

Planet Mozilla - Latest News

- [Don Marti: More money and better stuff for people in the UK](#) (2025/03/28 00:00)

Some good news last week: Meta settles UK 'right to object to ad-tracking' lawsuit by agreeing not to track plaintiff. Tanya O'Carroll, in the UK, has settled a case with Meta, and the company must stop using her data for ad targeting when she uses its services. It's not a change for everyone, though, since the settlement is just for one person. O'Carroll said she is unable to disclose full details of the tracking-free access Meta will be providing in her case but she confirmed that she will not have to pay Meta. The Open Rights Group now has a Meta opt-out page that anyone in the UK can use to do an opt out under the UK GDPR. If you use any Meta products - Facebook, Instagram, Meta Quest or VR, Threads or WhatsApp - you can use our tool to request that they no longer collect or process your data for advertising. This is known as your right to object, which is enshrined in data protection law. Meta had tried to get around GDPR, but by settling Tanya's case they have admitted that they need to give their users this right. If you're in the UK, you can either use the form on the site, or use the mailto link to open up a new regular email from your own account pre-populated with the opt out text. This is a win not just because it could mean less money for a transnational criminal organization and more money staying in the UK, but also because it's going to mean better products and services for the people who do it. Opt outs are one layer in the onion. Don't do a surveilled activity Block the transfer of tracking data Generate tracking data that is hard to link to you Set an opt out while doing the surveilled activity Send an opt out or Right to Delete after doing the surveilled activity Having access to this new tool doesn't mean not to do the others. Even if I could figure out how to use the Meta apps in a way that's totally safe for me, it's still a win to switch away because it helps build network effects for the alternatives and more safety for other people. So even if you do this opt out, it's also a good idea to do the other effective privacy tips. How this gets you more money and better stuff Turning off the personalized ads is a bigger deal than it looks like. The arguments from advertising personalization fans don't reflect the best research on the subject. Ad personalization systems, especially on Facebook, are designed to give some hard-to-overcome advantages to deceptive advertisers. Any limitations to personalization look like a big win, shopping-wise. In one study, turning on an Apple privacy setting reduced reported fraud losses by 4.7%. The personalization of ads on Facebook helps vendors of crappy, misrepresented goods match their products to the shoppers who are most likely to fall for their bullshit. Yes, you can follow the advice in articles like Don't Get Scammed! Tips For Spotting AI-Generated Fake Products Online on Bellingcat, but it's a time-saver and an extra layer of protection not to get the scam ad in the first place. Privacy tools and settings that limit ad personalization have been available for quite a while. If people who use them were buying worse stuff, the surveillance industry would have said so by now. Anyway, if you're in the UK, go do the Meta opt-out. In other countries, other effective privacy tips are still a win. Related Click this to buy better stuff and be happier Feeling left out because you're not in the UK? This tip works everywhere that I know of. Bonus links THE WHITE COAT DIDN'T BETRAY YOU—THE PIXEL DID: Judge Keeps Florida Wiretap Case Against Hospital Alive by Blake Landis. (Interesting legal direction in the USA: many states have wiretapping laws that may or may not apply to the Meta Pixel and CAPI.) Applying the Fundamental Axioms to Reduce Uncertainty by Joe Gregorio. Would it be bad form to point out that the entire edifice of "Agile" software development is built on a bed of lies? (I don't know. Read the whole thing and make up your own mind? Related: Day-to-day experiences with bug futures Why We Need Shortwave 2.0 by Kim Andrew Elliott on Radio World. Because Shortwave Radiogram is transmitted on a regular amplitude-modulated shortwave

transmitter, it can be received on any shortwave radio, from inexpensive portable radios with no sideband capability, to more elaborate communications receivers, amateur transceivers (most of which nowadays have general coverage receivers), and software defined radios (SDRs). (Then you need a program to convert the encoded signal into text and/or images—or this functionality could be built into future inexpensive radios.)

- [The Mozilla Blog: 4 accessibility tools to try in Firefox](#) (2025/03/27 17:41)

The internet is for everyone, but not everyone experiences it the same way. For many of us, harsh brightness, hard-to-read text or a webpage that doesn't work with a screen reader can turn simple tasks into frustrating obstacles. While we've built accessibility features directly into Firefox, we also support extensions that let you customize your online experience. Accessibility isn't one size-fits-all and the right tools can make a huge difference, including for folks who are not disabled. We've put together a list of extensions that address different accessibility challenges. Here are four that our users love:

1. **Dark Reader: Make websites easier on the eyes** What it does: Dark Reader applies a dark theme to websites, helping reduce glare and making text easier to read. Who it helps: People with light sensitivity, vision impairments or anyone who finds bright screens uncomfortable. Dark Reader is great for reducing eye strain. It increases contrast in a way that works best for the user and better supports browsing when eyes are tired. It's especially useful for sites like Google Docs, which don't support dark mode natively. It lets users adjust brightness and contrast to their preference, whether that's full dark mode, sepia, or an off-white background. Some sites have dark mode built-in, but not all of them do. Dark Reader helps bridge that gap. It doesn't solve everything — for example, images remain unchanged — but for text-heavy sites, it's a great option.
2. **Bionic Reader: Partially bolded text for smoother reading** What it does: Bionic Reader bolds parts of words to help guide the eye while reading. Who it helps: People with dyslexia, those learning a new language or anyone who struggles with dense text. Bionic Reader makes text easier to process by bolding parts of words to guide the eye. It's useful for a range of reading challenges — not just dyslexia but also for people learning a new language and need help parsing words and characters. It can be useful for anyone who finds certain fonts difficult to read or just prefers an easier-to-follow text format.
3. **Zoom Page WE: Text size that's just right** What it does: Zoom Page WE allows users to set custom zoom levels, override website restrictions and enforce minimum font sizes. Who it helps: People with low vision, those who strain to read small text or anyone who prefers larger font sizes. Many websites prevent users from resizing text, which can make reading difficult for people with low vision. Zoom Page WE overrides those restrictions, allowing for custom zoom levels and minimum font sizes. Firefox itself has built-in zoom features, but some people prefer the customization options of an extension like this. It allows you to jump directly to a specific zoom level instead of adjusting incrementally.
4. **Read Aloud: Let the web read to you** What it does: Read Aloud converts text to speech, offering customizable voices, reading speeds, and text highlighting. Who it helps: People with vision impairments, those who prefer listening to content rather than reading it or anyone who benefits from auditory learning. Read Aloud makes long articles more manageable by converting them into speech. Firefox has a built-in "narrate" feature in Reader View, but this extension offers more customization, including different voices, reading speeds, and even text highlighting. It's all about flexibility.

Accessibility is a work in progress. We're on it. We're always finding new ways to make Firefox more accessible. Right now, we're working with Fable, a company that connects us with disabled expert users of assistive technology to test and improve features. The feedback we're getting from real users is shaping Firefox for the better. We're committed to building more accessibility features and making sure everyone has the tools they need to browse with ease. If there's something that would make Firefox work better for you, we'd love to hear about it. Get the browser that puts your privacy first — and always has Download Firefox

The post [4 accessibility tools to try in Firefox](#) appeared first on [The Mozilla Blog](#).

- [Mozilla Privacy Blog: Mozilla shares 2025 Policy Priorities and Recommendations for Creating an Internet Where Everyone Can Thrive \(2025/03/27 15:49\)](#)

Mozilla envisions a future where the internet is a truly global public resource that is open, accessible, and safe for all. An internet that benefits people using online services, prioritizes the right to privacy, and enables economic dynamism. Our commitment to this vision stems from Mozilla's foundational belief that the internet was built by people, for people and that its future should not be dictated by a few powerful organizations. When technology is developed solely for profit, it risks causing real harm to people. True choice and control for Americans can only be achieved through a competitive ecosystem with a range of services and providers that foster innovation. However, today's internet is far from this ideal state, and without action, is only set to become increasingly consolidated in the age of AI. Today, Mozilla is setting out our 2025 - 2026 Policy Vision for the United States as we look to a new administration and a new congress. Our policy vision is anchored in our guiding principles for a healthy internet, and outlines policy priorities that we believe should be the 'north star' for U.S. policymakers and regulators. Some recommendations are long overdue, while others seek to ensure the development of a healthy and competitive internet moving forward. Here's how we can work together to make this happen.

Priority 1: Openness, Competition, and Accountability in AI Promoting open source policies and approaches in AI has the potential not just to create technology that benefits individuals, but also to make AI systems safer and more transparent. Open approaches and public investment can spur increased research and development, create products that are more accessible and less vulnerable to cyberattacks, and help to catalyze investment, job creation, and a more competitive ecosystem. Mozilla's key recommendations include: Increase government use of, and support for, open-source AI. The U.S. federal government procures billions of dollars of software every year. The government should use these resources to promote and leverage open source AI when possible, to drive growth and innovation. Develop and fund public AI infrastructure. Supporting initiatives like the National AI Research Resource (NAIRR) and the Department of Energy's FASST program is crucial for developing public AI infrastructure that provides researchers and universities with access to AI tools, fosters innovation and ensures benefits for all. Grow the AI talent ecosystem. It is critical that America invests in education programs to grow the domestic AI talent ecosystem. Without this talent, America will face serious difficulties competing globally. Provide access to AI-related resource consumption data. At its current growth trajectory, AI could end up consuming tremendous amounts of natural resources. The government should work with the AI industry (from semiconductor developers to cloud providers to model deployers) to provide open access to resource consumption data and increase industry transparency; this can help to prevent expensive and dangerous grid failures and could lead to lower energy prices for consumers. Clarify a federal position on open source AI export controls that maintains an open door for innovation. By affirming a federal position on open source AI export controls to reflect those of NTIA, and emphasizing the benefits of open models, the administration can spur further advancement in the field.

Priority 2: Protecting Online Privacy and Ensuring Accountability Today's internet economy is powered by people's information. While this data can deliver massive innovation and new services, it can also put consumers and trust at risk. Unchecked collection and opaque practices can leave people susceptible to deceptive and invasive practices, and discrimination online. The rise of generative AI makes the issue of online privacy more urgent than ever. Mozilla believes that privacy and innovation can coexist. With action and collaboration, policymakers can shift the internet economy toward one that prioritizes users' rights, transparency, and trust — where privacy is not a privilege but a guarantee for all. We recommend that policymakers: Pass strong comprehensive federal privacy legislation and support state efforts. Congress must enact a sufficiently strong comprehensive federal privacy law that addresses AI-specific privacy protections, upholds data minimization, ensures the security protections that encryption provides, and covers children and adults, setting a high bar for meaningful

protections. This is how Congress can create an environment where people can truly benefit from the technologies they rely on without paying the premium of exploitation of their personal data. States should also move to enact strong laws to fill the gap and protect their constituents. Support the adoption of privacy-enhancing technologies (PETs). This includes funding NIST and the National Science Foundation to advance fundamental and applied research, while establishing strong privacy protections that incentivize companies to prioritize PETs. The global standards development and consensus process is essential for privacy preserving technologies to develop in a sustainable manner, in particular around areas like advertising. Legislation can also incentivize companies to adopt more privacy-preserving business practices, ultimately benefiting users and supporting their right to privacy online. Provide necessary resources and tools to data privacy regulators. Congress and the administration must enable and empower relevant federal regulators by providing additional resources and authorizations to facilitate privacy-related investigations and enforcement. Efforts should, in particular, target data brokers who traffic sensitive data. Support critical independent research. Policymakers should ensure meaningful access to important data from major platforms for academia and civil society, enabling better research into big tech's harms and stronger accountability. Transparency efforts like the bipartisan Platform Accountability and Transparency Act (PATA) are essential to advancing transparency while protecting public-interest research. Expanding such legislation to include AI platforms and model providers is critical to addressing privacy-related harms and ensuring accountability in the evolving digital landscape. Respect browser opt-out signals. We encourage lawmakers at the state and federal level to support key privacy tools, like the Global Privacy Control (GPC) that Firefox uses. Mozilla supports bills like California's AB 566, which would require browsers and mobile operating systems to include an opt-out setting. We encourage lawmakers at the state and federal level to advance this key privacy tool in law and meet the expectations that consumers rightly have about treatment of their personal information. Priority 3: Increasing Choice for Consumers Real choice and control for consumers require an open, fair, and competitive ecosystem where diverse services and providers can thrive. Updated competition laws - and an understanding of the importance of competition at every layer of the ecosystem - are essential for the internet to be private, secure, interoperable, open, transparent, and to balance commercial profit with public benefit. Mozilla is committed to this future. To achieve this, we must advance the below. Update antitrust legislation to address anti-competitive business practices, such as harmful self-preferencing, that stymie innovation and limit consumer choice. Congress must pass antitrust legislation that addresses these practices and provides the necessary resources, expertise, and authority to relevant regulatory agencies. Tackle harmful design practices. Harmful deceptive design practices not only manifest at the interface level, but also deeper at the operating system level - particularly in cases of vertical integration of services and features. Deploying manipulative, coercive, and deceptive tactics such as aggressive and misleading prompts, messages, and pop-ups risk overriding user choice entirely. Policymakers must hold bad actors accountable. Foster competition across the ecosystem. Independent browser and browser engine developers, like Mozilla, have a long history of innovating and offering privacy- and security-conscious users a meaningful alternative to big tech browser engines. Policymakers should recognize the importance of independent browsers and browser engines in maintaining a safe, open, and interoperable web that provides meaningful choice. So what is the path forward? We see our vision as a roadmap to a healthier internet. We recognize that achieving this vision means engaging with legislators, regulators, and the wider policy community. Mozilla remains committed to our mission to ensure the internet is a space for innovation, transparency, and openness for all. Read our Vision for the United States: 2025 - 2026 for a more comprehensive look at our priorities and recommendations to protect the future of the internet. The post Mozilla shares 2025 Policy Priorities and Recommendations for Creating an Internet Where Everyone Can Thrive appeared first on Open Policy & Advocacy.

- [The Mozilla Blog: A smarter VPN experience: Introducing the Mozilla VPN extension for Windows](#) (2025/03/26 18:23)

VPNs are great for keeping your connection secure and your activity private, but they can also get in the way. From triggering captchas to making some websites harder to access, traditional VPNs often force users into a trade-off between privacy and convenience. The new Mozilla VPN extension for Firefox — now available on Windows — gives users more control and keeps you protected without the hassle. We get it — VPNs can be frustrating. I'm the product manager for Mozilla VPN, and I know firsthand how important — and frustrating — a VPN can be. One of my biggest frustrations with VPNs is their rigid, all-or-nothing approach. Many websites — whether news platforms, social media, banking services or streaming sites — often misinterpret VPN traffic as suspicious. This can lead to access restrictions, additional authentication steps, or slowdowns. What's worse, too often I've disabled my VPN just to visit a specific site, only to realize later that I failed to turn it back on. I also spend a lot of time online keeping up with world events, and a VPN doesn't just protect my privacy — it also helps me gain a local perspective by letting me experience the web as if I were browsing from different locations. Sometimes I want to access content as if I were in my home country; other times, I need locally relevant information from my current location. Unfortunately, traditional VPNs apply a blanket location setting to all traffic, making it cumbersome to switch between vantage points. The Mozilla VPN extension for Firefox makes it easy. Through conversations with our users, we discovered that frustrations with VPNs were widespread. In fact, difficulty accessing specific sites was one of the top reasons people abandoned VPNs. With that in mind, we developed the Mozilla VPN extension for Windows, built in collaboration with Firefox, to offer a more adaptable, flexible, and secure browsing experience. Here's what it does:

1. Choose which websites bypass VPN The new extension lets you select websites that don't use VPN protection, helping you avoid captchas and other access issues without turning off your VPN entirely. Once set, your preferences stay in place — no need to adjust them each time. What this means for you: No more toggling your VPN on and off just to access certain sites. A clear indicator in the URL bar shows when a site isn't using VPN protection, making it easy to turn it back on if needed.
2. Set different VPN locations per site Assign different VPN locations to different websites. Browse one site as if you're in the U.S. while accessing another as if you're in Paris — all in the same Firefox session. What this means for you: Whether you're checking region-specific content, local news, or work resources, this feature gives you precise control without affecting your entire connection. One of our biggest takeaways from speaking with users: People don't just want security — they want control. Traditional VPNs force users to apply the same settings across all websites, but real-world browsing isn't that simple. Our goal is to make privacy tools more practical — so you can stay secure without the usual trade-offs. More improvements and access to come We're actively improving the extension, focusing on stability, reliability and performance. Plus, we know that many macOS and Linux users are eager for this functionality too — we hear you! Stay tuned, as we're working to bring these features to more platforms soon. Your feedback matters If you're on Windows, install the Mozilla VPN Extension for Firefox to experience a more flexible VPN firsthand. And please let us know what you think! You can reach us on Mozilla Connect and, if you are an extension user, we've added a "Give Feedback" link directly there, making it easy for you to share your thoughts. Your input helps us refine the experience and make Mozilla VPN even better for everyone. Thank you for choosing Mozilla VPN, and happy (safe) browsing! A smarter VPN experience Try the Mozilla VPN extension for Windows The post A smarter VPN experience: Introducing the Mozilla VPN extension for Windows appeared first on The Mozilla Blog.

- [This Week In Rust: This Week in Rust 592](#) (2025/03/26 04:00)

Hello and welcome to another issue of This Week in Rust! Rust is a programming language empowering everyone to build reliable and efficient software. This is a weekly summary of its progress and community. Want something mentioned? Tag us at @ThisWeekInRust on X (formerly

Twitter) or @ThisWeekinRust on mastodon.social, or send us a pull request. Want to get involved? We love contributions. This Week in Rust is openly developed on GitHub and archives can be viewed at this-week-in-rust.org. If you find any errors in this week's issue, please submit a PR. Want TWIR in your inbox? Subscribe here. Updates from Rust Community Foundation Ferrous Systems Donates Ferrocene Language Specification to Rust Project Project/Tooling Updates Fastrace: A Modern Approach to Distributed Tracing in Rust rust-analyzer changelog #278 Introducing Cot v0.2: A new version of the Rust web framework for lazy developers Observations/Thoughts Does unsafe undermine Rust's guarantees? Notes on coreutils in Rust Rust in 2025: Language interop and the extensible compiler Dyn async traits, part 10: Box box box Dyn you have idea for dyn? Safe Delayed Initialization for Lifetime Extension Just write a test for it [audio] ExpressVPN with Pete Membrey Building a fast website with the MASH stack Rust Walkthroughs A Daft proc-macro trick: How to Emit Partial-Code + Errors Vendoring C/C++ Dependencies in Rust Fastest Vec Update on My Computer A 10x faster batch job by batching PostgreSQL inserts/updates with Rust and SQLx Bridging the Efficiency Gap Between FromStr and String [video] Build with Naz : traits, subtyping, polymorphism in Rust [video] Rust and embedded programming with Leon Vak Crate of the Week This week's crate is jiff, a datetime library for Rust. Thanks to Filip T for the suggestion! Please submit your suggestions and votes for next week! Calls for Testing An important step for RFC implementation is for people to experiment with the implementation and give feedback, especially before stabilization. If you are a feature implementer and would like your RFC to appear in this list, add a call-for-testing label to your RFC along with a comment providing testing instructions and/or guidance on which aspect(s) of the feature need testing. No calls for testing were issued this week by Rust, Rust language RFCs or Rustup. Let us know if you would like your feature to be tracked as a part of this list. Call for Participation; projects and speakers CFP - Projects Always wanted to contribute to open-source projects but did not know where to start? Every week we highlight some tasks from the Rust community for you to pick and get started! Some of these tasks may also have mentors available, visit the task page for more information. If you are a Rust project owner and are looking for contributors, please submit tasks here or through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! CFP - Events Are you a new or experienced speaker looking for a place to share something cool? This section highlights events that are being planned and are accepting submissions to join their event as a speaker. If you are an event organizer hoping to expand the reach of your event, please submit a link to the website through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! Updates from the Rust Project 496 pull requests were merged in the last week Compiler lower to a memset(undef) when Rvalue::Repeat repeats uninit Library MaybeUninit inherent slice methods part 2 core/slice: mark some split_off variants unstably const core: optimize RepeatN implement default methods for io::Empty and io::Sink optimize io::Write::write_fmt for constant strings simplify PartialOrd on tuples containing primitives reduce FormattingOptions to 64 bits Cargo add custom completer for cargo <TAB> to complete aliases defined in config.toml Rustdoc be more strict about "Methods from Deref" gate unstable doc(cfg()) predicates use own logic to print #[repr(..)] attributes in JSON output Clippy wildcard_imports: lint on pub use if asked to add MSRV check for question_mark add ignore_without_reason lint emit collapsible_match at the right node expand neg_multiply to lint float numbers as well fix suggestion for assignments have enclosing parentheses under needless_late_init fix: borrow_deref_ref suggests wrongly when coerce to mut fix: filter_map_bool_then suggest wrongly when the closure cannot be decompose directly fix: manual_find suggests wrongly when early return fix: missing_const_for_fn false positive on unstable const traits fix: nonminimal_bool wrongly showed the macro definition fix: option_if_let_else false positive when value partially moved fix: redundant_clone false positive on enum cast improve string_to_string lint in case it is in a map call lint more cases in collapsible_if make never_loop applicability more flexible move unlined_format_args back to style reinstate single_match/single_match_else lints with comments suggest is_some_and instead of map_or in case_sensitive_file_extension_comparisons unify

manual_unwrap_or and manual_unwrap_or_default code use code for references to other lints in as_conversions docs Rust-Analyzer fix ide-assist let else to if let else add diagnostic for missing ambiguity error for impl trait add postfix completion for const block add text edit support for return type hints on non-block body closures analysis-stats: emit lines of code and item tree counts for workspace; dependencies parse unsafe record fields fix missing syntax highlighting for &raw const / &raw mut in all files fix closure return inlayhints using macro ranges handle multiple #[repr(..)] attrs correctly properly calculate the layouts of tuple ptrs whose last fields are DST render layout and other extra information on hovering Self speed up resolving a "Generate delegate method" assist Rust Compiler Performance Triage A nearly noise-free week, which is exciting, with a number of fairly large improvements landing for a cumulative average speed up 0.5%, possibly larger if we ignore the likely to be fixed or reverted regressions from #138674. Triage done by @simulacrum. Revision range: 493c38ba..4510e86a 3 Regressions, 4 Improvements, 2 Mixed; 3 of them in rollups 35 artifact comparisons made in total Read the full report for more details. Approved RFCs Changes to Rust follow the Rust RFC (request for comments) process. These are the RFCs that were approved for implementation this week: No RFCs were approved this week. Final Comment Period Every week, the team announces the 'final comment period' for RFCs and key PRs which are reaching a decision. Express your opinions now. Tracking Issues & PRs Rust Deprecate the unstable concat_idents! Stabilize #![feature(precise_capturing_in_traits)] Rust RFCs Tracking Issue for slice::array_chunks stabilize const_cell Remove backticks from ShouldPanic::YesWithMessage's TrFailedMsg Use BinOp::Cmp for iNN::signum Prefer built-in sized impls (and only sized impls) for rigid types always Other Areas *No Items entered Final Comment Period this week for Cargo, Language Team, Language Reference or Unsafe Code Guidelines. Let us know if you would like your PRs, Tracking Issues or RFCs to be tracked as a part of this list. New and Updated RFCs No New or Updated RFCs were created this week. Upcoming Events Rusty Events between 2025-03-26 - 2025-04-23 Virtual 2025-03-27 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-01 | Virtual (Buffalo, NY, US) | Buffalo Rust Meetup Buffalo Rust User Group 2025-04-02 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2025-04-03 | Virtual (Nürnberg, DE) | Rust Nurnberg DE Rust Nürnberg online 2025-04-05 | Virtual | Ardan Labs Communicate with Channels in Rust 2025-04-05 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup 2025-04-08 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2025-04-10 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-15 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2025-04-16 | Virtual (Vancouver, BC, CA) | Vancouver Rust Rust Study/Hack/Hang-out 2025-04-17 | Virtual and In-Person (Redmond, WA, US) | Seattle Rust User Group April, 2025 SRUG (Seattle Rust User Group) Meetup 2025-04-22 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday Asia 2025-03-28 | Kowloon Tong, HK | Rust Asia Rust Asia 2025 2025-04-05 | Bangalore/Bengaluru, IN | Rust Bangalore April 2025 Rustacean meetup 2025-04-22 | Tel Aviv-Yafo, IL | Rust TLV In person Rust April 2025 at Braavos in Tel Aviv in collaboration with StarkWare Europe 2025-03-26 | Frankfurt, DE | Rust Rhein-Main "Beyond blazingly fast!" Performance Optimierungen in Rust 2025-03-26 | Manchester, UK | Rust Manchester Rust Manchester Talks March 2025-03-26 | Warsaw, PL | Rustikon Rustikon 2025-03-27 | Augsburg, DE | Rust Meetup Augsburg Rust Meetup #12: Testing in Rust 2025-03-27 | Copenhagen, DK | Copenhagen Rust Community Rust meetup #56 2025-03-29 | Stockholm, SE | Stockholm Rust Ferris' Fika Forum #10 2025-04-02 | Cambridge, UK | Cambridge Rust Meetup Monthly Rust Meetup 2025-04-02 | München, DE | Rust Munich Rust Munich 2025 / 1 - hybrid 2025-04-02 | Oxford, UK | Oxford Rust Meetup Group Oxford Rust and C++ social 2025-04-02 | Stockholm, SE | Stockholm Rust Rust Meetup @Funnel 2025-04-03 | Oslo, NO | Rust Oslo Rust Hack'n'Learn at Kampen Bistro 2025-04-08 | Olomouc, CZ | Rust Moravia 3. Rust Moravia Meetup (Real Embedded Rust) 2025-04-09 | Girona, ES | Rust Girona Rust Girona Hack & Learn 04 2025 2025-04-09 | Reading, UK | Reading Rust Workshop Reading Rust Meetup 2025-04-10 | Karlsruhe, DE | Rust Hack & Learn Karlsruhe Karlsruhe Rust Hack and Learn Meetup bei BlueYonder 2025-04-15 | Leipzig, DE | Rust - Modern Systems Programming in

Leipzig Topic TBD 2025-04-15 | London, UK | Women in Rust WIR x WCC: Finding your voice in Tech 2025-04-19 | Istanbul, TR | Türkiye Rust Community Rust Konf Türkiye 2025-04-23 | London, UK | London Rust Project Group Fusing Python with Rust using raw C bindings North America 2025-03-26 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2025-03-26 | Boston, MA, US | Boston Rust Meetup Allston Rust Dinner, March 26th! 2025-03-26 | New York, NY, US | Rust NYC Rust NYC: I can't believe that's legal Rust with Michael Gattozzi (NEW LOCATION) 2025-03-27 | Atlanta, GA, US | Rust Atlanta We're going again! 2025-03-29 | Boston, MA, US | Boston Rust Meetup North End Rust Lunch, Mar 29 2025-03-31 | Boulder, CO, US | Solid State Depot Boulder Rust: Bryan presents Rusted Hardware 2025-04-03 | Chicago, IL, US | Chicago Rust Meetup Rust Happy Hour 2025-04-03 | Montréal, QC, CA | Rust Montréal April Monthly Social 2025-04-03 | Saint Louis, MO, US | STL Rust icu4x - resource-constrained internationalization (i18n) 2025-04-06 | Boston, MA, US | Boston Rust Meetup Kendall Rust Lunch, Apr 6 2025-04-10 | Portland, OR, US | PDXRust TetaNES: A Vaccination for Rust—No Needle, Just the Borrow Checker 2025-04-14 | Boston, MA, US | Boston Rust Meetup Coolidge Corner Brookline Rust Lunch, Apr 14 2025-04-17 | Nashville, TN, US | Music City Rust Developers Using Rust For Web Series 1 : Why HTMX Is Bad 2025-04-17 | Redmond, WA, US | Seattle Rust User Group April, 2025 SRUG (Seattle Rust User Group) Meetup 2025-04-23 | Austin, TX, US | Rust ATX Rust Lunch - Fareground Oceania 2025-04-14 | Christchurch, NZ | Christchurch Rust Meetup Group Christchurch Rust Meetup 2025-04-22 | Barton, AC, AU | Canberra Rust User Group April Meetup South America 2025-03-27 | Medellín, CO | Rust Medellín Multithreading y Terminal User Interfaces con Rust 2025-04-03 | Buenos Aires, AR | Rust en Español Abril - Lambdas y más! If you are running a Rust event please add it to the calendar to get it mentioned here. Please remember to add a link to the event too. Email the Rust Community Team for access. Jobs Please see the latest Who's Hiring thread on r/rust Quote of the Week Did it work? It's Rust, so it worked on the first try! - James Calligeros on the Asahi progress report Thanks to yerke for the suggestion! Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell, llogiq, cdmistman, ericseppanen, extrawurst, U007D, joelmarcey, mariannegoldin, bennyvasquez, bdillo Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- [The Rust Programming Language Blog: Adopting the FLS](#) (2025/03/26 00:00)

Adopting the FLS Some years ago, Ferrous Systems assembled a description of Rust called the FLS1. They've since been faithfully maintaining and updating this document for new versions of Rust, and they've successfully used it to qualify toolchains based on Rust for use in safety-critical industries. Seeing this success, others have also begun to rely on the FLS for their own qualification efforts when building with Rust. The members of the Rust Project are passionate about shipping high quality tools that enable people to build reliable software at scale. Such software is exactly the kind needed by those in safety-critical industries, and consequently we've become increasingly interested in better understanding and serving the needs of these customers of our language and of our tools. It's in that light that we're pleased to announce that we'll be adopting the FLS into the Rust Project as part of our ongoing specification efforts. This adoption is being made possible by the gracious donation of the FLS by Ferrous Systems. We're grateful to them for the work they've done in assembling the FLS, in making it fit for qualification purposes, in promoting its use and the use of Rust generally in safety-critical industries, and now, for working with us to take the next step and to bring the FLS into the Project. With this adoption, we look forward to better integrating the FLS with the processes of the Project and to providing ongoing and increased assurances to all those who use Rust in safety-critical industries and, in particular, to those who use the FLS as part of their qualification efforts. This adoption and donation would not have been possible without the efforts of the Rust Foundation, and in particular of Joel Marcey, the Director of Technology at the Foundation, who has worked tirelessly to facilitate this on our behalf. We're grateful to him and to the Foundation for this support. The Foundation has published its own post about this adoption. I'm relying on the FLS today; what should I expect?

