Walk-through 005

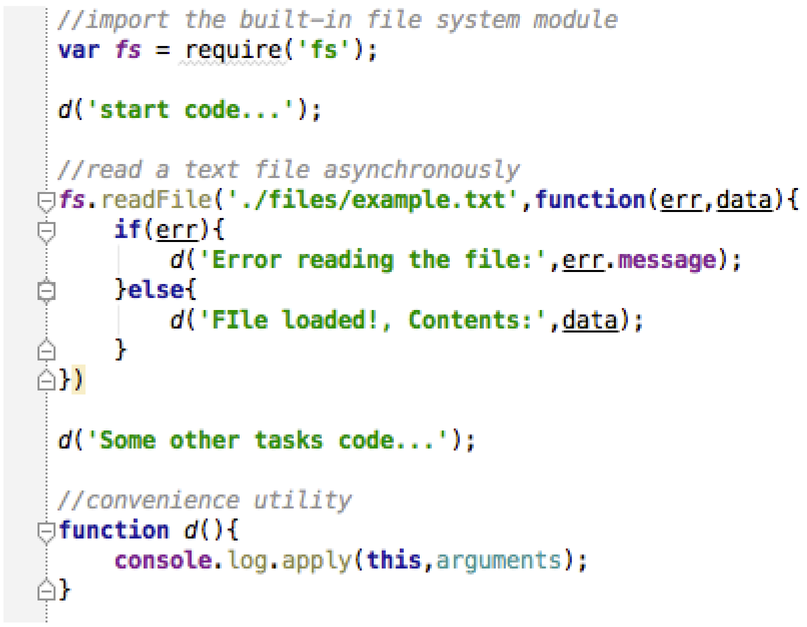
FILE SYSTEM OPERATIONS

Objectives:

1. learn basic file handling operations: read / write / parse / watch
2. refactor WT004 to load host and port from a config.json file
3. be able to refactor routing to send back static html pages

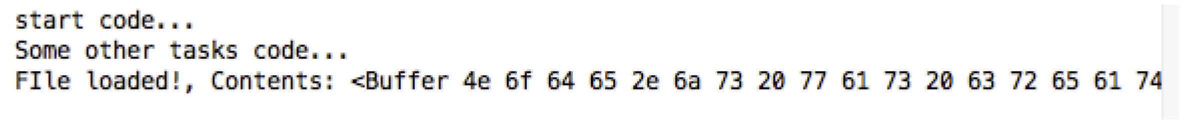
Steps:

1. Create a web module / new folder under your project and call it: 02_filesHandling
2. Create an **app.js** file, a **package.json** file and an **example.txt** file.
3. Open the **example.text** file and paste some text into it, save it in a new folder named '**files**' and close it.
4. Open the **app.js** and write the following and run the app:



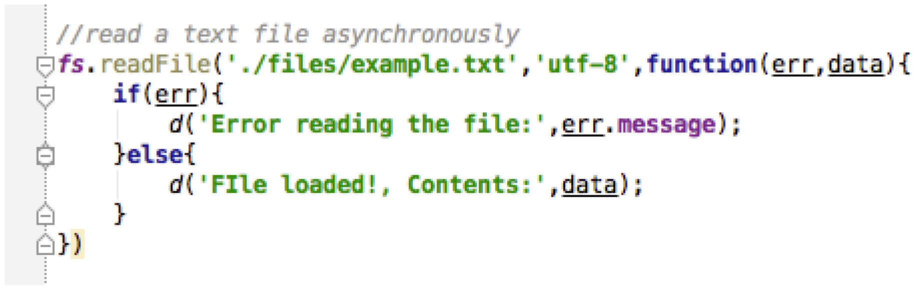
```
//import the built-in file system module
var fs = require('fs');

d('start code...');

//read a text file asynchronously
fs.readFile('./files/example.txt',function(err,data){
    if(err){
        d('Error reading the file:',err.message);
    }else{
        d('FIle loaded!, Contents:',data);
    }
})

d('Some other tasks code...');

//convenience utility
function d(){
    console.log.apply(this,arguments);
}
```

5. Notice the execution sequence we see in the console:



```
start code...
Some other tasks code...
FIle loaded!, Contents: <Buffer 4e 6f 64 65 2e 6a 73 20 77 61 73 20 63 72 65 61 74
```

