Google is killing the open web

Google is managing to achieve what Microsoft couldn't: killing the open web. The efforts of tech giants to gain control of and enclose the commons for extractive purposes have been clear to anyone who has been following the history of the Internet for at least the last decade, and the adopted strategies are varied in technique as they are in success, from Embrace, Extend, Extinguish (#EEE) to monopolization and lock-in.

What I want to talk about in this article is the war Google has been waging on XML for over a decade, why it matters that they've finally encroached themselves enough to get what they want, and what we can do to fight this.

A little bit of history

Google entered the browser market at a time when web development was starting to see the light again after Microsoft's “win” of the First browser war through the abuse of its operating system's monopoly by shipping its Internet Explorer for free and thus cutting off «Netscape's air supply», as intended.

What managed to break through Microsoft's short-lived victory was an alliance of browsers (my favorite Opera on its Presto engine, Mozilla's Firefox on its Gecko engine, and the newborn Safari from Apple, whose WebKit engine was forked from the KHTML engine that was being developed for the KDE Linux desktop environment) that decided to leverage their standards compliance to reinforce each other's position against the crippling effect of Microsoft's dominance —a dominance that Microsoft tried to protect resorting to the vilest tricks.

Google entered the market heavily abusing its dominance in web search to push the adoption of its Chrome browser, a practice not unlike the one used by Microsoft to push the adoption of IE, and of equally questionable legality and moral standing, a thing which was frequently overlooked with several excuses, not least the fact that Chrome was built on an open source core, Chromium, that was mostly assembled from software and libraries developed by other companies (primarily, Mozilla and Apple).

In the years of Chrome's release, the Internet was undergoing massive changes, with the emergence of centralized social media platforms like Facebook that started eroding the previous distributed social network of blogging platforms, Google's own Gmail mail service gaining ground over both ISP offering and other “cloud” offers like Yahoo!'s and Microsoft's Hotmail, and mobile connectivity growing beyond “professionals”, thanks mostly to Apple's iPhone and Google's own at-the-time recent acquisition of Android, plus some soon-to-be minor players I've talked about in the past.

For the purposes of our discussion, these changes had two major points of focus in terms of website development.

On the one hand, web developers started giving more attention to standards compliance, as it gave them more opportunities towards the growing user base of mobile users, which were unlikely to have the desktop-dominant Internet Explorer as browser. This helped accelerate the demise of IE (which was still going strong when Chrome was first released) —whose flaky standards compliance was ultimately responsible for its demise nearly a decade later, and subsequently for the complete discontinuation of its line (after the brief attempt of a reprise under the legacy Edge moniker)— and emboldened the “underdogs” of the time (Mozilla, Apple, Opera).

On the other hand, there was a distinct shift towards centralization of web services, which in turn accelerated the development of web applications, graphical user interfaces for the underlying (centralized) services that effectively relied on the browser(s) as cross-platform toolkits, an approach that would later give birth to the abomination known as Electron and the security nightmare better known as node.js.

Of course, Google had a primary interest in making web apps a credible alternative to desktop applications, what with their already-mentioned mail service and the recently-acquired-and-turned-web-app Google Maps. And since their browser was mostly a collection of existing #FLOSS software stapled together, they could focus their development effort in creating a faster implementation of #JavaScript, better known as V8. Never mind the fact that even years later native implementations of any useful feature would remain faster and cheaper than JavaScript.

But even before their direct involvement in browser development, Opera and Mozilla had started taking their distance from the W3C standardizing efforts and set up the WHATWG, a consortium of browser developers dedicated to coordinate rapid development of new web features without passing through the perceived slow W3C standardization process.

In truth, as it would become clear a few years later —and even more so with Google effectively taking over the WHATWG and turning into a sockpuppet to give a semblance of independence to their choices— the main purpose of the WHATWG was to hijack the development of web technologies to the benefits of the corporate investors, whereas the W3C, with all its flaws, had mostly given priority to features that would be of more general interest.

(It is not by chance that the most controversial standard to ever come out of the W3C has probably been the Encrypted Media Extensions, released as a failed attempt to remain relevant in the web space, and resulting instead of a critical strike against their own credibility as stewards of the open web.)

