



*Title:* **Assignment 2 Part 1: Introduction to Node.js**

*Submitted by:* **Colm Doyle**

*Student Number:* **C15387731**

*Date Submitted:* **9 October 2018**

*Submitted for*

*Module:* **COMP4000 Internet of Things**

*Programme :* **DT021A/4**

*Lecturers:* **Damon Berry**

**Frank Duignan**

**Michael Core**

**Paula Kelly**

## Contents

Exercise 1 - Named and Anonymous Callback Functions .....	3
Exercise 2 - Synchronous Vs Asynchronous Functions .....	5
Exercise 3 - Events and Event Handling in NodeJS.....	7
Exercise 4 - fs and http modules .....	9
Exercise 5 - Using the Node.js API.....	12
Bibliography .....	14
Figure 1: Web browser view of "index.html" .....	3
Figure 2: Output of Sample Code "main1.js" .....	4
Figure 3: Output when "main.js" is run .....	4
Figure 4: Output of "main.js" when the file "index.html" is deleted.....	4
Figure 5: The output of the 'sync.js' and a folder showing the copied version of the file.....	5
Figure 6: The output of the 'async.js' and a folder showing the copied version of the file.....	6
Figure 7: Output at the server end .....	11
Figure 8: Result after making a request to the Node.js server .....	11
Figure 9: Output result when 'time.js' is executed .....	13
Table 1: "main1.js" Sample Code using named function.....	3
Table 2: 'Hello World' Code used in index.html.....	3
Table 3: Rewritten Sample Code "main.js" .....	4
Table 4: File "sync.js" synchronously copies the file index.html .....	5
Table 5: File "async.js" asynchronously copies the file index.html .....	5
Table 6: Event handling code .....	7
Table 7: Edited code for new EventEmitter object and handler.....	8
Table 8: Web Server Code.....	9
Table 9: 'index.html' code used to test the 'server.js' code .....	9
Table 10: Server.js code used to create a Web Server .....	10
Table 11: setTimeout Function .....	12
Table 12: The code used for "time.js" .....	13

## Exercise 1 - Named and Anonymous Callback Functions

Create a file named “index.html” and save it in the same folder as the sample code shown below in Table 1.

```
// Exercise 1
// main.js
// Named Function

var fs = require('fs');

fs.readFile("index.html", writeFileContentToConsole);

function writeFileContentToConsole(err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log(data.toString());
  }
}
```

Table 1: "main1.js" Sample Code using named function

The file “index.html” was created and the ‘hello world’ code shown below in Table 2 was written in it using html coding notation.

```
// Exercise 1
// index.html

<!DOCTYPE html>
<html>
  <head>
    <title>Basic Web Page</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

Table 2: 'Hello World' Code used in index.html

Figure 1 below shows how the file “index.html” looks in chrome.

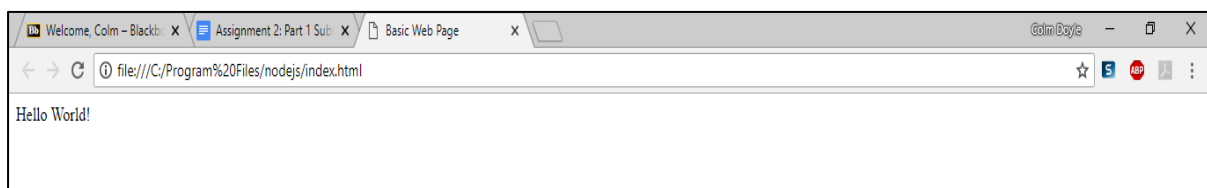


Figure 1: Web browser view of "index.html"

```
c:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\main1.js"
<!DOCTYPE html>
<html>
  <head>
    <title>Basic Web Page</title>
  </head>
  <body>
Hello World!
  </body>
</html>
```

Figure 2: Output of Sample Code "main1.js"

The code from Table 1 was run using Windows Command prompt and output is shown above in Figure 2.

Using the sample code in Table 1 for a NodeJS "fs" module, rewrite the named function, writeFileContentToConsole, as an anonymous function and run the code.

```
// Exercise 1
// Anonymous Function

var fs = require('fs');

fs.readFile("index.html", function(err, data){
  if (err) return console.error(err);
  console.log(data.toString());
});
```

Table 3: Rewritten Sample Code "main.js"

The original code is rewritten and the function 'writeFileContentToConsole' is now an anonymous function instead of a named function. The code is then run and the output is shown below in Figure 3.

