Node.js - File System

Node implements File I/O using simple wrappers around standard POSIX functions. 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. It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

Example

Create a text file named **input.txt** with the following content −

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Let us create a js file named **main.js** with the following code −

```
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");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Synchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!

Program Ended
Asynchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

The following sections in this chapter provide a set of good examples on major File I/O methods.

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.

Example

Let us create a js file named **main.js** having the following code to open a file input.txt for reading and writing.

```
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!");
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to open file!
File opened successfully!
```

Get File Information

Syntax

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

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

Parameters

Here is the description of the parameters used −

- **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.

Apart from the important attributes which are printed below in the example, there are several useful methods available in **fs.Stats** class which can be used to check file type. These methods are given in the following table.

Sr.No.	Method & Description
1	**stats.isFile()** Returns true if file type of a simple file.
2	**stats.isDirectory()** Returns true if file type of a directory.
3	**stats.isBlockDevice()** Returns true if file type of a block device.
4	**stats.isCharacterDevice()** Returns true if file type of a character device.
5	**stats.isSymbolicLink()** Returns true if file type of a symbolic link.
6	**stats.isFIFO()** Returns true if file type of a FIFO.
7	**stats.isSocket()** Returns true if file type of asocket.

Example

Let us create a js file named **main.js** with the following code −

```
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());
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to get file info!
{
   dev: 1792,
   mode: 33188,
   nlink: 1,
   uid: 48,
   gid: 48,
   rdev: 0,
   blksize: 4096,
   ino: 4318127,
   size: 97,
   blocks: 8,
   atime: Sun Mar 22 2015 13:40:00 GMT-0500 (CDT),
   mtime: Sun Mar 22 2015 13:40:57 GMT-0500 (CDT),
   ctime: Sun Mar 22 2015 13:40:57 GMT-0500 (CDT)
}
Got file info successfully!
isFile ? true
isDirectory ? false
```

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.

Example

Let us create a js file named **main.js** having the following code −

Live Demo

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

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

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

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

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to write into existing file
Data written successfully!
Let's read newly written data
Asynchronous read: Simply Easy Learning!
```

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).

Example

Let us create a js file named **main.js** with the following code −

```
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());
      }
   });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to open an existing file
File opened successfully!
Going to read the file
97 bytes read
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

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.

Example

Let us create a js file named **main.js** having the following code −

```
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);
      }

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

      // Close the opened file.
      fs.close(fd, function(err) {
         if (err) {
            console.log(err);
         }
         console.log("File closed successfully.");
      });
   });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to open an existing file
File opened successfully!
Going to read the file
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!

File closed successfully.
```

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.

Example

Let us create a js file named **main.js** having the following code −

```
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 truncate the file after 10 bytes");
   
   // Truncate the opened file.
   fs.ftruncate(fd, 10, function(err) {
      if (err) {
         console.log(err);
      }
      console.log("File truncated successfully.");
      console.log("Going to read the same file");
      
      fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
         if (err) {
            console.log(err);
         }

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

         // Close the opened file.
         fs.close(fd, function(err) {
            if (err) {
               console.log(err);
            }
            console.log("File closed successfully.");
         });
      });
   });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to open an existing file
File opened successfully!
Going to truncate the file after 10 bytes
File truncated successfully.
Going to read the same file
Tutorials
File closed successfully.
```

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.

Example

Let us create a js file named **main.js** having the following code −

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

console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to delete an existing file
File deleted successfully!
```

Create a Directory

Syntax

Following is the syntax of the method to create a directory −

```
fs.mkdir(path[, mode], callback)
```

Parameters

Here is the description of the parameters used −

- **path** − This is the directory name including path.
- **mode** − This is the directory permission to be set. Defaults to 0777.
- **callback** − This is the callback function No arguments other than a possible exception are given to the completion callback.

Example

Let us create a js file named **main.js** having the following code −

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

console.log("Going to create directory /tmp/test");
fs.mkdir('/tmp/test',function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Directory created successfully!");
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to create directory /tmp/test
Directory created successfully!
```

Read a Directory

Syntax

Following is the syntax of the method to read a directory −

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

Parameters

Here is the description of the parameters used −

- **path** − This is the directory name including path.
- **callback** − This is the callback function which gets two arguments (err, files) where files is an array of the names of the files in the directory excluding '.' and '..'.

Example

Let us create a js file named **main.js** having the following code −

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

console.log("Going to read directory /tmp");
fs.readdir("/tmp/",function(err, files) {
   if (err) {
      return console.error(err);
   }
   files.forEach( function (file) {
      console.log( file );
   });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test
test.txt
```

Remove a Directory

Syntax

Following is the syntax of the method to remove a directory −

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

Parameters

Here is the description of the parameters used −

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

Example

Let us create a js file named **main.js** having the following code −

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

console.log("Going to delete directory /tmp/test");
fs.rmdir("/tmp/test",function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Going to read directory /tmp");

   fs.readdir("/tmp/",function(err, files) {
      if (err) {
         return console.error(err);
      }
      files.forEach( function (file) {
         console.log( file );
      });
   });
});
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test.txt
```

Methods Reference

> Following is a reference of File System module available in Node.js. For more detail you can refer to the official documentation.