Google's war on XML as a proxy for the war against the open web

Arguably, the turning point for the centralization of the web was the year 2013. This is essentially the year where GAFAM stopped trying to pretend they liked to play nice, and started to “pull the reins in” on interoperability. Coincidentally, it's also the year Opera stopped being Opera, but I'll talk about this some more in the afterword.

Let's see a few of the major events relevant to our discussion:

1. 2013 is the year Google decides to sunset Google Reader, a (if not the most) widely used web feed aggregator (for #RSS and Atom feeds);
2. 2013 is the year Google decides to close XMPP server-to-server federation in their Google Chat service; Facebook will to the same with their Messenger product the following year;
3. 2013 is the year Google first proposes the removal of XSLT, a proposal that is so unpopular that it will continue receiving comments *against* it as far as 5 years later (the last comment in the thread is from 2018);
4. 2013 is the year Google removes the just-introduced MathML support from Chrome; it will take 10 years and an external company to bring #MathML support back into the browser.

This was just the beginning. Several other actions were undertaken or attempted in the following years.

1. in 2015, the WHATWG introduces the Fetch API, purportedly intended as the modern replacement for the old XMLHttpRequest; prominently missing from the new specification is any mention or methods to manage #XML documents, in favor of JSON that instead gets a dedicated document body presentation method;
2. in 2015, Google proposes deprecating SMIL, the standard for declarative animation and interactivity in SVG; I have written in the past about the usefulness of #SMIL and why not only it must not be deprecated, but its use should actually be integrated into #HTML, as noted by the W3C;
3. in 2015, Google also announces the Accelerated Mobile Pages project, purportedly as a way to make web pages more accessible and faster to load on mobile, which coincidentally relied heavily on leveraging large CDNs like Google to cache contents (and optionally pre-render it); nevermind the facts that the seminal Responsive Web Design article on how to design for different screen sizes was from 2010, that the `srcset` attribute for images to support different-sized screens was already supported by at-the-time current desktop and mobile browsers, and that the primary reasons why webpages weren't fast to load on mobile was because of the so-called web obesity crisis which had been known since 2012 at least, and that the primary reason why AMP pages loaded faster was because they came with one tenth of the useless crap attached to the “regular” pages —so the only actual benefit from AMP was to *force* webdevs into writing leaner pages, with at least a modicum of responsivity, (and of course, for Google, to encourage them to funnel everything through Google's —or any other tech giant— servers for easier metric collection, faster ad serving, and more user profiling);
4. still in 2015, Google announces the intent to deprecate the `keygen` element, a little-known but powerful security feature that simplified the generation of user-controlled cryptographic key pairs for secure communication between the client and server; you can read more about it in Tim Berners-Lee reaction, and in Hugo Landau's relevant “Memoir from the old web”; of note, TBL's primary interest in this element was to help build Solid, an incremental improvement on the WWW to make it more resistant to the centralization his original idea had been perverted into (see also the relevant issue in Solid's issue tracker); the importance of simplified handling of user certificates and the role they play in Mutual authentication can also be surmised by it being one

of the features of the lightweight Gemini protocol that was also born as a response to the centralization and consequent complexification of the World Wide Web;