```
c:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\main.js"
<!DOCTYPE html>
<html>
  <head>
    <title>Basic Web Page</title>
  </head>
  <body>
Hello World!
  </body>
</html>
```

Figure 3: Output when "main.js" is run

Delete the "index.html" file, rerun the code. Record the error message that is generated here:

The file "index.html" is deleted and the code "main.js" is rerun, the output is shown below with the following error message.

```
c:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\main.js"
{ Error: ENOENT: no such file or directory, open 'c:\Program Files\nodejs\index.html'
  errno: -4058,
  code: 'ENOENT',
  syscall: 'open',
  path: 'c:\Program Files\nodejs\index.html' }
```

Figure 4: Output of "main.js" when the file "index.html" is deleted

## Exercise 2 - Synchronous Vs Asynchronous Functions

Using the NodeJS API write the code to make a copy of the index.html file from exercise 1. Use both the synchronous and asynchronous versions of the copy file function.

```
// Exercise 2
// Synchronous
// sync.js

var fs = require('fs');

fs.copyFileSync("index.html", "indexcopy.html");

console.log("index.html copied to indexcopysync.html");
```

Table 4: File “sync.js” synchronously copies the file index.html

The code in Table 4 above synchronously copies index.html to indexcopy.html. By default, indexcopy.html is overwritten if it already exists. “Node.js makes no guarantees about the atomicity of the copy operation. If an error occurs after the destination file has been opened for writing, Node.js will attempt to remove the destination.” [1]

Name	Date modified	Type	Size
async	08/10/2018 16:44	JavaScript File	1 KB
ex1.1	08/10/2018 16:44	JavaScript File	1 KB
ex1	08/10/2018 16:44	JavaScript File	1 KB
index	08/10/2018 16:44	Firefox HTML Doc...	1 KB
indexcopy	08/10/2018 16:44	Firefox HTML Doc...	1 KB
sync	08/10/2018 16:44	JavaScript File	1 KB
server	08/10/2018 16:44	JavaScript File	2 KB
time	08/10/2018 16:44	JavaScript File	1 KB

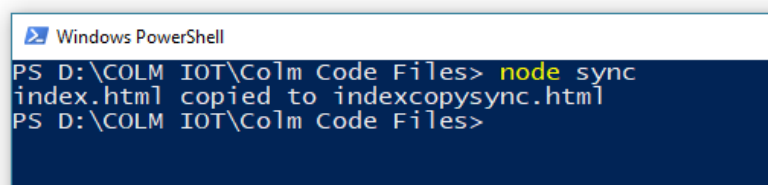


Figure 5: The output of the 'sync.js' and a folder showing the copied version of the file

```
// Exercise 2
// Asynchronous
// async.js

var fs = require('fs');

fs.copyFile('index.html', 'indexcopy.html', (err) => {
  if (err) throw err;
});
```

Table 5: File “async.js” asynchronously copies the file index.html

The code in Table 5 above asynchronously copies index.html to indexcopy.html. By default, indexcopy.html is overwritten if it already exists. “No arguments other than a possible exception are given to the callback function. Node.js makes no guarantees about the atomicity of the copy operation. If an error occurs after the destination file has been opened for writing, Node.js will attempt to remove the destination.” [2]

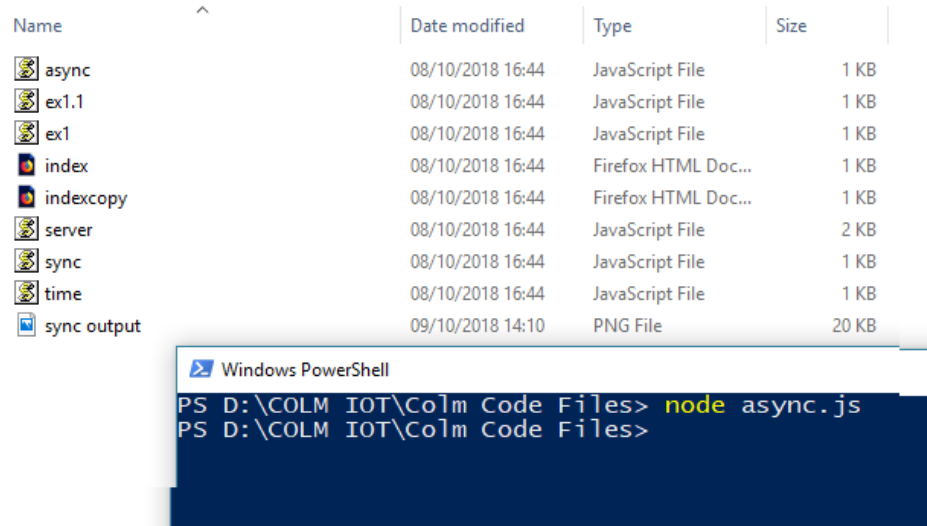


Figure 6: The output of the 'async.js' and a folder showing the copied version of the file

## Exercise 3 - Events and Event Handling in NodeJS

Using the following code in Table 6 taken from [https://www.w3schools.com/nodejs/nodejs\\_events.asp](https://www.w3schools.com/nodejs/nodejs_events.asp) as a basis, complete the following coding tasks:

- Add another eventEmitter object,
- Add another event handler function to the code
- Bind an event to the event handler function
- Fire/emit the new event and test that the new event handler code works
- Comment each line of your code to demonstrate that you understand it.

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

var btEventHandler = function () {
  console.log('discovered a Bluetooth device in the room!');
}

eventEmitter.on('BTdiscover', btEventHandler);

eventEmitter.emit('BTdiscover');
```

