Working with the File System

In this lab, you will learn to use Node’s File System module by watching a directory for file updates and file & directory creations and deletions.

# Objectives

In this lab, you will learn to

* use various functions from Node's File System module, and
* write asynchronous callback functions to handle events from the file system.

# Set things up

Let's start by setting up our script with a little bookkeeping.

## Require the File System module

1. Open the JavaScript file watch.js and declare a variable for Node’s File System module, by adding the line

var fs = require('fs');

## Declare a data structure to keep track of files & directories encountered

2. Declare a data structure that will hold our initial view of files and directories. You can use what you want, but an obvious choice would be map of lists:

var entries = { files : [], dirs : [] };

## Store the directory to watch as a variable

3. Declare a variable to hold the name of the directory that we'll be watching (due to the behavior of the file-related callbacks that we'll be using later on):

var dir = './dir';

Ok, we're now ready to start coding our directory watching logic!

# Initialize the data structure with files & directories

The first thing that we need to do is write logic to populate our data structure with the initial lists of files and directories in the directory that we're watching.

## Read the directory

4. Use the function fs.readdir to get all of the entries in the directory dir.

fs.readdir takes the directory you want to read and a callback of the form function(err, filenames), where filenames is an array of filenames relative to the directory being read.

It is a good idea to always name your callback functions, because when things go wrong, you can identify them more easily in stack traces. Also, remember that on success, err will be falsey, otherwise truey; if it's truey, make sure to throw it.

## Add file & directory names to the data structure

5. Use Array's forEach method to process each entry. forEach takes a callback of the form function(element, index array).

The fs.readdir function, when invoking your callback, returns filenames that are relative to the directory being read. For this particular use case, we'll only need to use the first argument (the array element), so your callback can take the form function(filename).

6. In your forEach callback, first prefix the given filename with the name of the directory being read.

7. Use fs.stat to find out if the given filename represents a file or directory.

fs.stat takes a callback of the form function(err, stats), where stats is an object of type fs.Stats, and has methods isDirectory() and isFile(). Use those methods to determine if the given filename represents a file or directory, and add it to the appropriate list (via Array's push method).

## Run what you've got so far

This is a good place to see whether things are working so far.

8. Sprinkle some console.log statements in with your forEach callback to see that your data structure is getting populated properly. If your data structure is a map called entries, you can just log the map itself with a call to console.log(entries).

9. Run your code either in your IDE or at the command line by invoking node watch.js in the directory containing the file watch.js.

Once you see that your code is populating your data structure ok, move on to the next step!

# Write the main program logic

It's time to write the meaty part of our application.

## Watch the target directory

10. Use the fs.watch function, which takes the target directory and a callback of the form function(event, filename). Write the call to fs.watch and include an empty callback for now; we'll complete the callback next.

fs.watch emits the rename event for all file & directory deletions, creations & renames. On some operating systems (like Mac OS X), a rename actually consists of a deletion and subsequent creation. Your mileage may vary.

## Complete the directory event callback function

11. Make the first line of your fs.watch callback function replace the contents of the filename variable with the filename prefixed with the target directory.

Just as the fs.readdir callback receives filenames relative to the directory it was given, so does the fs.watch callback.

The event given to your callback is actually a string with one of three values: 'error', 'change', or 'rename'.

12. Write skeleton code to handle each event value (hint: this is a good place to use a switch statement).

## Handle errors

13. Write code to handle the 'error' event. Simply throw an Error object with a message that indicates there was an error watching the target directory.

## Handle changes

14. Write code to handle the 'change' event that logs a message indicating which file changed and what its new size is. Use filename and the function fs.stat again to get the size of the file and echo the filename and its new size to the console (via console.log). Don't forget to add your break statement if you're using a switch!

15. Run your code again, then open and save a file in the directory you're watching. You should see messages indicating that file's changes.

Once that's working, move on to the next step!

## Handle file & directory deletions, creations & renames

16. Write code to handle the 'rename' event, using fs.exists to determine whether or not the given file exists. If it does, then you know that it's a creation.

fs.exists takes a filename and a callback of the form function(exists), where exists is a boolean indicating whether the file exists.

17. If fs.exists calls back with a true value, use fs.stat to determine whether it's a file or directory, and log a message to the console stating which type of entry (file or directory) that event is for, the name of the file or directory involved, and finally, how many files & directories are now being watched (information which can be gleaned from your data structure).

18. If event is for a file or directory that no longer exists, then you know that it was a deletion.

You can tell whether or not the file or directory existed by using your data structure; simply see if the file or directory name is present in your list of files or directories, respectively.

Remove the entry from the appropriate list (via Array's splice method), and then log to the console which type of entry event was for (file or directory), the file or directory's name, and the number of files and directories now being watched.

19. Run your application again. This time, trying adding then removing files or directories in the target directory and see what happens.

When you're seeing the messages that you expect, you have completed the lab!