5. in 2018, Mozilla removes RSS support from Firefox starting from version 64, and actively prevents opening them in-browser, giving them an *even worse treatment* than generic XML files, for which it keeps showing the structure (for example: compare how your browser handles the usage stats XML for this column with the way it handles the RSS feed and the Atom feed); the official reason is that the “Live Bookmarks” feature couldn't be easily ported to the new architecture; the fact that support for RSS could still be implemented via extensions, that Mozilla did not ship an extension to replace even just partially the Live Bookmarks feature —leaving its users in the hands of potentially insecure third-part extensions— and that feeds got an even worse treatment than generic XML document show that the official reason is just an excuse; this is one of the major cracks in the Mozilla façade, as it starts to show that their existence is just controlled opposition for Google to avoid antitrust issues — what Google wants goes, and Google doesn't want web feeds, so web feeds have to go;
6. in 2019, Google announces a number of changes to purportedly make browser extensions “safer” for users, starting the work for what would later become the Extension Manifest V3; it is immediatelly apparent that at least some of the changes introduced are primarily intended to prevent adblockers from working, and don't actually do much to improve security or privacy; despite several reports against the at best ineffective and at worst detrimental changes proposed, in the next years Google will move on with the timeline to deprecate the previous extension APIs and finally succeed in its ad-blocking-blocking efforts; although this change is not *currently* relevant for the XML/XSLT focus of this article, I mention it not only because it is one of the many examples of Chrome becoming less of a User Agent and more of a “Google tool on your computer” over time, but because *this* aspect is important for the future of client-side XML and XSLT, as I will discuss later;
7. in 2023, Google renames their chatbot from Bard to Gemini thereby completely eclipsing the 4-year-old independent protocol by the same name; this is *possibly* coincidental, which would make it the *only* unintentional attack on the open web by Google in the last 15 or so years —and at this point even that is doubtful;
8. in 2023, Google proposes the Web Environment Integrity API, of which I've talked at the time; although this is only

tangentially related to the XML-focused initiatives that are the subject of this article, it is relevant to mention here as it is another example indicative of the push to make browsers less User Agents and more corporate-controlled spyware;

9. in 2023, Google kills off support for the JPEG XL image format, introduced barely two years before, depriving the Internet of a format that would have finally delivered on the promise of a unified format to provide competitive compression —both lossless and lossy—, progressive decoding, transparency, and animation, which would have allowed it to replace the widespread (and less efficient) JPEG, PNG and GIF formats that have been the staple of the web for the last decades; this also is not directly related to XML (unless the reason for the hate is that JPEG XL supports XMP metadata), but should be filed under "against the open and indie web" as it prevents *at the very least* the reduction of hosting and bandwidth costs that a transition to JPEG XL would offer.
10. in 2023, after downranking plain HTTP websites for years, Google announces an even more aggressive stance to push for HTTPS adoption; I have a lot to say about the purported "security" of HTTPS (and in particular about how it doesn't mean what most people think it means, particularly concerning the distinction between the integrity of the connection between the client and server versus the authenticity of the content, particularly of relevance for both corporate silos and federated social networks), but that's material for a different article, so here I'll just link to a few writeups by Dave Winer (one of the inventors of RSS), especially this particularly prophetic one, and point out the hypocrisy of claiming an interest for security by the same company that pushed for the removal of `keygen`;
11. in 2025 Google announces a change in their Chrome Root Program Policy that within 2026 they will stop supporting certificate with an Extended Key Usage that includes any usage other than server (relevant Fediverse thread, other relevant Fediverse thread); this effectively kills certificates commonly used for mutual authentication (hey look, it's the `keygen` suppression theme again!) that include both client and server roles; *coincidentally* this also makes it harder to implement S/MIME, unless you go through Google's services, of course —but Google's war on self-hosted email deserves its own article, so that will be for another time.

And we finally get to these days. Just as RSS feeds are making a comeback and users are starting to grow skeptic of the corporate silos, Google makes another run to kill XSLT, this time using the

WHATWG as a sock puppet. Particularly of note, the corresponding Chromium issue was created *before* the WHATWG Github issue. It is thus to no one's surprise that the *overwhelmingly negative* reactions to the issue, the detailed explanations about why #XSLT is important, how instead of removing it browsers should move to more recent versions of the standard, and even the indications of existing better and more secure libraries to base such new implementations on, *every* counterpoint to the removal have gone *completely* ignored.

Still, the negative reactions were so extensive that the issue has been ultimately locked —particularly when people started pointing out that «we don't have enough resources to spend on this» was a completely idiotic excuse from billion-dollar companies, or even from smaller enterprises like Mozilla that apparently have enough money to waste on features nobody wants like LLM chat integration: this has ultimately confirmed that the purpose of the issue was never to actually discuss whether or not XSLT should be removed, but only to provide a flimsy excuse to pretend the removal was driven by a consensus rather than a top-down directive from Google.

The only true sentences stated by the Googler responsible for this issue were that browsers have been stuck with an obsolete version of XSLT for over two decades, and that the implementations they (Google and Apple) rely on has some security issues. The Googler in question also conveniently omitted several other important facts.