Table 6: Event handling code

Using the code in Table 6 taken from [https://www.w3schools.com/nodejs/nodejs\\_events.asp](https://www.w3schools.com/nodejs/nodejs_events.asp) as a basis the following code in Table 7 was written to satisfy the added requirements.

In Node.js it is common for objects to emit events, like an `fs.copyFile` emits an event when the file is copied. When an object emits an event it is said to be an instance of `event.EventEmitter`, the `EventEmitter` class lies within the 'event' module so it must be accessed before creating any `EventEmitter` objects.

“When an `EventEmitter` instance faces any error, it emits an 'error' event. When a new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired. `EventEmitter` provides multiple properties like `on` and `emit`. `on` property is used to bind a function with the event and `emit` is used to fire an event.” [3]

```
// Exercise 3
// main.js

// Accessing EventEmitter class in events module
var events = require('events');

// Original Event Emitter
var eventEmitter = new events.EventEmitter();

// New events emitter object
var newEventEmitter = new events.EventEmitter();

// Original Event Handler
var btEventHandler = function () {
  console.log('discovered a Bluetooth device in the room!');
}

// New Event Handler
var btPairEventHandler = function () {
  console.log('New Bluetooth device paired with your device');
}

// Original Event Emitter (listener)
eventEmitter.on('BTdiscover', btEventHandler);

// New event emitter (listener) this controls what happens when an
'Event Emit' occurs
newEventEmitter.on('BTconnect', btConnectEventHandler)

// Original Event Emitter Emits the Event 'BTconnect' causing the
listener to react and call the btEventHandler
eventEmitter.emit('BTdiscover');

// New Event Emitter Emits the Event 'BTpair' causing the listener
to react and call the btPairEventHandler
newEventEmitter.emit('BTpair');
```

Table 7: Edited code for new eventEmitter object and handler



## Exercise 4 - fs and http modules

Using the following sample code for a NodeJS HTTP server from [https://www.w3schools.com/nodejs/nodejs\\_http.asp](https://www.w3schools.com/nodejs/nodejs_http.asp) as a basis, combine this code with the code from exercise 1 above to output the contents of the file to a browser window (instead of to the console) when the browser sends a request to the server on localhost:8080.

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

http.createServer(function (req, res) {
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Table 8: Web Server Code

“The basic objective of the web server is to store, process and deliver web pages to the users. This intercommunication is done using Hypertext Transfer Protocol (HTTP). These web pages are mostly static content that includes HTML documents, images, style sheets, test etc. Apart from HTTP, a web server also supports SMTP (Simple Mail transfer Protocol) and FTP (File Transfer Protocol) protocol for emailing and for file transfer and storage.” [4]

Web servers display website content, so to request a website a URL is typed into a browser such as Google or Firefox, which in turn sends a request to the Internet for a web page for that address. A DNS converts this URL into an IP address which then points to a Web Server.

Node.js provides an http module which can be used to create an HTTP client of a server.

```
// Exercise 4
// index.html

<!DOCTYPE html>
<html>
  <head>
    <title>Basic Web Page</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

Table 9: 'index.html' code used to test the 'server.js' code

The html file “index.html” is created in the same directory as server.js and is used to test the Web Server created using the server.js code in Table 10.

