

WUNDERLANDMEDIA

The Polyfills You Can Finally Delete in 2026

Every status verified against api.webstatus.dev, not against a release announcement.

By Kemal | wunderlandmedia.com

20

Interop 2026 Focus Areas

3

Baseline Tiers

120M+

Weekly npm Downloads Still Shipping

1 Newly Available Is Not Widely Available

This distinction is the whole document

Everything else in here is a date. This page is the reason the dates matter. A feature that went Newly available in March 2026 is safe for a developer dashboard and genuinely risky for a checkout flow.

THE THREE BASELINE STATES

1a Limited availability

At least one of the four core engines does not support it. Keep the library.

1b Newly available

All four engines support it in their current stable versions, desktop and mobile: Chrome, Edge, Firefox, Safari.

1c Widely available

30 months have passed since the newly available date. That is the only difference between the two green badges.

30 months between Newly available and Widely available. It is not a formality. It is the industry's rough estimate for how long a feature takes to reach roughly 95% of users globally.

WHY THE GAP EXISTS

- ☐ **The day Safari ships is not the day Safari users have it**
Shipping a feature and users receiving it are two separate events, months to years apart.
- ☐ **People do not update**
Not out of stubbornness. Out of indifference and notification fatigue.
- ☐ **Devices stop getting updates**
An Android phone two years past its support window is frozen at whatever it had.
- ☐ **Corporate fleets freeze versions**
Managed IT environments pin browser versions deliberately. Your B2B audience lives here.

WHICH TIER YOUR AUDIENCE NEEDS

- ☐ **Internal tooling or a developer dashboard**
Newly available is fine. You know the browsers, they are current, and you can tell people to update.
- ☐ **Client marketing site with a broad consumer audience**
Widely available for anything structural. Newly available only where it degrades gracefully.
- ☐ **Checkout, forms, or anything that takes money**
Widely available only. A layout bug here is not a cosmetic problem.
- ☐ **Heavy traffic from older Android devices**
Same feature, same date, opposite decision from a US enterprise SaaS. Check your own numbers on page 8.

The failure mode to avoid

Ship a Newly available feature to a client site, get a bug report from someone on an older iPhone, spend a day confused about it, because the badge on the documentation page said green. When someone writes "X is now Baseline," ask which kind.

INTEROP 2026: 20 FOCUS AREAS, NOT 26

- ☐ **The number is 20, plus four investigation efforts**
The Interop 2026 README in the web-platform-tests repo lists 20. Fifteen are new, five carry over from 2025. The widely repeated 26 is wrong.
- ☐ **An Interop focus area is a warning, not an endorsement**
The browsers picked it because the pass rates are uneven. Scroll-driven animations being on the 2026 list means it is not done.
- ☐ **Only five of the 20 touch normal client work**
Dialogs and popovers, view transitions, scroll-driven animations, scroll snap, and `attr()`. You are not shipping JSPI for Wasm on a dentist's website.

2

Delete Today Without Thinking

Baseline Widely available, no asterisk

Every date below was verified against `api.webstatus.dev`, not a blog post. These have all cleared the 30-month rule. There is no audience-dependent judgement call on this page.

Feature	Widely available since	What it replaces	Polyfill still shipping (weekly)
<code>scroll-behavior: smooth</code>	14 Sep 2024	Smooth-scroll shims and scroll animation helpers	<code>smoothscroll-polyfill</code> 1,125,110
<code><dialog></code>	14 Sep 2024	Your modal library, for most simple cases	<code>dialog-polyfill</code> 253,743
Cascade layers	14 Sep 2024	Specificity conventions, part of what BEM compensated for	None (conventions, not a package)
<code>Array.at()</code> , <code>structuredClone()</code>	14 Sep 2024	Legacy JS transpilation for browsers nobody uses	<code>core-js</code> 71,000,000
Container queries	14 Aug 2025	ResizeObserver plumbing written to fake them	<code>container-query-polyfill</code> 74,300
<code>color-mix()</code>	9 Nov 2025	Sass colour functions, and the build step kept alive for them	None (a whole toolchain)
Subgrid	15 Mar 2026	Nested-grid hacks and <code>display: contents</code> tricks	None
CSS nesting	11 Jun 2026	The main reason small projects still run a preprocessor	None (a preprocessor)
<code>:has()</code>	19 Jun 2026	Class-toggling JavaScript for parent selection	<code>css-has-pseudo</code> 7,700,000
Scroll snap	15 Jul 2022	Carousel libraries whose only job is horizontal snapping	Assorted carousel libs

71,000,000

`core-js` downloads last week. Most projects ship it because a build config from 2019 targets browsers nobody uses. That is not a polyfill decision, it is a `browserslist` decision. See page 7.