For example, he omitted that two new major versions of XSLT have been released since this technology was first implemented in the browsers: XSLT 2 in 2007, and XSLT 3 in 2017. This means that when Google first proposed to kill XSLT, a newer, considerably more powerful version of the standard had been released for six years already. And already at the time people were pleading for browsers support to be upgraded to the new version.

It is thus not by chance or by lack of resources that browsers are stuck with the 1999 XSLT 1: it has been an intentional choice *against the users' will* since *at least* 2013, the year we already mentioned as the turning point for the centralization of the web. XSLT has been *intentionally boycotted* by Google, Apple and Mozilla: using the excuse that it is not widely used today, after decades of undercutting any efforts in adoption, refusing to fix bugs or even to provide meaningful errors to assist in debugging related issues, is a complete mockery of the victims of these *policy*.

The Googler also omits to mention that both Google's and Apple's XSLT implementation (not Mozilla's, that developed their own) relies on a set of free-software libraries whose maintainer has recently undergone a bombardment of borderline abusive issue reports from the characteristically extractive corporate exploitation of #FLOSS, with requests to provide professional services without actually paying for it in any way. Let's repeat that again: we're talking about billion-dollar companies that have been exploiting the labor of free-software maintainers, *demanding* a preferential treatment at no cost for them, limiting their efforts to finding bugs, without raising a finger to actually fix them —almost as if the primary intent was to find excuses to expunge the library rather than working to improve the commons. (And this is before even going into the irresponsible way in which these libraries were being used.)

But of course anyone questioning the motives of the corporations controlling the WHATWG or pointing out the abundance of resources they have, and how these could easily be spent in bringing XSLT support to the XXI century instead of being spent in user-antagonistic features, is “off topic” and “in violation of the code of conduct”.

In the end, the WHATWG was forced to close down comments to the Github issue to stop the flood of negative feedback, so that the Googler could move on to the next step: commencing the process of formalizing the dismissal of XSLT

Why it matters

When the XML specification was released in 1998, it gained traction very quickly, despite its increased verbosity, because by losing some of the flexibility of SGML (the overreaching specification of which HTML was the most famous incarnation[1]) it favored disambiguation and simplified parsing of documents of arbitrary kind. Combined with XSLT, it allowed documents of any kind to become “Internet ready”, and most importantly ready for the *World Wide Web*, helping driving the WWW towards its designed goal of a «universal linked information system».

Although the benefits of XML and the transformative power of XSLT mostly caught the attention of professionals in a variety of fields, at the turn of the century their flexibility reached also into the more general population of web users through the specific incarnation of RSS and Atom web feeds, which allowed users to remain informed about news and updates on their favorite websites without constantly “making the rounds”.

RSS and other XML-based technologies such as Pingbacks were the backbone of blogging, the distributed social network that characterized the first decade of the XXI century.

With blogging common and *distributed* across multiple platforms, the possibility to *aggregate* information from disparate sources, and still see it presented as a regular web page, across browsers, without any need for scripting, in a time where implementations were slow and (thanks to Microsoft, intentionally) incompatible with each other, was seen as a clear win.

Despite the efforts by Google to kill it since 2013, the RSS format remains an essential component of an open and independent web, still in widespread usage both server and client side: there's an estimate 500+ million websites using WordPress, and they *all* feature RSS feeds, even when not properly advertised; most if not all Fediverse platforms also offer RSS feeds, and some (e.g. Friendica) can also import them and thus work as aggregators; and possibly most important, RSS are *the* fundamental component of podcasts («it's not a podcast if it's not RSS»), a multimedia distribution format with hundreds of millions if not billions of users worldwide.

As already mentioned, it's now seeing a resurgence as people have started realizing how catastrophic for the web was the centralization driven by GAFAM during the second decade of the XXI century (even though too many have failed to learn the correct lesson, and have just jumped from one Nazi bar to the next, or have fallen for the cosplay of federation because it's shinier than actual federation).

