Web Development: Module 2, Lesson 2  
Node CLI 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 Node CLI and 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 text editor
* Windows PowerShell, Mac Terminal, or some other shell with node.js installed
* You should have completed Module 2 Lesson 1

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Using Node CLI and npm
* Exercise 2: Implementing a Node CLI Timer

## Exercise 1: Using Node CLI and npm

In this exercise, you will use the command line to configure npm.

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, installing some libraries.
2. Next install express, mocha and lodash using the following commands:

$> npm install express

$> npm install mocha --save-dev

$> npm install lodash --save

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.

1. Use the following commands to first view, then edit config values:

$ 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. Publishing your npm module:

Sign up on the npm website:

$ npm config adduser

Add package.json, then publish:

$ cd my-cool-app

$ npm publish

Use escape.js and escape.test.js from code/lesson2 to publish

1. List currently installed npm using the command

$ npm ls

$ npm ls -g

The –g flag lists globally installed modules.

1. Search for npm modules using the command

$ npm search [keyword]

which queries the npm registry, retrieves search results and prints them out to standard output

1. To update or remove npm modules use the following commands:

$ npm update [package\_name]

$ npm rm [package\_name]

$ npm rm [package\_name] -g

The –g flag removes globally installed modules

1. Node comes with a built-in debugger. 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. Install and launch Node Inspector, a GUI similar to Google Chrome’s DevTools:

$ npm install -g node-inspector

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

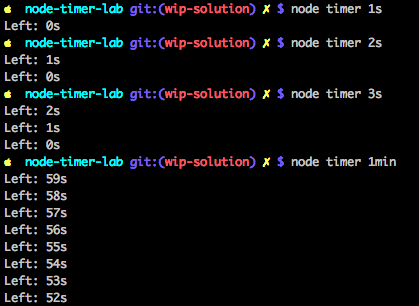
Checkout <https://github.com/node-inspector/node-inspector> for more about Node Inspector

## 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. Most of front-end developers are already familiar with setInterval() so we won't duplicate and explain it here. But here are the links if you need a refresher:

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



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 global.setInterval() and global.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 process.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.

## 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