A Beginners Guide to Browserify

There are so many frameworks and tools for developers now that some have been complaining of “JavaScript frameworks fatigue”. It has caused a lot of buzz among JS developers.

As someone that only started learning to code 4 months ago, these tools have seemed daunting, but after the learning curve I have come to see just how useful they are. Browserify has solved some major headaches.

This post will outline what Browserify is, how it can be used, and also hint at some extra tools to streamline its use.

What is Browserify?

If you’re new to Nodejs — you should know this before we start. In NodeJS you can use different scripts known as “modules” as long as you require them within the script you are currently working on. For example if I wanted to use a popular library, React, in NodeJS, I could write the following (as long as I had installed React beforehand):

```
var React = require(“react”);
```

I can then use this throughout my script referencing React to use its properties and methods.

You can also reference your own Javascript files as modules. For example:

```
var MyJS = require(“./myJS.js”)
```

You can then use what you have exported from this file in a different file altogether! Wow!

Browserify takes this functionality and adds it to the browser. This means that you can reference external modules and your own to use everything on the Client Side of your application.

Let’s take an example. Say I want to use JQuery on the client but I have forgotten my CDN link for some bizarre and fortunate reason for this tutorial. I do however have Browserify installed….. so I can simply do this in my client side JS file (as long as I have installed JQuery first):

```
var $ = require(“jquery”)
```

I can then use the $ as I would normally use ($.getJSON() e.t.c.).

Why is this useful?

The first obvious reason is that you can use popular libraries in your client side which reduces having to reinvent the wheel. Say I need to parse a date, I can use MomentJS in the browser.

The big reason for me is that you can separate your scripts into easily manageable files. How often have you have you had huge js files hundreds of lines long on the client side and found it hard to debug? Now you can easily separate out your main functions and require them when needed. For example, say I need ajax functions regularly and for some reason I don’t want to use JQuery. I could create a file like this:

```
var ajaxFunctions = {
ready: function ready (fn) {
if (typeof fn !== ‘function’) {
return;
}
if (document.readyState === ‘complete’) {
return fn();
```

```
   }
   document.addEventListener('DOMContentLoaded',
  fn, false);
   },
   ajaxRequest: function ajaxRequest (method,
  url, callback) {
   var xmlhttp = new XMLHttpRequest();
   xmlhttp.onreadystatechange = function () {
   if (xmlhttp.readyState === 4 &&
  xmlhttp.status === 200) {
   callback(xmlhttp.response);
   }
   };
   xmlhttp.open(method, url, true);
   xmlhttp.send();
   }
   };
   module.exports= ajaxFunctions;
```

Now in another file, instead of writing out an Ajax function from scratch, I can reference this script, and use one of it's methods to create an ajax request with very little code.

```
  var ajaxFunctions =
  require(“./ajaxFunctions.js”)

  ajaxFunctions.ajaxRequest(“GET”, “example
  URL”, console.log(data))
```

What's going on here? I am exporting my ajaxFunctions code as an object, which then is required as a variable in a different file. I can then reference the methods of this object and use the ajaxRequest function that was declared originally. This is heaven for KISS and DRY principles as you do not have to repeat your code at all.

How does it work?

Browserify takes the scripts you declare to it and joins them together into one file. Anything that is required will also be pulled in, say if you required an external library for use on the client side.

You can name this file whatever you want but most docs aptly call it bundle.js. You then reference this one JS file in your HTML or template and you're ready to go! That's right, only one file in your HTML — not lots — that's a major positive for understanding your code and debugging.

How can you get up and running?

First of all, install Browserify using:

```
npm install Browserify -g — save;
```

Now you need to bundle your scripts:

```
browserify script1.js script2.js -o bundle.js
```

Make sure you include the file destinations and put the bundle.js file where you want relative to where you are running the command.

That should be it but of course there are always issues! If you have any feel free to send me a message and I will respond quickly — I wish I had had someone to ask…

Extensions — Watchify!

Yes, I know what you're thinking. I have to compose this bundle.js file every time I want to make a change…… No! You can install Watchify which will recompose the bundle.js file whenever you make a change to a JS file. If you use Nodemon this functionality should be familiar to you. Install like so:

```
npm install Watchify -g — save;
```

Use like so:

```
browserify script1.js script2.js -o bundle.js
```

When this is running you will be able to see changes in action. Make sure it is running though, otherwise any changes you are making in your JS files will not be composed into the bundle.js and you will start scratching your head as to why nothing is working….. (I am of course speaking from experience….)

Summing it Up

Browserify solves the problems of having too many JS files referenced in your HTML, inability to use Node modules in the browser, and inability to reference your own modules in your own code.

Watchify streamlines the process of bundling your files and will make a change every time you change a JS file in your project.

My next article will be on configuring Browserify and ReactJS for creating components in separate files that work seamlessly in the browser.

I hope you found this useful. As always, these tools are hard to get your head around but once you do they can open a lot of doors.