XSLT is an *essential* companion to RSS, as it allows the feed itself to be perused in the browser (unless, of course, the browser makes the extra effort to prevents you from visualizing it at all, like Firefox does). This allows sites with hundreds of feeds to use *the feed itself* (styled with XSLT) as index page, reducing hosting and bandwidth costs. And of course it can also be used to style any other “standard” XML document that may be found on a site: for example, I have recently discovered thanks to @aslakr @mastodon.social, that WordPress provides a default XSLT stylesheet for its sitemaps (curiously, apparently not one for its web feeds, though?)

And that's just the beginning: as I've shown on this same site, it's possible to use XSLT to plot XML data, and in general to produce rich, complex documents *without JavaScript*, and again with potentially significant reductions in hosting and bandwidth costs.

Bonus points: it seems that the horde of LLM scrapers that are causing troubles all around have some difficulties with general XML, so switching to XML+XSLT could actually work for self-protection.

Remember AMP? If you really wanted to keep shipping the usual tons of useless crap on desktop, but not on mobile, you could put the actual content in an XML file, and then provide two separate, trivial XSLT stylesheets, one to transform it into the usual bloated desktop page, and one to transform it into the stripped-down (and less bloated) abomination that is AMP HTML —which would have come in handy when Google introduced the requirement that the AMP and standard page had to present the same content. But then again, why even ship those tons of useless crap on desktop in the first place?

And to be honest, HTML templates look thoroughly unimpressive compared to XSLT.

But most importantly, even if you personally don't like XML and/or XSLT, why are we letting Google decide what is acceptable and what is not (and most importantly, *not anymore*) on the World Wide Web?

Dorian Taylor on XSLT

If you've read this far, I would encourage you to also read the passionate defense of XSLT by Dorian Taylor on the Github issues that Google is using as an excuse to kill the standard. In case GAFAM gets touchy and decides to purge it, I'm taking the liberty to reproduce it here for archival purposes:

> I have been using XSLT since it was in beta and the only browser that implemented it was MSIE 5.5. I designed and implemented an internationalized content pipeline using XSLT and DocBook at the job I had from 2002-2005. Since about 2007 I've been using XSLT regularly on the client side to transform (X)HTML into itself (as well as SVG and Atom), because it excels at bolting presentation markup onto plain semantic markup, and thus makes for an extremely lazy templating language that exists separately from the JavaScript ecosystem. I use it on my own Web properties, and I use it on projects (I mainly do intranets). I have made libraries for seamless transclusion and querying RDFa, and I use those on projects (like Sense Atlas, a nascent knowledge graph product I'm working on, and Intertwingler, the application server that powers it).
>
> Why I still use XSLT:
>
> - it's a standard
> - it's fast (at least nominally)
> - it's declarative
> - it's orthogonal to JS

- it can mix any number of back-ends (because it's a standard and operates over standard inputs; I regularly use it to mix static and dynamic resources on the same page)
- it can only operate over information you give it (modulo zero-days, apparently)
- it operates over wholes, i.e., it doesn't stitch together markup as text but rather operates over intact DOM structures.

The first and last points are probably the biggest reasons I still use it, and I suppose the latter may need some unpacking. An XSLT stylesheet is a well-formed XML document and only operates over well-formed XML documents, and (unless you put in the effort) is only capable of producing well-formed (X|HT)ML documents. So you have a validity check baked in at a very low level. Every other templating language I've seen, going all the way back to server-side includes (with one esoteric exception), seems to not be shy about chopping up the syntax of the target language.

I anticipate the knee-jerk reaction to this is "so what?". Why should you care whether your template language breaks the syntax of the target? The tooling can compensate and it's intact when you render it. I mean, I guess? But then you need more tooling when otherwise an off-the-shelf validator would do. But that I think is not even the main differentiator.

The key difference, and why I've stuck with XSLT for almost 25 years, is cognitive. When I make a hypermedia resource (I am deliberately not using the word "page"), I think about it as a discrete, atomic whole. I can consider that object (and the server-side code that generates it) in isolation. It loads in the browser and is well-formed and intact and navigable. Then I can think about applying transformations to that object, and/or composing related objects together, as a separate act. When I write an XSLT template, I think about it like a function (in the mathematical sense) rather than a procedure. I see my job as not to stitch together fragments of markup but to describe the node tree that results from an input tree. When I look at so-called "modern" frameworks, I (still) don't see any of that.

