Botlhale Village
Working together for ICT innovation and growth in Africa

BELGIUM CAMPUS iTversity
It's the way we're wired

WPR252

BELGIUM CAMPUS
iTversity
It's the way we're wired

Node.Js Core Modules

Modules

- Modules are reusable functionality. It can be a function, can be a class, can be an object or even simple variables.
- Consuming core modules is very similar to consuming file-based modules that you write yourself. You still use the require function.
- **NOTE**: New version of Node.js supports es6 notation `import` and `export` statements.

Path Module

- The path module exports functions that provide useful string transformations common when working with the file system.
- The key motivation for using the path module is to remove inconsistencies in handling file system paths.
- path.normalize(str)
- This function fixes up slashes to be OS specific, takes care of . and .. in the path, and also removes duplicate slashes.

```
let path=require('path');
console.log(path.normalize('/folder//fold/...'))
// Also removes duplicate '//' slashes
```

- path.join([str1], [str2], …)

This function joins any number of paths together, taking into account the operating system.

```
console.log(path.join('documents','/wpr252','modules'))
```

dirname, basename, and extname

- These functions are three of the most useful functions in the path module.
- path.dirname gives you the directory portion of a specific path string (OS independent), and path.basename gives you the name of the file. path.extname gives you the file extension

```
let compeletepath='/documents/nodejs/hello.html';
console.log(path.dirname(compeletepath)); //documents/nodejs
console.log(path.basename(compeletepath));/// hello.html
console.log(path.extname(compeletepath));//.html
```

fs Module

- The fs module provides access physical file system.
- the fs module has functions for renaming files, deleting files, reading files, and writing to files.
- The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Reading File

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

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.

```
fs.readFile('data.txt', function(err, data) {
 if (err) throw err;
 console.log(data.toString()); });
```

Writing File

Use fs.writeFile() method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

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

Parameter Description:

- filename: Full path and name of the file as a string.
- Data: The content to be written in a file.
- options: The options parameter can be an object or string which can include encoding, mode 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 write operation completes.

```
•fs.writeFile('data.txt', 'Hello world!',
function (err) {
 if(err) { throw err; }
 console.log('It is saved!');
});
```

use fs.appendFile() method to append the content to an existing file. Same syntax as write file code only change the fs.writefile to fs.appendFile

Delete File

Use fs.unlink() method to delete an existing file.

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

```
fs.unlink('datat.txt', (err) => {
    if (err) console.log('the file is not availabale in the given directory');
    console.log('file deleted');
})
```

Create a Directory

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

```
fs.mkdir('./new folder', function (err) {
    if (err) console.log(err);
    console.log('folder successfully created');
})
```

Delete File

Read a Directory

fs.readdir(path, callback)

Delete a Directory

fs.rmdir(path, callback)

Streaming Data

- *Streaming data* is a phrase that means an application processes the data while it's still receiving it.
- This feature is useful for extra large datasets such as video or database migrations.
- there are four types of streams –

❖**Readable** – Stream which is used for read operation.

❖**Writable** – Stream which is used for write operation.

❖**Duplex** – Stream which can be used for both read and write operation.

❖**Transform** – A type of duplex stream where the output is computed based on input.

Streaming Data

- Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times. For example, some of the commonly used events are –

➢ **data** – This event is fired when there is data is available to read.

➢ **end** – This event is fired when there is no more data to read.

➢ **error** – This event is fired when there is any error receiving or writing data.

➢ **finish** – This event is fired when all the data has been flushed to underlying system.

```
var data = 'Streams are the objects that facilitate you to read data from a source and
write data to a destination. There are four types of streams in Node.js.';
// Create a writable stream
var writerStream = fs.createWriteStream('stream.txt');
// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');
// Mark the end of file
writerStream.end();
---------------------------------------------------------------------------------
//read stream
var data = '';
// Create a readable stream
var readerStream = fs.createReadStream('stream.txt');
// Set the encoding to be utf8.
readerStream.setEncoding('UTF8');
// Handle stream events --> data, end, and error
readerStream.on('data', function(chunk) {
 data += chunk;
});
readerStream.on('end',function() {
 console.log(data);
});
```

Piping the Streams

- Piping is a mechanism where we provide the output of one stream as the input to another stream.
- It is normally used to get data from one stream and to pass the output of that stream to another stream. There is no limit on piping operations.

```
//pipe a stream
// Create a readable stream
var readerStream = fs.createReadStream('stream.txt');
// Create a writable stream
var writerStream =
fs.createWriteStream('streamoutput.txt');
// Pipe the read and write operations
readerStream.pipe(writerStream);
• ----------------------------------------
///chain stream
var zlib = require('zlib');
// Compress the file input.txt to input.txt.gz
fs.createReadStream('stream.txt').pipe(zlib.createGzip())
    .pipe(fs.createWriteStream('stream.txt.gz'));
```

OS Module

The os module provides a few basic (but vital) operating-system related utility functions and properties.

```
let os = require('os');
console.log(os.platform());
console.log(os.hostname());
console.log(os.release());
//current system memory usage
var gigaByte = 1 / (Math.pow(1024, 3));
console.log('Total Memory', os.totalmem() * gigaByte, 'GBs');
console.log('Available Memory', os.freemem() * gigaByte, 'GBs');
console.log('Percent consumed', 100 * (1 - os.freemem() /
os.totalmem()));
console.log('This machine has', os.cpus().length, 'CPUs');
```