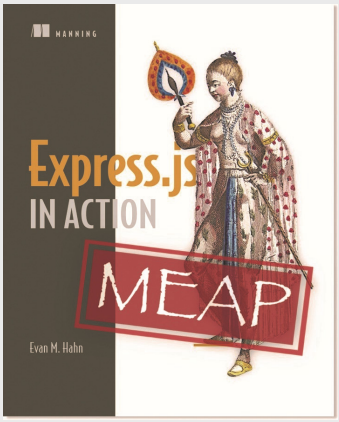

Using Browserify to require modules in the browser, just like in Node

By Evan Hahn, Express.js in Action

Client-side projects can be problematic for many reasons but having no reliable module system is a big one. Browserify solves the module problem by letting you require modules exactly like you would in Node. In this article, we take a look at how Browserify works.

This article is excerpted from Express.js in Action. Save 39% on Express.js in Action with code 15dzamia at manning.com.

Browserify is a tool for packaging JavaScript that allows you to use the `require` function just like you do in Node. And I love Browserify. I just want to get that out of the way. I freakin' love this thing.

I once heard someone describe browser-based programming as "hostile." I love making client-side projects, but I must admit that there are a lot of potholes in the road: browser inconsistencies, no reliable module system, an overwhelming number of packages that vary in quality, no real choice of programming language...the list goes on. Sometimes it's great, but sometimes it sucks! Browserify solves the module problem in a clever way: it lets you require modules exactly like you would in Node (in contrast to things like RequireJS, which are asynchronous and require an ugly callback). This is powerful for a few reasons.

First, it lets you easily define modules. If Browserify sees that `evan.js` requires `cake.js` and `burrito.js`, it'll package up `cake.js` and `burrito.js` and concatenate them into the compiled output file.

Second, it's almost completely consistent with Node modules. This is huge—both Nodebased and browser-based JavaScript can require Node modules, letting you share code between server and client with no extra work. You can even require most native Node modules in the browser, and many Node-isms like `__dirname` are resolved.

I could write sonnets about Browserify. This thing is truly great. Let me show it to you.

A simple Browserify example

Let's say you want to write a webpage that generates a random color and sets the background to that color. Maybe you want to be inspired for the next great color scheme.

We'll use an npm module called `random-color`, which just generates a random RGB color string. If you check out the source code for this module, you'll see that it knows nothing about the browser—it's only designed to work with Node's module system.

Make a new folder to build this. We'll make a `package.json` that looks something like this (your package versions may vary):

Listing 1 package.json for our simple Browserify example

```
{
  "private": true,
  "scripts": {
    "build-my-js": "browserify main.js -o compiled.js"
  },
  "dependencies": {
    "browserify": "^7.0.0",
    "random-color": "^0.0.1"
  } }
```

Run `npm install` and then create a file called `main.js`. Put this inside:

Listing 2 main.js for our simple Browserify example

```
var randomColor = require("random-color"); document.body.style.background =
    randomColor();
```

Note that this file uses the `require` statement, but it's made for the browser, which doesn't have that natively. Get ready for your mind to be blown!

Finally, define a simple HTML file with the following contents:

Listing 3 HTML file for our simple Browserify example

```
<!DOCTYPE html>
<html>
<body>
  <script src="compiled.js"></script>
</body>
</html>
```

Now, if you save all that and run `npm run build-my-js`, Browserify will compile `main.js` into a new file, `compiled.js`. Open the HTML file you saved to see a webpage that generates random colors every time you refresh!

You can open `compiled.js` to see that your code is there, as is the `random-color` module.

The code will be ugly, but here's what it looks like:

```
(function e(t,n,r){function s(o,u){if(!n[o]){if(!t[o]){var a=typeof
      require=="function"&&require;if(!u&&a)return a(o,!0);if(i) return
      i(o,!0);var f=new Error("Cannot find module '"+o+
"'");throw f.code="MODULE_NOT_FOUND",f}var l=n[o]={
      exports:{}};t[o][0].call(l.exports,function(e){var n=t[o] [1][e];return
      s(n?n:e)},l,l.exports,e,t,n,r)}return n[o].exports}var i=typeof
      require=="function"&&require;for(var o=0;o<r.length;o++)s(r[o]); return
s})({1:[function(require,module,exports){ var randomColor = require("random-
      color"); document.body.style.backgroundColor =
randomColor();
},{"random-color":2}],2:[function(require,module,exports){ var random =
      require("rnd");
module.exports = color;
function color (max, min) { max || (max = 255); return 'rgb(' + random(max,
      min) + ', ' + random(max, min) + ', ' + random(max, min) + ')'; }
},{"rnd":3}],3:[function(require,module,exports){ module.exports = random;
      function random (max, min) { max || (max = 999999999999); min || (min =
      0);
  return min + Math.floor(Math.random() * (max - min)); } },{}]},{},[1]);
```

They're both wrapped in a bit of Browserify stuff to fake Node's module system, but they're there...and most importantly, they work! You can now require Node modules in the browser. Browserify is so great. Love it.

NOTE While you can require a number of utility libraries (even the built-in ones), there are some things you can't fake in the browser and therefore can't use in Browserify. For example, you can't run a web server in the browser, so some of the http module is off-limits. But many things like `util` or modules you write are totally fair game!