We'll be bringing the FLS within the Project, so expect some URLs to change. We plan to release updates to the FLS in much the same way as they have been happening up until now. We're sensitive to the fact that big changes to this document can result in costs for those using it for qualification purposes, and we don't have any immediate plans for big changes here. What's this mean for the Rust Reference? The Reference is still the Reference. Adopting the FLS does not change the status of the Reference, and we plan to continue to improve and expand the Reference as we've been doing. We'll of course be looking for ways that the Reference can support the FLS, and that the FLS can support the Reference, and in the long term, we're hopeful we can find ways to bring these two documents closer together. The FLS stood for the "Ferrocene Language Specification". The minimal fork of Rust that Ferrous Systems qualifies and ships to their customers is called "Ferrocene", hence the name. We'll be dropping the expansion and just calling it the FLS within the Project. ↩

- [Hacks.Mozilla.Org: Improving Firefox Stability in the Enterprise by Reducing DLL Injection](#) (2025/03/25 18:31)

Beginning in version 138, Firefox will offer an alternative to DLL injection for Data Loss Prevention (DLP) deployments in enterprise environments. DLL Injection DLL injection into Firefox is a topic we've covered on the Hacks blog before. In 2023, we blogged about the Firefox capability to let users block third-party DLLs from being loaded. We explained what DLL injection is, how we deal with problematic third-party modules, our `about:third-party` page, and our third-party injection policy. Earlier, in 2019, we released a study of DLL injection Firefox bugs in collaboration with Polytechnique Montréal. We return to this topic now, in the context of enterprise Firefox installations. First, a reminder of what DLL injection is and why it continues to be problematic. DLL injection is the term we use to describe third-party Windows software injecting its own DLL module code into Firefox. Third parties develop DLLs for injecting into applications to extend their functionality in some way. This is prevalent in the Windows ecosystem. When third-party code is injected, the injected code interacts with the internals of the application. While it is not unusual for software to work together, and the internet is built on software interoperating over documented standards, DLL injection differs in that the undocumented internals of an application are not intended to be a stable interface. As such, they are a poor foundation to build software products on. When the underlying application is changed, it can result in incompatibilities, leading to crashes or other unexpected behavior. In a modern web browser like Firefox, new features and fixes, big and small, are developed and released on a monthly schedule. Normal browser development can therefore cause incompatibilities with injected software, resulting in Firefox crashes, bypassing of security features, or other unpredictable buggy behavior. When these problems arise, they require emergency troubleshooting and engineering of workarounds for users until the problems are addressed by software updates. This often requires collaboration between the browser and the third-party application's developers. The type of software injected into Firefox varies from small open source projects to widely-deployed enterprise security products. In an attempt to eliminate some of the most difficult DLL injection issues, we've turned our attention to Data Loss Prevention enterprise applications. Data Loss Prevention (DLP) in the Enterprise Data Loss Prevention (DLP) products are a type of software that is widely deployed by organizations to prevent unintended leaks of private data. Examples of private data include customer records such as names, addresses, credit card information or company secrets. Much like how anti-virus software is deployed across a corporation's fleet of laptops, so too is DLP software. These deployments have become increasingly common, in large part due to compliance and liability concerns. How does this relate to Firefox? DLP software typically uses DLL injection to monitor applications such as Firefox for activity that might leak private data. This only applies to specific operations that can leak sensitive information such as file uploads, pasting (as in copy-and-paste), drag-and-drop, and printing. DLP and Firefox Today Today, DLP software typically monitors Firefox activity via DLL injection as described above. Firefox and web browsers are not unique in this respect, but they are heavily used and under constant development, making DLL injection more dangerous. DLP software is

typically deployed to a fleet of corporate computers that are managed by an IT department. This includes deployment of the software that injects into applications. DLP vendors take efforts to ensure that their products are compatible with the latest version of Firefox by testing beta versions and updating their DLLs as needed, but problems still occur regularly. A common issue is that a problem is encountered by corporate users who report the problem to their IT department. Their IT staff then work to debug the problem. They may file a bug report with Firefox or the DLP vendor. When a Firefox bug is filed, it can be a challenge for Mozilla to determine that the bug was actually caused by external software. When we learn of such problems, we alert the vendor and investigate workarounds. In the interim, users have a poor experience and may have to work around problems or even use another browser. When the browser is not functional, the problem becomes a high severity incident where support teams work as quickly as possible to help restore functionality.

Browsing Privacy When users browse on company-owned computers, their browsing privacy is often subject to corporate-mandated software. Different regions have different laws about this and the disclosures required, but on a technical level, when the device is controlled by a corporation, that corporation has a number of avenues at its disposal for monitoring activity at whatever level is dictated by corporate policy. Firefox is built on the principle that browsing activity belongs only to the user, but as an application, it cannot reasonably override the wishes of the device administrator. Insofar as that administrator has chosen to deploy DLP software, they will expect it to work with the other software on the device. If a well-supported mechanism is not available, they will either turn to opaque and error-prone methods like DLL injection, or replace Firefox with another browser.

What's New - Reducing DLL Injection in the Enterprise Starting with Firefox 138, DLP software can work with Firefox without the use of DLL injection. Firefox 138 integrates the Content Analysis SDK and it can be enabled with Enterprise Policies. The SDK, developed by Google and used in Chrome Enterprise, is a lightweight protocol between the browser and a DLP agent, with the implementation being browser-specific. In other words, Firefox has its own implementation of the protocol. The integration allows Firefox to interact with DLP software while reducing the injection of third-party code. This will improve the stability for Firefox users in the enterprise and, as more DLP vendors adopt the SDK, there will be less third-party code injected into Firefox. With vendors and browsers using the same SDK, vendors can know that a single DLP agent implementation will be compatible with multiple browsers. During development of the Firefox implementation, we've been working with some leading DLP vendors to ensure compatibility. In addition to stability, Firefox will display an indicator when the DLP SDK is used, providing more transparency for users. For Enterprise Use Firefox will only enable the Content Analysis SDK in configurations where a Firefox Enterprise Policy is used. Firefox Enterprise Policies are used by organizations to configure Firefox settings across a fleet of computers. They allow administrators to configure Firefox, for example, to limit which browser extensions can be installed, set security-related browser settings, configure network proxy settings, and more. You can learn more about Firefox Enterprise Policies on our support article [Enforce policies on Firefox for Enterprise](#). The post [Improving Firefox Stability in the Enterprise by Reducing DLL Injection](#) appeared first on [Mozilla Hacks - the Web developer blog](#).

- [Niko Matsakis: Dyn you have idea for `dyn`?](#) (2025/03/25 17:19)

Knock, knock. Who's there? Dyn. Dyn who? Dyn you have ideas for dyn? I am generally dissatisfied with how dyn Trait in Rust works and, based on conversations I've had, I am pretty sure I'm not alone. And yet I'm also not entirely sure the best fix. Building on my last post, I wanted to spend a bit of time exploring my understanding of the problem. I'm curious to see if others agree with the observations here or have others to add. Why do we have dyn Trait? It's worth stepping back and asking why we have dyn Trait in the first place. To my mind, there are two good reasons. Because sometimes you want to talk about "some value that implements Trait" The most important one is that it is sometimes strictly necessary. If you are, say, building a multithreaded runtime like rayon or tokio, you are going to need a list of active tasks somewhere, each of

which is associated with some closure from user code. You can't build it with an enum because you can't enumerate the set of closures in any one place. You need something like a `Vec<Box<dyn AsyncTask>>`. Because sometimes you don't need to so much code. The second reason is to help with compilation time. Rust land tends to lean really heavily on generic types and `impl Trait`. There are good reasons for that: they allow the compiler to generate very efficient code. But the flip side is that they force the compiler to generate a lot of (very efficient) code. Judicious use of `dyn Trait` can collapse a whole set of "almost identical" structs and functions into one. These two goals are distinct. Right now, both of these goals are expressed in Rust via `dyn Trait`, but actually they are quite distinct. For the first, you really want to be able to talk about having a `dyn Trait`. For the second, you might prefer to write the code with generics but compile in a different mode where the specifics of the type involved are erased, much like how the Haskell and Swift compilers work. What does "better" look like when you really want a `dyn`? Now that we have the two goals, let's talk about some of the specific issues I see around `dyn Trait` and what it might mean for `dyn Trait` to be "better". We'll start with the cases where you really want a `dyn` value. Observation: you know it's a `dyn`. One interesting thing about this scenario is that, by definition, you are storing a `dyn Trait` explicitly. That is, you are not working with a `T: ?Sized + Trait` where `T` just happens to be `dyn Trait`. This is important because it opens up the design space. We talked about this some in the previous blog post: it means that you don't need working with this `dyn Trait` to be exactly the same as working with any other `T` that implements `Trait` (in the previous post, we took advantage of this by saying that calling an `async` function on a `dyn trait` had to be done in a `.box` context). Able to avoid the `Box` For this pattern today you are almost certainly representing your task a `Box<dyn Task>` or (less often) an `Arc<dyn Task>`. Both of these are "wide pointers", consisting of a data pointer and a vtable pointer. The data pointer goes into the heap somewhere. In practice people often want a "flattened" representation, one that combines a vtable with a fixed amount of space that might, or might not, be a pointer. This is particularly useful to allow the equivalent of `Vec<dyn Task>`. Today implementing this requires unsafe code (the `anyhow::Anyhow` type is an example). Able to inline the vtable Another way to reduce the size of a `Box<dyn Task>` is to store the vtable 'inline' at the front of the value so that a `Box<dyn Task>` is a single pointer. This is what C++ and Java compilers typically do, at least for single inheritance. We didn't take this approach in Rust because Rust allows implementing local traits for foreign types, so it's not possible to enumerate all the methods that belong to a type up-front and put them into a single vtable. Instead, we create custom vttables for each (type, trait) pair. Able to work with self methods Right now `dyn traits` cannot have self methods. This means for example you cannot have a `Box<dyn FnOnce()>` closure. You can workaround this by using a `Box<Self>` method, but it's annoying:

```
trait Thunk { fn call(self: Box<Self>); }
impl<F> Thunk for F where F: FnOnce(), { fn call(self: Box<Self>) { (*self)() } }
fn make_thunk(f: impl FnOnce()) -> Box<dyn Thunk> {
  Box::new(f) }
Able to call Clone One specific thing that hits me fairly often is that I want the ability to clone a dyn value: 

```
trait Task: Clone { // -----
 Error: not dyn compatible fn method(&self); }
fn clone_task(task: &Box<dyn Task>) { task.clone() }
This is a hard one to fix because the Clone trait can only be implemented for Sized types. But dang it would be nice. Able to work with (at least some) generic functions Building on the above, I would like to have dyn traits that have methods with generic parameters. I'm not sure how flexible this can be, but anything I can get would be nice. The simplest starting point I can see is allowing the use of impl Trait in argument position:

```
trait Log { fn log_to(&self, logger: impl Logger); // <-- not dyn safe today }
Today this method is not dyn compatible because we have to know the type of the logger parameter to generate a monomorphized copy, so we cannot know what to put in the vtable. Conceivably, if the Logger trait were dyn compatible, we could generate a copy that takes (effectively) a dyn Logger - except that this wouldn't quite work, because impl Logger is short for impl Logger + Sized, and dyn Logger is not Sized. But maybe we could finesse it. If we support impl Logger in argument position, it would be nice to support it in return position. This of course is approximately the problem we are looking to solve to support dyn async trait: 

```
trait Signal { fn signal(&self) ->
```


```


```


```

impl Future<Output = ()>; } Beyond this, well, I'm not sure how far we can stretch, but it'd be nice to be able to support other patterns too. Able to work with partial traits or traits without some associated types unspecified One last point is that sometimes in this scenario I don't need to be able to access all the methods in the trait. Sometimes I only have a few specific operations that I am performing via dyn. Right now though all methods have to be dyn compatible for me to use them with dyn. Moreover, I have to specify the values of all associated types, lest they appear in some method signature. You can workaround this by factoring out methods into a supertrait, but that assumes that the trait is under your control, and anyway it's annoying. It'd be nice if you could have a partial view onto the trait. What does "better" look like when you really want less code? So what about the case where generics are fine, good even, but you just want to avoid generating quite so much code? You might also want that to be under the control of your user. I'm going to walk through a code example for this section, showing what you can do today, and what kind of problems you run into. Suppose I am writing a custom iterator method, alternate, which returns an iterator that alternates between items from the original iterator and the result of calling a function. I might have a struct like this: struct Alternate<I: Iterator, F: Fn() -> I::Item> { base: I, func: F, call_func: bool, } pub fn alternate<I, F>(base: I, func: F,) -> Alternate<I, F> where I: Iterator, F: Fn() -> I::Item, { Alternate { base, func, call_func: false } } The Iterator impl itself might look like this: impl<I, F> Iterator for Alternate<I, F> where I: Iterator, F: Fn() -> I::Item, { type Item = I::Item; fn next(&mut self) -> Option<I::Item> { if !self.call_func { self.call_func = true; self.base.next() } else { self.call_func = false; Some((self.func)()) } } } Now an Alternate iterator will be Send if the base iterator and the closure are Send but not otherwise. The iterator and closure will be able to use of references found on the stack, too, so long as the Alternate itself does not escape the stack frame. Great! But suppose I am trying to keep my life simple and so I would like to write this using dyn traits: struct Alternate<Item> { // variant 2, with dyn base: Box<dyn Iterator<Item = Item>>, func: Box<dyn Fn() -> Item>, call_func: bool, } You'll notice that this definition is somewhat simpler. It looks more like what you might expect from Java. The alternate function and the impl are also simpler: pub fn alternate<Item>(base: impl Iterator<Item = Item>, func: impl Fn() -> Item,) -> Alternate<Item> { Alternate { base: Box::new(base), func: Box::new(func), call_func: false } } impl<Item> Iterator for Alternate<Item> { type Item = Item; fn next(&mut self) -> Option<Item> { // ...same as above... } } Confusing lifetime bounds There a problem, though: this code won't compile! If you try, you'll find you get an error in this function: pub fn alternate<Item>(base: impl Iterator<Item = Item>, func: impl Fn() -> Item,) -> Alternate<Item> {...} The reason is that dyn traits have a default lifetime bound. In the case of a Box<dyn Foo>, the default is 'static. So e.g. the base field has type Box<dyn Iterator + 'static>. This means the closure and iterators can't capture references to things. To fix that we have to add a somewhat odd lifetime bound: struct Alternate<'a, Item> { // variant 3 base: Box<dyn Iterator<Item = Item> + 'a>, func: Box<dyn Fn() -> Item + 'a>, call_func: bool, } pub fn alternate<'a, Item>(base: impl Iterator<Item = Item> + 'a, func: impl Fn() -> Item + 'a,) -> Alternate<'a, Item> {...} No longer generic over Send OK, this looks weird, but it will work fine, and we'll only have one copy of the iterator code per output Item type instead of one for every (base iterator, closure) pair. Except there is another problem: the Alternate iterator is never considered Send. To make it Send, you would have to write dyn Iterator + Send and dyn Fn() -> Item + Send, but then you couldn't support non-Send things anymore. That stinks and there isn't really a good workaround. Ordinary generics work really well with Rust's auto trait mechanism. The type parameters I and F capture the full details of the base iterator plus the closure that will be used. The compiler can thus analyze a Alternate<I, F> to decide whether it is Send or not. Unfortunately dyn Trait really throws a wrench into the works - because we are no longer tracking the precise type, we also have to choose which parts to keep (e.g., its lifetime bound) and which to forget (e.g., whether the type is Send). Able to partially monomorphize ("polymorphize") This gets at another point. Even ignoring the Send issue, the Alternate<'a, Item> type is not ideal. It will make fewer copies, but we still get one copy

per item type, even though the code for many item types will be the same. For example, the compiler will generate effectively the same code for `Alternate<_, i32>` as `Alternate<_, u32>` or even `Alternate<_, [u8; 4]>`. It'd be cool if we could have the compiler go further and coalesce code that is identical.¹ Even better if it can coalesce code that is “almost” identical but pass in a parameter: for example, maybe the compiler can coalesce multiple copies of `Alternate` by passing the size of the `Item` type in as an integer variable. Able to change from `impl Trait` without disturbing callers I really like using `impl Trait` in argument position. I find code like this pretty easy to read: `fn for_each_item<Item>(base: impl Iterator<Item = Item>, mut op: impl FnMut(Item),) { for item in base { op(item); } }` But if I were going to change this to use `dyn` I can't just change from `impl` to `dyn`, I have to add some kind of pointer type: `fn for_each_item<Item>(base: &mut dyn Iterator<Item = Item>, op: &mut dyn FnMut(Item),) { for item in base { op(item); } }` This then disturbs callers, who can no longer write: `for_each_item(some_iter, |item| process(item));` but now must write this `for_each_item(&mut some_iter, &mut |item| process(item));` You can work around this by writing some code like this... `fn for_each_item<Item>(base: impl Iterator<Item = Item>, mut op: impl FnMut(Item),) { for_each_item_dyn(&mut base, &mut op) }` `fn for_each_item_dyn<Item>(base: &mut dyn Iterator<Item = Item>, op: &mut dyn FnMut(Item),) { for item in base { op(item); } }` but to me that just begs the question, why can't the compiler do this for me dang it? Async functions can make send/sync issues crop up in functions In the iterator example I was looking at a struct definition, but with `async fn` (and in the future with `gen`) these same issues arise quickly from functions. Consider this `async fn`: `async fn for_each_item<Item>(base: impl Iterator<Item = Item>, op: impl AsyncFnMut(Item),) { for item in base { op(item).await; } }` If you rewrite this function to use `dyn`, though, you'll find the resulting future is never `send` nor `sync` anymore: `async fn for_each_item<Item>(base: &mut dyn Iterator<Item = Item>, op: &mut dyn AsyncFnMut(Item),) { for item in base { op(item).box.await; // <-- assuming we fixed this } }` Conclusions and questions This has been a useful mental dump, I found it helpful to structure my thoughts. One thing I noticed is that there is kind of a “third reason” to use `dyn` - to make your life a bit simpler. The versions of `Alternate` that used `dyn Iterator` and `dyn Fn` felt simpler to me than the fully parameteric versions. That might be best addressed though by simplifying generic notation or adopting things like implied bounds. Some other questions I have: Where else does the `Send` and `Sync` problem come up? Does it combine with the first use case (e.g., wanting to write a vector of heterogeneous tasks each of which are generic over whether they are `send/sync`)? Maybe we can categorize real-life code examples and link them to these patterns. Are there other reasons to use `dyn trait` that I didn't cover? Other ergonomic issues or pain points we'd want to address as we go? If the code is byte-for-byte identical, In fact LLVM and the linker will sometimes do this today, but it doesn't work reliably across compilation units as far as I know. And anyway there are often small differences. ←

- [The Mozilla Blog: Open-source AI is hard. Blueprints can help!](#) (2025/03/25 16:54)

“I spend 8 hours per week trying to keep up to date, it's overwhelming!” “Integrating new libraries is difficult. They're either poorly maintained or updated in ways that break compatibility.” “I want to be able to experiment quickly, without relying on APIs for closed-source models.” These were just a few of the challenges we heard from developers during months of interviews. Today, we're excited to introduce Blueprints and the Blueprints Hub! Meet Mozilla.ai Blueprints The Blueprints Hub is designed to cut through the clutter of clunky tool integration and outdated resources, so you can focus on building, not troubleshooting. It's a showcase for the best of the open-source builder community. What are Blueprints? Blueprints are your go-to, customizable workflows that enable you to prototype AI applications with trusted open-source tools. Each Blueprint tackles a real-world challenge, giving you a robust foundation to work from: Open-source power: Fully hosted on GitHub and built with the community. Ready out-of-the-box: Get started instantly with accessible setup options. Customizable and extendable: Use it as-is or extend it

to fit your own needs. Consistent and templated: Every Blueprint follows a core template to keep your workflow smooth. Community-driven: Contribute, collaborate, and be part of something bigger. Our launch lineup Kick off your journey with these five practical Blueprints: Document-to-Podcast: Turn your text into lively, multi-voice audio clips with minimal fuss. Structured Question Answering: Extract answers from structured documents with a simple workflow. Finetuning Speech-to-text: Fine-tune speech models locally for multiple languages or your own dataset. OpenStreetMap AI Helper: Use computer vision to detect and map features on OpenStreetMap, with Human Verification. Finetuning an LLM with Federated AI: Collaboratively fine-tune models across data owners without sharing raw data. Build your own Timeline Algorithm: Visualize, search, and re-rank social posts using AI without data leaving your computer. Explore the Blueprints Hub Our new Hub is built for ease and exploration: Instant demos: Play around with Blueprints live in the hosted demo. No installation required. Video walkthroughs: Follow our video guides for a step-by-step introduction Technical insights: Understand the technical choices made during development of each Blueprint Practical use-cases: See how other developers are customizing and extending these Blueprints for their needs. Join our community: Share your blueprints, learn from fellow innovators, and help expand the hub. Ready to transform your AI projects? Join us and see how Mozilla.ai Blueprints Hub can speed up your development and spark your creativity. Visit our website now to explore, experiment, and become part of our vibrant community. Your next great idea is just a click away! Ready to transform your AI projects? Explore the Blueprints Hub The post Open-source AI is hard. Blueprints can help! appeared first on The Mozilla Blog.