```
// Exercise 4
// fs, url and http modules
// server.js

var http = require('http');
var fs = require('fs');
var url = require('url');

// Create a Server
http.createServer(function (request, response)
{
    // Parse the request containing file name
    var pathname = url.parse(request.url).pathname;
    // Print the name of the file for which request is made
    console.log("Request for " + pathname + " received.");

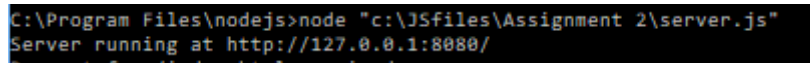
    // Read requested file content from file system
    fs.readFile(pathname.substr(1), function (err, data)
    {
        if (err)
        {
            console.log(err)
            // HTTP Status: 404 : NOT FOUND
            // Content Type: text/plain
            response.writeHead(404, {'Content-Type': 'text/html'});
        }
        else
        {
            //Page found
            // HTTP Status: 200 : OK
            // Content Type: text/plain
            response.writeHead(200, {'Content-Type': 'text/html'});

            // Write the content of the file to response body
            response.write(data.toString());
        }

        // Send the response body
        response.end();
    });
}).listen(8080); //the server object listens on port 8080

// Console will print the message
console.log('Server running at http://127.0.0.1:8080/');
```

Table 10: Server.js code used to create a Web Server

A terminal window with a black background and red and white text. The text shows the command to run a Node.js server and the output indicating the server is running on a specific IP and port.

```
C:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\server.js"  
Server running at http://127.0.0.1:8080/
```

Figure 7: Output at the server end

The output is verified at the server end as seen in Figure 5. The address <http://127.0.0.1:8080/index.html> is typed into a browser as shown in Figure 6 to see the following result which is the contents of index.html as seen in Chrome.

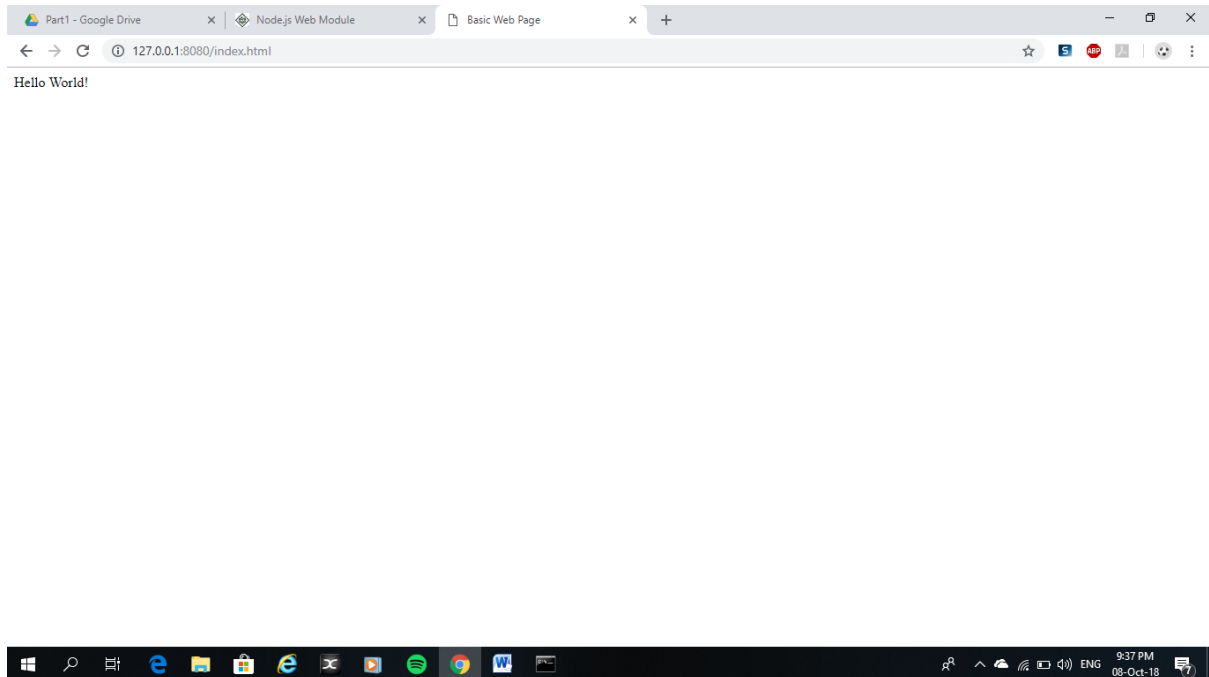


Figure 8: Result after making a request to the Node.js server

## Exercise 5 - Using the Node.js API

`setTimeout` is a basic asynchronous function defined in the `timers` module of Node.js (<https://nodejs.org/api/timers.html>). The function takes 2 or more parameters the first two of which are 1) a callback function and 2) a delay value (for the amount of time to delay before the callback function is executed). The sample code below shows a simple use of the `setTimeout` function being called with the 2 mandatory parameters.