WHAT TO ACTUALLY DO WITH THIS PAGE

- ☐ **Grep your lockfile for all five polyfill names above**
Direct dependency or transitive, they are all deletable. Start with the lockfile, not `package.json`.
- ☐ **Check whether `postcss-preset-env` is still transpiling `:has()`**
`css-has-pseudo` pulls 7.7 million weekly downloads mostly as a passenger inside `preset-env`, not because anyone chose it.
- ☐ **Delete the modal library if it only does open, close, and backdrop**
A quarter of a million people a week are still installing `dialog-polyfill` for behaviour the browser has had since 2024.
- ☐ **Kill the Sass build step if colour maths was its last job**
`color-mix()` plus relative colour syntax covers most of it, and nesting covers the rest.
- ☐ **Audit carousels before anything else**
Scroll snap crossed Widely available four years ago, in July 2022. It is the single biggest win in a typical cleanup pass.

3

Newly Available: Ship With Judgement

All four engines, but not aged 30 months

Ship these if you understand your audience, not because a badge told you to. The "safe if" column is the actual decision rule.

Feature	Newly available	What it replaces	Safe if
<code>text-wrap: balance</code>	13 May 2024	<code>balance-text</code> and headline-balancing scripts	Always. Pure progressive enhancement, degrades to normal wrapping.
<code>light-dark()</code>	13 May 2024	Duplicated custom properties in your dark mode setup	You keep a fallback colour declared before it.
Relative colour syntax	16 Sep 2024	More Sass colour maths	Your palette has static fallbacks for the computed values.
Popover API	27 Jan 2025	Show-hide logic and light-dismiss handling in tooltips and dropdowns	Only after testing. It does not degrade: content is just visible.
View transitions	14 Oct 2025	Page-transition animation libraries	Always. Degrades to a normal instant swap.
Trusted Types	24 Feb 2026	Ad-hoc DOM sanitisation	You have audited your CSP and third-party scripts first.
<code>@scope</code>	24 Mar 2026	Scoping conventions and CSS-in-JS scoping	Not on a client project's critical path. Six months old.
<code>contrast-color()</code>	10 Apr 2026	Manual light/dark text-on-background calculations	You declare an explicit colour as the fallback.
<code>field-sizing</code>	16 Jun 2026	The auto-growing textarea script	Degrades to a fixed-height textarea. Low risk, but three months old.

THE TWO WORTH EXTRA CARE

- ☐ Popover API: test the unsupported path before shipping
- On a browser without it, the popover content is simply visible on the page, which looks broken rather than plain. Gated by Safari iOS 18.3.
- ☐ Trusted Types: the Baseline date is the least interesting part
- What matters is what it does to your CSP and your third-party scripts once you turn it on.

GOOD REASONING

View transitions had Chrome in March 2023 and Safari in September 2024. The whole gap was Firefox, closed by Firefox 144 in October 2025. It degrades to an instant swap, so the downside of being early is nothing.

BAD REASONING

"View transitions are here" was written for two and a half years while an entire engine did not have them. Chrome shipping a feature is not the feature being available.

"I like `@scope` a lot and I am not putting it in a client project's critical path yet. It is a judgement call dressed up as a rule, and I would rather admit it than pretend the dates make the decision for me."

Kemal, Wunderlandmedia

4

Where the Announcement Was Wrong

Five features the ecosystem rounded up

Somebody shipped it in Chrome, wrote about it well, and "available" quietly became "safe." Every status below is what `api.webstatus.dev` returns, not what the coverage says.