The reason why the implementations are riddled with CVEs, in my opinion, is because of neglect. I am old enough to remember when HTML5 was competing with

XHTML2✱ as the proposed next-generation HTML standard. It turned out that the pedantry of the XML parser was not only reviled by developers (and remains a source of confusion for users if they hit a bad patch of it), for markup it was actually unnecessary. Tastes changed, and people moved on. The browser vendors keep the parsers around, but they demonstrably put as little effort into them as they can get away with. (The biggest shortcomings of XSLT 1.0 were fixed in 2.0—in 2007—but of course the browsers never implemented it.)

> ✱ XHTML2 actually had some really good ideas (like transclude all the things), but its mission (something like "how do we make the best XML-based hypertext markup language") was ultimately wrong-headed. I am also old enough to remember, however, that one of the central arguments for HTML5 (now just "HTML", of course) was not breaking backward-compatibility.

My proposal, then, is not to scrap XSLT, but to rehabilitate it. When it first shipped, XSLT was a solid, open-standard solution to the bog-standard problem of generating presentation markup. How many times has the wheel of Web templating been reinvented for this framework or that? Where is the Open Web successor to XSLT? How about…XSLT?

At its core, XSLT is terrifically powerful, especially its latest incarnation (which, incidentally, can operate over JSON). There are, of course, challenges:

1. I would say problem number one is the syntax. XSLT is an extremely bulky, chatty language. Without syntax completion in your code editor you'd never get anything done. But, there are precedents for ameliorating this, like the compact syntax for RelaxNG, or the Turtle or JSON-LD syntaxes for RDF.
2. XSLT only operates over XML (except of course for 3.0 which made an accommodation for JSON). Well that's simple, bump XSLT to 3.1 and spec out how it should operate over HTML DOMs (case-insensitive tags, whatever), as well as an invocation hook analogous to the XSLT processing instruction.
3. Namespaces: Apparently people hate them? This is something I have never understood (because if you don't use namespaces you just end up reinventing them badly), but whatever, fine. You won't need

namespaces in your XPath anyway if you're just transforming HTML.
4. XPath: I would actually put money on the likelihood that CSS selector semantics can embed fully into XPath, especially given that XPath 3.1 itself is extensible (worst case scenario is you cheat and just make a `css` function).
5. Debugging: currently sucks. This I would chalk up to the same neglect as the CVEs.

It would be eminently feasible to make a "*SWeT*", Standard Web Templates:

- easy, neat, declarative syntax, comparable to Sass or RNC (I sketched one out in like 2019)
- isomorphic (or at least injective onto) XSLT 3.0 (3.1?); compiles to it
- wouldn't have to touch namespaces or even XPath if you didn't want to (use CSS selectors instead)
- still capable of existing outside of the JS ecosystem, but can be accessed from JS/DOM just like XSLT 1.0 can

Now I can imagine somebody saying well I can go off and do that anyway; there's a reference implementation of XSLT 3.0 I can compile against (written, actually, by the spec's author), etc etc. I think that kind of misses the point of having a *standard* templating language that you can rely on being baked into every Web browser. At least, I suppose, until they rip it out.

So I guess my ultimate question is, is there truly no appetite for a standard language for transforming markup, a thing we all have to do, on every project, all the time? A thing that for lack of a standard, locks us into this or that framework, or stymies casual system heterogeneity? A thing that would make it even easier to build the Web? Seems like a sensible idea, doesn't it?

(Again: source, for reference.)

And now, onwards to what's ahead.

What we can do about it

Make yourself be seen

The first step is to actually use XML and XSLT. Visit sites that use them. If you have your own website, seek out opportunities to rely on this tech. If you don't know how, there's apparently a growing number of tutorials around, both in text and video form. Stealing the links to the text tutorials from the above link, we have for example.

- Styling Your RSS For a Better Web
- Styling My RSS Feed
- Styling an RSS Feed With XSLT
- Style your RSS feed

(notice a pattern there?)

