FILE SYSTEMS IN NODEJS

To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js gives the functionality of file I/O by providing wrappers around the standard POSIX functions. All file system operations can have synchronous and asynchronous forms depending upon user requirements.

To use the file system module, we use the require method:

Var fs=require(‘fs’);

Common use for File System module:

* Read Files
* Write Files
* Append Files
* Close Files
* Delete Files

To get started with the functionality of File System modules in NodeJs, we need to understand the concept of asynchronous and Synchronous functions and then we can implement different Synchronous and Asynchronous functions.

* Synchronous Methods: Also called Blocking function. Blocks execution of program until file operation is performed. These functions have File Descriptor as last argument. File descriptors refer to opened files. Some Synchronous functions are: fs.readFileSync(), fs.renameSync(),fs.writeSync(),fs.writeFileSync()
* Asynchronous Methods: Do not block execution of the program and command is executed after previous command even if previous command has not computed the result. The previous command runs in the background and loads the result once it has finished processing. They take a callback function as the last parameter. Preferred over Synchronous methods as they do not block. Some Asynchronous functions are: fs.readFile(), fs.writeFile().

Nodejs fs.readFile method (Asynchronous): The fs.readFile() method is an inbuilt method which is used to read the file. This method reads the entire file into buffer. To load the fs module, we use require() method. For example: var fs = require(‘fs’);

Syntax:

fs.readFile( filename, encoding, callback\_function )

Parameters: The method accepts three parameters as mentioned above and described below:

filename: It holds the name of the file to read or the entire path if stored at another location.

encoding: It holds the encoding of file. Its default value is ‘utf8’.

callback\_function: It is a callback function that is called after reading of file. It takes two parameters:

err: If any error occurred.

data: Contents of the file.

Node.js fs.writeFile method (Asynchronous): The fs.writeFile() method is used to asynchronously write the specified data to a file. By default, the file would be replaced if it exists. The ‘options’ parameter can be used to modify the functionality of the method.

Syntax: fs.writeFile( file, data, options, callback)

Parameters: This method accepts four parameters as mentioned above and described below:

file: It is a string, Buffer, URL, or file description integer that denotes the path of the file where it must be written. Using a file descriptor will make it behave like fs.write() method.

data: It is a string, Buffer, TypedArray or DataView that will be written to the file.

options: It is a string or object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

encoding: It is a string value that specifies the encoding of the file. The default value is ‘utf8’.

mode: It is an integer value that specifies the file mode. The default value is 0o666.

flag: It is a string value that specifies the flag used while writing to the file. The default value is ‘w’.

callback: It is the function that would be called when the method is executed.

err: It is an error that would be thrown if the operation fails.

Node.js appendFile() method (Asynchronous): The fs.appendFile() method is used to asynchronously append the given data to a file. A new file is created if it does not exist. The options parameter can be used to modify the behavior of the operation.

Syntax: fs.appendFile( path, data [, options], callback)

Parameters: This method accepts four parameters as mentioned above and described below:

path: It is a String, Buffer, URL, or number that denotes the source filename or file descriptor that will be appended to.

data: It is a String or Buffer that denotes the data that must be appended.

options: It is a string or an object that can be used to specify optional parameters that will affect the output. It has three optional parameters:

encoding: It is a string which specifies the encoding of the file. The default value is ‘utf8’.

mode: It is an integer which specifies the file mode. The default value is ‘0o666’.

flag: It is a string which specifies the flag used while appending to the file. The default value is ‘a.’

callback: It is a function that would be called when the method is executed.

err: It is an error that would be thrown if the method fails.

Node.js fs.chmod() method: The fs.chmod() method is used to change the permissions of a given path. These permissions can be specified using string constants or octal numbers that correspond to their respective file modes.

Syntax: fs.chmod( path, mode, callback)

Parameters: This method accepts three parameters as mentioned above and described below:

path: It is a string, Buffer or URL that denotes the path of the file of which the permission has to be changed.

mode: It is string or octal integer constant that denotes the permission to be granted. The logical OR operator can be used to separate multiple permissions.

callback: It is a function that would be called when the method is executed.

err: It is an error that would be thrown if the method fails.

Node.js fs.mkdir() method: The fs.mkdir() method i Node.js is used to create a directory asynchronously.

Syntax: fs.mkdir(path, mode, callback)

Parameters: This method accepts three parameters as mentioned above and described below:

path: This parameter holds the path of the directory must be created.

mode: This parameter holds the recursive boolean value. The mode option is used to set the directory permission, by default it is 0777.

callback: This parameter holds the callback function that contains error. The recursive option if set to true will not give an error message if the directory to be created already exists.

To use the synchronous form of an Asynchronous function , add a ‘Sync’ to the end of method name.

For example: fs.readFile(Asynchronous) --> fs.readFileSync(Synchronous);