Web Development: Lesson 4  
npm/Debugger Introduction, and Timer Hands-On Lab

## Overview

In this lab, we will introduce the node command line interface. This lab will run through installing npm modules and debugging in node. Additionally, we'll be using the process and timer functions to build a command line interface application that takes a number of seconds or minutes and counts them down. By now you should now how to access CLI arguments and use setTimeout().

## Objectives

In this hands-on lab you will learn how to:

* Use npm to install express, mocha and lodash
* Use the Node debugger
* Install and run node-inspector (a GUI node debugger)
* Implement a Node CLI timer app which works from CLI (takes a time and counts down like a kitchen timer)
* Implement accepting CLI arguments: a number of seconds or minutes
* Implement the asynchronous sleep function so that the app knows when the time is up
* Implement the current countdown so that the users can see how much time is left

## Prerequisites

The following are required to complete this hands-on lab:

* A code editor
* Windows PowerShell, Mac Terminal, or some other shell with node.js installed
* You should have completed [Module 2 Lesson 1 and Lesson 2](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module2/Lessons)

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Learn how to use npm, built-in Node debugger and Node Inspector
* Exercise 2: Implementing a Node CLI Timer

## Exercise 1: Learn how to use npm, built-in Node debugger and Node Inspector

In this exercise, you will use the command line to configure npm. This exploratory exercise will give you an opportunity to familiarize yourself with node inspector and node debugger.

NOTE: The following resources will detail the packages installed in this lab:

* <https://www.npmjs.com/package/express>
* <https://www.npmjs.com/package/mocha>
* <https://www.npmjs.com/package/lodash>

1. In your desired directory run the command npm init and respond appropriately to the prompts. Step through creating package.json. In the end you will have a file called package.json. It's recommended to start every new Node project with the creation of this file.
2. Next, install some libraries to learn how to work with npm. Install express, mocha and lodash using the following commands:

$ npm install express

$ npm install mocha --save-dev

$ npm install lodash --save

$ npm i express

Notice the results of save-dev and save when you execute these commands (there are different entries in package.json). It is a good to have mocha installed locally in devDependencies, because it allows using different versions of it with different projects. There are aliases to

--save-dev which is -D and to --save which is -S. i is an alias for install.

1. In case you need to configure npm, use the following commands to first view, then edit config values such as name and email in the example below:

$ npm config list

$ npm set init.author.name "Your Name"

$ npm set init.author.email "you@example.com"

$ npm set init.author.url http://yourblog.com

1. It's convenient to see what modules/packages you have already in your project or computer (global installation). List currently installed npm using the command

$ npm ls

$ npm ls -g

The –g flag lists globally installed modules. To install modules globally means to have them available in shell/terminal in any folder of your computer. You do it with npm i -g [package\_name] or npm install --global [package\_name].

1. If you want to update a package version, simply install with explicit version using @ and semantic version number (e.g., 4.2.9) or word latest, and to remove npm modules use the following rm or uninstall commands:

$ npm i [package\_name]@latest

$ npm install [package\_name]@4.2.9

$ npm rm [package\_name]

$ npm rm [package\_name] -g

$ npm uninstall [package\_name] --global

The –g flag removes globally installed modules.

1. Node comes with a built-in debugger. If you want to debug a program all you need to do is to start the program in a debug mode:

$ node debug program.js

The prompt will turn into a REPL debug>. You can get a list of commands by using the help command.

The debugger commands are similar to those used when debugging in other languages like Ruby or browser JavaScript (with the help of DevTools). To name a few:

debug> run //runs program

debug> cont //(c): continue, i.e., proceed with the execution until a breakpoint

debug> next //(n): step to the next line

debug> step (s): step in (go deeper into the execution context)

debug> out (o): step out (go out of the execution, skipping the deeper context)

debug> setBreakpoint //(sb): sets breakpoint(e.g. setBreakpoint(20) sets breakpoint to line 20)

debug> clearBreakpoint //(cb): removes breakpoint(e.g. clearBreakpoint('script.js', 1) clears the break point in the script.js file on line 1)

1. Node Inspector is the recommended tool to debug Node. Install and launch Node Inspector, a GUI similar to Google Chrome’s DevTools:

$ npm install -g node-inspector

$ node-debug [my\_cool\_app].js

Checkout <https://github.com/node-inspector/node-inspector> for more about Node Inspector.