- [Mozilla Privacy Blog: Mozilla Respond to the White House’s RFI on AI](#) (2025/03/24 19:31)

The Future of AI Must Be Open, Competitive, and Accountable The internet has always thrived on openness, access, and broad participation. But as we enter the AI era, these core principles are at risk. A handful of dominant tech companies are positioned to control major AI systems, threatening both competition and innovation. At Mozilla, we believe AI should serve the public interest—not just corporate bottom lines. Earlier this month, we responded to the White House Office of Science and Technology Policy’s request for input on AI policy, where we offered a roadmap for a more open and trustworthy AI future (view Mozilla’s full submission here). Here’s what we think should happen. 1. AI Policy Must Prioritize Openness, Competition, and Accountability Right now, too much AI development stays behind closed doors. Proprietary models dominate, creating a landscape where users and developers have little insight—or control—over the AI systems shaping our digital lives. If we want AI that benefits everyone, we need strong policies that promote: Openness: Encouraging open-source AI development ensures transparency, security, and accessibility. Competition: Preventing monopolistic control keeps AI innovation dynamic and diverse. Accountability: Effective governance can mitigate AI’s risks while fostering responsible development. By advancing these principles, we can build an AI ecosystem that empowers users rather than locking them into closed, corporate-controlled systems. 2. The Government Should Support Public AI Infrastructure AI’s future shouldn’t be dictated solely by private companies. Public investment is key to ensuring broad access to AI tools and research. We support initiatives like the National AI Research Resource (NAIRR), which would provide universities, researchers, and small businesses with AI computing power and resources. We hope to see federal, state, and local governments increasingly adopt open source AI models in their workflows, which can help save taxpayers money, increase efficiency, and prevent vendor lock-in. Public AI infrastructure levels the playing field, allowing more voices to shape AI’s future and facilitating innovation across America, not just in a few tech hubs. 3. Open-Source AI Should Be Encouraged, Not Restricted Discussions about restricting open-source AI through export controls often miss the point about how to ensure national leadership in AI. Open-source AI fosters innovation, improves security, and lowers costs—critical benefits for businesses, researchers, and everyday users around the world. For the United States, promoting open source AI means more global products would be built

on top of American AI innovation. A 2025 McKinsey report, “Open source in the age of AI,” created in collaboration with Mozilla, found that 60% of decision-makers reported lower implementation costs with open-source AI compared to proprietary tools. Restricting open models would stifle progress and put the U.S. at a competitive disadvantage. Instead, we urge policymakers to support the open-source AI ecosystem and resist governance approaches that restrict AI models overall rather than making more targeted and precise interventions for AI harms. 4. AI Energy Consumption Needs Transparency AI systems consume enormous amounts of energy, and this demand is only growing. To prevent AI from straining our power grids and driving up costs, we need better transparency into AI’s resource consumption so that we can plan infrastructure development more effectively. The federal government should work with the industry to collect and share data on AI energy use. By understanding AI’s impact on infrastructure, we can promote sustainable innovation. 5. The U.S. Must Invest in AI Talent Development AI leadership isn’t just about technology—it’s about people. To remain competitive, the U.S. needs a strong, diverse workforce of AI researchers and practitioners. That means investing in: Community colleges and public universities to train the next generation of AI professionals. Apprenticeship and retraining programs to help workers adapt to AI-driven industries and adopt AI in every type of business across the economy from manufacturing to retail. Public-private partnerships that create novel education pathways for students, like Dakota State University’s collaboration with ArmyCyber. By growing the AI talent ecosystem, we ensure that AI works for people—not the other way around. The Path Forward AI is one of the most transformative technologies of our time. But without strong policies, it risks becoming yet another tool for big tech consolidation and unchecked corporate power. At Mozilla, we believe in an AI future that is open, competitive, and accountable. We call on policymakers to take bold steps—supporting open-source AI, investing in public infrastructure, and fostering fair competition—to ensure AI works for everyone. The post Mozilla Respond to the White House’s RFI on AI appeared first on Open Policy & Advocacy.

- [Niko Matsakis: Dyn async traits, part 10: Box box box](#) (2025/03/24 19:00)

This article is a slight divergence from my Rust in 2025 series. I wanted to share my latest thinking about how to support dyn Trait for traits with async functions and, in particular how to do so in a way that is compatible with the soul of Rust. Background: why is this hard? Supporting async fn in dyn traits is a tricky balancing act. The challenge is reconciling two key things people love about Rust: its ability to express high-level, productive code and its focus on revealing low-level details. When it comes to async function in traits, these two things are in direct tension, as I explained in my first blog post in this series – written almost four years ago! (Geez.) To see the challenge, consider this example Signal trait: trait Signal { async fn signal(&self); } In Rust today you can write a function that takes an impl Signal and invokes signal and everything feels pretty nice: async fn send_signal_1(impl_trait: &impl Signal) { impl_trait.signal().await; } But what I want to write that same function using a dyn Signal? If I write this... async fn send_signal_2(dyn_trait: &dyn Signal) { dyn_trait.signal().await; // ----- ERROR } ...I get an error. Why is that? The answer is that the compiler needs to know what kind of future is going to be returned by signal so that it can be awaited. At minimum it needs to know how big that future is so it can allocate space for it¹. With an impl Signal, the compiler knows exactly what type of signal you have, so that’s no problem: but with a dyn Signal, we don’t, and hence we are stuck. The most common solution to this problem is to box the future that results. The async-trait crate, for example, transforms async fn signal(&self) to something like fn signal(&self) -> Box<dyn Future<Output = ()> + '_>. But doing that at the trait level means that we add overhead even when you use impl Trait; it also rules out some applications of Rust async, like embedded or kernel development. So the name of the game is to find ways to let people use dyn Trait that are both convenient and flexible. And that turns out to be pretty hard! The “box box box” design in a nutshell I’ve been digging back into the problem lately in a series of conversations with Michal Goulet (aka, compiler-errors) and it’s gotten me thinking about a fresh approach I call “box box box”. The “box box

box” design starts with the call-site selection approach. In this approach, when you call `dyn_trait.signal()`, the type you get back is a `dyn Future` – i.e., an unsized value. This can’t be used directly. Instead, you have to allocate storage for it. The easiest and most common way to do that is to box it, which can be done with the new `.box` operator: `async fn send_signal_2(dyn_trait: &dyn Signal) { dyn_trait.signal().box.await; // ----- //` Results in a `Box<dyn Future<Output = ()>>` . } This approach is fairly straightforward to explain. When you call an async function through dyn Trait, it results in a dyn Future, which has to be stored somewhere before you can use it. The easiest option is to use the .box operator to store it in a box; that gives you a Box<dyn Future>, and you can await that. But this simple explanation belies two fairly fundamental changes to Rust. First, it changes the relationship of Trait and dyn Trait. Second, it introduces this .box operator, which would be the first stable use of the box keyword2. It seems odd to introduce the keyword just for this one use – where else could it be used? As it happens, I think both of these fundamental changes could be very good things. The point of this post is to explain what doors they open up and where they might take us.`

Change 0: Unsized return value methods Let’s start with the core proposal. For every trait `Foo`, we add inherent methods³ to `dyn Foo` reflecting its methods: For every `fn f` in `Foo` that is `dyn compatible`, we add a `<dyn Foo>::f` that just calls `f` through the `vtable`. For every `fn f` in `Foo` that returns an `impl Trait` value but would otherwise be `dyn compatible` (e.g., no generic arguments⁴, no reference to `Self` beyond the `self` parameter, etc), we add a `<dyn Foo>::f` method that is defined to return a `dyn Trait`. This includes `async fns`, which are sugar for functions that return `impl Future`. In fact, method dispatch already adds “pseudo” inherent methods to `dyn Foo`, so this wouldn’t change anything in terms of which methods are resolved. The difference is that `dyn Foo` is only allowed if all methods in the trait are `dyn compatible`, whereas under this proposal some non-`dyn-compatible` methods would be added with modified signatures.

Change 1: `Dyn compatibility` Change 0 only makes sense if it is possible to create a `dyn Trait` even though it contains some methods (e.g., `async functions`) that are not `dyn compatible`. This revisits RFC #255, in which we decided that the `dyn Trait` type should also implement the trait `Trait`. I was a big proponent of RFC #255 at the time, but I’ve since decided I was mistaken⁵. Let’s discuss. The two rules today that allow `dyn Trait` to implement `Trait` are as follows: By disallowing `dyn Trait` unless the trait `Trait` is `dyn compatible`, meaning that it only has methods that can be added to a `vtable`. By requiring that the values of all associated types be explicitly specified in the `dyn Trait`. So `dyn Iterator<Item = u32>` is legal but not `dyn Iterator` on its own. “`dyn compatibility`” can be powerful The fact that `dyn Trait` implements `Trait` is at times quite powerful. It means for example that I can write an implementation like this one: `struct RcWrapper<T: ?Sized> { r: Rc<RefCell<T>> } impl<T> Iterator for RcWrapper<T> where T: ?Sized + Iterator, { type Item = T::Item; fn next(&mut self) -> Option<T::Item> { self.borrow_mut().next() } }` This `impl` makes `RcWrapper<I>` implement `Iterator` for any type `I`, including `dyn trait` types like `RcWrapper<dyn Iterator<Item = u32>>`. Neat. “`dyn compatibility`” doesn’t truly live up to its promise Powerful as it is, the idea of `dyn Trait` implementing `Trait` doesn’t quite live up to its promise. What you really want is that you could replace any `impl Trait` with `dyn Trait` and things would work. But that’s just not true because `dyn Trait` is `?Sized`. So actually you don’t get a very “smooth experience”. What’s more, although the compiler gives you a `dyn Trait: Trait impl`, it doesn’t give you `impls` for references to `dyn Trait` – so e.g. given this trait `trait Compute { fn compute(&self); }` If I have a `Box<dyn Compute>`, I can’t give that to a function that takes an `impl Compute` `fn do_compute(i: impl Compute) { } fn call_compute(b: Box<dyn Compute>) { do_compute(b); // ERROR }` To make that work, somebody has to explicitly provide an `impl` like `impl<I> Compute for Box<I> where I: ?Sized, { // ... }` and people often don’t. “`dyn compatibility`” can be limiting However, the requirement that `dyn Trait` implement `Trait` can be limiting. Imagine a trait like `trait ReportError { fn report(&self, error: Error); fn report_to(&self, error: Error, target: impl ErrorTarget); // ----- // Generic argument. }` This trait has two methods. The `report` method is `dyn-compatible`, no problem. The `report_to` method has an `impl Trait` argument is therefore generic, so it is not `dyn-compatible`⁶ (well, at least not

under today's rules, but I'll get to that). (The reason `report_to` is not dyn compatible: we need to make distinct monomorphized copies tailored to the type of the target argument. But the vtable has to be prepared in advance, so we don't know which monomorphized version to use.) And yet, just because `report_to` is not dyn compatible mean that a dyn `ReportError` would be useless. What if I only plan to call `report`, as in a function like this? `fn report_all(errors: Vec<Error>, report: &dyn ReportError,) { for e in errors { report.report(e); } }` Rust's current rules rule out a function like this, but in practice this kind of scenario comes up quite a lot. In fact, it comes up so often that we added a language feature to accommodate it (at least kind of): you can add a `where Self: Sized` clause to your feature to exempt it from dynamic dispatch. This is the reason that `Iterator` can be dyn compatible even when it has a bunch of generic helper methods like `map` and `flat_map`. What does all this have to do with AFIDT? Let me pause here, as I imagine some of you are wondering what all of this "dyn compatibility" stuff has to do with AFIDT. The bottom line is that the requirement that dyn `Trait` type implements `Trait` means that we cannot put any kind of "special rules" on dyn dispatch and that is not compatible with requiring a `.box` operator when you call async functions through a dyn trait. Recall that with our `Signal` trait, you could call the `signal` method on an impl `Signal` without any boxing: `async fn send_signal_1(impl_trait: &impl Signal) { impl_trait.signal().await; }` But when I called it on a dyn `Signal`, I had to write `.box` to tell the compiler how to deal with the dyn `Future` that gets returned: `async fn send_signal_2(dyn_trait: &dyn Signal) { dyn_trait.signal().box.await; }` Indeed, the fact that `Signal::signal` returns an impl `Future` but `<dyn Signal>::signal` returns a dyn `Future` already demonstrates the problem. All impl `Future` types are known to be `Sized` and dyn `Future` is not, so the type signature of `<dyn Signal>::signal` is not the same as the type signature declared in the trait. Huh. Associated type values are needed for dyn compatibility Today I cannot write a type like dyn `Iterator` without specifying the value of the associated type `Item`. To see why this restriction is needed, consider this generic function: `fn drop_all<I: ?Sized + Iterator>(iter: &mut I) { while let Some(n) = iter.next() { std::mem::drop(n); } }` If you invoked `drop_all` with an `&mut dyn Iterator` that did not specify `Item`, how could the type of `n`? We wouldn't have any idea how much space it needs. But if you invoke `drop_all` with `&mut dyn Iterator<Item = u32>`, there is no problem. We don't know which `next` method is being called, but we know it's returning a `u32`. Associated type values are limiting And yet, just as we saw before, the requirement to list associated types can be limiting. If I have a dyn `Iterator` and I only call `size_hint`, for example, then why do I need to know the `Item` type? `fn size_hint(iter: &mut dyn Iterator) -> bool { let sh = iter.size_hint(); }` But I can't write code like this today. Instead I have to make this function generic which basically defeats the whole purpose of using dyn `Iterator`: `fn size_hint<T>(iter: &mut dyn Iterator<Item = T>) -> bool { let sh = iter.size_hint(); }` If we dropped the requirement that every dyn `Iterator` type implements `Iterator`, we could be more selective, allowing you to invoke methods that don't use the `Item` associated type but disallowing those that do. A proposal for expanded dyn `Trait` usability So that brings us to full proposal to permit dyn `Trait` in cases where the trait is not fully dyn compatible: dyn `Trait` types would be allowed for any trait.⁷ dyn `Trait` types would not require associated types to be specified. dyn compatible methods are exposed as inherent methods on the dyn `Trait` type. We would disallow access to the method if its signature references associated types not specified on the dyn `Trait` type. dyn `Trait` that specify all of their associated types would be considered to implement `Trait` if the trait is fully dyn compatible.⁸ The `box` keyword A lot of things get easier if you are willing to call `malloc`. – Josh Triplett, recently. Rust has reserved the `box` keyword since 1.0, but we've never allowed it in stable Rust. The original intention was that the term `box` would be a generic term to refer to any "smart pointer"-like pattern, so `Rc` would be a "reference counted box" and so forth. The `box` keyword would then be a generic way to allocate boxed values of any type; unlike `Box::new`, it would do "emplacement", so that no intermediate values were allocated. With the passage of time I no longer think this is such a good idea. But I do see a lot of value in having a keyword to ask the compiler to automatically create boxes. In fact, I see a lot of places where that could be useful. boxed expressions The first

place is indeed the `.box` operator that could be used to put a value into a box. Unlike `Box::new`, using `.box` would allow the compiler to guarantee that no intermediate value is created, a property called emplacement. Consider this example: `fn main() { let x = Box::new([0_u32; 1024]); }` Rust's semantics today require (1) allocating a 4KB buffer on the stack and zeroing it; (2) allocating a box in the heap; and then (3) copying memory from one to the other. This is a violation of our Zero Cost Abstraction promise: no C programmer would write code like that. But if you write `[0_u32; 1024].box`, we can allocate the box up front and initialize it in place.⁹ The same principle applies calling functions that return an unsized type. This isn't allowed today, but we'll need some way to handle it if we want to have `async fn` return `dyn Future`. The reason we can't naively support it is that, in our existing ABI, the caller is responsible for allocating enough space to store the return value and for passing the address of that space into the callee, who then writes into it. But with a `dyn Future` return value, the caller can't know how much space to allocate. So they would have to do something else, like passing in a callback that, given the correct amount of space, performs the allocation. The most common case would be to just pass in `malloc`. The best ABI for unsized return values is unclear to me but we don't have to solve that right now, the ABI can (and should) remain unstable. But whatever the final ABI becomes, when you call such a function in the context of a `.box` expression, the result is that the callee creates a `Box` to store the result.¹⁰

boxed `async` functions to permit recursion If you try to write an `async` function that calls itself today, you get an error: `async fn fibonacci(a: u32) -> u32 { match a { 0 => 1, 1 => 2, _ => fibonacci(a-1).await + fibonacci(a-2).await } }` The problem is that we cannot determine statically how much stack space to allocate. The solution is to rewrite to a boxed return value. This compiles because the compiler can allocate new stack frames as needed. `fn fibonacci(a: u32) -> Pin<Box<impl Future<Output = u32>>> { Box::pin(async move { match a { 0 => 1, 1 => 2, _ => fibonacci(a-1).await + fibonacci(a-2).await } }) }` But wouldn't it be nice if we could request this directly? `box async fn fibonacci(a: u32) -> u32 { match a { 0 => 1, 1 => 2, _ => fibonacci(a-1).await + fibonacci(a-2).await } }`

boxed structs can be recursive A similar problem arises with recursive structs: `struct List { value: u32, next: Option<List>, // ERROR }` The compiler tells you `error[E0072]: recursive type `List` has infinite size --> src/lib.rs:1:1 | 1 | struct List { | ^^^^^^^^^^^^^^^ 2 | value: u32, 3 | next: Option<List>, // ERROR | ---- recursive without indirection | help: insert some indirection (e.g., a `Box`, `Rc`, or `&`) to break the cycle | 3 | next: Option<Box<List>>, // ERROR | + + + + + As it suggests, to work around this you can introduce a Box: struct List { value: u32, next: Option<Box<List>>, } This though is kind of weird because now the head of the list is stored "inline" but future nodes are heap-allocated. I personally usually wind up with a pattern more like this: struct List { data: Box<ListData> } struct ListData { value: u32, next: Option<List>, } Now however I can't create values with List { value: 22, next: None } syntax and I also can't do pattern matching. Annoying. Wouldn't it be nice if the compiler just suggest adding a box keyword when you declare the struct: box struct List { value: u32, next: Option<List>, // ERROR } and have List { value: 22, next: None } automatically allocate the box for me? The ideal is that the presence of a box is now completely transparent, so I can pattern match and so forth fully transparently: box struct List { value: u32, next: Option<List>, // ERROR }`

`fn foo(list: &List) { let List { value, next } = list; // etc }`

boxed enums can be recursive and right-sized Enums too cannot reference themselves. Being able to declare something like this would be really nice: `box enum AstExpr { Value(u32), If(AstExpr, AstExpr, AstExpr), ... }` In fact, I still remember when I used Swift for the first time. I wrote a similar enum and Xcode helpfully prompted me, "do you want to declare this enum as indirect?" I remember being quite jealous that it was such a simple edit. However, there is another interesting thing about a `box` enum. The way I imagine it, creating an instance of the enum would always allocate a fresh box. This means that the enum cannot be changed from one variant to another without allocating fresh storage. This in turn means that you could allocate that box to exactly the size you need for that particular variant.¹¹ So, for your `AstExpr`, not only could it be recursive, but when you allocate an `AstExpr::Value` you only need to allocate space for a `u32`,

whereas a `AstExpr::If` would be a different size. (We could even start to do “tagged pointer” tricks so that e.g. `AstExpr::Value` is stored without any allocation at all.) boxed enum variants to avoid unbalanced enum sizes Another option would to have particular enum variants that get boxed but not the enum as a whole: `enum AstExpr { Value(u32), box If(AstExpr, AstExpr, AstExpr), ... }` This would be useful in cases you do want to be able to overwrite one enum value with another without necessarily reallocating, but you have enum variants of widely varying size, or some variants that are recursive. A boxed variant would basically be desugared to something like the following: `enum AstExpr { Value(u32), If(Box<AstExpr>, ...) struct AstExprIf(AstExpr, AstExpr, AstExpr);` `clippy` has a useful lint `large_enum_variant` that aims to identify this case, but once the lint triggers, it’s not able to offer an actionable suggestion. With the `box` keyword there’d be a trivial rewrite that requires zero code changes. `box` patterns and types If we’re enabling the use of `box` elsewhere, we ought to allow it in patterns: `fn foo(s: box Struct) { let box Struct { field } = s; }` Frequently asked questions Isn’t it unfortunate that `Box::new(v)` and `v.box` would behave differently? Under my proposal, `v.box` would be the preferred form, since it would allow the compiler to do more optimization. And yes, that’s unfortunate, given that there are 10 years of code using `Box::new`. Not really a big deal though. In most of the cases we accept today, it doesn’t matter and/or LLVM already optimizes it. In the future I do think we should consider extensions to make `Box::new` (as well as `Rc::new` and other similar constructors) be just as optimized as `.box`, but I don’t think those have to block this proposal. Is it weird to special case `box` and not handle other kinds of smart pointers? Yes and no. On the one hand, I would like the ability to declare that a struct is always wrapped in an `Rc` or `Arc`. I find myself doing things like the following all too often: `struct Context { data: Arc<ContextData> } struct ContextData { counter: AtomicU32, }` On the other hand, `box` is very special. It’s kind of unique in that it represents full ownership of the contents which means a `T` and `Box<T>` are semantically equivalent – there is no place you can use `T` that a `Box<T>` won’t also work – unless `T: Copy`. This is not true for `T` and `Rc<T>` or most other smart pointers. For myself, I think we should introduce `box` now but plan to generalize this concept to other pointers later. For example I’d like to be able to do something like this... `#[indirect(std::sync::Arc)] struct Context { counter: AtomicU32, }` ...where the type `Arc` would implement some trait to permit allocating, deref’ing, and so forth: `trait SmartPointer: Deref { fn alloc(data: Self::Target) -> Self; }` The original plan for `box` was that it would be somehow type overloaded. I’ve soured on this for two reasons. First, type overloads make inference more painful and I think are generally not great for the user experience; I think they are also confusing for new users. Finally, I think we missed the boat on naming. Maybe if we had called `Rc` something like `RcBox<T>` the idea of “box” as a general name would have percolated into Rust users’ consciousness, but we didn’t, and it hasn’t. I think the `box` keyword now ought to be very targeted to the `Box` type. How does this fit with the “soul of Rust”? In my [soul of Rust blog post], I talked about the idea that one of the things that make Rust Rust is having allocation be relatively explicit. I’m of mixed minds about this, to be honest, but I do think there’s value in having a property similar to `unsafe` – like, if allocation is happening, there’ll be a sign somewhere you can find. What I like about most of these `box` proposals is that they move the `box` keyword to the declaration – e.g., on the struct/enum/etc – rather than the use. I think this is the right place for it. The major exception, of course, is the “marquee proposal”, invoking `async` fns in `dyn` trait. That’s not amazing. But then... see the next question for some early thoughts. If traits don’t have to be `dyn` compatible, can we make `dyn` compatibility opt in? The way that Rust today detects automatically whether traits should be `dyn` compatible versus having it be declared is, I think, `nogr eat`. It creates confusion for users and also permits quiet `semver` violations, where a new defaulted method makes a trait no longer be `dyn` compatible. It’s also a source for a lot of soundness bugs over time. I want to move us towards a place where traits are not `dyn` compatible by default, meaning that `dyn Trait` does not implement `Trait`. We would always allow `dyn Trait` types and we would allow individual items to be invoked so long as the item itself is `dyn` compatible. If you want to have `dyn Trait` implement `Trait`, you should declare it, perhaps with a `dyn`

keyword: dyn trait Foo { fn method(&self); } This declaration would add various default impls. This would start with the dyn Foo: Foo impl: impl Foo for dyn Foo /*[1]*/ { fn method(&self) { <dyn Foo>::method(self) // vtable dispatch } // [1] actually it would want to cover `dyn Foo + Send` etc too, but I'm ignoring that for now } But also, if the methods have suitable signatures, include some of the impls you really ought to have to make a trait that is well-behaved with respect to dyn trait: impl<T> Foo for Box<T> where T: ?Sized { } impl<T> Foo for &T where T: ?Sized { } impl<T> Foo for &mut T where T: ?Sized { } In fact, if you add in the ability to declare a trait as box, things get very interesting: box dyn trait Signal { async fn signal(&self); } I'm not 100% sure how this should work but what I imagine is that dyn Foo would be pointer-sized and implicitly contain a Box behind the scenes. It would probably automatically Box the results from async fn when invoked through dyn Trait, so something like this: impl Foo for dyn Signal { async fn bar(&self) { <dyn Signal>::signal(self).box.await } } I didn't include this in the main blog post but I think together these ideas would go a long way towards addressing the usability gaps that plague dyn Trait today. Side note, one interesting thing about Rust's async functions is that their size must be known at compile time, so we can't permit alloca-like stack allocation. ← The box keyword is in fact reserved already, but it's never been used in stable Rust. ← Hat tip to Michael Goulet (compiler-errors) for pointing out to me that we can model the virtual dispatch as inherent methods on dyn Trait types. Before I thought we'd have to make a more invasive addition to MIR, which I wasn't excited about since it suggested the change was more far-reaching. ← In the future, I think we can expand this definition to include some limited functions that use impl Trait in argument position, but that's for a future blog post. ← I've noticed that many times when I favor a limited version of something to achieve some aesthetic principle I wind up regretting it. ← At least, it is not dyn compatible under today's rules. Convievably it could be made to work but more on that later. ← This part of the change is similar to what was proposed in RFC #2027, though that RFC was quite light on details (the requirements for RFCs in terms of precision have gone up over the years and I expect we wouldn't accept that RFC today in its current form). ← I actually want to change this last clause in a future edition. Instead of having dyn compatibility be determined automatically, traits would declare themselves dyn compatible, which would also come with a host of other impls. But that's worth a separate post all on its own. ← If you play with this on the playground, you'll see that the memcpy appears in the debug build but gets optimized away in this very simple case, but that can be hard for LLVM to do, since it requires reordering an allocation of the box to occur earlier and so forth. The .box operator could be guaranteed to work. ← I think it would be cool to also have some kind of unsafe intrinsic that permits calling the function with other storage strategies, e.g., allocating a known amount of stack space or what have you. ← We would thus finally bring Rust enums to "feature parity" with OO classes! I wrote a blog post, "Classes strike back", on this topic back in 2015 (!) as part of the whole "virtual structs" era of Rust design. Deep cut! ←

- [William Durand: Firefox AI & WebExtensions](#) (2025/03/24 00:00)

