Browser usage share statistics in the modern web development landscape

Chris Thornton

bitsbelow.us

published 2019-02-28

At a time with evergreen web browsers and frontend tooling that autoprefixes and transpiles the latest JavaScript and CSS features down to something the user's browser will understand, it seems like browser usage share statistics (BUSS) are losing their significance.

Certainly global browser usage share statistics (GBUSS) are meaningless when applied to your website: (1) StatCounter data, the most commonly cited source of GBUSS, is derived from slightly less than 1% of the web,[1] that percentage is decreasing year-over-year,[2] and it's known to be skewed—counting hits instead of unique visits;[3,4] (2) GBUSS includes both mobile and desktop usage share yet many website's actual BUSS is either nearly all mobile or nearly all desktop; and (3) even country-specific BUSS, whether combined or filtered by desktop or mobile, is unlikely to reflect your website's BUSS.

Each niche has its own identity and demographics. Each website exists to fulfill the needs and wants of its particular niche. From this angle, the inapplicability of GBUSS to individual websites isn't surprising. However, this perspective seems to be overlooked when it comes to using GBUSS to justify using a particular feature, especially in our own projects.

Global skepticism

When you check a feature's viability with CanIUse, you should be skeptical. CanIUse uses StatCounter's skewed "global" BUSS for desktop and its mobile data is opaquely "extrapolated from other sources"[5]. Despite how brilliantly StatCounter's marketing plays off of our drive to have a website with global reach, StatCounter's "Global Stats" is only applicable to StatCounter and its clients, not to you and not to your niche. The next time someone states you should or should not do something according to GBUSS, you should be skeptical. Citing GBUSS does not necessarily disqualify a claim, it's just that GBUSS ought to be ignored rather than assumed to be applicable to your website. There are better options and they're readily available.

Real world browser usage patterns seem to be more niche-market based rather than having a global or geographic pattern. Corporate and business-to-business websites generally have higher Edge, Chrome

for Enterprise, maybe IE11, and maybe Firefox ESR usage. Technology-related websites tend to have users that adopt the latest browser versions the moment they're released paired with a small percentage of users running atypical browsers, such as Brave, w3m, and lynx. On fashion, grooming, and make-up websites you'll likely see higher mobile and less desktop traffic.

The next time you're on CanIUse go to "Settings" then to "Import" and import your Google Analytics visitor statistics,[6] or from the same "Settings" menu, toggle the appropriate browsers for your website. The inapplicability of GBUSS to your website isn't really CanIUse's fault. Showing some BUSS by default seems to be better than not showing anything at all. But perhaps more attention should be given to the importance of using your own BUSS.

In build pipelines

If you have a JavaScript build pipeline then there is a good chance your frontend toolchain is driven by StatCounter's GBUSS. Browserslist[7]—"[t]he config to share target browsers and Node.js versions between different frontend tools"—uses CanIUse. Babel, PostCSS, their plugins, and many other frontend tools use Browerslist, and consequently, rely on CanIUse and StatCounter's GBUSS under the hood.

If Browserslist is in your toolchain then understand what your query is doing, use Browserslist-GA,[8] or supply a custom dataset[9]. A properly tuned Browserslist query provides your toolchain with the information it needs to transpile and polyfill only the features not present in your users' browsers. In the best case scenario, this reduces the size of your JavaScript bundle. In the worst case, you've found the upper limit on your bundle's size: you're now serving a JavaScript bundle that is as large as it needs to be and no larger[10].

In project management

Beyond frontend tooling, project management continues to be the place where website-specific BUSS is most useful. BUSS is a powerful aid for determining (1) when and how to implement a particular feature; and (2) when legacy code is able to be refactored or removed.

When we write code, we are limited by the tools available in our users' current web browsers. Some features can be taken advantage of immediately and do not interfere with older browser versions, e.g., features unlocked via HTTP headers, such as `Content-Security-Policy` or `Feature-Policy`. Some features require the usage share of an older browser to decline before any hacks, workarounds, or less-than-ideal approaches can be refactored using a relatively new feature.

Consider all the less-than-ideal approaches to layouts: table-based layouts (either HTML tables or CSS's `display: table`), float-and-clear layouts and their variants, and inline-block-based layouts. You didn't need flexbox to implement a holy-grail style layout[11]. It used to be done with tables and it's been

done with floats, but all approaches had their fair share of bugs and maintainability problems. Flexbox just happened to make that particular layout less buggy, easier to implement, and easier to maintain.

I document these less-than-ideal situations with `*-misuse` tags. For example, when I coerce nested flexboxes into a multicolumn layout, I create an issue and tag it as `flexbox-misuse`. This way, I can find code that may be cleaned up once CSS grid or CSS multicol is viable according to my website's BUSS. Hacks and workarounds for specific browsers can be handled in a similar way. When adding a `<meta http-equiv='X-UA-Compatible' content='ie=edge'>` to your codebase, document this as a workaround for IE10 and earlier by creating an issue and tagging it `ie<=10-workaround`. Once IE10 is practically nonexistent according to your website's BUSS, find all issues tagged `ie<=10-workaround` and remove the workarounds.