Scroll-driven animations are still Limited availability

Not Newly available. Limited. Chrome shipped them in July 2023, Safari 26 in September 2025, and Firefox has not shipped them at all. `api.webstatus.dev` returns no Firefox entry. It is an Interop 2026 focus area precisely because it is not done. If you have seen a "scroll-driven animations are Baseline now" post, it was wrong, or it was describing Chrome. Keep the library.

`text-wrap: pretty` is Limited availability

Chrome 117 in September 2023, Safari 26 in September 2025, no Firefox. Its sibling `text-wrap: balance` reached Newly available in May 2024 and `pretty` did not. The two get written about together constantly, which makes this the easiest trap on the list to fall into.

The Temporal API is Limited availability

Firefox 139 in May 2025, Chrome 144 in January 2026, and no Safari. So the answer to "can I finally delete `moment`" is: yes, but not by using Temporal natively. You still need `temporal-polyfill` (3.5 million weekly downloads), or you use `Intl.DateTimeFormat` and native `Date` for the 80% of cases that are just formatting.

`accent-color` is Limited availability

Which I did not believe until I read the browser table. Chrome desktop 93, Firefox 92, Edge 93, Safari 26.2 in December 2025, and no entry for Chrome on Android. One missing row in one browser on one platform, and the whole feature sits below the line. Worth knowing before you rely on it for a form's visual identity.

CSS anchor positioning is contested

`api.webstatus.dev` reports the composite feature as Limited, some coverage in early 2026 called it Newly available, and there is an open argument in the `web-features` repo about how the feature should be split. The disagreement is real, and it is about how the feature is defined, not about who is lying. Full breakdown in [CSS Anchor Positioning Shipped Everywhere. Here's What You Can Delete.](#)

WHAT TO KEEP, AND WHY

- ☐ **Keep the scroll animation library**
Firefox has no implementation at all, so there is no fallback to degrade to. This is not a judgement call.
- ☐ **Use `text-wrap: pretty` as an enhancement only**
It costs nothing where it works and does nothing where it does not. Just do not treat it like its sibling.
- ☐ **Keep `temporal-polyfill`, or skip Temporal entirely**
For the 80% of cases that are just formatting, `Intl.DateTimeFormat` plus native `Date` replaces `moment` today with no polyfill at all.
- ☐ **Do not build a form's visual identity on `accent-color`**
One missing row for Chrome on Android. Fine as a nicety, wrong as the thing that makes the brand legible.
- ☐ **Re-check all five in six months**
Four of them are one browser release away from moving. The status is a snapshot, not a property of the feature.

WHAT A CORRECT CLAIM LOOKS LIKE

"Scroll-driven animations are Limited availability: Chrome July 2023, Safari 26 September 2025, no Firefox. An Interop 2026 focus area."

WHAT THE ECOSYSTEM WROTE INSTEAD

"Scroll-driven animations are Baseline now." Somebody shipped it in Chrome, wrote about it well, and "available" quietly became "safe."

36,910,362

`moment` downloads last week, for a library that has been in maintenance mode for years, waiting on a Temporal API that Safari has not shipped.

5 How to Check Instead of Guessing

"Check the API, not the announcement, then check your own traffic."

Kemal, Wunderlandmedia

QUERY API.WEBSTATUS.DEV DIRECTLY

- ☐ **Look up the feature by its web-features ID**
One request returns the Baseline status, both dates, and the per-browser version table. That is everything the decision needs.
- ☐ **Read the `baseline.status` field, not the headline**
It returns one of three values: `limited`, `newly`, or `widely`. There is no fourth state and no "mostly."
- ☐ **Read the browser rows individually when the status surprises you**
This is how `accent-color` turns out to be Limited: one missing row for Chrome on Android, invisible in any summary.
- ☐ **Check both dates, not just the newly-available one**
`baseline_low_date` is Newly available. `baseline_high_date` is Widely available, 30 months later. Most coverage quotes the first and calls it Baseline.

```
# One feature, everything you need to decide
curl "https://api.webstatus.dev/v1/features/container-queries"

# Everything that is still Limited in a search
curl "https://api.webstatus.dev/v1/features?q=baseline_status:limited"

# The three fields that matter in the response
baseline.status      -> "limited" | "newly" | "widely"
baseline.low_date    -> "newly available" date
baseline.high_date   -> "widely available" date, 30 months later
```

