Progressive Enhancement Matters

> **Progressive Enhancement** is a powerful methodology that allows Web developers to concentrate on building the best possible websites while balancing the issues inherent in those websites being accessed by multiple unknown user-agents. Progressive Enhancement (PE) is the principle of starting with a rock-solid foundation and then adding enhancements to it if you know certain visiting user-agents can handle the improved experience. PE differs from Graceful Degradation (GD) in that GD is the journey from complexity to simplicity, whereas PE is the journey from simplicity to complexity. PE is considered a better methodology than GD because it tends to cover a greater range of potential issues as a baseline. PE is the whitelist to GD’s blacklist. Part of the **appeal of PE is the strength of the end result**. PE forces you to initially plan out your project as a functional system using only the most basic of Web technologies. This means that you know you’ll always have a strong foundation to fall back on as complexity is introduced to the project.
>
> — Progressive Enhancement: What It Is, And How To Use It?

Progressive enhancement (PE) is the principle of starting with a rock-solid foundation and then adding enhancements to it. Every so often the discussion between PE, its cousing Graceful Degradation (GD) and the need to provide such featurees will rear its head in debates about Javascript not being available or not needing to think about PE or GD in a world where browsers are evergreen.

This is not a post on what is Progressive Enhancements. If you’re interested in that area, Aaron Gustafson wrote three articles on the subject, the fist of which is a good introduction to the subject. The articles are:

1. Understanding Progressive Enhancement
2. Progressive Enhancement with CSS
3. Progressive Enhancement with JavaScript

How we build a base experience?

One of the hardest things to decide when it comes to progressive enhancement is what makes for a core experience. How do we decide on the baseline we want to provide to all our users?

It would be tempting to make the baseline modules and grid, but that wouldn't be as useful as we think.

Grid is supported in all modern browsers so it's not an issue moving forward. However there are three cases to be made for providing grid as a progressive enhancement:

- Earlier versions of our evergreen browsers did not support the feature
- Browsers like IE and Opera Mini don't support Grid at all (and save yourself the comment on how not even Microsoft supports IE… there are plenty of people still using it)
- Having to work around interoperability bugs makes

If you want to write defensive CSS you can use feature queries like this.

```css
div {
  float: right;
}

@supports (display: grid) {
  div {
    display: grid;
  }
}
```

If you're using Javascript then the way to check for CSS feature support looks like this:

```javascript
hasGrid = CSS.supports("display: grid");

if (hasGrid) {
  console.log('We support grid');
  'We support grid'// Do something for browsers that support gridle.log('
  // Do something f'Grid is not supported'// Do something for browsers th
}
```

I’m not saying don’t use Grids… quite the opposite. I’m saying to provide a base experience that will support as many browsers as possible and then use Grids as an enchancement for those browsers that can work with them.

In Javascript take `modules` and `async/await` as an example. All versions of modern browsers that support modules support async/await but not all versions of modern browsers that support async/await support modules. So you get to decide which supported features are more important for your application.

Another question that you need to ask is whether transpilation is needed for your target browsers. Tools like Babel will convert your modern JavaScript (ES2015 and later) into an older version of Javascript for browsers that don’t support. Using the env preset and a list of the oldest browser versions you want to support you can write your code once and let Babel deal with making the code work in your older supported browsers.

The biggest challenge is how to make the transpiled bundles as performant as the original code.

How to enhance it?

As with many things in web development and design, it depends. It depends on the type of site you’re building, how much data it needs to load from the server and how you’re caching the content for future re-use.

If we are building a content site we may want to populate the base content first and then run CSS and JavaScript to enhance the base content or add additional material.

If build a catalog page for a store the most expedient way may be to create templates that get populated from data from the server. But because we are sensitive to network hicups, and a number of other reasons why Javascript may time out or otherwise fail to load, particularly in older or lower-end devices.

Once we have our core experience, one that will work without CSS and whether JavaScript is enabled or with as little Javascript as possible we can start thinking about how to enhance it and how to do it in an accessible way.

Conclusion

I know that you don’t have to make the experience identical for all devices but, to me, that doesn’t mean that we should provide a subpar experience to those browsers that “don’t cut the mustard”, particularly when you don’t have to.

> I like an escalator because an escalator can never break, it can only become stairs. There would never be an escalator temporarily out of order sign, only an escalator temporarily stairs. Sorry for the convenience.
>
> Mitch Hedberg, Comedy Central Presents

We should make our apps into escalators, not part of the wealthy western web.

Links and Resources

- Enough with the Javascript Already!
- Stumbling on the escalator
- Progressive enhancement is still important
- Progressive enhancement is faster
- Progressive Enhancement: Zed’s Dead, Baby