# adding browserslist support to your project

A lot of the tools in my projects like `Autoprefixer`, Babel's `preset-env` and `preset-modules` use browserslist (yes that's the name, no typo) to decide if adding prefixes or transpiling is required.

The hardest thing for me to decide is what versions to support and whether they will be the same for all projects or all tools.

For example, I'm all for supporting ES2017 and earlier in my projects. ES 2017 support means that we're supporting all the features I would like to use in my projects without having to be afraid of substantial differences in support or broken implementations (broken implementations are always possible but less likely).

The table below shows the earliest browser that supports module scripts and the full set of Ecmascript 2017 features.

| Feature Supported | Chrome | Edge | Firefox | Safari |
|---|---|---|---|---|
| <script type="module"> | 61+ | 16+ | 60+ | 10.1+ |
| All ES2017 features (minus atomics+shared memory) | 58+ | 16+ | 53+ | 10.1+ |

This could be easily included in a `.browserslistrc` configuration file or as part of a `package.json` file. A `.browserslistrc` file for the browser versions that support the `<script type="module">` looks like this:

```
Chrome > 61
Edge > 16
Firefox > 60
Safari > 10.1
```

and the equivalent JSON that can be inserted in your `package.json` implementation, looks like this:

```
  "browserslist": [
    "Chrome > 61",
 "Chrome > 61"",
    "Firefox > 60",
    "Safari > 10.1"
  ]
```

We can use the browserslist configuration as the first step when we use the `module/nomodule` pattern.

Using these settings as the compilation target we'll get a module that will work in all our target browsers and be ignored by browsers that don't support modules.

A second compilation target can address the browsers that don't support modules and that will likely not support the language features we want to use.

Yes, I am aware of the small gap between full ES2017 support and module type script support. I'm OK with it; evergreen browsers will continue to update to where it becomes irrelevant.

Publish, ship, and install modern JavaScript for faster applications) provides examples of how we might do this using WebPack and other tools.