I gave an introduction to the Firefox AI runtime and WebExtensions at a French local conference this month. This article is a loose transcript of what I said. Let's talk about Firefox, AI, and WebExtensions. Browser extensions Browser extensions are tiny applications that modify and/or add features to a web browser. Nowadays, these small programs can be written in such a way that they should be compatible with different browsers. That's because there exists a cross-browser system called "WebExtensions", which - among other things - provides a set of common APIs that browser extensions can use. In addition to that, browsers can also expose their own APIs, and we'll see that in a moment. You'll find a lot more information on this MDN page. Note: During my talk, I used the Borderify extension to walk the audience through an example of a web extension. I then concluded that it's super easy to get started but also very powerful. Extensions like uBlock Origin, Dark Reader, 1Password, etc. are rather powerful and sophisticated features. Firefox AI runtime Firefox has a new component based on Transformers.js and the ONNX runtime to perform

local inference directly in the browser. In short, this runtime allows to use a model from Hugging Face (like GitHub but for Machine Learning models) directly in Firefox¹, without the need to send data to any servers. Every operation is performed on the user's machine once the model files have been downloaded by Firefox. While every website could technically load Transformers.js and a model, this isn't very efficient. Say two websites use the same model, you end up with two copies of the model files. And those aren't exactly small. This Firefox component – also known as the Firefox (AI) runtime – addresses this problem by ensuring that models are shared. In addition, this runtime takes care of managing resources, and model inference is isolated from the rest of Firefox in an inference process. Note: During my talk, I mentioned that – while we call this “AI” now – Mozilla has been doing Machine Learning (ML) for a very long time². For instance, Firefox Translations isn't exactly new, and whether you like it or not, this is clearly an application of Generative AI³. Same thing, still. Anyway, let's see how we can interact with this runtime. We're going to generate text that describes an image in the rest of this section. The “hacker's approach” is probably to open a Browser console in Nightly, and run some privileged JavaScript code: `const { createEngine } = ChromeUtils.importESModule("chrome://global/content/ml/EngineProcess.sys.mjs"); const options = { taskName: "image-to-text", modelHub: "mozilla", }; const engine = await createEngine(options); const [res] = await engine.run({ args: ["https://williamdurand.fr/images/posts/2014/12/brussels.jpg"], }); // res => { generated_text: "A large poster of a man on a wall." }` As mentioned previously, the Firefox runtime is based on Transformers.js, which is why the code looks familiar when you know Transformers already. For instance, instead of passing an actual model here, we pass a task name. That's an abstraction coming from Transformers. Don't worry, we can also pass a model name and a lot of other (pipeline) options! For those looking for a more graphical approach to play with this AI runtime, Firefox Nightly provides an `about:inference` page that looks like this: That's cool but... Why? Well, it turns out this example isn't a random example. This code snippet is an overly simplified version of a feature in Firefox's PDF reader (PDF.js ♥): alt text generation⁴. Screenshot of an “Edit alt text” dialog in PDF.js (inside Firefox): the image description has been automatically generated. Note: During my talk, someone asked a question about the use of GPU, for which I didn't have the answer. I do have it now, though. The Firefox AI runtime runs on CPU by default, but it is possible to run on GPU via WebGPU. It's worth mentioning that this runtime doesn't feel as fast as a more “native” solution (like Ollama) yet but the team at Mozilla is working on it! Anyhow, let's move to the final part of this talk article. WebExtensions ML API The best of both worlds, yay! We shipped an experimental WebExtensions API to allow extensions to do local inference (docs), leveraging the Firefox AI runtime under the hood. Expect things to evolve, it's bleeding edge technology! At the time of writing this, we can rewrite the example from the previous section into “extension code” as follows: `const options = { taskName: "image-to-text", modelHub: "mozilla", }; await browser.trial.ml.createEngine(options); const [res] = await browser.trial.ml.runEngine({ args: ["https://williamdurand.fr/images/posts/2014/12/brussels.jpg"], }); // res => { generated_text: "A large poster of a man on a wall." }` This looks similar, right? That's on purpose. For extension developers, the WebExtensions API namespace is `trial.ml`, and the associated permission is named `trialML`, which extensions must request at runtime. What can we do with that, though? Well, what if we were to provide the alt-text-generation feature not just in PDFs but for any image on any website? That's what we have done in a demo extension (code, docs), which we can see in action in the screencast below: At 00:00, the demo extension has been loaded in Firefox Nightly. At 00:02, we open the context menu on an image in the current web page, and we click on “Generate Alt Text”. This menu entry has been added by the extension using the `menus` API by the way. At 00:05, we can see that Firefox is downloading the model files (the UI is provided by the extension, which receives events from Firefox). This means the model was not used before so the Firefox AI runtime has to download the model files first. This step is only necessary when Firefox doesn't already have the model used by the extension. At 00:09, the model inference starts. At 00:12, the result of the inference,

which is the description of the image in this case, is returned to the extension, and the extension shows it to the user. Your browser doesn't support HTML video. Previously, I mentioned that browser extensions can be cross-browser. They can also run on different platforms as well. In Bug 1954362, I updated this demo extension so that it can run on Firefox for Android. The screencast below shows the same extension running in Firefox for Android: At 00:00, we can see a dialog because the extension has just been installed. At 00:01, the extension opened a page in a new tab to request permission to interact with the Firefox AI runtime. This is pretty standard for browser extensions to request permissions ahead of time. At 00:06, we load a web page with an image. At 00:10, we use long-press on the image to trigger the extension because the menus API is not supported on Android yet (Bug 1595822). At 00:12, similar to the previous screencast, Firefox starts by downloading the model files. This takes a lot of time because my emulator isn't exactly fast. Do remember that this is only needed once, though. At 01:09, the model inference starts. At 01:12, the result of the inference, which is – again – the description of the image, is returned to the extension, and the extension shows it to the user. Your browser doesn't support HTML video. And that's basically it. I am personally looking forward to see what extension developers could do with this new capability in Firefox! And since this is related to my work at Mozilla, feel free to get in touch if you have questions. Mozilla has its own “hub” too, so it isn't just from Hugging Face. ← I wrote a bit about my use of ML at work in 2019 in this article. ← Firefox Translations generates text so this can be considered Generative AI. A major difference is that this doesn't rely on a Large Language Model (LLM). Instead, Translations uses (Marian) Neural Machine Translation (NMT) models and the Bergamot runtime. ← My colleague Tarek wrote an extensive Hacks article about this in 2024. ←

- [Don Marti: power moves, signaling, and a helpful book for understanding Big Tech](#) (2025/03/23 00:00)

I'm still waiting for my copy of *Careless People* by Sarah Wynn-Williams, so I don't have anything more on the content of the book than what I have seen in other reviews. The local bookstore had a stack—more than they normally order for new hardcovers—but I hesitated and they were gone next time I went in there. So yes, I am a little behind on this. But come on, people. Does anyone seriously think that Meta execs don't understand the Streisand effect? Did Meta lawyers really not read the arbitration agreement and understand that it doesn't apply to the publisher? *Careless People* is a best-seller because Meta decision-makers want it to be a best-seller. In other Big Tech news, Google is delivering ads for obvious malware, with a landing page featuring an unauthorized copy of one of Google's own logos. Even worse, they got spotted placing ads on Child Sexual Abuse Material. At first these look like embarrassing self-owns, especially for a company that's contending for favorable PR in the AI business. Is their AI really that bad at classifying landing pages, extension listings, and the content of sites where their ads appear? The search ad malware thing is particularly egregious—the whole point of the deceptive ads that are all over Google Search now is to impersonate some well-known company. It should be a high school level coding project to filter out some of these. But Big Tech's apparent eagerness to appear in bad news makes sense when you look at the results. Out of all the people who read and were outraged by *Careless People* over the weekend, how many are going to come in to work on Monday and delete their Meta tracking pixel or turn off Meta CAPI? And how many people concerned about Google's malware, CSAM, and infringing content problems are going to switch to inclusion lists and validated SupplyChain objects and stop with the crappy, often illegal ad placements that Google recommends and legit ad agencies don't? For Big Tech, doing crimes in an obvious way is a power move, a credible, costly signal. If there were a Meta alternative that didn't do genocide, or an honest alternative to Google search advertising, then advertising decision-makers would have switched to them already. All these embarrassing-looking stories are a signal: don't waste your time looking for an alternative to paying us. The publisher's page for *Careless People* has a Meta pixel on it. I do have a book recommendation that might be a little easier to get a hold of. *Codes of the Underworld* by Diego Gambetta was the weekly book

recommendation on *A Collection of Unmitigated Pedantry*. I'm glad to see that it is still in print, because it's a useful way to help understand the Big Tech companies. Actions that might not have made sense in a company's old create more value than you capture days are likely to be easier to figure out after understanding the considerations applied by other criminal organizations. Criminals have hard-to-satisfy communications needs, such as the need to convey a credible threat to a victim without attracting the attention of enforcers. This is related to the signaling problem faced by honest advertisers, but in reverse. How can a representative of a protection racket indicate to a small business that they represent a true threat, and aren't just bluffing? Gambetta digs into a variety of signaling problems. It's a 2009 book, so many of the Big Tech firms were still legit when it came out, but a lot of the communications methods from back then apply to the companies of today. Is there a solution? As Gambetta points out, real-life organized crime perpetrators tend to copy from the movies, and today they're copying the partnership with a friendly government subplot from *The Godfather Part II*. Maybe it's time to watch that movie again. Related some ways that Facebook ads are optimized for deceptive advertising trying to think about European tech policy in context Bonus links Meta settles UK right to object to ad-tracking lawsuit by agreeing not to track plaintiff by Natasha Lomas. (No details on how to replicate this result, though.) (discuss) Google and Its Confederate AI Platforms Want Retroactive Absolution For AI Training Wrapped in the American Flag by Chris Castle. (If only they could be this patriotic when it's time to pay their damn taxes.) Privacy-Respecting European Tech Alternatives by Jonah Aragon. [T]he United States certainly does not have a monopoly on the best technologies, and many of our favorite recommended tools come from Europe and all over the world. Tools from the European Union also generally benefit from much stronger data protection laws, thanks to the EU's General Data Protection Regulation (GDPR). Related: But how to get to that European cloud? Please stop externalizing your costs directly into my face by Drew DeVault. Whether it's cryptocurrency scammers mining with FOSS compute resources or Google engineers too lazy to design their software properly or Silicon Valley ripping off all the data they can get their hands on at everyone else's expense... Instead of F-35, Portugal turns to Europe in search of new fighter by Ricardo Meier. Melo stated that the predictability of our allies is a greater asset to take into account. We have to believe that, in all circumstances, these allies will be on our side.

- [The Mozilla Blog: Reaching readers, one TikTok at a time](#) (2025/03/20 17:58)

<figcaption class="wp-element-caption">Sanibel Lazar is using TikTok to break beyond traditional book promotion. </figcaption> Spoiler: The internet's not finished. Welcome to "Web in Progress," a series celebrating the internet as a space you can shape, not just scroll through. Just as Firefox empowers you to take charge of your online experience, this series highlights how individuals and communities are shaping an internet that truly serves their needs. In this installment, see how a debut novelist is using TikTok to break beyond traditional book promotion. By focusing on niche interests, she found new ways to connect with readers who might never have picked up her book. Her experience is a testament to how digital platforms can open unexpected doors. Before I started promoting my debut novel, "To Have and Have More," I hadn't posted on any social media platform since 2015. Creating content wasn't in my plan — until I realized it was the most practical way to get my book noticed. Working with a brand-new press meant I had to carve out my own opportunities. Social media was a means to feel like I was in the driver's seat as my book went out into the world. Instead of feeling overwhelmed by the need to promote my book, I leaned into what I could control. I started creating videos on TikTok, not as part of BookTok, but tailored to themes from my book like class, privilege and wealth. They led me to unexpected audiences. I've ended up on PrivilegeTok, WealthTok, StatusSymbolTok and even QuietLuxuryTok — places where I stand out as the only person talking about a novel. My videos are a way to access audiences who might not otherwise pick up a book. When one of my videos hit a couple hundred thousand views, I checked my Amazon ranking and watched my book climb. Social media has also brought me opportunities I

didn't anticipate. Rather than chasing podcast invites or op-eds, I've been getting invitations to do readings and guest spots (it's thanks to social media that I got tapped to write this article) — all because people discovered me through my content. I'm not on TikTok to recommend books or talk about author life; instead I riff on social etiquette, classism, and luxury brands. My strategy isn't about jumping on trend bandwagons but about getting people interested in my book. I call this approach "Oblique Content," inspired by perfume ads that sell a mood or idea rather than focusing on product specs. In my videos, I talk about everything from toxic wealth to throwback millennial fashion trends — and I plug my book for ten seconds at the end. I got a DM recently from a follower who said she was shocked to see a certain high-end brand at TJMaxx and thought of me. That message was a small but significant sign: My content was resonating. People were connecting my name and voice with the themes of my book. For creatives, finding success on social media isn't as simple as racking up views. You want your followers to be interested in your body of work and your ideas — not just your ability to "stop the scroll." My advice? Experiment widely and don't pigeonhole yourself in the conventions of your genre. And don't get sidetracked scrolling for inspiration; focus on creating. Sanibel Lazar is a writer based in New York City. She earned her MFA from The New School and her BA in Classical Studies from the University of Pennsylvania. Her debut novel, "To Have and Have More," will be published in April 2025. Sanibel's work has appeared in New York Magazine, ELLE, Air Mail, Literary Hub and more. The post Reaching readers, one TikTok at a time appeared first on The Mozilla Blog.

- [This Week In Rust: This Week in Rust 591](#) (2025/03/19 04:00)

Hello and welcome to another issue of This Week in Rust! Rust is a programming language empowering everyone to build reliable and efficient software. This is a weekly summary of its progress and community. Want something mentioned? Tag us at @ThisWeekInRust on X (formerly Twitter) or @ThisWeekinRust on mastodon.social, or send us a pull request. Want to get involved? We love contributions. This Week in Rust is openly developed on GitHub and archives can be viewed at this-week-in-rust.org. If you find any errors in this week's issue, please submit a PR. Want TWIR in your inbox? Subscribe here. Updates from Rust Community Official Announcing Rust 1.85.1 Hiring for Rust program management March 2025 Leadership Council Update Project/Tooling Updates rust-analyzer #277 Git 2.49 Released With Faster Packing, Rust Foreign Language Interface Rust CUDA project update Big Rust Update Merged For GCC 15 - Lands The Polonius Borrow Checker Oxidizing Ubuntu: adopting Rust utilities by default Apache OpenDAL 2025 Roadmap: Perfecting Production Adoption bevy_lint v0.2.0: lint your Bevy projects Why Yōzefu? A terminal user interface to search for data in a Kafka cluster Announcing AIScript and How I Built It Announcing mocktail: HTTP & gRPC server mocking for Rust Observations/Thoughts How to speed up the Rust compiler in March 2025 Carefully But Purposefully Oxidising Ubuntu Below: World Writable Directory in /var/log/below Allows Local Privilege Escalation Extending Future In Rust Writing Terrible Code alp-rs is faster than c++ reference [video] It's Not As Simple As "Use A Memory Safe Language" [video] Ubuntu Will Replace GNU Core Utilities With Rust [video] What's up with Rust? Rust Walkthroughs Create A Web + Desktop Application With Rust Transition Systems in Rust Nine Pico PIO Wats with Rust: Raspberry Pi programmable IO Pitfalls Illustrated with a Musical Example (Part 2) [series] Building a search engine from scratch, in Rust [video] Build with Naz : newtype design pattern, and impl Into T for ergonomic APIs Miscellaneous February 2025 Rust Jobs Report Crate of the Week This week's crate is dom_smoothie, a crate for extracting readable content from web pages. Despite a lack of suggestions this week, llogiq is pleased with his choice. Please submit your suggestions and votes for next week! Calls for Testing An important step for RFC implementation is for people to experiment with the implementation and give feedback, especially before stabilization. If you are a feature implementer and would like your RFC to appear in this list, add a call-for-testing label to your RFC along with a comment providing testing instructions and/or guidance on which aspect(s) of the feature need testing. No calls for testing were issued this week by Rust, Rust language RFCs or Rustup. Let us know if you would

like your feature to be tracked as a part of this list. Call for Participation; projects and speakers CFP - Projects Always wanted to contribute to open-source projects but did not know where to start? Every week we highlight some tasks from the Rust community for you to pick and get started! Some of these tasks may also have mentors available, visit the task page for more information. If you are a Rust project owner and are looking for contributors, please submit tasks here or through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! CFP - Events Are you a new or experienced speaker looking for a place to share something cool? This section highlights events that are being planned and are accepting submissions to join their event as a speaker. If you are an event organizer hoping to expand the reach of your event, please submit a link to the website through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! Updates from the Rust Project 468 pull requests were merged in the last week Compiler perf:allow bounds checks when enumerating IndexSlice to be elided stabilize asm_goto feature gate Miri native_calls: ensure we actually expose mutable provenance to the memory FFI can access alloc_addresses: use MemoryKind instead of tcx query to determine global allocations Libraries add From<{integer}> for f16/f128 impls denote ControlFlow as #[must_use] optimize multi-char string patterns stabilize std::io::ErrorKind::InvalidFilename stablize anonymous pipe Cargo add custom completer for cargo +<TAB> to complete toolchain name deduplicate crate types in cargo rustc command Rustdoc add RTN support to rustdoc rustdoc-json: don't also include #[deprecated] in Item::attrs Rustfmt rustfmt: allow also allow literals as first item of single line let chain Clippy new lint: doc_comment_double_space_linebreaks incompatible_msrv: lint function calls with any argument count needless_pass_by_value: reference the innermost Option content question_mark: avoid incorrect suggestion when ref binding used fix from_over_into lint suggesting invalid code fix incorrect suggestions related to parentheses in needless_return fix unnecessary_safety_comment false positive on desugared assign Rust-Analyzer add icons to views analysis-stats: run Salsa's LRU at the end of analysis display varargs in completion detail do not error for actions with no data to resolve for loop to while let assist fix testing packages with multiple targets avoid recursively debug printing crates fix stale Building CrateGraph report observe unsafeness when generating manual impls of former derives preparation to Return Type Notation (RTN) port rust-analyzer to new salsa salsify the crate graph Rust Compiler Performance Triage A relatively busy week with a large amount of regressions in rollups which made investigations more tricky. Luckily overall the week was an improvement due to some medium sized improvements through improving target feature computation and a type systems internals fix. Triage done by @rylev. Revision range: 9fb94b32..493c38ba Summary: (instructions:u) mean range count Regressions (primary) 1.7% [0.2%, 3.0%] 18 Regressions (secondary) 0.8% [0.2%, 2.7%] 37 Improvements (primary) -1.0% [-10.3%, -0.2%] 157 Improvements (secondary) -1.7% [-8.8%, -0.2%] 158 All (primary) -0.8% [-10.3%, 3.0%] 175 5 Regressions, 5 Improvements, 3 Mixed; 5 of them in rollups 44 artifact comparisons made in total Full report here. Approved RFCs Changes to Rust follow the Rust RFC (request for comments) process. These are the RFCs that were approved for implementation this week: RFC for doc_cfg, doc_cfg_auto, doc_cfg_hide and doc_cfg_show features RFC: Demote i686-pc-windows-gnu to Tier 2 Final Comment Period Every week, the team announces the 'final comment period' for RFCs and key PRs which are reaching a decision. Express your opinions now. Tracking Issues & PRs Rust Deprecate the unstable concat_idents! Stabilize #![feature(precise_capturing_in_traits)] Rust RFCs [disposition: close] RFC: Add descriptive names to doctests Other Areas No Items entered Final Comment Period this week for Cargo, Language Team, Language Reference or Unsafe Code Guidelines. Let us know if you would like your PRs, Tracking Issues or RFCs to be tracked as a part of this list. New and Updated RFCs RFC: -crate-attr Upcoming Events Rusty Events between 2025-03-19 - 2025-04-16 Virtual 2025-03-19 | Virtual (Vancouver, BC, CA) | Vancouver Rust Bacon & Performance Benchmarking 2025-03-20 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - - Rust and embedded programming with Leon Vak (online in English) 2025-03-25 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2025-03-25 | Virtual (London, UK) | Women

in Rust Lunch & Learn: SKIing into Rust - crafting a simple interpreter 2025-03-27 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-01 | Virtual (Buffalo, NY, US) | Buffalo Rust Meetup Buffalo Rust User Group 2025-04-02 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2025-04-03 | Virtual (Nürnberg, DE) | Rust Nurnberg DE Rust Nürnberg online 2025-04-05 | Virtual | Ardan Labs Communicate with Channels in Rust 2025-04-05 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup 2025-04-08 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2025-04-10 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-15 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2025-04-16 | Virtual (Vancouver, BC, CA) | Vancouver Rust Rust Study/Hack/Hang-out Asia 2025-03-19 | Tel Aviv-Yafo, IL | Rust TLV In person Rust March 2025 at Jit in Tel Aviv 2025-03-28 | Kowloon Tong, HK | Rust Asia Rust Asia 2025 2025-04-05 | Bangalore/Bengaluru, IN | Rust Bangalore April 2025 Rustacean meetup Europe 2025-03-20 | Edinburgh, UK | Rust and Friends March Talks! (Two) 2025-03-20 | Prague, CZ | Rust Prague Rust/C++ Meetup Prague (March 2025) 2025-03-25 | Aarhus, DK | Rust Aarhus Hack Night - Robot Edition 2025-03-25 | Eindhoven, NL | RustNL Rust x Julia Meetup Eindhoven 2025-03-25 | London, UK | London Rust Project Group Deep Dive into Async Rust 2025-03-26 | Frankfurt, DE | Rust Rhein-Main "Beyond blazingly fast!" Performance Optimierungen in Rust 2025-03-26 | Manchester, UK | Rust Manchester Rust Manchester Talks March 2025-03-26 | Warsaw, PL | Rustikon Rustikon 2025-03-27 | Augsburg, DE | Rust Meetup Augsburg Rust Meetup #12: Testing in Rust 2025-03-29 | Stockholm, SE | Stockholm Rust Ferris' Fika Forum #10 2025-04-02 | Cambridge, UK | Cambridge Rust Meetup Monthly Rust Meetup 2025-04-02 | München, DE | Rust Munich Rust Munich 2025 / 1 - hybrid 2025-04-02 | Oxford, UK | Oxford Rust Meetup Group Oxford Rust and C++ social 2025-04-02 | Stockholm, SE | Stockholm Rust Rust Meetup @Funnel 2025-04-03 | Oslo, NO | Rust Oslo Rust Hack'n'Learn at Kampen Bistro 2025-04-08 | Olomouc, CZ | Rust Moravia 3. Rust Moravia Meetup (Real Embedded Rust) 2025-04-09 | Girona, ES | Rust Girona Rust Girona Hack & Learn 04 2025 2025-04-09 | Reading, UK | Reading Rust Workshop Reading Rust Meetup 2025-04-10 | Karlsruhe, DE | Rust Hack & Learn Karlsruhe Karlsruhe Rust Hack and Learn Meetup bei BlueYonder 2025-04-15 | Leipzig, DE | Rust - Modern Systems Programming in Leipzig Topic TBD 2025-04-15 | London, UK | Women in Rust WIR x WCC: Finding your voice in Tech North America 2025-03-20 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2025-03-20 | Nashville, TN, US | Music City Rust Developers Rust Game Development Series 3: Community Presentations 2025-03-20 | Redmond, WA, US | Seattle Rust User Group March, 2025 SRUG (Seattle Rust User Group) Meetup 2025-03-21 | México City, MX | Rust MX Rust y AWS Lambda. Preparando el camino para desplegar ML/AI 2025-03-26 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2025-03-26 | New York, NY, US | Rust NYC Rust NYC: I can't believe that's legal Rust with Michael Gattozzi (NEW LOCATION) 2025-03-27 | Atlanta, GA, US | Rust Atlanta We're going again! 2025-03-31 | Boulder, CO, US | Solid State Depot Boulder Rust: Bryan presents Rusted Hardware 2025-04-03 | Chicago, IL, US | Chicago Rust Meetup Rust Happy Hour 2025-04-03 | Montréal, QC, CA | Rust Montréal April Monthly Social 2025-04-03 | Saint Louis, MO, US | STL Rust icu4x - resource-constrained internationalization (i18n) 2025-04-10 | Portland, OR, US | PDXRust TetaNES: A Vaccination for Rust—No Needle, Just the Borrow Checker South America 2025-04-03 | Buenos Aires, AR | Rust en Español Abril - Lambdas y más! If you are running a Rust event please add it to the calendar to get it mentioned here. Please remember to add a link to the event too. Email the Rust Community Team for access. Jobs Please see the latest Who's Hiring thread on r/rust Quote of the Week Probably a terrible idea, but I enjoy throwing ideas at the wall, and seeing how sharp their broken fragments are. - Katt on the RFC #3762 discussion Thanks to Jacob Lifshay for the suggestion! Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell, llogiq, cdmistman, ericseppanen, extrawurst, U007D, joelmarcey, mariannegoldin, bennyvasquez, bdillo Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- [Niko Matsakis: Rust in 2025: Language interop and the extensible compiler](#) (2025/03/18 15:34)

For many years, C has effectively been the “lingua franca” of the computing world. It’s pretty hard to combine code from two different programming languages in the same process—unless one of them is C. The same could theoretically be true for Rust, but in practice there are a number of obstacles that make that harder than it needs to be. Building out silky smooth language interop should be a core goal of helping Rust to target foundational applications. I think the right way to do this is not by extending rustc with knowledge of other programming languages but rather by building on Rust’s core premise of being an extensible language. By investing in building out an “extensible compiler” we can allow crate authors to create a plethora of ergonomic, efficient bridges between Rust and other languages. We’ll know we’ve succeeded when... When it comes to interop... It is easy to create a Rust crate that can be invoked from other languages and across multiple environments (desktop, Android, iOS, etc). Rust tooling covers the full story from writing the code to publishing your library. It is easy¹ to carve out parts of an existing codebase and replace them with Rust. It is particularly easy to integrate Rust into C/C++ codebases. When it comes to extensibility... Rust is host to wide variety of extensions ranging from custom lints and diagnostics (“clippy as a regular library”) to integration and interop (ORMs, languages) to static analysis and automated reasoning[^][math].

Lang interop: the least common denominator use case

In my head, I divide language interop into two core use cases. The first is what I call Least Common Denominator (LCD), where people would like to write one piece of code and then use it in a wide variety of environments. This might mean authoring a core SDK that can be invoked from many languages but it also covers writing a codebase that can be used from both Kotlin (Android) and Swift (iOS) or having a single piece of code usable for everything from servers to embedded systems. It might also be creating WebAssembly components for use in browsers or on edge providers. What distinguishes the LCD use-case is two things. First, it is primarily unidirectional—calls mostly go from the other language to Rust. Second, you don’t have to handle all of Rust. You really want to expose an API that is “simple enough” that it can be expressed reasonably idiomatically from many other languages. Examples of libraries supporting this use case today are uniffi and diplomat. This problem is not new, it’s the same basic use case that WebAssembly components are targeting as well as old school things like COM and CORBA (in my view, though, each of those solutions is a bit too narrow for what we need). When you dig in, the requirements for LCD get a bit more complicated. You want to start with simple types, yes, but quickly get people asking for the ability to make the generated wrapper from a given language more idiomatic. And you want to focus on calls into Rust, but you also need to support callbacks. In fact, to really integrate with other systems, you need generic facilities for things like logs, metrics, and I/O that can be mapped in different ways. For example, in a mobile environment, you don’t necessarily want to use tokio to do an outgoing networking request. It is better to use the system libraries since they have special cases to account for the quirks of radio-based communication. To really crack the LCD problem, you also have to solve a few other problems too: It needs to be easy to package up Rust code and upload it into the appropriate package managers for other languages. Think of a tool like maturin, which lets you bundle up Rust binaries as Python packages. For some use cases, download size is a very important constraint. Optimizing for size right now is hard to start. What’s worse, your binary has to include code from the standard library, since we can’t expect to find it on the device—and even if we could, we couldn’t be sure it was ABI compatible with the one you built your code with.

