CSC336 – Web Technologies

File System

Common use for the File System module:

1. Read files
2. Create files
3. Update files
4. Delete files
5. Rename files

FS Model

The Node File System (fs) module can be imported using the following syntax −

var fs = require("fs")

Synchronous vs Asynchronous

Every method in the fs module has synchronous as well as asynchronous forms.

Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.

Example

```
var fs = require("fs");
  // Asynchronous read
 fs.readFile('input.txt', function (err, data)
 { if (err) { return console.error(err); }
 console.log("Asynchronous read: " + data.toString());
});
// Synchronous read
 var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());
console.log("Program Ended");
```

Important method of fs module

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(src path, dst path, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.

Important method of fs module

fs.utimes(path, atime, mtime, callback)
Changes the timestamp of the file.

fs.exists(path, callback)
Determines whether the specified file exists or not.

fs.access(path[, mode], callback)
Tests a user's permissions for the specified file.

fs.appendFile(file, data[, options], callback)
Appends new content to the existing file.

Opening Files

1) Open a File

Syntax : open a file in asynchronous mode –
fs.open (path, flags[, mode], callback)

Parameters-

***1.path** – This is the string having file name including path.*
***2.flags** – Flags indicate the behavior of the file to be opened. All possible values have been mentioned below.*
***3.mode** – It sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.*

4.callback – *This is the callback function which gets two arguments (err, fd).*
Flags
r *: Open file for reading. An exception occurs if the file does not exist.*
r+ *: Open file for reading and writing. An exception occurs if the file does not exist.*
rs *: Open file for reading in synchronous mode.*
rs+: *Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.*
w *: Open file for writing. The file is created (if it does not exist) or truncated (if it exists).*
wx *: Like 'w' but fails if the path exists.*
w+ *: Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).*
wx+ *: Like 'w+' but fails if path exists.*
a *: Open file for appending. The file is created if it does not exist.*
ax *: Like 'a' but fails if the path exists.*
a+ *: Open file for reading and appending. The file is created if it does not exist.*
ax+ *: Like 'a+' but fails if the the path exists.*

Example

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

Reading Files

2) Reading a File

Syntax –

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

uses file descriptor to read the file.

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).*

Example

```
var fs = require("fs");
var buf = new Buffer(1024);
console.log("Going to open an existing file");
 fs.open('input.txt', 'r+', function(err, fd)
  { if (err) { return console.error(err); }
 console.log("File opened successfully!");
 console.log("Going to read the file");
  fs.read (fd, buf, 0, buf.length, 0, function(err, bytes)
  { if (err){ console.log(err); }
  console.log(bytes + " bytes read");
  // Print only read bytes to avoid junk.
  if(bytes > 0)
{ console.log(buf.slice(0,bytes).toString());
  } });
  })
```

Use fs.readFile() method to read the physical file asynchronously.

Syntax:

fs.readFile(fileName [,options], callback)

Parameter Description:

filename: Full path and name of the file as a string.

options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when readFile operation completes.

Reading Files : example

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

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

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

run

The following is a sample TextFile.txt file.

TextFile.txt

```
This is test file to test fs module of Node.js
```

Now, run the above example and see the result as shown below.

```
C:\> node server.js
This is test file to test fs module of Node.jsC:\> node server.js
This is test file to test fs module of Node.js
```

Use fs.readFileSync() method to read file synchronously as shown below.

Example: Reading File Synchronously

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

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

Writing a File

Syntax :

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

filename *– 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.*

Writing in File

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

fs.writeFile('mynewfile3.txt', 'This is my text',
  function (err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

Append file

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

fs.appendFile('mynewfile1.txt', 'HelloContent!',
  function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Other File Operations

Delete File

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

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename

```
var fs = require('fs');
fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

Get File Information

Syntax : Following is the syntax of the method to get the information about a file −

fs.stat(path, callback)

path − This is the string having file name including path.
callback − This is the callback function which gets two arguments (err, stats) where stats is an object of fs.Stats type which is printed below in the example.

Get File Information

Method & Description

stats.isFile()	*: Returns true if file type of a simple file.*
stats.isDirectory()	*: Returns true if file type of a directory.*
stats.isBlockDevice()	*: Returns true if file type of a block device.*
stats.isCharacterDevice()	*: Returns true if file type of a character device.*
stats.isSymbolicLink()	*: Returns true if file type of a symbolic link.*
stats.isFIFO()	*: Returns true if file type of a FIFO.*
stats.isSocket()	*: Returns true if file type of a socket.*

Get File Information

Example

```
var fs = require("fs");
  console.log("Going to get file info!");
  fs.stat('input.txt', function (err, stats)
  { if (err)
          { return console.error(err); }
console.log(stats);
console.log("Got file info successfully!");
        // Check file type
console.log("isFile ? " + stats.isFile());
console.log("isDirectory ? " + stats.isDirectory()); });
```

Question Bank—
Assignment-
1.Explain syntax, run example & display output of each operation on file
1) Readfile()
2) readFileSync()
3) Read()
4) Open()
5) Truncate()
6) ftruncate()
7) append file
8) Rename()
9) Write file
10) Writefilesync()
11) Write()
12) Chmod
13) Stat()
14) Fstat()
15) Link
16) symlink
17)Readlink
18)Unlink()
19)chown()
20)lchown()
21)fchown()

Any Questions?