Async.js

In this lab, you will familiarize yourself with the Node module Async.js, which assists in various ways to perform parallel & serial control flow as well as to iterate collections in both parallel & serial modes.

# Objectives

You will learn to use async.auto to more cleanly control program flow among interdependent functions.

# Install Async.js

The first thing we have to do is to use Node Package Manager, npm, to install the 'async' module. Do this by opening a command prompt in the lesson directory and issuing the command npm install async. This will download & install the latest version of the Async.js module into a directory called node\_modules.

NOTE: The version of async in the solutions folder is 1.4.x and the version obtained with npm install async is version 2.0.0-rc3. These are NOT compatible. Feel free to examine the online documentation to determine the differences. HINT: the order of the parameters is different. We will change the labs when 2.0 is released. If the task has NO dependencies, it has only one parameter, the callback. If the task has dependencies, the task has two parameters, the current results and the callback.

# Lab Steps

Now that Async.js is installed, we can begin the lab, whose goal is to use a simple (and admittedly buggy) minification algorithm to reduce the size of a JavaScript file.

## Add tasks

Since async.auto is a function that takes an object representing the tasks to be performed and a callback to be invoked when all tasks have been completed, we need to fill in the stubbed out tasks variable in auto.js with our initial requirements.

Notice that we've defined several functions: exists, read, create, write & close. These all deal with the JavaScript file we're reading from and the minified JavaScript file we're writing to. The functions' interdependencies are the following:

* read must be called only after exists is successfully called,
* write must be called only after read & create are successfully called, and
* close must be called only after write is successfully called.

Let's start slowly, and simply add our first function to the tasks object that calls exists.

1. Add a property named exists to variable tasks whose value is a reference to the exists function.

## Invoke async.auto

2. Now that we've got a minimal tasks object created, add the invocation of async.auto that takes the tasks object and a callback of the form function(err, results) that checks for the existence of an error and, if one exists, logs it to the console, otherwise simply logs a message that the file was minified ok.

3. Execute your tiny workflow consisting simply of the call to exists by executing node auto.js at a command prompt.

Once you see your 'minified ok' message, move on to the next step.

## Add the read task

The next thing to do is to add to the workflow the call to the read function, but only after the call to exists has completed successfully.

4. Update your tasks object to contain a property called read whose value is an array of a string containing the name of the key you used for the task that confirms file existence (exists) and a reference to the read function.

5. Execute your workflow again via node auto.js at the command prompt. This time, you should observe your exists function being invoked before the read function.

Once you see the order of invocations that you expect and your 'minified ok' message again, move on to the next step.

## Add the create task

Before we can write anything to the destination file, let's add a call to the create function, which ensures that the file is created and opened with a file descriptor. Note that create has no prerequisites because we're always creating & overwriting the destination file.

6. Update the tasks object with a property create whose value is a reference to the create function.

7. Execute your workflow again via node auto.js.

Once you see your create function being called and your 'minified ok' message, move on to the next step.

## Add the write and close tasks

We're now going to finish our minification workflow by adding two more tasks: one to write the contents of the source file and one to close the destination file after it's been written.

Notice that the buffer we're going to write to the file is set to null initially. Your job is to create a new Buffer whose contents are equal the array-ified string contents read from the read function. Where can you find that? Thanks to async.auto, we can find it in the results object that's given to the write function!

You see, async.auto tracks all values returned by tasks and stores them as properties on a results object that is given to any function that wants them (including the final completion callback given to async.auto itself).

8. Set the buffer to a new Buffer object whose value is simply results.read.join(' '), which is the result of the elements of the array returned by the read function (results.read) concatenated with a space (join(' ')).

9. Set the file descriptor fd to results.create, which is the file descriptor returned by the function create.

Lastly, we want to be good resource citizens and close the destination file.

10. Implement the close function by invoking fs.close using the same file descriptor used above.

11. Now, update the tasks object to include a write property that depends on read and create & invokes write, and a close property that depends on write & invokes close.

12. Execute node auto.js, which now has our complete workflow, and observe each function invocation happening only after each function's prerequisites have been satisfied.

Once you see the correct invocations and your 'minified ok' message, you have completed this lab!