READING THE BASELINE DASHBOARD

- ☐ **The dashboard is the same data with a UI on it**
Use it to browse, use the API to verify. They cannot disagree, because they are the same source.
- ☐ **Cross-check anything on the Interop 2026 list**
Being a focus area means the pass rates are uneven. Look at the actual per-engine support before you trust it.
- ☐ **Treat a green badge in documentation as a prompt to check, not an answer**
The badge does not tell you which of the two green states it means, and that is the entire decision.

A blog post is not a source

Release announcements are written by the people who shipped the thing, and they are optimistic by design. I was told Interop 2026 had 26 focus areas and repeated it before checking. It has 20. That is exactly the habit this document argues against, and I am not exempt from it.

SOURCE

`api.webstatus.dev`, the `web-features` repo, and the Interop README in `web-platform-tests`. Machine-readable, dated, and arguable in public.

NOT A SOURCE

A vendor release post, a conference talk, a bookmarked article from last year, or a screenshot of a support table with no date on it.

6 Browserslist in Practice

This is the fix for the core-js problem

Browserslist supports Baseline queries directly now. Most projects ship legacy JavaScript to modern browsers because of a config nobody has touched in four years, not because they need to.

Query	What it resolves to	Use it when
baseline widely available	The browser versions that have every Widely available feature: everything past the 30-month line.	Client work. The safe default for anything with a broad public audience.
baseline newly available	The current stable versions of all four engines. The aggressive setting.	Internal tools, dashboards, and audiences you can actually name.
baseline 2025	Everything that reached Newly available by the end of 2025.	You want a pinned, reproducible target that does not drift as browsers ship.

```
// package.json — the safe default for client work
"browserslist": [ "baseline widely available" ]

// Aggressive: internal tooling, known audience
"browserslist": [ "baseline newly available" ]

// Pinned to a year, does not drift as browsers ship
"browserslist": [ "baseline 2025" ]

# See what any of them actually resolves to
npx browserslist "baseline widely available"
```

THE MIGRATION, IN ORDER

6a Print what your current config resolves to

Run `npx browserslist` with no arguments. Most people are surprised by what is still in the list.

6b Find where the old config came from

Usually a framework default from 2019, a copied `.browserslistrc`, or a `> 0.5%`, `last 2 versions`, `IE 11` string nobody re-read.

6c Swap in a Baseline query and diff the resolved list

Compare before and after. The features you lose support for are the ones to evaluate, not the whole config.

6d Rebuild and check the bundle size delta

This is where `core-js` stops being injected. The number is usually large enough to end the debate on its own.

6e Check `postcss-preset-env` against the same target

It reads browserslist too. Fixing the target stops it transpiling `:has()`, nesting, and cascade layers you no longer need transpiled.

6f Remove the now-unused polyfill packages

Changing the target stops the injection. It does not remove the direct dependencies. Do that pass separately, from the lockfile.

GOOD

```
"browserslist": ["baseline widely available"]
```

Tied to a public, dated definition that updates itself as the 30-month line moves.

BAD

```
"browserslist": [ "> 0.5%", "last 2 versions", "IE 11" ]
```

Written in 2019, never re-read, and quietly responsible for 71 million weekly `core-js` downloads.

7 Checking Your Own Audience

95% of users **globally**. That is what the 30-month rule is calibrated to. Your users are not global, so the rule is a starting point, not an answer.

THE THREE-MINUTE CHECK

7a Open your analytics and go to browser and OS version

Version, not browser name. "Safari 12%" tells you nothing. Safari 15 versus Safari 18 is the entire question.

7b Set the window to the last 30 days

Long enough to be representative, short enough that it reflects who is on the site now rather than last year.

7c Find the version that shipped the feature

The per-browser table on `api.webstatus.dev` gives you the exact version number for each engine.

7d Sum the sessions below that version

One number. That is the percentage of your traffic that will not get the feature.

