console.log(data);

////////////////////////////////////////////////////////////////////////////////
Writing a file:
////////////////////////////////////////////////////////////////////////
var fs = require("fs");

console.log("Going to write into existing file");
fs.writeFile('input.txt', 'PES University!', function(err) {
   if (err) {
      return console.error(err);
   }

   console.log("Data written successfully!");
   console.log("Let's read newly written data");

   fs.readFile('data.txt', function (err, data) {
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " + data.toString());
   });
});

////////////////////////////////////////////////////////////////////////
Reading from a file
////////////////////////////////////////////////////////////////////////

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

```
fs.open('data.txt', 'r', function (err, fd) {

                        if (err) {
                        return console.error(err);
    }

                        var buffr = new Buffer(1024);
    console.log("File opened successfully!");
   console.log("Going to truncate the file after 3 bytes");

   // Truncate the opened file.
   fs.ftruncate(fd, 3, function(err) {
      if (err) {
         console.log(err);
      }
      console.log("File truncated successfully.");
      console.log("Going to read the same file");

    fs.read(fd, buffr, 0, buffr.length, 0, function (err, bytes) {

                        if (err) throw err;

                        // Print only read bytes to avoid junk.
                        if (bytes > 0) {
            console.log(buffr.slice(0, bytes).toString());
        }

                        // Close the opened file.
        fs.close(fd, function (err) {
                        if (err) throw err;
        });
    });
});

////////////////////////////////////////////////////////////////////////////
Delete a file
////////////////////////////////////////////////////////////////////////////
var fs = require('fs');

fs.unlink('data.txt', function () {

    console.log('write operation complete.');

});
```