And styling RSS is just the most common use-case of XSLT. You can use it to plot data and create sparklines, like I've done. You can use it in place of server-side includes. You can use it to render tabular data you have in XML form. You can use it to adjust the (X)HTML structure of your documents to compensate for CSS limitations.

By the way, if you can find new and interesting applications of XSLT in browsers, do let me (and the world) know (see bottom of this article for information about how to contact me on the Fediverse).

Make yourself be heard

Complain. Complain. Complain. Comment on every relevant issue. Vote against the issue in the issue trackers. Let the browser developers know you are affected.

If (or when) the changes still pass, open new tickets. *Demand* that XSLT support be reinstated. And while you're at it, *demand* that it get upgraded to XSLT 3.0 at least. And *demand* that it be enabled for plain HTML.

Voice your opinion on social media. Post about it. Tag the social media profiles of the WHATWG, of Google, Apple, Mozilla, Microsoft, of Chrome, Safari, Firefox, Edge, and let them know that such a change will not be accepted.

Do not let the lies prevail. The choice to suppress XSLT is not due to technical reasons, and it's not due to lack of resources. It's entirely a policy choice intended to obstruct and limit the expressivity of the open and indie web, and this needs to be remarked to anyone believing otherwise.

Build the alternative

If (or when) the changes pass, our best option is to push through with a polyfill like the SaxonJS mentioned by Dorian Taylor|. It will not be as efficient as a native implementation, it will not be as fast, andit will not be enough to allow clients to open and visualize XML files directly, but it *will* allow us to build the case for a return to XSLT as a significant web technology, and become an important instrument in pressuring vendors for new native implementations, not unlike how MathJax has been a useful bridge to native implementations of MathML.

For pure XML files … maybe an extension? This is most likely possible for Firefox, but I don't know enough about the more restrictive rules implemented in Chrome to tell if it would be possible or not. But of course, even if such an extension was possible today, there is no guarantee that Chrome won't push for *another* change in the API to disable it, like it did with ad blockers.

Who knows, it might as well be that the Streisand effect on this umpteenth attempt by Google to kill XSLT will be the chance for its rebirth.

Afterword

With as much I hate Microsoft, its anticompetitive practices, and the way their Wintel monopoly has stymied software and hardware development, killed companies and destroyed innovation in the desktop and workstation space, *one* thing I can say about the First browser war is that —at least while it was ongoing— it led to a lot of innovation in the web space. Microsoft were the first to implement client-side XSLT, they were the ones that opened the gateway to AJAX through their proprietary XMLHTTP ActiveX control that was reimplemented into other browsers as the XMLHttpRequest object, and they were the ones that tried to add SMIL to (X)HTML through the TIME extension, which I wish hadn't failed the way it did (we would have to wait nearly another decade before a limited subset of the functionality would finally get into HTML via CSS animations).

It's possible that this is was at least in part due to the fact that, as it has been said, Microsoft didn't “get” the Internet, but I suspect that the primary reason was that there was some actual competition going on —competition that since the creation of the WHATWG has been replaced by what is, for all intents and purposes, a *cartel*.

The intent to bypass the W3C for some decisions *did* have some merit at the time of creation; looking at the the document whose rejection led to the creation of the WHATWG, for example, we see among the design principles:

Well-defined error handling

> Error handling in Web applications must be defined to a level of detail where User Agents do not have to invent their own error handling mechanisms or reverse engineer other User Agents'.

Users should not be exposed to authoring errors

> Specifications must specify exact error recovery behaviour for each possible error scenario. Error handling should for the most part be defined in terms of graceful error recovery (as in CSS), rather than obvious and catastrophic failure (as in XML).

which I can't disagree with. On the other hand, it's also clear that two other design principles, *backwards compatibility* and *open process*, have been consistently violated since Opera dropped out (oh, how prescient I was in that article!) and the WHATWG was taken over by Google and its lapdogs (Mozilla) and frenemies (Microsoft and Apple).