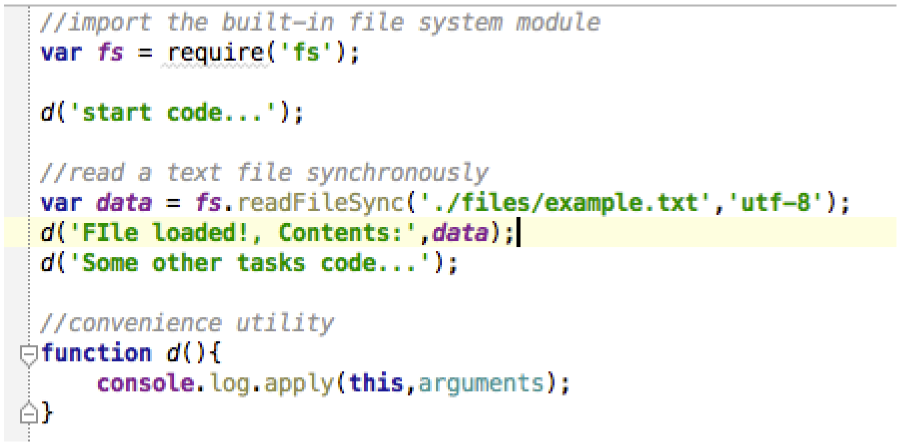
First we get the starting log, then the last log, and when the file was loaded, only then the contents of the file. notice the contents is not showing as text, this is due to encoding issues, if you come across a similar print you can very easily fix it either by adding a **toString()** call to the data object, or specify the encoding when loading the text file like so:



```
//read a text file asynchronously
fs.readFile('./files/example.txt','utf-8',function(err,data){
    if(err){
        d('Error reading the file:',err.message);
    }else{
        d('FIle loaded!, Contents:',data);
    }
})
```

Run the app again and see the text contents of the file in the console.

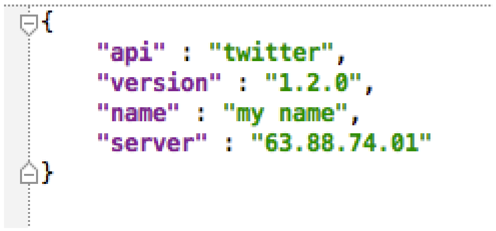
6. Next we'll load the file synchronously. This is a good fit for situations you want to make sure the rest of the code executes only after the content of the file is accessible, like loading a config file for example.
   Refactor the code like so:



```
//import the built-in file system module
var fs = require('fs');

d('start code...');

//read a text file synchronously
var data = fs.readFileSync('./files/example.txt','utf-8');
d('FIle loaded!, Contents:',data);
d('Some other tasks code...');

//convenience utility
function d(){
    console.log.apply(this,arguments);
}
```

   When you'll run this code, you'll see the 'other tasks' log is being printed only after the contents of the file was loaded and printed. Also notice that when we load files synchronously, we don't supply a callback function but instead, store the contents of the loaded file in a variable.

7. Next, let's use the **readFileSync** method to load a config file, and parse it straight after.
   Create a new file in the files folder and name it **config.json**
   Edit its content to contain the following data and save it.

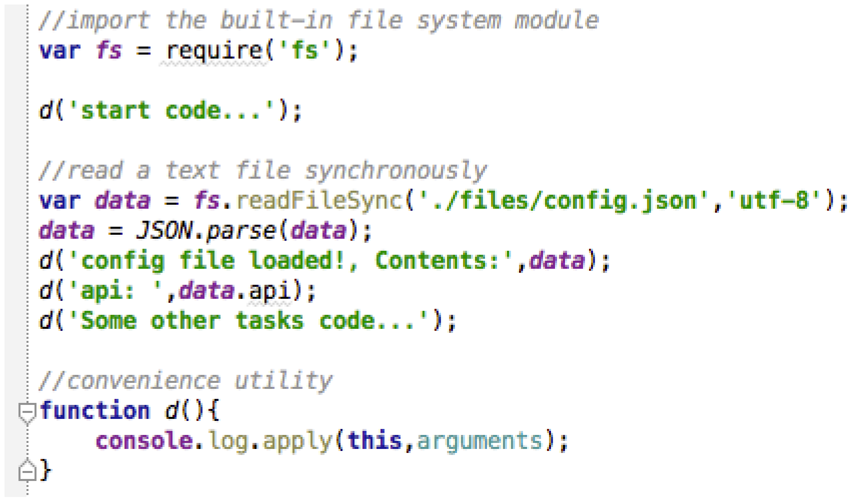


```
{
    "api" : "twitter",
    "version" : "1.2.0",
    "name" : "my name",
    "server" : "63.88.74.01"
}
```