7e Apply the threshold

Under 0.5% and it degrades gracefully: ship it. 4% and it breaks the layout when missing: do not. That is the whole decision.

7f Repeat per feature, not per project

Three minutes each. The threshold moves with how badly the feature fails, so a graceful one and a breaking one get different answers on the same traffic.

SHIP IT

B2B SaaS, US enterprise audience, 0.2% of sessions below the version that shipped `text-wrap: balance`, and the failure mode is slightly worse line breaks.

DO NOT SHIP IT

Consumer site, heavy traffic from older Android devices, 4% below the version that shipped the Popover API, and the failure mode is a tooltip permanently visible mid-page.

Do not use a global support chart as your answer

A B2B SaaS with a US enterprise audience has a completely different Safari-version distribution than a consumer site with heavy traffic from older Android devices. A global percentage describes neither of them. Your own analytics is the only chart that describes your users.

The direction of the error

Almost nobody is being too cautious. The bundles I open are full of workarounds for problems that were fixed while the project was in maintenance mode. If you are unsure whether to run this pass, that uncertainty is itself the signal.

8 Quick Reference: Every Feature

Feature	Status	Date	Verdict
Scroll snap	Widely available	15 Jul 2022	Delete the carousel library
<code>scroll-behavior: smooth</code>	Widely available	14 Sep 2024	Delete <code>smoothscroll-polyfill</code>
<code><dialog></code>	Widely available	14 Sep 2024	Delete <code>dialog-polyfill</code>
Cascade layers	Widely available	14 Sep 2024	Delete the specificity conventions
<code>Array.at()</code> , <code>structuredClone()</code>	Widely available	14 Sep 2024	Fix <code>browserslist</code> , drop <code>core-js</code>
Container queries	Widely available	14 Aug 2025	Delete <code>container-query-polyfill</code>
<code>color-mix()</code>	Widely available	9 Nov 2025	Drop the Sass colour functions
Subgrid	Widely available	15 Mar 2026	Delete the nested-grid hacks
CSS nesting	Widely available	11 Jun 2026	Drop the preprocessor
<code>:has()</code>	Widely available	19 Jun 2026	Delete <code>css-has-pseudo</code>
<code>text-wrap: balance</code>	Newly available	13 May 2024	Ship it, degrades gracefully
<code>light-dark()</code>	Newly available	13 May 2024	Ship with a fallback colour
Relative colour syntax	Newly available	16 Sep 2024	Ship with static fallbacks
Popover API	Newly available	27 Jan 2025	Test first, does not degrade
View transitions	Newly available	14 Oct 2025	Ship it, degrades to instant swap
Trusted Types	Newly available	24 Feb 2026	Audit CSP and third parties first
<code>@scope</code>	Newly available	24 Mar 2026	Not on the critical path yet
<code>contrast-color()</code>	Newly available	10 Apr 2026	Ship with an explicit fallback
<code>field-sizing</code>	Newly available	16 Jun 2026	Low risk, but three months old
<code>text-wrap: pretty</code>	Limited	No Firefox	Keep it as enhancement only
Scroll-driven animations	Limited	No Firefox	Keep the library
<code>accent-color</code>	Limited	No Chrome Android	Do not rely on it for form identity
Temporal API	Limited	No Safari	Keep <code>temporal-polyfill</code> , or use <code>Intl</code>
CSS anchor positioning	Limited (contested)	Definition disputed	See the dedicated post

The rule, in one line

Widely available means ship it. Newly available means ship it only if it degrades gracefully. Limited means keep the library.

"Almost nobody is being too cautious. The bundles I open are full of workarounds for problems that were fixed while the project was in maintenance mode, and the fix is the same every time: check the API, not the announcement, then check your own traffic."

Kemal, Wunderlandmedia

120M+

weekly npm downloads across the polyfills on this list alone, for problems the browser already solved. Some of that is yours.

Want someone to run this pass over a real project?

Read the full post at wunderlandmedia.com/polyfills-you-can-delete-2026, or get in touch at wunderlandmedia.com and I will tell you exactly what comes out of your bundle.