A couple years ago I wrote a CSS Properties Data Visualization experiment to teach myself how to use D3 version 3 and how to build a tree-based layout. I had also donated the work to the Web Platform Documentation Project for their use (as far as i know it was never used).

I've stayed away from D3 in particular and data visualization in general since I completed the project because I haven't had the need to do data visualizations. I have the spare cycles to look at D3 again and, now that version 4 of D3 has been released, it's time to come back and take another look.

What version 4 brings to the table

The main thing in version 4, for me, is modularity. It makes the code bundles for D3 smaller and it forces me to learn what it is that I'm working with so I can then bundle it to use. ## Technical: Shaking the tree

D3 now leverages ES2015 modules to create a modularized build where we can download only those packages that we need for our project rather than use the complete D3 package.

During development we can install the packages we need using NPM and then, when we're ready to deploy the application to staging and/or production we can bundle it together to reduce the load in HTTP servers unable to preload or push content. We may also have to create an unbundled version for HTTP/2 servers where this is not an issue.

The NPM install process is simple, we use NPM install and make sure to save the package as a dependency, like so:

```
npm install --save d3-scale
```

You can still download and use modules manually and reference all the related dependencies manually as well. To use `d3-scale` without NPM the links in the home page would look like this:

```
<script src="https://d3js.org/d3-array.v1.min.js"></script>
<script src="https://d3js.org/d3-collection.v1.min.js"></script>
<script src="https://d3js.org/d3-color.v1.min.js"></script>
```

```
<script src="https://d3js.org/d3-format.v1.min.js"></script>
<script src="https://d3js.org/d3-interpolate.v1.min.js"></script>
<script src="https://d3js.org/d3-time.v1.min.js"></script>
<script src="https://d3js.org/d3-time-format.v2.min.js"></script>
<script src="https://d3js.org/d3-scale.v1.min.js"></script>
```

Of course you can always install the full package (whether you need all of it or not) just like you did with version 3:

```
<script src="https://d3js.org/d3.v4.js"></script>
```

Working with modules directly

In browsers that support modules directly we should be able to do something like this to import only the needed symbols for your application. In this case we import `event`, `selection`, `select` and `selectAll` from the `d3-selection` module and the entire `d3-transition` module.

```
<script type="module">
export {
  event,
  selection,
  select,
  selectAll
} from "d3-selection";

import "d3-transition";

// Rest of the modules goes here

</script>
```

Demo: CSS Property tree revisited

This project was one of the first things I did in D3. I'm revisting it as a way to explore how to identify modules we need to import to accomplish a given task.

Doing it for one project is easier than doing it for a larger project.

The idea is that rather than use version 3 of the D3 library I'll create a customized bundle of version 4 and link to that as my D3 library for the project. This will test my ability to figure out what to bundle, the bundling of modules themselves and whether the resulting bundle works or not; Unless there are major changes to the codebase, this should work as is.

The steps to bundle our custom D3 library are:

1. Install Rollup as a global Node application
2. Initialize `package.json` to hold our D3 modules
3. Install Rollup and dependencies for the project using NPM
4. Identify the components to bundle
5. Install the packages that contain the components identified above using NPM
6. Create `index.js` containing all the files to bundle
7. Create the rollup configuration file
8. Run Rollup with your configuration

Install Rollup and Uglify as global Node applications

Just to make sure I don't forget later in the process, I'll install Rollup as a global Node application. This will put it in my shell's path and will save me from typying full paths to my project's `node_modules` bin directory

```
npm install -g rollup uglify-es
```

initialize a package.json file

If this is a new project you need to initialize a `package.json` file to store dependencies and other configuration options. We'll run the command with the `--yes` flag to take all the default values without typing them in. We cam

```
npm init --yes
```

Install Rollup and dependencies for the project using NPM

We've already installed Rollup as a global plugin for N

```
npm i -D rollup rollup-plugin-node-resolve uglify-es
```

Identify the components to bundle

Working from the API docs I've identified the following items to bundle to get the application working.

`d3.hierarchy` contains the tree layout we use in the project and `d3.selection` holds the navigation and data related utilities we need to make the layout work. `d3.request` contains the json function that will load the data for the project. `d3-collection` and `d3-dispatch` are imported by `d3-request`.