If you are using VS Code, then you have debugger built-in into the editor. It's very similar to Node Inspector and DevTools. The advantage of using the VS Code debugger is that developers don't have to leave the application and switch to terminal and then to browser. The disadvantage is that if you are using another editor or in a GUI-less environment (CentOS machine in the cloud) then you can really use it. Webstorm IDE also offers debugging.

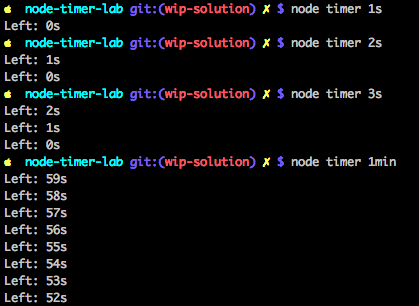
## Exercise 2: Implementing a Node CLI Timer

In this exercise, you will write a timer function that accepts command line input and counts down to 0, displaying the time left (in seconds) as it counts down.

NOTE: You might want to consider using setInterval() and clearInterval() from Node Timers API which is identical to browser timer API. If you are not already familiar with setInterval()here are some helpful links:

* <https://nodejs.org/api/timers.html>
* <https://developer.mozilla.org/en-US/Add-ons/Code_snippets/Timers>

The example lab output



To create Timer, we will follow these steps:

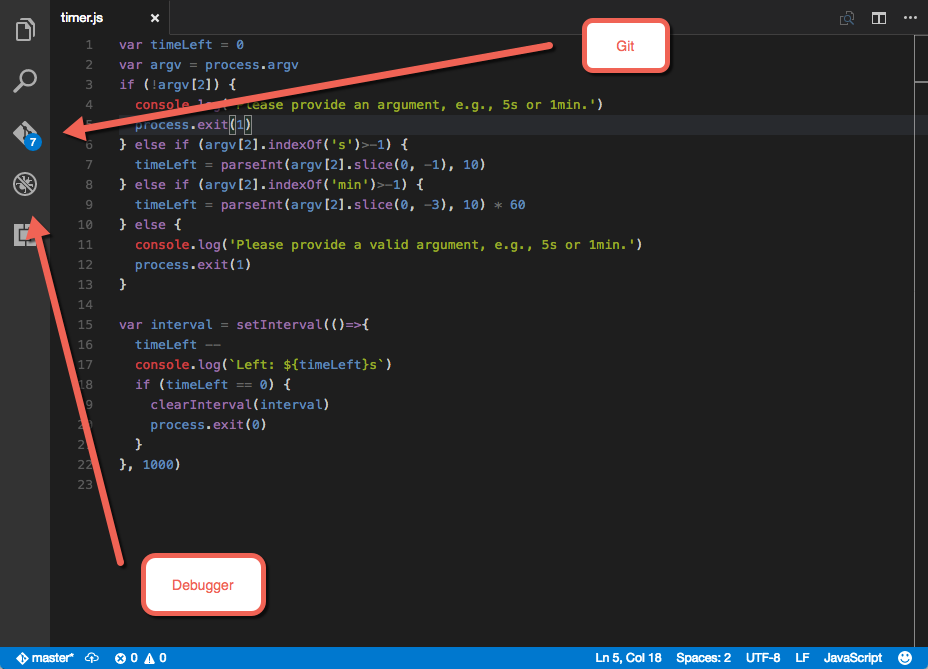
1. Create a CLI app that takes a number of seconds in the following format: ns for n seconds, e.g., and 5s for five seconds. The command for 5 seconds must look like node timer 5s.
2. Implement the timer in timer.js by printing the number of seconds left on a new line. For example, if you passed 10s as your argument, you should see the text 'Left: 10s', 'Left: 9s', etc. printed to the console.
3. Use global objects setInterval() and clearInterval()
4. Enhance the app by adding minutes in the following format nmin, e.g. 1min for 1 minute. The command for 1 minute must look like node timer 1min.
5. Utilize exit() to exit when there's an error and when there's no error, but the time is up.
6. Use process to access arguments from the command line.
7. Test Timer with npm test. Create a directory in your cluster home directory.