```
function callbackFunc() {
    console.log("2 seconds later...callbackFunc is executed");
}
setTimeout(callbackFunc, 2000);
//next line gets executed while waiting for the delay to complete
console.log("this is executed while waiting for the callback
function");
```

Table 11: `setTimeout` Function

Using the API at <https://nodejs.org/api/timers.html> for the `setTimeout()` function write the code for a similar named callback function that in addition to the two mandatory parameters it also expects two numbers to be passed into it as additional parameters and that then prints out to the console the product of these two numbers, i.e.

```
console.log("2 seconds later...and the answer is ..... " + a*b);
```

```
// Exercise 5
// time.js

var a = 2;
var b = 3;

function callBackFunc(x,y)
{
    console.log("Waited 2 seconds. ");
    console.log("The answer is: ", x*y);
};

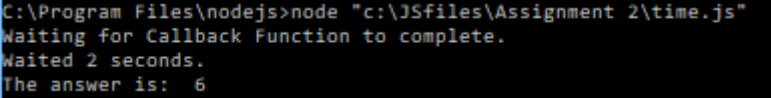
// Now call above function after (2000 ms) 2 seconds
setTimeout(callBackFunc, 2000, a, b);

// Next line gets executed while waiting for the delay to complete
as it is asynchronous
console.log("Waiting for Callback Function to complete.");
```

Table 12: The code used for "time.js"

The code written in Table 12 uses the `setTimeout` function to call a callback function after a specified delay, in this example the callback function is named `callBackFunc(x,y)`. `callBackFunc(x,y)` is function that multiplies the two parameters, `x` and `y`, that are passed to it and prints out the answer. The `setTimeout` function calls this callback function and the desired delay length is specified. But as the `callBackFunc(x,y)` function requires two parameters to be passed to it the `setTimeout()` function must also specify which variables are going to be passed to the `callBackFunc(x,y)` function, in this case the variable `a` and `b` are the variables passed to the function.

As this is an asynchronous code the last line is executed prior to the callback function being called as it is not delay constrained. The output of the program can be seen from the terminal image in Figure 7.

A terminal window showing the execution of a Node.js script. The prompt is 'C:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\time.js"'. The output consists of three lines: 'Waiting for Callback Function to complete.', 'Waited 2 seconds.', and 'The answer is: 6'.

```
C:\Program Files\nodejs>node "c:\JSfiles\Assignment 2\time.js"
Waiting for Callback Function to complete.
Waited 2 seconds.
The answer is: 6
```

Figure 9: Output result when 'time.js' is executed

## Bibliography

- [1] Joyent, Inc., "File System | Node.js v10.11.0 Documentation," [Online]. Available: [https://nodejs.org/api/fs.html#fs\\_fs\\_copyfile\\_src\\_dest\\_flags\\_callback](https://nodejs.org/api/fs.html#fs_fs_copyfile_src_dest_flags_callback). [Accessed 2nd October 2018].
- [2] Joyent, Inc., "File System | Node.js v10.11.0 Documentation," [Online]. Available: [https://nodejs.org/api/fs.html#fs\\_fs\\_copyfilesync\\_src\\_dest\\_flags](https://nodejs.org/api/fs.html#fs_fs_copyfilesync_src_dest_flags). [Accessed 2nd October 2018].
- [3] TutorialsPoint, "Node.js Event Emitter," [Online]. Available: [https://www.tutorialspoint.com/nodejs/nodejs\\_event\\_emitter.htm](https://www.tutorialspoint.com/nodejs/nodejs_event_emitter.htm). [Accessed 8 October 2018].
- [4] The Economic Times, "What is Web Server - The Economic Times," [Online]. Available: <https://economictimes.indiatimes.com/definition/web-server>. [Accessed 8 October 2018].