- d3.selection
    - selection.select - select a descendant element for each selected element
    - selection.selectAll - select multiple descendants for each selected element
    - selection.attr - get or set an attribute
    - selection.style - get or set a style property
    - selection.append - create, append and select new elements
    - selection.data - join elements to data
    - selection.enter - get the enter selection (data missing elements
    - selection.exit - get the exit selection (elements missing data)
- d3.hierarchy
    - tree
- d3.shape
    - link
- d3.request
    - json
- Required by d3.request
    - d3-collection
        - map
    - d3-dispatch
        - dispatch

Install the packages that contain the components identified above using NPM

Now that we've identified the packages we need for the bundle, the next step is to install them using NPM. This is the reason why we initialized a `package.json` earlier... now we need to install the files and save themas development dependencies. The command looks like this:

```
npm install -D d3-hierarchy d3-selection d3-request \
d3-collection d3-dispatch
```

You don't have to save them but in doing so we ensure that the build is repeatable without having to run the installer on every machine we move this build to. It also ensures that we can upload the application to version control without having to download the packages every time.

TODO: Figure out how deep you need to import... right now it looks like a nightmare

Create `index.js` containing all the files to bundle

`index.js` is the entry point to the bundle we're creating. Rollup will use this entry point when creating the bundle later on. The file itself is a simple Javascript file that contains all the module imports we need for the project.

```
export {
  style,
  event,
  selection,
  select,
  selectAll
} from "d3-selection";

export {
  link
```

```
} from "d3-shape";

export {
  tree
} from "d3-hierarchy";

export {
  json
} from "d3-request";

export {
  map
} from "d3-collection";

export {
  dispatch
} from "d3-dispatch";
```

Create the rollup configuration file

For this project, Rollup's configuration is fairly easy. We create a module and tell that we want to use the `rollup-plugin-node-resolve` plugin. This will enable Rollup to locate modules using the Node resolution algorithm to search for third party modules in node_modules

In the file (`rollup.config.js`) we import the resolve symbol from the plugin and then create an export default that uses `node()` as a plugin.

```
import resolve from 'rollup-plugin-node-resolve';

export default {
  plugins: [ node() ]
};
```

Run Rollup with your configuration

Ok, now that'we gone through the pain of configuration the rest will be, almost, anticimatic. We need to run both the rollup and Uglify commands to generate the minimized version of the bundle to use.

The command is shown below:

```
rollup -c -f umd -n d3 -o d3.js -- index.js && uglifyjs d3.js -c -m -o d3
```

To make it easier to digest we'll discuss each individual command separately. We start with `rollup`. In this command we want to generate the bundled `d3.js` file so we use the following configuration flags.

Short Option	Long Option	Description
-c	--conftig	Use this config file. if argument is used but value is unspecified, defaults to `rollup.config.js`
-f	--format	the output format. For this example we're using `umd`
-n	--name	name of bundle in UMD/IIFE output
-o	--output	output. If absent, prints to stdout
-m	--sourcemap	generate sourcemap

This will create the bundle but it's still to big for my liking so I'll run it through uglify-es to shrink it further.

The flags we use for uglify are:

Short Option	Long Option	Description
-c	--compress	Enable compressor
-m	--mangle	Mangle names
-o	--output	Output file path (default STDOUT)

Rather than write the command out every time we can add a script to the

`package.json` file to do this for us. We'll call the script `bundle` and add it like so:

```
  "scripts": {
    "bundle": "rollup -c -f umd -n d3 -o d3.js -- index.js && uglifyjs d3
  },
```

and instead of typing that entire command every time we want to change the bundle, we can just type the following in terminal:

```
npm run bundle
```

And it will execute the command specified.

The final test is to change the link in the HTML file to point to our bundle and see if it still works. If it does then we're good, if not we'll have to gnp back to the drawing board and figure out why.

There are significant changes to make to the existing code and I'm having a hard time wrapping my head around the new way of working with the code.

Conclusion

This does a lot. It reduces the size of the minimized bundle to ***25k*** which can be further reduced if your server can send gzipped content and your client can read it.

Links and Resources

- Changes documentation in the D3 distribution
- D3 V4 - What's new?
- D3 Version 4 API
- Examples of using Rollout to create custom D3 bundles
    - Example 1
    - Example 2
- Additional information
    - Rollup
    - Uglify ES