Needed: the “serde” of language interop

Obviously, there’s enough here to keep us going for a long time. I think the place to start is building out something akin to the “serde” of language interop: the `serde` package itself just defines the core trait for serialization and a `derive`. All of the format-specific details are factored out into other crates defined by a variety of people. I’d like to see a universal set of conventions for defining the “generic API” that your Rust code follows and then a tool that extracts these conventions and hands them off to a backend to do the actual language specific work. It’s not essential, but I think this core dispatching tool should live in the `rust-lang` org. All the language-specific details, on the other hand, would live in `crates.io` as crates that can be

created by anyone. Lang interop: the “deep interop” use case The second use case is what I call the deep interop problem. For this use case, people want to be able to go deep in a particular language. Often this is because their Rust program needs to invoke APIs implemented in that other language, but it can also be that they want to stub out some part of that other program and replace it with Rust. One common example that requires deep interop is embedded developers looking to invoke gnarly C/C++ header files supplied by vendors. Deep interop also arises when you have an older codebase, such as the Rust for Linux project attempting to integrate Rust into their kernel or companies looking to integrate Rust into their existing codebases, most commonly C++ or Java. Some of the existing deep interop crates focus specifically on the use case of invoking APIs from the other language (e.g., `bindgen` and `duchess`) but most wind up supporting bidirectional interaction (e.g., `pyo3`, `[npapi-rs][[]`, and `neon`). One interesting example is `cxx`, which supports bidirectional Rust-C++ interop, but does so in a rather opinionated way, encouraging you to make use of a subset of C++’s features that can be readily mapped (in this way, it’s a bit of a hybrid of LCD and deep interop). Interop with all languages is important. C and C++ are just more so. I want to see smooth interop with all languages, but C and C++ are particularly important. This is because they have historically been the language of choice for foundational applications, and hence there is a lot of code that we need to integrate with. Integration with C today in Rust is, in my view, “ok” – most of what you need is there, but it’s not as nicely integrated into the compiler or as accessible as it should be. Integration with C++ is a huge problem. I’m happy to see the Foundation’s Rust-C++ Interoperability Initiative as well a projects like Google’s `crubit` and of course the venerable `cxx`. Needed: “the extensible compiler” The traditional way to enable seamless interop with another language is to “bake it in” i.e., Kotlin has very smooth support for invoking Java code and Swift/Zig can natively build C and C++. I would prefer for Rust to take a different path, one I call the extensible compiler. The idea is to enable interop via, effectively, supercharged procedural macros that can integrate with the compiler to supply type information, generate shims and glue code, and generally manage the details of making Rust “play nicely” with another language. In some sense, this is the same thing we do today. All the crates I mentioned above leverage procedural macros and custom derives to do their job. But procedural macros today are the “simplest thing that could possibly work”: tokens in, tokens out. Considering how simplistic they are, they’ve gotten us remarkably, but they also have distinct limitations. Error messages generated by the compiler are not expressed in terms of the macro input but rather the Rust code that gets generated, which can be really confusing; macros are not able to access type information or communicate information between macro invocations; macros cannot generate code on demand, as it is needed, which means that we spend time compiling code we might not need but also that we cannot integrate with monomorphization. And so forth. I think we should integrate procedural macros more deeply into the compiler.² I’d like macros that can inspect types, that can generate code in response to monomorphization, that can influence diagnostics³ and lints, and maybe even customize things like method dispatch rules. That will allow all people to author crates that provide awesome interop with all those languages, but it will also help people write crates for all kinds of other things. To get a sense for what I’m talking about, check out `F#’s` type providers and what they can do. The challenge here will be figuring out how to keep the stabilization surface area as small as possible. Whenever possible I would look for ways to have macros communicate by generating ordinary Rust code, perhaps with some small tweaks. Imagine macros that generate things like a “virtual function”, that has an ordinary Rust signature but where the body for a particular instance is constructed by a callback into the procedural macro during monomorphization. And what format should that body take? Ideally, it’d just be Rust code, so as to avoid introducing any new surface area. Not needed: the Rust Evangelism Task Force So, it turns out I’m a big fan of Rust. And, I ain’t gonna lie, when I see a prominent project pick some other language, at least in a scenario where Rust would’ve done equally well, it makes me sad. And yet I also know that if every project were written in Rust, that would be so sad. I mean, who would we steal good ideas from? I really

like the idea of focusing our attention on making Rust work well with other languages, not on convincing people Rust is better 4. The easier it is to add Rust to a project, the more people will try it - and if Rust is truly a better fit for them, they'll use it more and more. Conclusion: next steps This post pitched out a north star where a single Rust library can be easily used across many languages and environments; Rust code can easily call and be called by functions in other languages; this is all implemented atop a rich procedural macro mechanism that lets plugins inspect type information, generate code on demand, and so forth. How do we get there? I think there's some concrete next steps: Build out, adopt, or extend an easy system for producing "least common denominator" components that can be embedded in many contexts. Support the C++ interop initiatives at the Foundation and elsewhere. The wheels are turning: tmandry is the point-of-contact for project goal for that, and we recently held our first lang-team design meeting on the topic (this document is a great read, highly recommended!). Look for ways to extend proc macro capabilities and explore what it would take to invoke them from other phases of the compiler besides just the very beginning. An aside: I also think we should extend rustc to support compiling proc macros to web-assembly and use that by default. That would allow for strong sandboxing and deterministic execution and also easier caching to support faster build times. Well, as easy as it can be. ← Rust's incremental compilation system is pretty well suited to this vision. It works by executing an arbitrary function and then recording what bits of the program state that function looks at. The next time we run the compiler, we can see if those bits of state have changed to avoid re-running the function. The interesting thing is that this function could as well be part of a procedural macro, it doesn't have to be built-in to the compiler. ← Stuff like the diagnostics tool attribute namespace is super cool! More of this! ← I've always been fond of this article Rust vs Go, "Why they're better together". ←

- [The Rust Programming Language Blog: Announcing Rust 1.85.1](#) (2025/03/18 00:00)

The Rust team has published a new point release of Rust, 1.85.1. Rust is a programming language that is empowering everyone to build reliable and efficient software. If you have a previous version of Rust installed via rustup, getting Rust 1.85.1 is as easy as: rustup update stable If you don't have it already, you can get rustup from the appropriate page on our website. What's in 1.85.1 Fixed combined doctest compilation Due to a bug in the implementation, combined doctests did not work as intended in the stable 2024 Edition. Internal errors with feature stability caused rustdoc to automatically use its "unmerged" fallback method instead, like in previous editions. Those errors are now fixed in 1.85.1, realizing the performance improvement of combined doctest compilation as intended! See the backport issue for more details, including the risk analysis of making this behavioral change in a point release. Other fixes 1.85.1 also resolves a few regressions introduced in 1.85.0: Relax some target_feature checks when generating docs. Fix errors in std::fs::rename on Windows 1607. Downgrade bootstrap cc to fix custom targets. Skip submodule updates when building Rust from a source tarball. Contributors to 1.85.1 Many people came together to create Rust 1.85.1. We couldn't have done it without all of you. Thanks!

- [Spidermonkey Development Blog: SpiderMonkey Newsletter \(Firefox 135-137\)](#) (2025/03/17 20:00)

Hello everyone, Matthew here from the SpiderMonkey team. As the weather whipsaws from cold to hot to cold, I have elected to spend some time whipping together a too brief newsletter, which will almost certainly not capture the best of what we've done these last few months. Nevertheless, onwards! ☐☐Outreachy We hosted an Outreachy intern, Serah Nderi, for the most recent Outreachy cycle, with Dan as her mentor. Serah worked on implementing the Iterator.range proposal as well as a few other things. We were happy to host her, and grateful to her for joining. Read about her internship project here. ☐HYTRADBOI: Have You Tried Rubbing a Database On It HYTRADBOI is an interesting independent online only conference, which this year had a strong programming languages track. Iain from the SpiderMonkey team was able to produce a

stellar video talk called A quick ramp-up on ramping up quickly, where he helps the audience reinvent our baseline interpreter in 10 minutes. The talk is fun and short, so go forth and watch it! ☞☞ New features & In Progress Standards Work We have done a whole bunch of shipping work this cycle. By far the most important thing is that Temporal has now been shipped on Nightly. We must extend our enormous gratitude to André Bargull, who has been implementing this proposal for years, providing reams of feedback to champions, and making it possible for us to ship so early. We've also been working on improving error messages reported to developers, and have a list of "good first bugs" available for people interested in getting started contributing to SpiderMonkey or Firefox. In addition to Temporal, Dan has worked on shipping a number of our complete proposal implementations: Math.sumPrecise Intl.DurationFormat and Atomics.pause. ☞ Performance New contributor abdoatef.ab got a nice speedup (2.3x on a micro-benchmark!) by hinting our object allocator on the final size an object literal. Jon added a slots-and-elements allocator which should reduce contention on the system allocator which is used for many other things. Jan added code to recycle LifoAllocs for IonCompilations, which reduces the amount of contention on the memory allocator where possible. Jan continued work on register allocation tuning, continuing on from where we left it last with Jan's blog post. Jan's been doing some work with fuses to take advantage of knowing the state of the VM more. ☞ SpiderMonkey Platform Improvements Iain landed the infrastructure for off-thread baseline compilation and batched baseline compilation. The hope is that this will eventually lead to some performance improvements but it's disabled while it is tuned for now. We now share the parsed version of our self-hosted code from parent process to child process on Android, leading to a small improvement in child process startup time on Android. Ryan added JitDump support for Wasm compilation, which means that now it shows up beautifully in Samply.

- [Don Marti: privacy laws for slacker states](#) (2025/03/15 00:00)

It has come to my attention that there are still 15 or so states in the USA without privacy laws. This is understandable. We all have a lot of stuff to deal with. And of course there's the problem of privacy law compliance turning into a time-suck for small businesses. The more that the laws and regulations pile up, the harder to pick out everything you need to do from all those damn PDFs. And it's not just small companies. Honda just got around to dealing with some obvious differences between GDPR compliance and CCPA compliance that I pointed out back in 2020. And that's an old PDF and a big company. But the good news for slacker states is that doing the most work, cranking out the most lines of code, or the most pages of PDFs, or whatever, does not necessarily produce the best results. Given the amount of work that other states, and jurisdictions like the European Union, have already done on privacy, a slacker state can, right now, get not just the best privacy protection but also save a lot of time and grief for state employees and for business people in your state. You need two laws. And we know that people are going to print them out, so please keep them short. (Maybe do a printer ink right to refill law next year?) First, surveillance licenses for Big Tech. This gets you a few benefits. Focus on the riskiest companies with the most money and staff for compliance—don't put extra work on small local businesses. Save your state's attorney general and their staff a bunch of time. They're not Big Tech's support department. If a Big Tech company drops the ball on user support, just suspend their surveillance license until they clean up their act, like a problem bar and their liquor license. You can define surveillance really briefly in the law and make the big out-of-state companies do the work of describing their surveillance practices in their license application. That one is pretty easy to do as long as you focus purely on inbound data, the surveillance part, and don't touch anything that sounds like speech from the company to others. And you can push most of the work off onto Big Tech and a new surveillance licensing board. I'm sure every state has people who would be willing to get on one of those. Second, copy all the details from other states and countries. The other law would be focused on maximum privacy, minimum effort. The goal is to make a law that small business people can comply with, without even reading it, because they already had to do some privacy thing for somewhere else. Two parts. Any privacy feature offered in some

other jurisdiction must be offered here, too. A company only breaks the law if someone out-of-state gets a privacy feature that someone in-state doesn't. This law may be enforced by anyone except a state employee. (Borrow the Texas S.B. 8 legal hack, to protect yourself from Big Tech industry groups trying to block the law by starting an expensive case.) A small business that operates purely locally can just do their thing. But if they already have some your California privacy rights feature or whatever, they just turn it on for this state too. Easier compliance project for the companies, better privacy for the users, no enforcement effort for the state, it's a win-win-win. After all, state legislators don't get paid by the page, and we each only get one set of carpal tunnels. Related there ought to be a law (Some stuff you can do to look busy next year maybe?) advertising personalization: good for you? predictions for 2025 Bonus links Meta, Apparently, Really Wants Everyone To Read This Book (By Trying To Ban It) by Mike Masnick. Macmillan showed up just long enough to point out the blazingly obvious: they never signed any agreement with Meta and thus can't be bound by arbitration. The arbitrator, displaying basic common sense, had to admit they had no jurisdiction over Macmillan. Micah Lee writes, Not only is Substack right-wing brologarchy garbage, it's way more expensive than Ghost Substack takes a 10% cut of every transaction, while Ghost doesn't take any cut at all. Instead, Ghost charges based on the number of newsletter subscribers you have. AI Search Has A Citation Problem by Klaudia Jaźwińska and Aisvarya Chandrasekar. Chatbots were generally bad at declining to answer questions they couldn't answer accurately, offering incorrect or speculative answers instead. (related: fix Google Search) It's Official: the Cybertruck is More Explosive than the Ford Pinto Update: In case you were wondering, are these sample sizes statistically significant? The resident scientist over at Some Weekend Reading demonstrates: yes they are! Tesla Cybertruck vs Ford Pinto: Which is the Bigger Fire-Trap? (The fatality rate may also be related to the electric doors problem: Testimony Reveals Doors Would Not Open on Cybertruck That Caught Fire in Piedmont, Killing Three. It's possible that some of the people listed as victims would have survived if they had been able to exit.)

- [Don Marti: Links for 14 March 2025: autonomous drones in the news](#) (2025/03/14 00:00)

How Ukraine integrates machine vision in battlefield drones by Oleksandr Matviienko, Bohdan Miroshnychenko & Zoriana Semenovych. In November 2024, the government procured 3,000 FPV drones with machine vision and targeting technologies. Reports also suggested that the procurement would be expanded to 10,000 units. Preparing for the next European war by Azeem Azhar. One challenge will be the simple rate of innovation in the actual battlefield. Drone warfare in Ukraine has shown iteration cycles measuring weeks not years. So any systems procured today need to be future-proofed for those dynamics. Thread by Trent Telenko The logistical facts are that the FM-MAG machine gun, the 60 mm & 81mm mortars, LAWS, Javelins, any infantry crew served weapon you care to name are all going to be most to fully replaced with drones and drone operators, because of the logistical leverage drones represent on the battlefield. Long-range drone strikes weakening Russia's combat ability, senior Ukrainian commander says by Deborah Haynes. Some of the drones are remotely piloted, others work via autopilot. Russia's war has forced Ukraine to use technology and innovation to fight back against its far more powerful foe. It has accelerated the use of autonomous machines in an irreversible transformation of the warzone that everyone is watching and learning from. Brigadier Shchygol said: Right now, Ukraine's battlefield experience is essentially a manual for the world. Ukraine Drives Next Gen Robotic Warfare by Mick Ryan. Another more interesting trend has arisen which will force policy makers and military strategists to undertake an even more careful analysis of Ukraine war trends, and how these trends apply in other theatres, particularly the Pacific. This trend, robotic teaming, has emerged over the past year with the advent on drone-on-drone combat in the air and on the ground. In particular, several recent combat actions in Ukraine provide insights that need to be studied and translated for their employment in the massive ocean expanses, tens of thousands of kilometres of littoral, thousands of large and small islands and at least three continents that constitute the Pacific theatre. DEEP DIVE: Taiwan miltech aims to undermine Chinese

components by Tim Mak. Taiwan has learnt the central tech lesson from the war in Ukraine: the next global conflicts will heavily feature cheap, small drones—and in large numbers. So as an electronics and hardware component giant—especially relative to its size and diplomatic status—it is trying not only to develop a domestic industry, but also become an arsenal for the free world, building drones and devices for allied militaries worldwide. Why America fell behind in drones, and how to catch up again by Cat Orman and Jason Lu. Also Building Drones for Developers: A uniquely open architecture on the F-11 means that every part of the drone is truly built around the [NVIDIA]n Orin [GPU]. This enables sophisticated autonomy applications in which ML models are able to not only analyze data obtained in-flight, but actually use that analysis to inform flight actions in real time.

- [Mozilla Thunderbird: VIDEO: The Thunderbird Design System](#) (2025/03/13 19:07)

In this month's Community Office Hours, Laurel Terlesky, Design Manager, is talking about the new Thunderbird Design System. In her talk from FOSDEM, "Building a Cross-Platform, Scalable, Open-Source Design System," Laurel describes the Thunderbird design journey. If you are interested in how the desktop and mobile apps have gotten their new look, or in the open source design process (and how to take part), this talk is for you! Next month, we'll be chatting with Vineet Deo, a Software Engineer on the Desktop team who will walk us through the new Account Hub on the Desktop app. If you want a sneak peak at this new streamlined experience, you can find it in the Daily channel now and the Beta channel starting March 25. February Office Hours: The Thunderbird Design System As Thunderbird has grown over the past few years, so has its design needs. The most recent 115 and 128 releases, Supernova and Nebula, have introduced a more modern, streamlined look to the Thunderbird desktop application. Likewise, the Thunderbird for Android app has incorporated Material 3 in its development from the K-9 Mail app. When we begin working on the iOS app, we'll need to work with Apple's Human Interface Guidelines. Thus, Laurel and her team have built a design system that provides consistency across our existing and future products. This system's underlying principles also embrace user choice and privacy while emphasizing human collaboration and high design standards. Watch, Read, and Get Involved We're so grateful to Laurel for joining us! We hope this video helps explain more about how we design our Thunderbird products. Want to know more about this new Thunderbird design system? Want to find out how to contribute to the design process? Watch the video and check out our resources below! VIDEO (Also on Peertube): Thunderbird Design Resources: Open Source Design Thunderbird UX Mailing List Thunderbird/Mozilla Careers The post VIDEO: The Thunderbird Design System appeared first on The Thunderbird Blog.

- [Mozilla Security Blog: Enhancing CA Practices: Key Updates in Mozilla Root Store Policy, v3.0](#) (2025/03/12 16:14)

Mozilla remains committed to fostering a secure, agile, and transparent Web PKI ecosystem. The new Mozilla Root Store Policy (MRSP) v3.0, effective March 15, 2025, introduces critical updates to strengthen Certificate Authority (CA) practices and enhance compliance. A major focus of MRSP v3.0 is tackling the long-standing challenge of delayed certificate revocation—an issue that has historically weakened the security and reliability of TLS certificate management. The updated policy establishes clearer revocation expectations, improved incident reporting, subscriber education by CAs, revocation planning, and automated certificate issuance to ensure that certificate replacement and revocation can be handled promptly and at scale. Beyond improving revocation, MRSP v3.0 also introduces policies to move CA operators toward dedicated hierarchies for TLS and S/MIME certificates and to enhance CA private key security with improved lifecycle tracking. All of these updates raise the bar for CA operations, reinforcing security and trust across the broader internet ecosystem. Addressing Delayed Certificate Revocation One of the most persistent challenges in certificate management has been ensuring that TLS server certificates are revoked quickly when necessary. Many website operators struggle to replace certificates efficiently, while security advocates emphasize the need for rapid revocation and automated

certificate lifecycle management to reduce risks. To strike the right balance between security, stability, and operational feasibility, MRSP v3.0 introduces several key changes towards clearer and more comprehensive revocation expectations. No Exceptions to Revocation Requirements Previously, some CA operators expressed uncertainty about whether Mozilla could grant exceptions to revocation timelines in certain situations. MRSP v3.0 explicitly reiterates that Mozilla does not grant exceptions to the TLS Baseline Requirements for revocation. This will promote more consistent enforcement of revocation policy. Stronger Subscriber Communication and Contractual Clarity CA operators must proactively warn subscribers about the risks of relying on publicly trusted certificates in environments that cannot tolerate timely revocation. Additionally, Subscriber Agreements must explicitly require cooperation with revocation timelines, ensuring CA operators can act without unnecessary delays. Mass Revocation Preparedness Historically, large-scale certificate revocations have been challenging, leading to operational slowdowns, and ecosystem-wide risks when urgent action is required. To prevent revocation delays, MRSP v3.0 mandates mass revocation readiness to help ensure that CA operators proactively plan for such scenarios. CA operators will be required to develop, maintain, and test comprehensive plans to revoke large numbers of certificates quickly when necessary. And, to further strengthen mass revocation preparedness, MRSP v3.0 introduces a third-party assessment requirement. Assessors will verify that CA operators: Maintain well-documented, actionable plans for large-scale revocation, Demonstrate feasibility through regular testing, and Continuously improve their approach based on lessons learned. These measures ensure CA operators are fully prepared for high-impact security events. By strengthening mass revocation preparedness—and investing in CRLite—Mozilla is working to make certificate revocation a reliable security control. Enhancing Automation in Certificate Issuance and Renewal Automation plays a critical role in ensuring certificates can be replaced in a timely manner. To further encourage adoption of automation, MRSP v3.0 introduces new requirements for CA operators seeking root inclusion with the “websites” trust bit enabled, including: offering automation options for Domain Control Validation (DCV), certificate issuance, and renewal (demonstrated by a publicly accessible test website demonstrating automated certificate replacement at least every 30 days). Test website details must be disclosed in the Common CA Database (CCADB), adding transparency to this requirement. This push for more automation aligns with industry best practices, reducing reliance on manual processes, improving security, and minimizing mismanagement risks. Phasing Out Dual-Purpose (TLS and S/MIME) Root CAs A significant change introduced in MRSP v3.0 is the phase-out of dual-purpose root CAs—those with both the “websites” trust bit and the “email” trust bit enabled. The industry is already moving toward separating TLS and S/MIME hierarchies due to their distinct security needs. Keeping these uses separate at the root certificate level ensures more focused compliance, increases CA agility, reduces complexity, and enhances security. Going forward, Mozilla’s Root Store will require that new root CA certificates are dedicated to either TLS or S/MIME, and CA operators with existing dual-purpose roots will need to submit a transition plan to Mozilla by April 15, 2026, and complete a full migration to separate roots by December 31, 2028. This move enhances clarity and security by ensuring TLS and S/MIME compliance requirements remain distinct and enforceable. Strengthening CA Key Security with “Cradle-to-Grave” Monitoring Another major enhancement in MRSP v3.0 is the introduction of stricter key lifecycle monitoring to protect “parked” CA private keys. A “parked” key is a private key that the CA operator has generated for future use, but not yet used in a CA certificate. MRSP v3.0 adds mandatory reporting of parked key public hashes (corresponding to the parked CA private key) in annual audits. By enforcing transparency and accountability, Mozilla strengthens protections against undetected key compromise or misuse. Conclusion MRSP v3.0 represents a major step forward in ensuring stronger CA accountability with more reliable certificate revocation processes, better automation and operational resilience, and enhanced security for CA private keys. In all, these changes help modernize the Web PKI and ensure that CA operations will remain transparent, accountable, and secure. We encourage you to engage with the Mozilla community and to contribute to

these efforts and our shared mission of ensuring a secure and trustworthy online experience for all users. The post Enhancing CA Practices: Key Updates in Mozilla Root Store Policy, v3.0 appeared first on Mozilla Security Blog.

- [This Week In Rust: This Week in Rust 590](#) (2025/03/12 04:00)