Today, I'm left wondering if the developers of browsers like Servo —the engine born out of the Mozilla experiments that were cut out (with the entire development team fired) at the start of the COVID-19 pandemic— or Pale Moon would even be accepted into the WHATWG today, since they could (and at the least the latter would) happily throw a wrench into the whole “fake public feedback” mockery we've been subject to this time.

Name and shame

The engineers working on these proposals should be ashamed of themselves. The names I could gather from the public discussions are:

- Adam Barth was the engineer who proposed to remove XSLT in 2013, with support from Eric Seidel, Ojan Vafai;
- Philip Rogers and Eric Willigers were the engineers who proposed to remove SMIL in 2015, with support from Chris Harrelson, Dimitri Glazkov, Philip Jägenstedt;
- Ryan Sleevi was the engineer who proposed to remove `keygen` in 2015;
- Ben Wiser, Borbala Benko, Philipp Pfeiffenberger, Sergey Kataev were the engineers who proposed the Web Environment Integrity;
- Mason Freed is the engineer pushing the XSLT removal in 2025, with support from Anne van Kesteren, the Mozilla employee that appropriately goes by the moniker `smaug`, Tab Atkins Jr., Domenic Denicola.

If you ever hear any of these blabber about open web, interoperability and standards, know that they are lying through their teeth.

And if any of you happen to read this: fuck you.

Made the news

I've apparently “made the news”.

- OSNews has an article that provides a very nice, short, on-point summary;
- Hacker News has a thread;
- here it is on Reddit, where people mostly focus on the website layout (welcome to the “This site's CSS sucks” club! I'm the president!);
- and here it is on Lobste.rs.

I have read the comments (yes, I know, you should never do that) and it's curious that those that didn't like or agree with the article can be grouped in three sets:

unconvinced by my timeline
: these are commenters that disagree on my list being enough to prove that Google is out to destroy the open web; that's OK, the list isn't there to prove anything, it's just to remind people (or inform youngsters who might not remember those days) about some relevant (and a couple less relevant) events in the last decade-plus that have significantly shaped the web; the list isn't even exaustive insofar Google is concerned (anyone wants some user stylesheets?), let alone all the crap, failed promises, rug pulls and abuses committed by the rest of the GAFAM crowd —the only reason I'm singling out Google here, and on those events in particular, is because the focus is on XML, XSLT, and the WHATWG takeover and unwillingness to listen to what users have to say;

disagreement on the assessment
: these are people who disagree on some of the events I reported being bad for the open web, or aimed at encircling it; so far, from what I see, these comments come mostly from Googlers or ex-Googlers; well, I'm sorry to burst your bubble; I'm sure the engineers working at Google have always had the conviction to be doing Good Stuff™ for the benefits of all; but see, that's kind of the problem of a lot of Big Tech employees: the lack of attention to the *implications* of what their brilliant ideas are going to be used for —and sometimes, possibly, even the time to stop and think if the “innovation” is even needed in the first place, or you're forgetting what older, well-established but less reknown tech can already do; a few others don't seem to be (ex-)Googlers, but have a pet peeve with this or that item being included in the list; all I have to say to them is: if that's your only gripe with the article, well, there's hope for you;

people that don't like XML
: a lot of people seem to be in this camp; and I have an important message for them: **you're missing the point**; this isn't about whether XML is nice or not, it's about the fact that

it exists, it's in use, and it's a powerful tool that web developers can and *do* use; you prefer #JSON to #XML? *That's fine*, and if anything that's one more reason to push browser developers to implement newer XSLT versions that *support JSON too*; or if you prefer: the question isn't whether or not XML and XSLT are worth saving; **it's whether or not you want Google to define what is allowed on the World Wide Web**. (Yes, I have edited the post to make this more clear.)

By the way, you can let me know directly about your thoughts about this article by commenting on this Fediverse thread.

---

1. it has been pointed out in some comments that HTML is not really an application of SGML; this is debatable: TBL intended it to be one, and even if there were some significant divergences in the first iterations, the HTML 4 spec, which was *the* standard definition of HTML in the time I'm referring to, actually defined it as a compliant SGML language —even if browsers never adopted it as such. ↩

---

permalink

Oblomov

Created dom 17 ago 2025, 17:10:00
Last edited mer 20 ago 2025, 00:32:26