Node.js - Streams

What are Streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js, 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.

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.

This tutorial provides a basic understanding of the commonly used operations on Streams.

Reading from a Stream

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

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

Create a js file named main.js with the following code −

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

// Create a readable stream
var readerStream = fs.createReadStream('input.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);
});

readerStream.on('error', function(err) {
   console.log(err.stack);
});

console.log("Program Ended");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

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

Writing to a Stream

Create a js file named main.js with the following code −

Live Demo

```
var fs = require("fs");
var data = 'Simply Easy Learning';

// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');

// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');

// Mark the end of file
writerStream.end();

// Handle stream events --> finish, and error
writerStream.on('finish', function() {
   console.log("Write completed.");
});
```

```
writerStream.on('error', function(err) {
   console.log(err.stack);
});

console.log("Program Ended");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Program Ended
Write completed.
```

Now open output.txt created in your current directory; it should contain the following −

```
Simply Easy Learning
```

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. Now we'll show a piping example for reading from one file and writing it to another file.

Create a js file named main.js with the following code −

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

// Create a readable stream
var readerStream = fs.createReadStream('input.txt');

// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');

// Pipe the read and write operations
// read input.txt and write data to output.txt
readerStream.pipe(writerStream);

console.log("Program Ended");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Program Ended
```

Open output.txt created in your current directory; it should contain the following −

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

Chaining the Streams

Chaining is a mechanism to connect the output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations. Now we'll use piping and chaining to first compress a file and then decompress the same.

Create a js file named main.js with the following code −

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

// Compress the file input.txt to input.txt.gz
fs.createReadStream('input.txt')
   .pipe(zlib.createGzip())
   .pipe(fs.createWriteStream('input.txt.gz'));

console.log("File Compressed.");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
File Compressed.
```

You will find that input.txt has been compressed and it created a file input.txt.gz in the current directory. Now let's try to decompress the same file using the following code −

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

// Decompress the file input.txt.gz to input.txt
fs.createReadStream('input.txt.gz')
   .pipe(zlib.createGunzip())
   .pipe(fs.createWriteStream('input.txt'));
```

```
console.log("File Decompressed.");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
File Decompressed.
```

Revision #1
Created 16 December 2022 10:26:22 by Admin
Updated 16 December 2022 10:26:56 by Admin