Hello and welcome to another issue of This Week in Rust! Rust is a programming language empowering everyone to build reliable and efficient software. This is a weekly summary of its progress and community. Want something mentioned? Tag us at @ThisWeekInRust on X (formerly Twitter) or @ThisWeekinRust on mastodon.social, or send us a pull request. Want to get involved? We love contributions. This Week in Rust is openly developed on GitHub and archives can be viewed at this-week-in-rust.org. If you find any errors in this week's issue, please submit a PR. Want TWIR in your inbox? Subscribe here. Updates from Rust Community Official Announcing rustup 1.28.1 Inferred const generic arguments: Call for Testing! This Month in Our Test Infra: January and February 2025 Project/Tooling Updates Native Git support in Zed - Zed Blog tfmcp : A Rust-Implemented Tool to Operate Terraform from LLMs What's new in SeaORM 1.1 Observations/Thoughts Rust in 2025: Targeting foundational software A Happy Day for Rust Rust Learning Resources 2025 Taming A Voracious Rust Proxy Succinct data structures When is "this trait can be implemented" part of the trait's public API? When are Rust's const fns executed? Rust trait object layout The Art of Formatting Code [video] Rust is the New C [audio] Rust with Guillaume Gomez Rust Walkthroughs Writing into uninitialized buffers in Rust Translating bzip2 with c2rust Nine Pico PIO Wats with Rust: Raspberry Pi programmable IO pitfalls illustrated with a musical example (Part 1) Async Rust for Dummies How we built our 2025 Embedded World Demos [video] Ratatui - terminal user interfaces in Rust with Orhun Parmaksız - build ratatop in pair programming [video] Derive Macros: Or, How I Learned to Stop Worrying and Love the proc_macro2::TokenStream [video] Porting the guff plot device to Rust Miscellaneous Rust Communities/User Groups World Map Crate of the Week This week's crate is eval-macro, a crate that allows to evaluate macros at compile time, giving similar feel to Zig's comptime. Thanks to Aleksander Krauze for the suggestion! Please submit your suggestions and votes for next week! Calls for Testing An important step for RFC implementation is for people to experiment with the implementation and give feedback, especially before stabilization. If you are a feature implementer and would like your RFC to appear in this list, add a call-for-testing label to your RFC along with a comment providing testing instructions and/or guidance on which aspect(s) of the feature need testing. No calls for testing were issued this week by Rust, Rust language RFCs or Rustup. Let us know if you would like your feature to be tracked as a part of this list. Call for Participation; projects and speakers CFP - Projects Always wanted to contribute to open-source projects but did not know where to start? Every week we highlight some tasks from the Rust community for you to pick and get started! Some of these tasks may also have mentors available, visit the task page for more information. No Calls for participation were submitted this week. If you are a Rust project owner and are looking for contributors, please submit tasks here or through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! CFP - Events Are you a new or experienced speaker looking for a place to share something cool? This section highlights events that are being planned and are accepting submissions to join their event as a speaker. GOSIM Rust Spotlight - Nominate and support your favorite projects! | Closes 2025-03-15 at 7:59am UTC | Utrecht, NL | 2025-05-13 - 2025-05-17 If you are an event organizer hoping to expand the reach of your event, please submit a link to the website through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! Updates from the Rust Project 555 pull requests were merged in the last week Compiler ergonomic ref counting on long spans, trim the middle of them to make them fit in the terminal width split the Edges iterator perf: change TaskDeps to start preallocated with 128 capacity perf: speed up target feature computation Library stabilize [T]::split_off... methods stabilize box_uninit_write stabilize const_char_classify, const_sockaddr_setters stabilize const_vec_string_slice stabilize string_extend_from_within stabilize feature const_copy_from_slice override default Write methods for cursor-like types specialize OsString::push

and OsString as From for UTF-8 perf: improve the generic MIR in the default PartialOrd::le and friends count char width at most once in Formatter::pad fix char count in Display for ByteStr fix crash in BufReader::peek() Cargo cargo tree: Add --depth public behind -Zunstable-options cargo: add terminal integration via ANSI OSC 9;4 sequences cargo: don't use \$CARGO_BUILD_TARGET in cargo metadata cargo: add completions for add --path cargo: add completions for install --path cargo: respect --frozen everywhere --offline or --locked is accepted Rustdoc fix O(tests) stack usage in edition 2024 mergeable doctests search: increase strictness of typechecking rustdoc: add attribute-related tests for rustdoc JSON hide item that is not marked as doc(inline) and whose src is doc(hidden) Clippy clippy: arbitrary_source_item_ordering: Make alphabetic ordering in module item groups optional clippy: unnecessary_to_owned: don't call iter() on a temporary object clippy: add missing tests annotations for ui-internal clippy: don't trigger blocks_in_conditions when the condition contains a return clippy: don't trigger unnecessary_debug_formatting in tests clippy: fix manual_let_else missing binding mode clippy: better help for mixed_case_hex_literals clippy: improve needless_pass_by_value suggestion clippy: make struct_field_names check private fields of public structs clippy: refactor function after adding a new diagnostic item clippy: remove Known problems section for vec_box clippy: rename the MSRV alias MANUAL_DIV_CEIL to DIV_CEIL clippy: use size_of from the prelude instead of imported clippy: io_error_other: walk back to the root context to compute the span Rust-Analyzer rust-analyzer: fix(hir): VariantDef is impl HasSource rust-analyzer: add missing name-ref parents to syntactic highlighting rust-analyzer: add warning and debug information when cargo metadata fails rust-analyzer: adjust relevance scoring threshold to consistent with existing imple... rust-analyzer: add diagnostic for dangling dyn and impl rust-analyzer: warn the user when a rename will change the meaning of the program rust-analyzer: path macro hygiene rust-analyzer: syntax highlightingg punct filtering ignoring mods rust-analyzer: fix diagnostics being cleared right after being received rust-analyzer: normalize projections in evaluated const display and layout calculation rust-analyzer: prevent wrong invocations of needs_parens_in with non-ancestral "parent"s rust-analyzer: highlight unsafe operations as unsafe, not definitions rust-analyzer: improve keyword completion for 'let' and 'let mut' rust-analyzer: log build script error output in load_cargo::load_workspace_at rust-analyzer: make GenericParamsCollector::type_or_consts not unnecessarily pub(crate) rust-analyzer: make change annotations per text-edit rust-analyzer: move loaded project MSRV back to 1.78, show notification for the warning rust-analyzer: rank ADT constructors as constructors for completion scoring Rust Compiler Performance Triage This week we had to merge a lot of large rollups due to many problems with our CI infrastructure, which made analysis harder. Even though the aggregated stats look like there were a lot of regressions, it is skewed by two large regressions happening on an uncommon optimized incremental build and a documentation build of a single crate. The documentation regression is being tracked, and fixes to some other regressions are already in progress. Triage done by @kobzol. Revision range: daf59857..9fb94b32 Summary: (instructions:u) mean range count Regressions (primary) 1.2% [0.2%, 58.8%] 149 Regressions (secondary) 4.2% [0.2%, 165.8%] 127 Improvements (primary) -1.1% [-14.0%, -0.3%] 31 Improvements (secondary) -2.9% [-38.4%, -0.1%] 43 All (primary) 0.8% [-14.0%, 58.8%] 180 2 Regressions, 2 Improvements, 5 Mixed; 4 of them in rollups 37 artifact comparisons made in total Full report here. Approved RFCs Changes to Rust follow the Rust RFC (request for comments) process. These are the RFCs that were approved for implementation this week: RFC: Deprecate the per-build-target edition field in Cargo.toml Final Comment Period Every week, the team announces the 'final comment period' for RFCs and key PRs which are reaching a decision. Express your opinions now. Tracking Issues & PRs Rust Uplift clippy::invalid_null_ptr_usage lint Rust RFCs RFC for doc_cfg, doc_cfg_auto, doc_cfg_hide and doc_cfg_show features Other Areas No Items entered Final Comment Period this week for Cargo, Language Team, Language Reference or Unsafe Code Guidelines. Let us know if you would like your PRs, Tracking Issues or RFCs to be tracked as a part of this list. New and Updated RFCs RFC: const ergonomics for NonZero<T> Upcoming Events Rusty Events between 2025-03-12 -

2025-04-09 | Virtual 2025-03-13 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-03-18 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens -
- | - | crum: Complex Numbers and Complex Matrices in Rust with Frans Slabber 2025-03-18 | Virtual (Washington, DC, US) | Rust DC Mid-month
Rustful—Rust GameDev Basics with Raylib by Tony Bradley 2025-03-19 | Virtual (Vancouver, BC, CA) | Vancouver Rust Bacon & Performance
Benchmarking 2025-03-20 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - | - | Rust and embedded programming with Leon Vak (online in English)
2025-03-25 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2025-03-25 | Virtual (London, UK) | Women in Rust Lunch & Learn:
SKling into Rust - crafting a simple interpreter 2025-03-27 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-01 | Virtual (Buffalo,
NY, US) | Buffalo Rust Meetup Buffalo Rust User Group 2025-04-02 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing
2025-04-03 | Virtual (Nürnberg, DE) | Rust Nurnberg DE Rust Nürnberg online 2025-04-05 | Virtual | Ardan Labs Communicate with Channels in
Rust 2025-04-08 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday Asia 2025-03-15 | Beijing, CN | WebAssembly and Rust
Meetup (Wasm Empowering AI) KCD Beijing 2025 2025-03-19 | Tel Aviv-Yafo, IL | Rust TLV In person Rust March 2025 at Jit in Tel Aviv
2025-03-28 | Kowloon Tong, HK | Rust Asia Rust Asia 2025 2025-04-05 | Bangalore/Bengaluru, IN | Rust Bangalore April 2025 Rustacean meetup
Europe 2025-03-12 | Reading, UK | Reading Rust Workshop Reading Rust Meetup 2025-03-13 | Biel, CH | Rust Bern 2025 Rust Talks Bern #2 @
BFH Biel 2025-03-14 | Paris, FR | Rust in Paris Rust in Paris 2025 2025-03-18 | Basel, CH | Rust Basel Rust Meetup #10 @ MDPI Basel 2025-03-18 |
Leipzig, DE | Rust - Modern Systems Programming in Leipzig (1) Mago; (2) Iggy; (2) Rust binary sizes 2025-03-20 | Edinburgh, UK | Rust and
Friends March Talks! (Two) 2025-03-20 | Prague, CZ | Rust Prague Rust/C++ Meetup Prague (March 2025) 2025-03-25 | Aarhus, DK | Rust Aarhus
Hack Night - Robot Edition 2025-03-25 | Eindhoven, NL | RustNL Rust x Julia Meetup Eindhoven 2025-03-25 | London, UK | London Rust Project
Group Deep Dive into Async Rust 2025-03-26 | Frankfurt, DE | Rust Rhein-Main "Beyond blazingly fast!" Performance Optimierungen in Rust
2025-03-26 | Manchester, UK | Rust Manchester Rust Manchester Talks March 2025-03-26 | Warsaw, PL | Rustikon Rustikon 2025-03-27 |
Augsburg, DE | Rust Meetup Augsburg Rust Meetup #12: Testing in Rust 2025-04-02 | München, DE | Rust Munich Rust Munich 2025 / 1 - hybrid
2025-04-02 | Oxford, UK | Oxford Rust Meetup Group Oxford Rust and C++ social 2025-04-02 | Stockholm, SE | Stockholm Rust Rust Meetup
@Funnel 2025-04-03 | Oslo, NO | Rust Oslo Rust Hack'n'Learn at Kampen Bistro 2025-04-08 | Olomouc, CZ | Rust Moravia 3. Rust Moravia Meetup
(Real Embedded Rust) 2025-04-09 | Reading, UK | Reading Rust Workshop Reading Rust Meetup North America 2025-03-13 | Chicago, IL, US |
Chicago Rust Meetup Rust Happy Hour 2025-03-13 | Portland, OR, US | PDXRust PDXRust Meetup: Finding One Way Out 2025-03-18 | San
Francisco, CA, US | San Francisco Rust Study Group Rust Hacking in Person 2025-03-18 | Spokane, WA, US | Spokane Rust Monthly Meetup: Intro
to Rust and Python; Using Rustup, Cargo, and Rust! 2025-03-20 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO
2025-03-20 | Nashville, TN, US | Music City Rust Developers Rust Game Development Series 3: Community Presentations 2025-03-20 | Redmond,
WA, US | Seattle Rust User Group March, 2025 SRUG (Seattle Rust User Group) Meetup 2025-03-21 | México City, MX | Rust MX Rust y AWS
Lambda. Preparando el camino para desplegar ML/AI 2025-03-26 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2025-03-27 | Atlanta, GA,
US | Rust Atlanta We're going again! 2025-03-31 | Boulder, CO, US | Solid State Depot Boulder Rust: Bryan presents Rusted Hardware 2025-04-03
| Montréal, QC, CA | Rust Montréal April Monthly Social 2025-04-03 | Saint Louis, MO, US | STL Rust icu4x - resource-constrained
internationalization (i18n) South America 2025-03-15 | São Paulo, BR | Rust São Paulo Meetup Encontro do Rust-SP na CloudWalk If you are
running a Rust event please add it to the calendar to get it mentioned here. Please remember to add a link to the event too. Email the Rust
Community Team for access. Jobs Please see the latest Who's Hiring thread on r/rust Quote of the Week Treat anything starting with cargo as if it
is cargo run. This applies even to commands that do not build anything, such as cargo clean, and third-party plugins, such as cargo audit. -

Sergey "Shnatsel" Davidoff on /r/rust Thanks to llogiq for the suggestion! Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell, llogiq, cdmistman, ericseppanen, extrawurst, U007D, joelmarcey, mariannegoldin, bennyvasquez, bdillo Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- [The Mozilla Blog: Misinformation in the age of AI: It's in the details \(like extra fingers\)](#) (2025/03/11 18:28)

As you scroll through social media, the posts blend together: a heroic cat, a shocking statistic, a breaking news clip. But in a world where AI blurs the line between fact and fiction, how do you tell what's real from what's misinformation? The short answer: Most of the time, you can't. Generative AI tools have made it faster, cheaper, and easier to create convincing fakes, flooding our feeds with AI-generated content. But here's what you can do - learn to spot the signs of misinformation. What should I think about when trying to detect AI? Just looking out for obvious AI will mean missing a lot of it. Retrain your brain to assess social media on a framework similar to the ones used by AI-misinformation researchers. Consider who's behind the post, whether the content makes sense, how it's written, the emotions it evokes and any signs of manipulation.

User Who is posting this? Is it a reliable source? Is this account tied to a real-world institution that I trust? What is the username length? Is it a long set of random numbers and letters? Is it a verified account? Does it only have a handful of followers (who also look random or not real)?

Content Does the framing make sense? What is this content about? Is it especially vague or seems so outrageous that it couldn't be true? Does this contradict what you already know about the topic? Are there platform flags that the content could be potentially misleading, or a comment section full of claims that it's false? Are there AI badges or indicator hashtags such as #AI, #satire, #skit, #spoofer or #AD? **Style** How is it written? Is there poor or wooden-sounding grammar? Is it flowery? Is there unnatural repetition, or has the user posted the same thing several times? Does it repeat often-used AI words such as "elevate," "captivate," "tapestry" or "delve"? Does it use known AI phrases such as "provided valuable insights," an "indelible mark," or "a rich tapestry"? (Of course, these words and phrases don't definitively mean that the content is AI-generated misinformation; they're just reasons to take a closer look.)

Emotion Is this an especially emotion-laded post? Is the level of emotion appropriate for the situation? Does the post appear to "weaponize" emotion or tell readers how to feel about the content, such as by using more anger and swear words? (Keep in mind that bots on social media can and do use profanity).

Manipulation What might someone have to gain by touching on your emotions in this way? What's the worst-case scenario if this turns out to not be true? What might a user (using AI) be hoping you don't look up? How do I spot AI-generated images or videos online? Gone are the days where every AI image looked like a wacky Pixar knockoff, but it's still worth checking for these known cues: Hands and teeth with too many fingers, too many hands, too many teeth or impossibly long arms Hyper-realistic images or those that look like paintings Texture issues in the background, midground and around the corners and edges of an image Unnatural smoothness, especially on faces, clothes or hair Shadows and light coming from the wrong place, or with only certain elements casting shadows Abrupt transitions, either in an image or a video

How do I verify if something is AI or not? Tools like TrueMedia.org scan social posts for false faces, voices, and information, while Mozilla's Deepfake Detector analyzes text using multiple detection engines and generates a confidence score indicating the likelihood that the content is AI-generated. But while AI detection accuracy is improving, it isn't perfect. It always helps to try to verify the information itself — search for it along with "fact check" and look for trusted sources. For images, try a reverse image search using Google Image Search or Bing Image Match.

What can misinformation look like on TikTok? Every social media site has its own AI landscape. Fake news, images and news clips targeting young voters and consumers are circulated particularly widely on TikTok due to its young user base. "Content farms" spin out inaccurate or misleading videos, often multiple a day in the distinctive TikTok style of on-screen text read by an AI voice. When scrolling on TikTok, be skeptical — or at least get a second opinion — on any informational

videos that aren't read by real people or only consist of captions to the AI voice (reputable news sites usually show who's talking to build trust). Profiles that look like news sites but that have no comments or likes (especially for celebrity news) are a red flag — as well as canned phrases like “creepy response” or “finally broke their silence” meant to drive clicks. What can misinformation look like on X? Though many AI-generated posts on X are largely innocuous text-based posts, videos in particular are more likely to be political deepfakes. And while the crowdsourced moderation feature, “Community Notes,” allows users to annotate posts with context or warnings, it replaced a more robust monitoring operation that means it's more likely users will encounter bots. Stay wary of accounts that only spam replies, or situations where multiple accounts are commenting similar things under a post. If a user only posts replies, especially to inflammatory content, it's a red flag that it's a bot searching for certain keywords. Also, user verification on X is the least trustworthy of the major social media sites as users can pay for “verified” status (in one Cambridge study, half of synthetic profiles studied had verified status). What can misinformation look like on Facebook? It's especially difficult to silo yourself from AI-generated content on Facebook, even if you're only interested in posts from family and friends. Over the past three years, there has been a “significant increase” in the rate of users seeing posts they held no “friend” connection to, thanks to the algorithm that surfaces high-engagement posts in users' feeds. Being disciplined about clicking “not interested” under the three dots on each post can help stem the flow, as well as staying skeptical of images and being wary of link-outs to “news” sites. Verify posts (even those that appear to be from a harmless, real person) about any news events independently. Misleading posts on Facebook are also especially focused on trying to get users off Facebook — directing them off the platform to content farms, fake stores and other scam sites. Stay alert and think critically online Humans often overestimate how good they are at detecting AI — nice art is sometimes AI-generated, and terrible grammar is sometimes very human. It's not easy to navigate a landscape designed to trick you, but your best call is to improve how you critically consume all information. Stay curious. After all, AI gets better every passing day — right down to drawing those tricky hands. Sarah Skinner is a senior editor at a NYC tech startup. She holds a degree from Cornell University on AI and empathy, and has previously worked for McKinsey & Company and the Associated Press. The post Misinformation in the age of AI: It's in the details (like extra fingers) appeared first on The Mozilla Blog.

- [Firefox Nightly: Turn Tabs To Their Side - These Weeks in Firefox: Issue 177](#) (2025/03/11 15:32)

Highlights Firefox 135 goes to release today (Feb 4th)! Lots of great changes in this release! We've landed Bug 1936379 - Add Sidebar and Vertical Tabs checkboxes in about:preferences and remove from Nightly Labs and the new sidebar and vertical tabs are riding the trains as options in the preferences UI (was in Labs) The Manifest V3 userScripts API has been enabled by default on Desktop builds and riding the Firefox 136 release train Documentation is forthcoming. In the meantime, this is a pull-request that shows an example of how the userScripts API can be used. A new browser.commands.openShortcutsSettings WebExtension API method allows extensions to direct users to the keyboard shortcuts management UI (in about:addons). Thanks to Matt Mower for this contribution! Alexandre Poirot [:ochameau] Added some crash UI to make it easy for users to report an issue on Bugzilla (#1941886) screenshot Friends of the Firefox team Resolved bugs (excluding employees) New contributors (□ = first patch) Christoph Moench-Tegeder extended BSD about:processes for FreeBSD □ Meike [:mei] added pt to size unit options in FontSize component Matt Mower added openShortcutSettings() to browser.commands extension API Project Updates Add-ons / Web Extensions Addon Manager & about:addons Fixed a regression related to using the mouse to select text inside an extension options page embedded in about:addons - Bug 1939206 Fixed a leak triggered by the private browsing checkbox included in the install dialog (caught by shutdown leaks failure hit by one of our tests) - Bug 1943031 Starting from Firefox 136, users will be able to undo the installation of a new theme directly from the notification popup - Bug 1931402 Thanks to Becca King for contributing this enhancement □ WebExtensions Framework A new

background.preferred_environment manifest property has been introduced in Firefox 136 in order to let cross-browser Manifest V3 extensions use a background service worker on Chrome and a background event page on Firefox with the same manifest file - Bug 1930334 DevTools Meike added the pt unit in the Fonts tab (#1940009) Julien Wajsberg added link to MDN for Cross-Origin-* headers in Netmonitor (#1943610) Tooru Fujisawa [:arai] fixed an issue where subsequent cached requests for scripts weren't displayed in Netmonitor (#1945492) Alexandre Piroit [:ochameau] fixed a performance regression that was making console evaluation slower than it should have been (#1933343) Alexandre Piroit [:ochameau] made JavaScript errors and warnings logged through MOZ_LOG (#1925003) Julian Descottes [:jdescottes] improved Netmonitor performance, specially when it contains a lot of requests (#1942149), but also when displaying large big data URIs (#1943284) Nicolas Chevobbe [:nchevobbe] made the Netmonitor "Raw" Response checkbox to persist (#1555647) Hubert Boma Manilla (:bomsy) fixed a hang when opening WASM files in the Debugger (#1935157), and thanks to that was able to enable codemirror 6 on release (#1942702) WebDriver BiDi Liam DeBeasi improved the browsingContext.contextDestroyed event to serialize the whole browsing context tree in the event's payload. Krzysztof Jan Modras (:chrmod) moved Marionette's Addon module to a shared module for the Remote Agent. Henrik updated crash reports from Marionette and Remote Agent to be annotated accordingly. Henrik also updated our vendored puppeteer version to v24.0.0 (note that it's the first version of puppeteer to remove support for Firefox over CDP) Sasha updated session.subscribe command to return subscription ID which can be used now in session.unsubscribe to remove the subscription. This will give clients more flexibility to add and remove event listeners during tests. Henrik updated Marionette and WebDriver BiDi to avoid a HTTPS-First upgrade when performing a navigation. Sasha improved the "script.addPreloadScript" command to support "userContexts" argument to allow clients setting preload scripts to a list of user contexts (also known as Firefox containers). Henrik added the "-allow-nonlocal-connections" argument to the Marionette mach command to allow accessing remote locations. Lint, Docs and Workflow Arai has removed the old support code for JSMS. This includes the ESLint support for the various import functions. Alex has merged the python linter requirements files, which should speed up linter setup a little bit. Ahal has stopped the "problem with lint setup, skipping android-api-lint, android-checkstyle, android-findbugs, android-lint, android-test" messages on Desktop builds. jdescottes improved some of our custom rules so that they do not throw when there's missing arguments. This makes the in-editor experience better. Information Management Various fixes to sidebar/toolbar interactions, some uplifts still to do. Changes to the sidebar settings: Bug 1939917 - Update Sidebar visibility settings and settings order New Tab Page mconley moved the vast majority of the New Tab code from browser/components/newtab to browser/extensions/newtab Punam has moved and decoupled a bunch of modules out from the New Tab codebase into shared module directories in browser and toolkit, since other components depend on them We're in the midst of a refresh of the stylings for top sites! This isn't ready to ship yet, but it can still be previewed by setting browser.newtabpage.activity-stream.newtabShortcuts.refresh to true in Nightly. Performance Some of the New Tab refactoring and movements have resulted in some memory-wins on our AWSY ("are we slim yet") memory test as well as startup paint times Performance Tools (aka Firefox Profiler) Profiles now contain markers for all Firefox data collection (both legacy telemetry and Glean) Example profile Search and Navigation Address bar Scotch Bonnet Project Improved feature callout for the Unified Search Button, enabled Scotch Bonnet by default in Beta, and re-enabled scotch bonnet in performance tests. (Bug 1937666, Bug 1942357, Bug 1923381) Removed redundant default search engine favicon in the Unified Search Button. (Bug 1908929) Suggest AMP & Dismissal Fixes: Hooked up AMP-suggestion-matching strategy and fixed dismissals for Merino suggestions when Suggest features are disabled. (Bug 1942222, Bug 1942435) Address bar Set deduplication threshold to zero, fixed Ctrl-Alt-b / Ctrl-Alt-f on Mac to properly move the cursor. (Bug 1943946, Bug 1481157) Added support for exponents for calculator and updated UrlbarUtils icon protocol list.

(Bug 1942623, Bug 1943272) Search Search improvements with Nimbus: Refactored the Search Nimbus set-up to split Search experiments between two nimbus feature sections - one for search feature related experiments, the other one for search configuration related experiments. (Bug 1830056, Bug 1883685, Bug 1942099, Bug 1942658) Places Added support for exporting interactions via CSV and resolved process crash caused by missing chrome/resource URLs. (Bug 1940782, Bug 1945152)

- [Mozilla Thunderbird: Thunderbird Monthly Development Digest - February 2025](#) (2025/03/11 14:32)

Hello again Thunderbird Community! Despite the winter seeming to last forever and the world being in a state of flux, the Thunderbird team has been hard at work both in development and planning strategic projects. Here's the latest from the team dedicated to making Thunderbird better each day: Monthly Releases are here! The concept of a stable monthly release channel has been in discussion for many years and I'm happy to share that we recently changed the default download on Thunderbird.net to point at our most feature-rich and up-to-date stable version. A lot of work went into this release channel, but for good reason - it brings the very latest in performance and UX improvements to users with a frequent cadence of updates. Meaning that you don't have to wait a year to benefit from features that have been tested and already spent time on our more experimental Daily and Beta release channels. Some examples of features that you'll find on the monthly release channel (but not on ESR) are: Linux System Tray Dark reader Support Folder compaction improvements Hundreds of UI enhancements ICS Import Calendar printing improvements Appearance settings UI Many, many more Download it over the top of your ESR installation and get the benefits today! Developing Standards As privacy and security legislation evolves, the Thunderbird team often finds itself in the heart of discussions that have the potential to define industry solutions to emerging problems. In addition to the previously-mentioned research underway to develop post-quantum encryption support, we're also currently considering solutions to EU laws (EU NIS2) that require multi-factor authentication be in place for critical digital infrastructure and services. We're committed to solving these issues in a way that gives users and system administrators other options besides Google & Microsoft, and we'll be sharing our thoughts on the matter soon, with the resulting decisions documented in our new ADR process. For now, you can follow a healthy and colourful discussion on the topic of OAuth2 Dynamic Client Registration here. Calendar UI Rebuild is underway The long awaited UI/UX rebuild of the calendar has begun, with our first step being a new event dialog that we're hoping to get into the hands of users on Daily via a preference switch. Turning the pref on will allow the existing calendar interface to launch the new dialog once complete. The following pieces of work have already landed: Dialog container Generic row container Calendar row Close button Generic subview Title Keep track of feature delivery via the [meta] bug Exchange Web Services support in Rust A big focus for February has been to grow our team so we've been busy interviewing and evaluating the tremendously talented individuals who have stepped forward to show interest in joining the team. In the remaining time, the team has managed to deliver another set of features and is heading toward a release on Daily that will result in most email features being made available for testing. Here's what landed and started in February: Display refactor Basic testing framework Sync folder - delete Sync folder read/unread Integration testing Complete composition support (reply/forward) Keep track of feature delivery here. Account Hub Since my last update, tasks related to density and font awareness, the exchange add-on and keyboard navigation were completed, with the details of each step available to view in our Meta bug & progress tracking. Watch out for this feature being rolled out as the default experience for the Daily build this week and on beta after the next merge on March 25th! Global Message Database The New Zealand team are in the middle of a work week to shout at the code together, have a laugh and console each other plan out work for the next several weeks. Their focus has been a sprint to prototype the integration of the new database with existing interfaces with a positive outcome meaning we're a little closer to producing a work breakdown that paints a more accurate picture of what lies ahead. Onward! In-App Notifications Phase 3 of the project is

underway to finalize our uplift stack and add in last-minute features! It is expected that our ESR version will have this new feature enabled for a small percentage of users at some point in April. If you use the ESR release, watch out for an introductory notification! Meta Bug & progress tracking. New Features Landing Soon Several requested features and fixes have reached our Daily users and include... Final folder compaction fixes relating to database writes! The RSS scrolling fix took a while but has now landed Overflow overrides causing scrolling problems are now a thing of the past and many more which we listed in release notes for beta. As usual, if you want to see things as they land, and help us squash some early bugs, you can always check the pushlog and try running daily, which would be immensely helpful for catching things early. If you're interested in joining the technical discussion around Thunderbird development, consider joining one or several of our mailing list groups here. — Toby Pilling Senior Manager, Desktop Engineering The post Thunderbird Monthly Development Digest – February 2025 appeared first on The Thunderbird Blog.