8. Next - load it synchronously and display its content.

   You’ll get a string representation of the json file. In order to parse it and use its properties, we’ll use the **JSON.parse** method which will convert the data to a JavaScript object, we can then use normally. Here is the complete example. Edit your **app.js** file to match it.

   This example loads the config file synchronously, parse it contents and print out one of its properties to demonstrate it is being treated like a JS object.

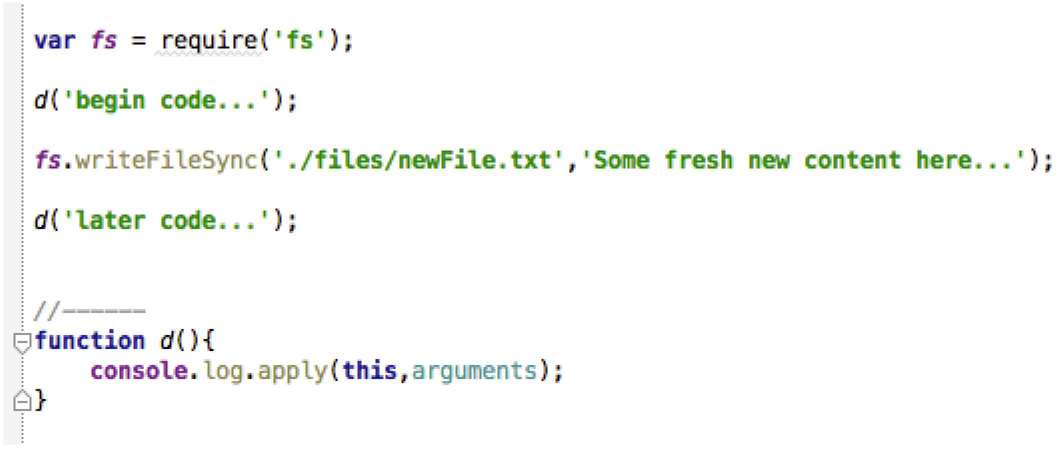


```
//import the built-in file system module
var fs = require('fs');

d('start code...');

//read a text file synchronously
var data = fs.readFileSync('./files/config.json','utf-8');
data = JSON.parse(data);
d('config file loaded!, Contents:',data);
d('api: ',data.api);
d('Some other tasks code...');

//convenience utility
function d(){
    console.log.apply(this,arguments);
}
```

9. Next we'll learn to write a file.

   Similar to **readFile** and **reaFileSync** methods**,** we can use **writeFile** and **writeFileSync**.

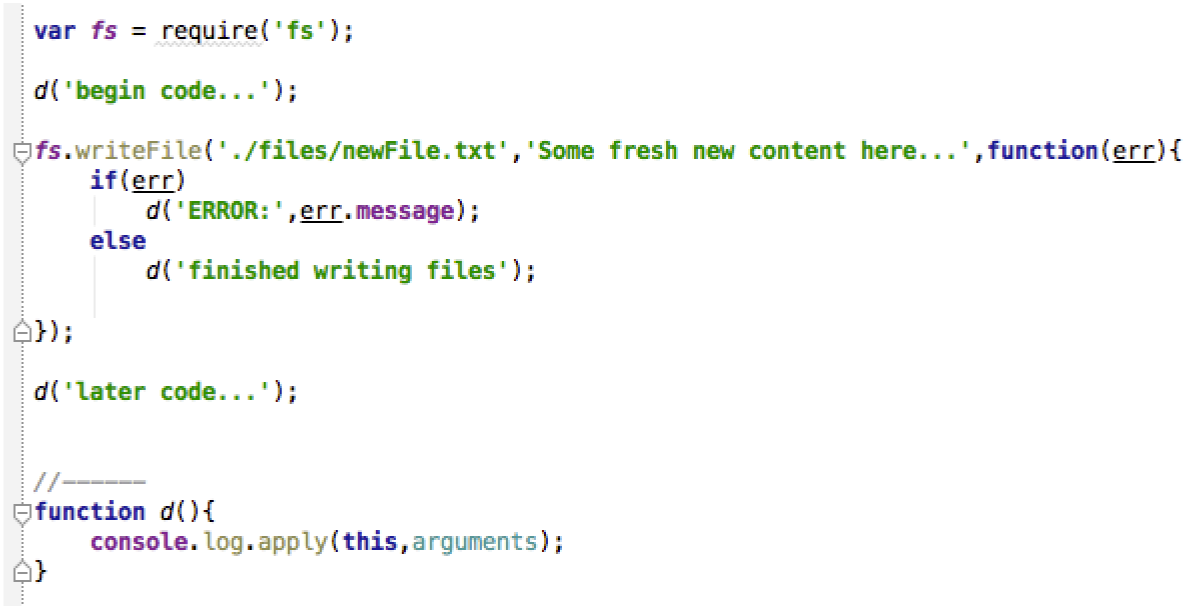
   Create a new app.js file and write the following:



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

d('begin code...');

fs.writeFileSync('./files/newFile.txt','Some fresh new content here...');

d('later code...');


//------
function d(){
    console.log.apply(this,arguments);
}
```

Save and run the script, then check out the new file created. Notice the file was created synchronously and the app waits for it to finish in order to continue executing the code.

In most cases we'll want to use **writeFile** that would write a file without blocking the app like so:

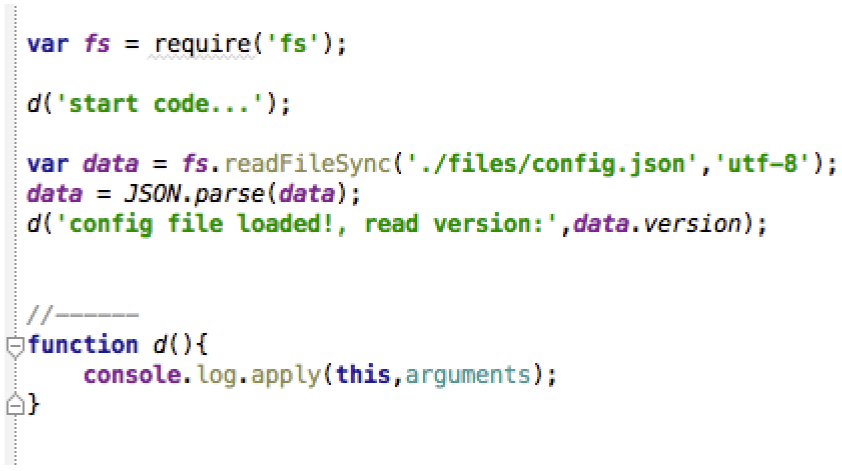


```
var fs = require('fs');

d('begin code...');

fs.writeFile('./files/newFile.txt','Some fresh new content here...',function(err){
    if(err)
        d('ERROR:',err.message);
    else
        d('finished writing files');

});

d('later code...');