Issues tagged `*-misuse` and `*-workaround` are cheap wins for reducing page weight, providing a smoother user experience, and easing maintenance work. For this to workout, you do need to stay up-to-date with modern web development features though. Time spent doing that certainly isn't free.

BUSS measurement and accuracy

If you need BUSS data and you aren't using analytics then you can find the raw hits and user-agent strings in your server logs. However, before tallying it up, realize that BUSS is harder to measure than it seems. The `User-Agent` header, which BUSS is typically derived from, is easily spoofable. Furthermore, many users, corporate environments, and even web browsers block JavaScript trackers.[1][2]

It's hard to get good data on public facing websites, on websites where the term "user" is fuzzy, and on websites that rely heavily on proxying or caching. For SaaS products, BUSS for logged in users should be easy to obtain, but you might benefit from a separate cohort for leads to make sure you're not excluding potential clients. Don't expect your BUSS to be 100% accurate. It will not be, even if you're paying for analytics. At an individual level BUSS is effectively meaningless. At a userbase level, it's assumed that not all your users are spoofing their `User-Agent` request header or blocking trackers, and therefore your BUSS has *some* meaning.

Don't let the inherit inaccuracy of BUSS lead you into thinking GBUSS are sufficient. It's not.

Conclusion

Unfortunately, we generally treat GBUSS and country-specific BUSS as if it had any relevance to our own websites. When we make "informed" decisions based on GBUSS we play into the idea that as an industry, we're data-driven, no matter how bad the data is.

That said, BUSS continues to be a fixture in the web development landscape. The way we interact

with BUSS has changed. Its become an obscure yet important piece in a huge, sprawling toolchain. But its use-case is still the same: informing us about the current and future viability of particular features so that we can better meet our users' wants and needs.

Thank you for reading, and happy hacking.

---

1. See the "Historical trend" graph on <w3techs.com/technologies/details/ta-statcounter>.
2. Same as 1 above, <w3techs.com/technologies/details/ta-statcounter>, but notice the graph has a downward trend. There is also the "Historical trends" report at <w3techs.com/technologies/history_overview/traffic_analy which appears to be the raw numbers used in the "Historical" graph.
3. From <gs.statcounter.com/faq#page-views-uniques>: "We measure internet usage trends. To accurately measure usage, we have to base our stats on page views (and not unique visitors)". StatCounter's FAQ even includes a video explanation. I disagree with their counting method. I think browser usage share is more accurately described by unique visits than page views. Page views skews data in favor of users and web browsers that send the most HTTP requests. But I agree with the difficulties StatCounter raises in regards to calculating BUSS, which is also why I don't think their statistics are useful for most people, and why I think their statistics are not what I'd consider to be "global". My disagreement comes down to an argument about what "usage" means and what browser usage share should represent.
4. Sampling bias in StatCounter's Global Stats data was spotlighted by <wikipedia.org/wiki/StatCounter>, and from there I verified that the claims are accurate.
5. "Information for mobile versions is extrapolated from other sources" from <caniuse.com/usage-table>.
6. I don't use Google Analytics. I'm stating this because Google Analytics is ubiquitous. Furthermore, I do not know what CanIUse's terms of use or privacy policy is when it comes to importing your data. I assume it's reasonable but find out for yourself.
7. See <github.com/browserslist/browserslist>
8. I don't use Google Analytics, so I'm assuming this works. It seems to be relatively popular. See <github.com/browserslist/browserslist-ga>.
9. See <github.com/browserslist/browserslist#custom-usage-data>.
10. From the perspective of features transpiled. We're omitting the conversation about whether all that JavaScript is necessary. The point is, once your users have native support for features like generators, async/await, ES2015 classes, etc., they no longer need to be transpiled, and this has a significant impact on bundle size.

11. But flexbox does make holy-grail style layouts significantly easier. For a great overview on the subject, see <wikipedia.org/wiki/Holy_grail_(web_design)>.
12. Blocking tracking is a certainly a good thing. I have mixed feelings on collecting `User-Agent` strings. As a privacy absolutist, I think code should work in "any browser" and websites shouldn't have to collect this data at all. On the other hand, as a developer, I value my time and engineering effort. Having BUSS helps me make informed decisions. Ultimately, there is real value in collecting some user or device-related data. We should focus on ways data can be collected that are morally acceptible and preserve privacy of the end-user. Collection efforts, such as: (1) only collect exactly the things you need and no more; (2) make aggregated data available from a high-level, e.g., minimum bucket sizes of 15k or 100k users and classify anything not meeting that criteria as "other"; (3) don't make the raw data available; (4) round timestamps to the nearest day or hour, if they're collected; (5) do not associate the data with an IP address or user; (6) limit collection into cycles, e.g., four weeks, and after these four weeks, keep the aggregated data and wipe all the raw data; (7) inform the user exactly what is being collected and for how long; and there are probably many other things that could be done.