- [Firefox Developer Experience: Firefox WebDriver Newsletter 136](#) (2025/03/11 13:12)

WebDriver is a remote control interface that enables introspection and control of user agents. As such it can help developers to verify that their websites are working and performing well with all major browsers. The protocol is standardized by the W3C and consists of two separate specifications: WebDriver classic (HTTP) and the new WebDriver BiDi (Bi-Directional). This newsletter gives an overview of the work we've done as part of the Firefox 136 release cycle. Contributions Firefox is an open source project, and we are always happy to receive external code contributions to our WebDriver implementation. We want to give special thanks to everyone who filed issues, bugs and submitted patches. In Firefox 136, several contributors managed to land fixes and improvements in our codebase: Liam (ldebeasi) improved the `browsingContext.contextDestroyed` event in WebDriver BiDi. Krzysztof Jan Modras (chrmod) refactored an internal module for handling addons/webextensions. WebDriver code is written in JavaScript, Python, and Rust so any web developer can contribute! Read how to setup the work environment and check the list of mentored issues for Marionette, or the list of mentored JavaScript bugs for WebDriver BiDi. Join our chatroom if you need any help to get started! General Bug Fixes Firefox now handles WebSocket port conflicts for the RemoteAgent more efficiently. If the port specified via the `--remote-debugging-port` command-line argument cannot be acquired within 5 seconds, such as when another Firefox process is already using it, Firefox will now shut down instead of hanging. Navigations using the HTTP scheme, triggered by the `WebDriver:Navigate` command in Marionette or `browsingContext.navigate` in WebDriver BiDi, will no longer be automatically upgraded to HTTPS. These requests will now remain on HTTP, as originally intended. WebDriver BiDi Updated: `session.subscribe` now returns a subscription ID The return value of the `session.subscribe` command has been updated to include a `session.Subscription` which is a unique ID identifying the event subscription. Example: adding a global subscription for network events: `-> { "method": "session.subscribe", "params": { "events": ["network"] }, "id": 1 } <- { "type": "success", "id": 1, "result": { "subscription": "9a29c77b-5e7b-43f1-bc6f-20b5228bf207" } }` New: subscriptions parameter for `session.unsubscribe` The subscription IDs can be used with the new subscriptions parameters of `session.unsubscribe` to remove specific event subscriptions. Previously, it was necessary to unsubscribe using the same attributes provided to `session.subscribe`. And if there were several calls to `session.unsubscribe` performed from different areas of the test, there could be unexpected side-effects. With the new subscriptions parameter, only the subscriptions matching the provided subscription IDs will be removed. The parameter must be an array of subscription IDs. If any ID doesn't match a known (and not yet removed) subscription ID, an `InvalidArgumentError` is raised. When subscriptions is provided, the events or contexts parameters should not be present, otherwise an `InvalidArgumentError` will be raised. Example: adding a second subscription for a specific context: `-> { "method": "session.subscribe", "params": { "events": ["browsingContext.contextCreated"], "contexts": ["304a76e3-`

```
e177-4355-b598-9f5f02808556"} }, "id": 2 } <- { "type": "success", "id": 2, "result": { "subscription": "e1a54927-dc48-4301-b219-812ceca13496" } } Now removing both subscriptions at once: -> { "method": "session.unsubscribe", "params": { "subscriptions": ["9a29c77b-5e7b-43f1-bc6f-20b5228bf207", "e1a54927-dc48-4301-b219-812ceca13496"] }, "id": 3 } <- { "type": "success", "id": 3, "result": { } } Using the subscriptions parameter allows to remove global and context-specific subscriptions simultaneously, without unexpected side effects. Deprecated: contexts parameter for session.unsubscribe It remains possible to globally unsubscribe by using only the events parameter. However the additional contexts parameter is now deprecated and will be removed in a future version. To remove context-specific subscriptions, please use the subscriptions parameter instead. New: userContexts parameter for script.addPreloadScript The script.addPreloadScript command now supports a userContexts parameter, allowing clients to specify in which user contexts (containers) a script should be automatically loaded — including new browsing contexts created within those user contexts. Note: If userContexts is provided, the contexts parameter must not be included. Using both at the same time will raise an InvalidArgumentError. Example: adding a preload script for a specific user context: -> { "method": "script.addPreloadScript", "params": { "arguments": [], "functionDeclaration": "(arg) => { console.log('Context created in test user context'); }", "userContexts": ["528545b6-89d6-486e-a85c-517212654c13"] }, "id": 4 } <- { "type": "success", "id": 4, "result": { "script": "168cff9c-9de0-44e7-af95-0c478ad2596f" } } Updated: browsingContext.contextDestroyed event returns full tree Thanks to Liam's contribution, the browsingContext.contextDestroyed event now returns the full serialized tree of destroyed contexts — including all its child contexts. Example: closing a tab with two iframes: { "type": "event", "method": "browsingContext.contextDestroyed", "params": { "children": [ { "children": [], "context": "30064771075", "originalOpener": null, "url": "https://example.bidi.com/frame1", "userContext": "1c88a947-3297-4733-9cd7-c16a863a602d" }, { "children": [], "context": "30064771076", "originalOpener": null, "url": "about:blank", "userContext": "1c88a947-3297-4733-9cd7-c16a863a602d" } ], "context": "ab8c33a7-6bc8-45cc-a068-7cdcef518577", "originalOpener": null, "url": "https://example.bidi.com/", "userContext": "1c88a947-3297-4733-9cd7-c16a863a602d", "parent": null } }
```

- [The Mozilla Blog: Mozilla's response to proposed remedies in U.S. v. Google](#) (2025/03/11 12:52)

Last week the Department of Justice and some state attorneys general filed revised proposed remedies in the U.S. v. Google LLC search case. If the proposed remedies barring all search payments to browser developers are adopted by the court, these misguided plans would be a direct hit to small and independent browsers—the very forces that keep the web open, innovative and free. This case was meant to promote search competition, yet somehow the outcome threatens to crush browser competition, making it even harder for challengers to stand up to dominant players like Google, Apple and Microsoft. “These proposed remedies prohibiting search payments to small and independent browsers miss the bigger picture—and the people who will suffer most are everyday internet users,” said Mark Surman, President of Mozilla. “Independent browsers like Firefox are on the frontlines of protecting consumer privacy, driving browser innovation, and giving people real choice on how they experience the web. But instead of promoting a fair fight, the DOJ’s remedies would tilt the playing field further into the hands of a few dominant players, diminishing consumer choice and weakening the broader internet ecosystem.” The DOJ’s proposal hurts, not helps, browser competition. Mozilla agrees that we need to improve search competition, but the DOJ’s proposed remedies unnecessarily risk harming browser competition instead. Here’s why: The DOJ wants to ban all search agreements between Google and browsers, even independent browsers that make up a smaller part of the market. Dominant players that own browsers, like Apple, don’t rely on search deals as they have significant revenue streams from other sources, like hardware, operating systems and app stores. Meanwhile, independent browsers like Firefox fund the development of their browsers mainly through search revenue—they require this revenue to survive. Search revenue underpins a large part of our work, keeping

Firefox competitive and ensuring that web users have privacy-first alternatives. Punishing independent browsers will not solve the problem. Judge Mehta found that independent browsers account for just 1.15% of U.S. search queries. This means that cutting off our access to search deals won't fix the issue of search dominance—not by a landslide. Instead, it hurts browser competition. “The big unintended consequence here is the handing of power from one dominant player to another. So, from Google Search to Microsoft, or Bing for example—while shutting out the smaller, independent challengers that actually drive browser innovation and offer web users privacy and choice,” Surman added. The last unicorn—the web can't afford to lose Mozilla's browser engine. Another thing missing from this conversation is something pretty important—browser engine competition. You see, browser engines power the web. They are central to a browser's speed, privacy and security functionality, and the browser's ability to innovate and do things differently. But they're very complex and require massive resources and a deep technical expertise to maintain—so much so, that right now only three major browser engines remain: Google's Chromium, Apple's Webkit (this engine is really only supported on Apple devices, and isn't considered “cross-platform”), and then there's Mozilla's Gecko (which happens to be the only true cross-platform alternative to Chromium). The DOJ's proposal to bar search payments to independent browser developers would put Mozilla's ability to develop and maintain Gecko at risk. If Mozilla is unable to sustain our browser engine, it would severely impact browser engine competition and mean the death of the open web as we know it—essentially, creating a web where dominant players like Google and Apple, have even more control, not less. “This isn't just about Firefox,” Surman explained. “If we lose our ability to maintain Gecko, it's game over for an open, independent web. Look, Microsoft—a \$3 trillion company—already gave up its browser engine in 2019 and Opera gave up theirs in 2013. If Mozilla is forced out, Google's Chromium becomes the only cross-platform browser engine left.” Mozilla's role in an open web is BIGGER than our market share. Nevermind our market share, Mozilla has played an outsized role in keeping the web open, private and advocating for choice. Firefox still serves 27 million monthly active users (MAU) in the U.S. and nearly 205 million MAU globally, but our real impact comes from making the internet better by: Shaping the future of web standards—maintaining our own browser engine, Gecko, gives us a voice in defining how the web works and making decisions that are in support of people, not the bottom-line. Ensuring interoperability—we fight for a web accessible to all—where anyone can create, access, and share content seamlessly, regardless of the devices or web services they use—not locked into a few ecosystems. Proving that privacy-respecting technology is possible—we build critical web technologies with security, privacy and user agency at the core. “This isn't something we do because it's profitable or easy,” said Surman. “We do it because it matters. The DOJ's proposal doesn't just miss the mark, it risks handing even more power to dominant industry players like Google or Apple, not less.” Mozilla calls on regulators and policymakers to recognize the vital role of independent browsers and take action to nurture competition, innovation, and protect the public interest in the evolving digital landscape. Mozilla is committed to ensuring a fair and competitive internet ecosystem, one where independent browsers can compete on a level playing field and consumers have real choice. The future of competition, innovation and the open internet depends on us. The post Mozilla's response to proposed remedies in U.S. v. Google appeared first on The Mozilla Blog.

- [Mozilla Addons Blog: Root certificate will expire on 14 March — users need to update Firefox to prevent add-on breakage](#) (2025/03/10 23:11)
UPDATE - 19 March We've discovered a bug impacting some older extensions for users on Firefox 128+ or ESR 115 (in this case even updating Firefox won't resolve the root certificate expiration). We're working on fixes that will ship soon. Otherwise, we recommend keeping Firefox and your extensions up to date. This will resolve the vast majority of root certification issues. On 14 March a root certificate (the resource used to prove an add-on was approved by Mozilla) will expire, meaning Firefox users on versions older than 128 (or ESR 115) will not be able to use their add-ons. We want developers to be aware of this in case some of your users are on older versions of Firefox that may be impacted. Should you

see bug reports or negative reviews reflecting the effects of the certificate expiration, we recommend alerting your users to this support article that summarizes the issue and guides them through the process of updating Firefox so their add-ons work again. The post Root certificate will expire on 14 March — users need to update Firefox to prevent add-on breakage appeared first on Mozilla Add-ons Community Blog.

- [Niko Matsakis: Rust in 2025: Targeting foundational software](#) (2025/03/10 13:33)

Rust turns 10 this year. It's a good time to take a look at where we are and where I think we need to be going. This post is the first in a series I'm calling "Rust in 2025". This first post describes my general vision for how Rust fits into the computing landscape. The remaining posts will outline major focus areas that I think are needed to make this vision come to pass. Oh, and fair warning, I'm expecting some controversy along the way—at least I hope so, since otherwise I'm just repeating things everyone knows. My vision for Rust: foundational software I see Rust's mission as making it dramatically more accessible to author and maintain foundational software. By foundational I mean the software that underlies everything else. You can already see this in the areas where Rust is highly successful: CLI and development tools that everybody uses to do their work and which are often embedded into other tools¹; cloud platforms that people use to run their applications²; embedded devices that are in the things around (and above) us; and, increasingly, the kernels that run everything else (both Windows and Linux!). Foundational software needs performance, reliability—and productivity The needs of foundational software have a lot in common with all software, but everything is extra important. Reliability is paramount, because when the foundations fail, everything on top fails also. Performance overhead is to be avoided because it becomes a floor on the performance achievable by the layers above you. Traditionally, achieving the extra-strong requirements of foundational software has meant that you can't do it with "normal" code. You had two choices. You could use C or C++³, which give great power but demand perfection in response⁴. Or, you could use a higher-level language like Java or Go, but in a very particular way designed to keep performance high. You have to avoid abstractions and conveniences and minimizing allocations so as not to trigger the garbage collector. Rust changed the balance by combining C++'s innovations in zero-cost abstractions with a type system that can guarantee memory safety. The result is a pretty cool tool, one that (often, at least) lets you write high-level code with low-level performance and without fear of memory safety errors. Empowerment and lowering the barrier to entry In my Rust talks, I often say that type systems and static checks sound to most developers like "spinach", something their parents forced them to eat because it was "good for them", but not something anybody wants. The truth is that type systems are like spinach—popeye spinach. Having a type system to structure your thinking makes you more effective, regardless of your experience level. If you are a beginner, learning the type system helps you learn how to structure software for success. If you are an expert, the type system helps you create structures that will catch your mistakes faster (as well as those of your less experienced colleagues). Yehuda Katz sometimes says, "When I'm feeling alert, I build abstractions that will help tired Yehuda be more effective", which I've always thought was a great way of putting it. What about non-foundational software? When I say that Rust's mission is to target foundational software, I don't mean that's all it's good for. Projects like Dioxus, Tauri, and Leptos are doing fascinating, pioneering work pushing the boundaries of Rust into higher-level applications like GUIs and Webpages. I don't believe this kind of high-level development will ever be Rust's sweet spot. But that doesn't mean I think we should ignore them—in fact, quite the opposite. Stretch goals are how you grow The traditional thinking goes that, because foundational software often needs control over low-level details, it's not as important to focus on accessibility and ergonomics. In my view, though, the fact that foundational software needs control over low-level details only makes it more important to try and achieve good ergonomics. Anything you can do to help the developer focus on the details that matter most will make them more productive. I think projects that stretch Rust to higher-level areas, like Dioxus, Tauri, and Leptos, are a great way to identify opportunities to make Rust programming more

convenient. These opportunities then trickle down to make Rust easier to use for everyone. The trick is to avoid losing the control and reliability that foundational applications need along the way (and it ain't always easy). Cover the whole stack There's another reason to make sure that higher-level applications are pleasant in Rust: it means that people can build their entire stack using one technology. I've talked to a number of people who expected just to use Rust for one thing, say a tail-latency-sensitive data plane service, but they wound up using it for everything. Why? Because it turned out that, once they learned it, Rust was quite productive and using one language meant they could share libraries and support code. Put another way, simple code is simple no matter what language you build it in.5 "Smooth, iterative deepening" The other lesson I've learned is that you want to enable what I think of as smooth, iterative deepening. This rather odd phrase is the one that always comes to my mind, somehow. The idea is that a user's first experience should be simple—they should be able to get up and going quickly. As they get further into their project, the user will find places where it's not doing what they want, and they'll need to take control. They should be able to do this in a localized way, changing one part of their project without disturbing everything else. Smooth, iterative deepening sounds easy but is in fact very hard. Many projects fail either because the initial experience is hard or because the step from simple-to-control is in fact more like scaling a cliff, requiring users to learn a lot of background material. Rust certainly doesn't always succeed—but we succeed enough, and I like to think we're always working to do better. What's to come This is the first post of the series. My current plan6 is to post four follow-ups that cover what I see as the core investments we need to make to improve Rust's fit for foundational software. In my mind, the first three talk about how we should double down on some of Rust's core values: achieving smooth language interop by doubling down on extensibility; extending the type system to achieve clarity of purpose; leveling up the Rust ecosystem by building out better guidelines, tools, and leveraging the Rust Foundation. After that, I'll talk about the Rust open-source organization and what I think we should be doing there to make contributing to and maintaining Rust as accessible and, dare I say it, joyful as we can. Plenty of people use ripgrep, but did you know that when you do full text search in VSCode, you are also using ripgrep? And of course Deno makes heavy use of Rust, as does a lot of Python tooling, like the uv package manager. The list goes on and on. ← What do AWS, Azure, CloudFlare, and Fastly all have in common? They're all big Rust users. ← Rod Chapman tells me I should include Ada. He's not wrong, particularly if you are able to use SPARK to prove strong memory safety (and stronger properties, like panic freedom or even functional correctness). But Ada's never really caught on broadly, although it's very successful in certain spaces. ← Alas, we are but human. ← Well, that's true if the language meets a certain base bar. I'd say that even "simple" code in C isn't all that simple, given that you don't even have basic types like vectors and hashmaps available. ← I reserve the right to change it as I go! ←

- [The Servo Blog: This month in Servo: new elements, IME support, delegate API, and more!](#) (2025/03/10 00:00)

Servo now supports more HTML and CSS features: the `<details>` element (@simonwuelker, #35261) the `<meter>` element (@simonwuelker, #35524) the `<progress>` element (@simonwuelker, #35531) the 'quotes' property (@xiaochengh, @Loirooriol, #34770, #35420) the 'isolation' property (@kkoyung, @Loirooriol, #35552) 'overflow: clip' (@longvatrong111, #35103) 'overflow' property with two values (@yehzizhen, @mrobinson, #35414) the '::slotted' selector (@simonwuelker, #35352) Plus several new web API features: contextmenu events (@pewsheen, #35364) WritableStream (@gterzian, @jdm, #34844) ReadableStreamBYOBRequest (@Taym95, #35074) initial support for FontFace and its load() method (@mukilan, #35262) toBlob() and toDataURL() on WebGPU canvases (@sagudev, #35237) bytes() on Request, Response, and Blob (@shanehandley, @yoseio, @gterzian, @Taym95, #35250, #35151) href, origin, protocol, username, password, host, hostname, port, pathname, search, and hash properties on HTMLAreaElement (@shanehandley, #35482) insertRule() with no argument on CSSGroupingRule (@Loirooriol, #35295) `<slot>` elements are now fully supported including layout (@simonwuelker, #35220, #35519), and we've also landed support for the

':slotted' selector (@simonwuelker, #35352). Shadow roots are now supported in devtools (@simonwuelker, #35294), and we've fixed some bugs related to shadow DOM trees (@simonwuelker, #35276, #35338), event handling (@simonwuelker, #35380), and custom elements (@maxtidev, @jdm, #35382). We've landed layout improvements around 'border-collapse' (@Loirooriol, #35219), 'align-content: normal' (@rayguo17, #35178), 'place-self' with 'position: absolute' (@Loirooriol, #35208), the intrinsic sizing keywords (@Loirooriol, #35413, #35469, #35471, #35630, #35642, #35663, #35652, #35688), and 'position: absolute' now works correctly in a 'position: relative' grid item (@stevennovaryo, #35014). Input has also been improved, with better IME support (@dklassic, #35535, #35623) and several fixes to touch input (@kongbai1996, @jschwe, @shubhamg13, #35450, #35031, #35550, #35537, #35692). Servo-the-browser (servoshell) Directory listings are now enabled for local files (@mrobinson, #35317). servoshell's dialogs now use egui (@chickenleaf, #34823, #35399, #35464, #35507, #35564, #35577, #35657, #35671), rather than shelling out to a program like zenity (@chickenleaf, #35674), making them more secure and no longer falling back to terminal input. We've also fixed a bug when closing a tab other than the current one (@pewsheen, #35569). Servo-the-engine (embedding) We've simplified our embedding API by merging all input event delivery into `WebView::notify_input_event` (@mrobinson, @mukilan, #35430), making bluetooth optional (@jdm, @webbeef, #35479, #35590), making the "background hang monitor" optional (@jdm, #35256), and eliminating the need to depend on `webxr` (@mrobinson, #35229). We've also moved some servoshell-only options out of `Opts` (@mrobinson, #35377, #35407), since they have no effect on Servo's behaviour. We've landed our initial delegate-based API (@delan, @mrobinson, @mukilan, #35196, #35260, #35297, #35396, #35400, #35544, #35579, #35662, #35672), which replaces our old message-based API for integrating Servo with your app (@mrobinson, @delan, @mukilan, #35284, #35315, #35366). By implementing `WebViewDelegate` and `ServoDelegate` and installing them, you can have Servo call back into your app's logic with ease. We've simplified the `RenderingContext` trait (@wusyong, @mrobinson, #35251, #35553) and added three built-in `RenderingContext` impls (@mrobinson, @mukilan, #35465, #35501), making it easier to set up a context Servo can render to: `WindowRenderingContext` renders to a whole window `OffscreenRenderingContext` renders to part of a window `SoftwareRenderingContext` renders to an image, without hardware acceleration We've heavily reworked and documented our webview rendering model (@mrobinson, @wusyong, @mukilan, #35522, #35621), moved image output and shutdown logic out of the compositor (@mrobinson, @wusyong, #35538), and removed some complicated logic around synchronous repaints when a window is resized (@mrobinson, #35283, #35277). These changes should make it a lot clearer how to get Servo's webviews onto your display. One part of this model that we're starting to move away from is the support for multiple webviews in one rendering context (@mrobinson, @wusyong, #35536). First landed in #31417, this was an expedient way to add support for multiple webviews, but it imposed some serious limitations on how webviews could be interleaved with other app content, and the performance and security was inadequate. We've updated our `winit_minimal` example to take advantage of these changes (@webbeef, #35350, #35686), simplify it further (@robertohuertasm, #35253), and fix window resizing (@webbeef, #35691). Perf and stability The compositor now notifies the embedder of new frames immediately (@mrobinson, #35369), not via the constellation thread. Servo's typical memory usage has been reduced by over 1% thanks to Node object optimisations (@webbeef, #35592, #35554), and we've also improved our memory profiler (@webbeef, #35618, #35607). We've fixed a bug causing very high CPU usage on sites like `wikipedia.org` (@webbeef, #35245), as well as bugs affecting `requestAnimationFrame` (@mukilan, #35387, #35435). You can now configure our tracing-based profiler (`--features tracing`) with `servo --tracing-filter` instead of `SERVO_TRACING` (@jschwe, #35370). We've continued reducing our use of `unsafe` in `script` (@nscaife, @stephenmuss, #35351, #35360, #35367, #35411), and moving parts of `script` to `script_bindings` (@jdm, #35279, #35280, #35292, #35457, #35459, #35578, #35620). Breaking up our massive `script` crate is absolutely critical

for reducing Servo's build times. We've fixed crashes that happen when moving windows past the edge of your monitor (@webbeef, #35235), when unpaired UTF-16 surrogates are sent to the DOM (@mrobinson, #35381), when focusing elements inside shadow roots (@simonwuelker, #35606), and when calling `getAsString()` on a `DataTransferItem` (@Gae24, #35699). We've also continued working on static analysis that will help catch crashes due to GC borrow hazards (@augustebaum, @yerke, @jdm, @Gae24, #35541, #35593, #35565, #35610, #35591, #35609, #35601, #35596, #35595, #35594, #35597, #35622, #35604, #35616, #35605, #35640, #35647, #35646). Donations Thanks again for your generous support! We are now receiving 4363 USD/month (+13.7% over January) in recurring donations. This helps cover the cost of our self-hosted CI runners and Outreachy internships. Servo is also on thanks.dev, and already 21 GitHub users (+5 over January) that depend on Servo are sponsoring us there. If you use Servo libraries like `url`, `html5ever`, `selectors`, or `cssparser`, signing up for thanks.dev could be a good way for you (or your employer) to give back to the community. 4363 USD/month 10000 As always, use of these funds will be decided transparently in the Technical Steering Committee. For more details, head to our Sponsorship page.

- [Don Marti: Pro tips: links for 9 March 2025](#) (2025/03/09 00:00)

Jason Lefkowitz covers how to set up the Compose key (and make everything you type awesome™), in [Make special characters stupid easy: meet the compose key switching software](#) offers Ethical, easy-to-use and privacy-conscious alternatives to well-known software Pro tip: avoid generative AI images in blog posts (even if your CMS says you should have one for SEO purposes) unless you want to make a political statement: [AI: The New Aesthetics of Fascism](#) by Gareth Watkins Got third-party tracking scripts or pixels on your site? Avoid legal grief, take them off. [Caught with Their Hand in the Cookie Jar: CNN's Privacy Lawsuit is Served Fresh and the Court is Taking a Bite](#) by Blake Landis. (Highest priority is to get rid of the Meta pixel. That's not just a pro-evil-dictator tattoo for your web site, it's really easy for lawyers to check for.) Add data poisoning for AI scrapers hitting your GitHub Pages site: [Trapping AI from the Algorithmic Sabotage Research Group \(ASRG\)](#) Got a small business? Like riding bikes? [Relocating to the Netherlands with DAFT](#) If you need an integer and all you have is four 2s, Eli Bendersky has some math advice: [Making any integer with four 2s Nearly a Year Later](#), Mozilla is Still Promoting OneRep (Part of the Mozilla Monitor Plus service. Pro tip: check [Have I Been Pwned](#) directly) Why you need a radio (yes, you!) by Audrey Eschright The Linux kernel project can't use code from sanctioned countries. Other projects need to check compliance with sanctions, too. [US Blocks Open Source 'Help' From These Countries](#) by Steven J. Vaughan-Nichols Jake Archibald covers [The case against self-closing tags in HTML](#) (you don't need `
` just `
`. John D. Cook makes rounding numbers much easier (if you use balanced ternary) in [A magical land where rounding equals truncation](#) Understanding the legal issues for small community sites under UK law: [#2: Five things you need if you run a small, low-risk user-to-user service](#) by Rachel Coldicutt

- [Don Marti: advertising personalization: good for you?](#) (2025/03/08 00:00)