//------
function d(){
    console.log.apply(this,arguments);
}
```

10. Next - we'll learn how to watch files.

    Watching files is a very popular and could be achieved very easily with node. What it means is we'll be notified whenever a file had been updated and could trigger another action as a result. For example a web app which is configured a certain way based on its config file could watch it and instead of restarting the server when something had been updated, the app could watch the config file and adjust accordingly.

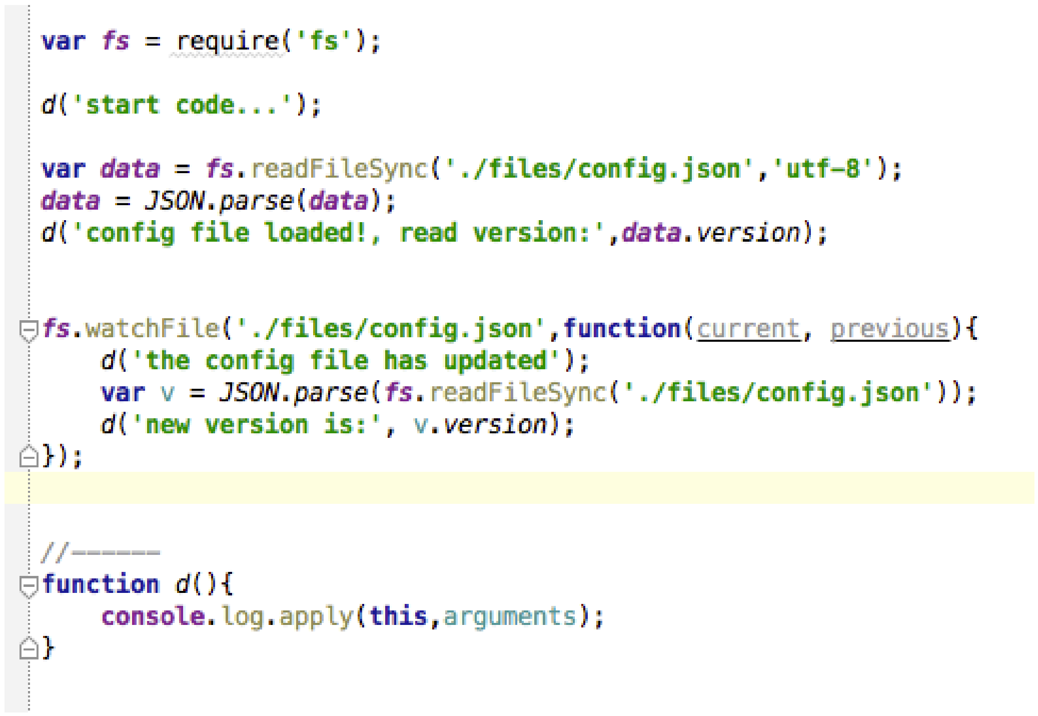
    We'll start by reading the config file synchronously like we did earlier:



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

d('start code...');

var data = fs.readFileSync('./files/config.json','utf-8');
data = JSON.parse(data);
d('config file loaded!, read version:',data.version);


//------
function d(){
    console.log.apply(this,arguments);
}
```

Run the app and notice the version we'll print out.

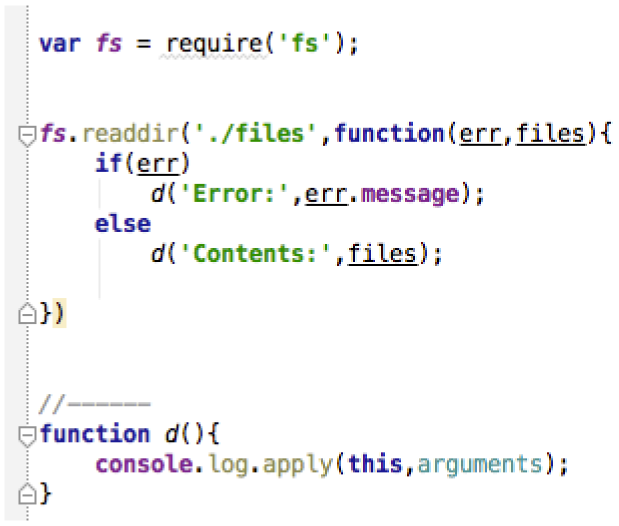
11. Next we'll watch the file passing the location of the file and a callback function that would execute as soon as the file will update, like so:



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

d('start code...');

var data = fs.readFileSync('./files/config.json','utf-8');
data = JSON.parse(data);
d('config file loaded!, read version:',data.version);


fs.watchFile('./files/config.json',function(current, previous){
    d('the config file has updated');
    var v = JSON.parse(fs.readFileSync('./files/config.json'));
    d('new version is:', v.version);
});


//------
function d(){
    console.log.apply(this,arguments);
}
```

12. To get a list of files in a given directory you can very easily use **readdir** and **readirSync**

Create a new app.js file and try the following:



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


fs.readdir('./files',function(err,files){
    if(err)
        d('Error:',err.message);
    else
        d('Contents:',files);

})


//------
function d(){
    console.log.apply(this,arguments);
}
```

As you can see, the **readdir** method of the fileSystem module accepts the path of the directory and a callback with two parameters - first one is error and the second one is an array of the files names. As in the read and write methods, **readdirSync** will not have a callback but instead the method will return the output which you can assign of course to a variable.

Notes:

- Be sure to go through the full file operations reference here - http://nodejs.org/api/fs.html