Let's get started. Open your terminal or command prompt and create a new folder and navigate to it using:

$ mkdir timer

$ cd timer

and create a new file timer.js. Feel free to use your favorite code editor such as [VS Code](http://code.visualstudio.com/), [Atom](https://atom.io/), [Sublime Text](https://www.sublimetext.com/), [Webstorm](https://www.jetbrains.com/webstorm) or similar.



The structure of the program will look as follows:

var timeLeft = 0

var argv = process.argv // Access CLI arguments

if (!argv[2]) { // Error if argument is not provided

console.log('Please provide an argument, e.g., 5s or 1min.')

process.exit(1)

} // Check if s or min present in the argument

else { // Error if s or min is not present

console.log('Please provide a valid argument, e.g., 5s or 1min.')

process.exit(1)

}

var interval = setInterval(()=>{

// Print time left and check if timeLeft is 0

}, 1000)

Now, let’s fill in the details of the implementations. First we set the value of timeLeft to 0 and access all CLI arguments from process.argv:

var timeLeft = 0

var argv = process.argv

Next, we must check if there is a CLI argument which would follow the file name. Remember, in JavaScript/Node arrays are 0-based so the first argument is node, second is the script name timer.js and the third which has the index of 2 is the value of time. Therefore, we can extract this argument from argv[2] and then check if it's truthy (not empty):

if (!argv[2]) { // Error if argument is not provided

console.log('Please provide an argument, e.g., 5s or 1min.')

process.exit(1)

After we ensured that the value is not empty so there would be no exceptions, we can check if there's an s character meaning it's the number of seconds, not minutes. We use slice to strip the s character and then we use parseInt with base 10 to parse the string into a number:

} else if (argv[2].indexOf('s')>-1) {

timeLeft = parseInt(argv[2].slice(0, -1), 10)

Similarly, if there's no s it could be minutes so we check for min, and then parse the value while also multiplying it by 60 to convert to seconds. This way, we can have just one function which reduces number of seconds for both minutes and seconds:

} else if (argv[2].indexOf('min')>-1) {

timeLeft = parseInt(argv[2].slice(0, -3), 10) \* 60

Finally, we should exit with 1 error code and display the message that neither s or min were provided when all the previous conditions fail:

} else {

console.log('Please provide a valid argument, e.g., 5s or 1min.')

process.exit(1)

}

So far we implemented checks and parsing of the values. We should have a numeric value of timeLeft. Next, we will launch the timer with setInterval() making sure it clears after time is up. Without clearing the timer, it will run forever which is not what we want. At the end, let's exit the script with success code 0.

var interval = setInterval(()=>{

timeLeft -- // Decrement one second

console.log(`Left: ${timeLeft}s`) // Display seconds left in terminal

if (timeLeft == 0) { // Check for 0

clearInterval(interval) // Clear time so it's not executing anymore

process.exit(0) // Exit with code 0 - success

}

}, 1000) // Use 1000 milliseconds for 1 second interval

Now you can save the file and test it by going to the terminal / command prompt and running the script with second, minutes or even without anything (should error):

node timer 1s

node timer.js 1s

node timer 1min

node timer

node timer 5min

Note that the .js extension is optional when running Node scripts.

Congratulations on finishing your first CLI program in Node!

## Summary

In this hands-on lab, you learned how to:

* Use the node CLI to debug and install npm modules
* Install and use Node Inspector
* Write a basic Node CLI app
* Use command line arguments
* Output to the command line