A new paper is out, collecting some of the top arguments in favor of personalized advertising: [The Intended and Unintended Consequences of Privacy Regulation for Consumer Marketing](#) by Jean-Pierre Dubé, John G. Lynch, Dirk Bergemann, Mert Demirer, Avi Goldfarb, Garrett Johnson, Anja Lambrecht, Tesary Lin, Anna Tuchman, Catherine E. Tucker It's probably going to get cited this privacy law season. But, as an Internet optimist, I'm still not buying the argument that personalized advertising has important benefits that need to be balanced with privacy. Looking at the literature, it is more likely that certain risks are inherent to personalization as such and that reducing personalization is more likely to be a bonus benefit of privacy protection than a trade-off. Some notes and links follow. p. 3 We do not consider legal arguments for consumer privacy as a fundamental right or concerns about access to personal data by malign actors or governments. Avoiding malign actors is the big reason for restricting personalized ads. And malign actors are numerous. The high-profile national security threats are already in the news, but most people

will encounter miscellaneous malware, scams, rip-offs and other lesser villainy enabled by ad personalization more often than they have to deal with state or quasi-state adversaries. There is no hard line between malign actors and totally legit sellers—not only does the personalized ad business have plenty of halfway crooks, you can find n/m-way crooks for arbitrary values of n and m. Ad personalization gives a bunch of hard-to-overcome advantages to deceptive sellers. Although scams are generally illegal and/or against advertising platform policies, personalization makes the rules easier to evade, as we see with some ways that Facebook ads are optimized for deceptive advertising. Most personalized ads aren't clustered at the good (high-quality pair of shoes in your size, on sale, next door!) or bad (malware pre-configured for your system) ends of the spectrum. Advertisers at all levels of quality and honesty are present, so any framework for thinking about ad personalization needs to take that variability into account. p. 3 Some privacy advocates assume, incorrectly, that personalized marketing based on granular consumer data is automatically harmful... Treating personalized advertising as harmful by default is not an assumption, but a useful heuristic based on both theoretical models and real-world experience. personally, I don't pay attention to your ad if it's personalized to me—it's as credible as a cold call. But I might pay attention to your ad if it's run in a place where the editors of sites that cover your industry would see it, or your mom would. Yes, it is possible for professors to imagine a hypothetical world in which personalization is beneficial, but that only works if you make the unrealistic simplifying assumption that all sellers are honest and that the only impact of personalization is to show people ads that are more or less well matched to them. The theoretical arguments in favor of personalized advertising break down as soon as you level up your economic model to consider the presence of both honest and deceptive advertisers in a market. See Gardete and Bart, Tailored Cheap Talk: The Effects of Privacy Policy On Ad Content and Market Outcomes. Our research suggests that another peril of sharing very high quality targeting information with advertisers is that ad content may become less credible and persuasive to consumers. An advertising medium that allows for personalization is incapable of conveying as much information from an honest seller to a potential buyer as an advertising medium that does not support personalization. Mustri et al., in Behavioral Advertising and Consumer Welfare, find that products found in behaviorally targeted ads are likely to be associated with lower quality vendors and higher product prices compared to competing alternatives found among search results. p. 8 Which Consumers Care Most About Privacy, and Do Privacy Policies Unintentionally Favor the Privileged? Lots of studies show that, basically, some people really want cross-context personalized advertising, some people don't, and for the largest group in the middle, it depends how you ask. (references at the 30-40-30 rule). But the difference in consumer preferences is not about privilege, it's about information level. See Turow et. al, Americans Reject Tailored Advertising and Three Activities That Enable It. That study includes a survey of privacy preferences before and after informing the participants about data practices—and people were more likely to say they do not want tailored advertising after getting the additional information. In the Censuswide study Who's In the Know: The Privacy Pulse Report, the experienced advertisers surveyed in the USA (people with 5 or more years of ad experience) were more likely than average to use an ad blocker (66% > 52%), and privacy is now the number one reason for people to use one. It is reasonable for policy-makers to consider the preferences of better-informed people—which is already a thing in fields such as transportation safety and public health. p. 11 Poorer consumers live in data deserts (Tucker 2023), causing algorithmic exclusion due to missing or fragmented data. This exclusion thwarts marketing outreach and may deprive them of offers, exacerbating data deserts and marginalization. Instead of speculating about this problem, personalized advertising proponents who are concerned about under-tracked consumers can already look at other good examples. Early adopters of privacy tools and preferences are helpfully acting as the experimental group for a study that the surveillance business hasn't yet run. If people on whom less data is collected are getting fewer win-win offers, then the privacy early adopters should have worse consumer outcomes than people who leave the personalization turned on. For

example, Apple iOS users with App Tracking Transparency (ATT) set to allow tracking should be reporting higher satisfaction and doing fewer returns and chargebacks. So far, this does not seem to be happening. (For a related result, see Bian et al., Consumer Surveillance and Financial Fraud. Consumers who deliberately placed themselves in a data desert by changing ATT to disallow tracking reported less fraud.) Click this to buy better stuff and be happier And there's little evidence to suggest that if a personalized ad system knows someone to be poor, that they'll receive more of the kind of legit, well-matched offers that are targeted to the more affluent. Poor people tend to receive more predatory finance and other deceptive offers, so may be better off on average with ads less well matched to their situation. p. 13 More broadly, without cross-site/app identity, consumers enjoy less free content This depends on how you measure content and how you define enjoy. The Kircher and Foerderer paper states that, although children's games for Android got fewer updates on average after a targeted advertising policy change by Google, Only exceptionally well-rated and demanded games experienced more feature updates, which could be interpreted as a sign of opportunity due to better monetization potential or weakened competition. However, considering that we observed these effects only for games in the highest decile of app quality and demand and given that the median user rating of a game is 4.1 of 5, our findings suggest widespread game abandonment. By Sturgeon's Law, a policy change that benefits the top 10% of games but not the bottom 90% (which, in total, account for a small fraction of total installs and an even smaller fraction of gameplay) is a win for the users. Another relevant paper is Kox, H., Straathof, B., and Zwart, G. (2014). Targeted advertising, platform competition and privacy. We find that more targeting increases competition and reduces the websites' profits, but yet in equilibrium websites choose maximum targeting as they cannot credibly commit to low targeting. A privacy protection policy can be beneficial for both consumers and websites. When both personalized and non-personalized ad impressions are available in the same market, the personalized impressions tend to go for about double the non-personalized. But it doesn't work to artificially turn off some data collection for a fraction of ad impressions, observe that revenue for those impressions is lower (compared to impressions with the data that are still available), and then extrapolate the revenue difference to a market in which no impressions have the data available. It is also important to consider the impact of extremely low-quality and/or illegal content in the personalized advertising market. Much of the economic role of ad personalization is not to match the right ad to the right user but to monetize a higher-value user on lower-value content. The surveillance economy is more like the commodification economy. Surveillance advertising companies are willing to pursue content commodification even to the point of taking big reputational risks from feeding ad money to the worst people on the Internet (Hiding in Plain Sight: The Ad-Supported Piracy Ring Driving Over a Billion Monthly Visits - deepsee.io, Senators Decry Adtech Failures as Ads Appear On CSAM Site). If advertising intermediaries were more limited in their ability to put a good ad on a bad site using user tracking, the higher-quality content sites would enjoy significantly increased market power. p. 14 Restrictions to limit the effectiveness of digital advertising would likely disproportionately disadvantage small businesses, since nine out of ten predominantly use digital advertising, especially on Meta Are small businesses really better off in the surveillance advertising era? Although personalized Big Tech advertising is the main ad medium available to small businesses today, there is clearly some survivorship bias going on here. The Kerrigan and Keating paper states that, While entrepreneurship has rebounded since the Great Recession and its aftermath, startup activity remains weak by historical standards. This period of time overlaps with the golden age of personalized advertising, after widespread adoption of smartphones but before Apple's ATT, the EU's GDPR, and California's CCPA. If personalized advertising is so good for small businesses, where are the extra small businesses enabled by it? We should have seen a small business boom in the second half of the 2010s, after most people in the USA got smartphones but before CCPA and ATT. Jakob Nielsen may have provided the best explanation in 2006's Search Engines as Leeches on the Web, which likely applies not just to

search, but to other auction-based ad placements like social media advertising. An auction-based advertising platform like those operated by Google and Meta is able to dynamically adjust its advertising rates to capture all of the expected incremental profits from the customers acquired through it. Part of the missing small business effect may also be caused by platform concentration. If, instead of an advertising duopoly, small businesses had more options for advertising, the power balance between platform (rentier) and small business (entrepreneur) might shift more toward the latter. See also Crawford et al., *The antitrust orthodoxy is blind to real data harms*. Policy makers might choose to prioritize pro-competition privacy legislation such as surveillance licensing for the largest, riskiest platforms in order to address competition concerns in parallel with privacy ones. p. 15 Since PETs are costly for firms to implement, forward-looking regulation should consider how to incentivize PET adoption and innovation further. In a section about how so-called privacy-enhancing technologies (PETs) have equal perceived privacy violation and bigger competition issues than conventional personalization, why recommend incentivizing PETs? The works cited would better support a recommendation to have a more detailed or informative consent experience for PETs than for cookie-based tracking. Because PETs obfuscate real-world privacy problems such as fraud and algorithmic discrimination, it would be more appropriate to require additional transparency, and possibly licensing, for PETs. PETs, despite their mathematical appeal to many at Big Tech firms, have a long list of problems when applied to the real world. The creeped-out attitude of users toward PETs is worth paying attention to, as people who grow up in market economies generally develop good instincts about information in markets—just like people who grow up playing ball games can get good at catching a ball without consciously doing calculus. Policymakers should pay more attention to user perceptions—which are based on real-world market activity—than to mathematical claims about developers’ PET projects. PETs should be considered from the point of view of regulators investigating discrimination and fraud complaints, which are often difficult to spot on large platforms. Because PETs have the effect of shredding the evidence of platform misdeeds, enabling the existing problems of adtech, just in a harder-to-observe way, they need more scrutiny, not incentivization. Coming soon: a useful large-scale experiment Policymakers may soon be able to learn from what could be the greatest experiment on the impact of ad personalization ever conducted. If Meta is required to offer Facebook users in the European Union a meaningfully de-personalized ad experience (and not just the less personalized ads option that still allows for personalization using fraud risk factors like age, gender, and location) then there will be a chance to measure what happens when users can choose personalized or de-personalized ads on a service that is otherwise the same. Personally, I bet that users with the personalization turned off will have better outcomes as consumers, but we’ll see. I’m pretty confident that personalized ads will turn out to be worse because tools and settings that tend to make personalization less effective have been available for a while, and if choosing the privacy option made you buy worse stuff, the surveillance companies would have said so by now. Conclusion I put these links and notes together to help myself out when someone drops a link to the Dubé et al. paper into an Internet argument, and put them up here in the hope that they will help others. Hardly anyone will read all the literature in this field, but a lot of the most interesting research is still found in corners of the library that Big Tech isn’t actively calling attention to. Thanks to Fengyang Lin for reviewing a draft of this post. Related Protecting Privacy, Empowering Small Business: A Path Forward with S.71 by Melanie Ensign, Founder and CEO, Discernible Inc. Small business owner testimony on S.71, the Vermont Data Privacy and Online Surveillance Act.

- [The Mozilla Blog: Browser choice? Here’s how EU’s DMA is helping make it real](#) (2025/03/06 16:32)

Too often, well-intentioned regulation misses the mark. This can be due to poor design, poor implementation, poor compliance, or failure to address unintended consequences. But the EU’s Digital Markets Act (DMA) has the potential to be different. The DMA is a regulatory framework that came into effect in the EU in March 2024. It covers a range of services, including browsers, where it empowers people in the EU to choose a

browser for themselves. Twelve months later, the verdict's in, and it proves what 98% of people told us before the DMA even kicked in last year: People want browser choice. And when they are given real choice, they will opt for a browser that serves their needs and preferences. For many EU consumers, this choice has been Firefox — an independent browser that offers the features they want in terms of privacy, safety, productivity and speed. Why is the DMA important? The DMA is a first-of-its-kind regulation. Its aim? Leveling the playing field so that EU consumers can discover and use products from smaller, innovative companies without being blocked by gatekeeper operating systems. In practice? When it comes to browsers, it means putting real choice in the hands of users. The desired outcome? Removing barriers to choice such as the complex device settings consumers have to navigate to change their pre-installed browser — and enabling them to keep their choice. Under the DMA, certain operating system providers are required to prompt users to actively select their preferred default web browser through a choice screen. What is a browser choice screen and when will I see one? Browser choice screens are a seemingly simple solution — a menu of options listed for you to choose your preferred default browser. The first DMA browser choice screens started rolling out in the EU in March 2024. Since then, they have slowly started appearing for: New and existing Android smartphones and tablet users with Chrome pre-set as a default browser (though rollout has been fairly inconsistent) New and existing iOS users with Safari as their default browser, who have iOS 18.2 and iPadOS 18.2 or later iOS versions installed on their device (initial roll out in iOS 17.4 was poorly designed) Unfortunately, if you're a smartphone user outside the EU — or a Windows/Mac user anywhere — you won't have seen the benefit of browser choice screens. Yet. Why does browser choice matter? From our research, we know that when well-designed and fully implemented, browser choice screens can be a powerful tool. They can not only provide people with real choice but can also have a huge impact on their levels of satisfaction with their tech by giving them freedom to customize their device experience more easily and quickly. Just as important, browser choice screens can promote user choice without degrading the user experience or causing unintended harm to consumers, competition and innovation. Browser choice screens also matter because they allow people to opt for independent alternatives like Firefox that are not tied to an operating system or device manufacturer. This makes them a critical intervention against the self-preferencing deployed by device manufacturers and operating systems, which push people to use their own label browsers and services. What happens when you choose for yourself? Despite vague compliance plans and some obvious gatekeeper non-compliance preventing the DMA from reaching its full potential, one year on we're starting to see how targeted regulation can help tackle some of the barriers to competition in the browser space and what can happen if the power of browser choice is in the hands of consumers. Since the launch of the first DMA browser choice screens on iOS in March 2024, people are making themselves heard: Firefox daily active users in Germany alone have increased by 99%. And in France, Firefox's daily active users on iOS grew by 111%. This growth has also been fueled by a number of new features coming to Firefox — from enhanced privacy controls and performance updates to new productivity tools — but the effect of the DMA is clear. When people are given browser choice, they vote with their feet for a product they love and stick with it. And we've found that when people choose Firefox via a DMA choice screen, they stick with it. Why choose Firefox? When Firefox was launched 20 years ago, our mission was to provide people with an alternative choice for a browser that prioritises user privacy, transparency and openness. The internet has changed dramatically since then, but our mission to keep the internet open and accessible to everyone is more important than ever. This is why we're continuously improving Firefox and working on providing users with real choice, giving them control of their internet experience through privacy and productivity enhancing features. Not convinced yet? Don't take our word for it and check out what users say they love about Firefox. Ready to make the switch? Don't wait on a choice screen, download Firefox or set it as your default browser now. Get Firefox Get the browser that protects what's important The post Browser choice? Here's how EU's DMA is helping make it real appeared first on The Mozilla

Blog.

- [Mozilla Thunderbird: Thunderbird for Android January/February 2025 Progress Report](#) (2025/03/06 15:12)

Hello, everyone, and welcome to the first Android Progress Report of 2025. We're ready to hit the ground running improving Thunderbird for Android experience for all of our users. Our January/February update involves a look at improvements to the account drawer and folders on our roadmap, an update on Google and K-9 Mail, and explores our first step towards Thunderbird on iOS. Account Drawer Improvements As we noted in our last post on the blog, improving the account drawer experience is one of our top priorities for development in 2025. We heard your feedback and want to make sure we provide an account drawer that lets you navigate between accounts easily and efficiently. Let's briefly go into the most common feedback: The accounts on the same domains or with similar names are difficult to distinguish from the two letters provided. It isn't clear how the account name influences the initials. The icons seemed to be jumping around, especially obvious with 3-5 accounts. There is a lot of spacing in the new drawer. Users would like more customization options, such as an account picture or icon. Some users would like to see a broader view that shows the whole account name. With just one account, the accounts sidebar isn't very useful. Our design folks are working on some mockups on where the journey is taking us. We're going to share them on the beta topicbox where you can provide more targeted feedback, but for a sneak peek here is a medium-fidelity mockup of what the new drawer and settings could look like: On the technical side, we've integrated an image loader for the upcoming pictures. We now need to gradually adapt the mockups. We will begin with the settings screen changes and then adapt the drawer itself to follow. Notifications and Error States Some of you had the feeling your email was not arriving quick enough. While email delivery is reliable, there are a few settings in Thunderbird for Android and K-9 mail that aren't obvious leading to confusion. When permissions are not granted, functionality is simply turned off instead of telling the user they actually need to grant the alarms permission for us to do a regular sync. Or maybe the sync interval is simply set to the default of 1 hour. We're still in the process of mapping out the best experience here, but will have more updates soon. See the notifications support article in case you are experiencing issues. A few things we're aiming for this year: Show an indicator in foreground service notification when push isn't working for all configured folders Show more detailed information when foreground service notification is tapped Move most error messages from the system notifications to an area in-app to clearly identify when there is an error Make authentication errors, certificate errors, and persistent connectivity issues use the new in-app mechanism Make the folder synchronization settings more clear (ever wondered why there is "sync" and "push" and if you should have both enabled or not?) Prompt for permissions when they are needed, such as aforementioned alarms permission Indicate to the user if permissions are missing for their folder settings. Better debug tool in case of notification issues. Road(map) to the Highway Our roadmap is currently under review from the Thunderbird council. Once we have their final approval, we'll update the roadmap documentation. While we're waiting, we would like to share some of the items we've proposed: Listening to community feedback on Mozilla Connect and implementing HTML signatures and quick filter actions, similar to the Thunderbird Desktop Backend refactoring work on the messages database to improve synchronization Improving the message display so that you'll see fewer prompts to download additional messages Adding Android 15 compatibility, which is mainly Edge to Edge support Improving the QR code import defaults (relates to notification settings as well) Making better product decisions by (re-)introducing a limited amount of opt-in telemetry. Does that sound exciting to you? Would you like to be a part of this but don't feel you have the time? Are you good at writing Android apps in Kotlin and have an interest in muti-platform work? Well, do I have a treat for you! We're hiring an Android Senior Software Engineer to work on Thunderbird for Android! K-9 Mail Blocked from Gmail We briefly touched on this in the last update as well: some of our users on K-9 Mail have noticed issues with an "App Blocked" error when trying to log into

certain Gmail accounts. Google is asking K-9 Mail to go through a new verification process and has introduced some additional requirements that were not needed before. Users that are already logged in or have logged in recently should not be affected currently. Meeting these requirements depended on several factors beyond our control, so we weren't able to resolve this immediately. If you are experiencing this issue on K-9 Mail, the quickest workaround is to migrate to Thunderbird for Android, or check out one of the other options on the support page. For those interested, more technical details can be found in issue 8598. We're using keys on this application that have so far not been blocked. Our account import feature will make this transition pretty seamless. We've been able to make some major progress on this, we have a vendor for the required CASA review and expect the letter of validation to be shared soon. We're still hitting a wall with Google, as they are giving us inconsistent information on the state of the review, and making some requirements on the privacy policy that sound more like they are intended for web apps. We've made an effort to clarify this further and hope that Google will accept our revised policy. If all goes well we'll get approval by the end of the month, and then need to make some changes to the key distribution so that Thunderbird and K-9 use the intended keys. Our Plans for Thunderbird on iOS If you watched the Thunderbird Community Office Hours for January, you might have noticed us talking about iOS. You heard right - our plans for the Thunderbird iOS app are getting underway! We've been working on some basic architectural decisions and plan to publish a barebones repository on GitHub soon. You can expect a readme and some basic tools, but the real work will begin when we've hired a Senior Software Engineer who will lead development of a Thunderbird app for the iPhone and iPad. Interviews for some candidates have started and we wish them all the best! With this upcoming hire, we plan to have alpha code available on Test Flight by the end of the year. To set expectations up front, functionality will be quite basic. A lot of work goes into writing an email application from scratch. We're going to be focusing on a basic display of email messages, and then expanding to triage actions. Sending basic emails is also on our list. FOSDEM Our team recently attended FOSDEM in Brussels, Belgium. For those unfamiliar with FOSDEM, it's the Free and Open Source Software Developers' European Meeting—an event where many open-source enthusiasts come together to connect, share knowledge and ideas, and showcase the projects they're passionate about. We received a lot of valuable feedback from the community on Thunderbird for Android. Some key areas of feedback included the need for Exchange support, improvements to the folder drawer, performance enhancements, push notifications (and some confusion around their functionality), and much more. Our team was highly engaged in listening to this feedback, and we will take all of it into account as we plan our future roadmap. Thunderbird has always been a project developed in tandem with our community and it was exciting for us to be at FOSDEM to connect with our users, contributors and friends. In other news... As always, you can join our Android-related mailing lists on TopicBox. And if you want to help us test new features, you can become a beta tester. This blog post talks a lot about the exciting things we have planned for 2025. We're also hiring for two positions, and may have a third one later in the year. While our software is free and open source, creating a world class email application isn't without a cost. If you haven't already made a contribution in January, please consider supporting our work with a financial contribution. Thunderbird for Android relies entirely on user funding, so without your support we could likely only get to a fraction of what you see here. Making a contribution is really easy if you have Thunderbird for Android or K-9 Mail installed, just head over to the settings and sign up directly from your device. See you next month, The post Thunderbird for Android January/February 2025 Progress Report appeared first on The Thunderbird Blog.

- [Spidermonkey Development Blog: Implementing `Iterator.range` in SpiderMonkey](#) (2025/03/05 07:00)

In October 2024, I joined Outreachy as an Open Source contributor and in December 2024, I joined Outreachy as an intern working with Mozilla. My role was to implement the TC39 Range Proposal in the SpiderMonkey JavaScript engine. `Iterator.range` is a new built-in method proposed for

JavaScript iterators that allows generating a sequence of numbers within a specified range. It functions similarly to Python's range, providing an easy and efficient way to iterate over a series of values: `for (const i of Iterator.range(0, 43)) console.log(i); // 0 to 42` But also things like: `function* even() { for (const i of Iterator.range(0, Infinity)) if (i % 2 === 0) yield i; }` In this blog post, we will explore the implementation of `Iterator.range` in the SpiderMonkey JavaScript engine. Understanding the Implementation When I started working on `Iterator.range`, the initial implementation had been done, ie; adding a preference for the proposal and making the builtin accessible in the JavaScript shell. The `Iterator.range` simply returned `false`, a stub indicating that the actual implementation of `Iterator.range` was under development or not fully implemented, which is where I came in. As a start, I created a `CreateNumericRangeIterator` function that delegates to the `Iterator.range` function. Following that, I implemented the first three steps within the `Iterator.range` function. Next, I initialised variables and parameters for the `NUMBER-RANGE` data type in the `CreateNumericRangeIterator` function. I focused on implementing sequences that increase by one, such as `Iterator.range(0, 10)`. Next, I created an `IteratorRangeGenerator*
function` (ie, step 18 of the Range proposal), that when called doesn't execute immediately, but returns a generator object which follows the iterator protocol. Inside the generator function you have `yield` statements which represents where the function suspends its execution and provides value back to the caller. Additionally, I updated the `CreateNumericRangeIterator` function to invoke `IteratorRangeGenerator*
function` with the appropriate arguments, aligning with Step 19 of the specification, and added tests to verify its functionality. The generator will pause at each `yield`, and will not continue until the next method is called on the generator object that is created. The `NumericRangeIteratorPrototype` (Step 27.1.4.2 of the proposal) is the object that holds the iterator prototype for the Numeric range iterator. The `next()` method is added to the `NumericRangeIteratorPrototype`, when you call the `next()` method on an object created from `NumericRangeIteratorPrototype`, it doesn't directly return a value, but it makes the generator yield the next value in the series, effectively resuming the suspended generator. The first time you invoke `next()` on the generator object created via `IteratorRangeGenerator*
function`, the generator will run up to the first `yield` statement and return the first value. When you invoke `next()` again, the `NumericRangeIteratorNext()` will be called. This method uses `GeneratorResume(this)`, which means the generator will pick up right where it left off, continuing to iterate the next `yield` statement or until iteration ends. Generator Alternative After discussions with my mentors Daniel and Arai, I transitioned from a generator-based implementation to a more efficient slot-based approach. This change involved defining slots to store the state necessary for computing the next value. The reasons included: Efficiency: Directly managing iteration state is faster than relying on generator functions. Simplified Implementation: A slot-based approach eliminates the need for generator-specific handling, making the code more maintainable. Better Alignment with Other Iterators: Existing built-in iterators such as `StringIteratorPrototype` and `ArrayIteratorPrototype` do not use generators in their implementations. Performance and Benchmarks To quantify the performance improvements gained by transitioning from a generator-based implementation to a slot-based approach, I conducted comparative benchmarks using a test in the current `bookmarks/central`, and in the revision that used generator-based approach. My benchmark tested two key scenarios: Floating-point range iteration: Iterating through 100,000 numbers with a step of 0.1 `BigInt` range iteration: Iterating through 1,000,000 `BigInts` with a step of 2 Each test was run 100 times to eliminate anomalies. The benchmark code was structured as follows: `// Benchmark for Number iteration var sum = 0; for (var i = 0; i < 100; ++i) { for (num of Iterator.range(0, 100000, 0.1)) { sum += num; } } print(sum); // Benchmark for BigInt iteration var sum = 0n; for (var i = 0; i < 100; ++i) { for (num of Iterator.range(0n, 1000000n, 2n)) { sum += num; } } print(sum);` Results Implementation Execution Time (ms) Improvement Generator-based 8,174.60 - Slot-based 2,725.33 66.70% The slot-based implementation completed the benchmark in just 2.7 seconds compared to 8.2 seconds for the generator-based approach. This represents a 66.7% reduction in execution time,

or in other words, the optimized implementation is approximately 3 times faster. Challenges Implementing BigInt support was straightforward from a specification perspective, but I encountered two blockers: 1. Handling Infinity Checks Correctly The specification ensures that start is either a Number or a BigInt in steps 3.a and 4.a. However, step 5 states: If start is $+\infty$ or $-\infty$, throw a RangeError. Despite following this, my implementation still threw an error stating that start must be finite. After investigating, I found that the issue stemmed from using a self-hosted isFinite function. The specification requires isFinite to throw a TypeError for BigInt, but the self-hosted Number_isFinite returns false instead. This turned out to be more of an implementation issue than a specification issue. See Github discussion here. Fix: Explicitly check that start is a number before calling isFinite: `// Step 5: If start is $+\infty$ or $-\infty$, throw a RangeError. if (typeof start === "number" && !Number_isFinite(start)) { ThrowRangeError(JSMSG_ITERATOR_RANGE_START_INFINITY); }` 2. Floating Point Precision Errors When testing floating-point sequences, I encountered an issue where some decimal values were not represented exactly due to JavaScript's floating-point precision limitations. This caused incorrect test results. There's a GitHub issue discussing this in depth. I implemented an approximatelyEqual function to compare values within a small margin of error. Fix: Using approximatelyEqual in tests: `const resultFloat2 = Array.from(Iterator.range(0, 1, 0.2)); approximatelyEqual(resultFloat2, [0, 0.2, 0.4, 0.6, 0.8]);` This function ensures that minor precision errors do not cause test failures, improving floating-point range calculations. Next Steps and Future Improvements There are different stages a TC39 proposal goes through before it can be shipped. This document shows the different stages that a proposal goes through from ideation to consumption. The Iterator.range proposal is currently at stage 1 which is the Draft stage. Ideally, the proposal should advance to stage 3 which means that the specification is stable and no changes to the proposal are expected, but some necessary changes may still occur due to web incompatibilities or feedback from production-grade implementations. Currently, this implementation is in its early stages of implementation. It's only built in Nightly and disabled by default until such a time the proposal is in stage 3 or 4 and no further revision to the specification can be made. Final Thoughts Working on the Iterator.range implementation in SpiderMonkey has been a deeply rewarding experience. I learned how to navigate a large and complex codebase, collaborate with experienced engineers, and translate a formal specification into an optimized, real-world implementation. The transition from a generator-based approach to a slot-based one was a significant learning moment, reinforcing the importance of efficiency in JavaScript engine internals. Beyond technical skills, I gained a deeper appreciation for the standardization process in JavaScript. The experience highlighted how proposals evolve through real-world feedback, and how early-stage implementations help shape their final form. As Iterator.range continues its journey through the TC39 proposal stages, I look forward to seeing its adoption in JavaScript engines and the impact it will have on developers. I hope this post provides useful insights into SpiderMonkey development and encourages others to contribute to open-source projects and JavaScript standardization efforts. If you'd like to read more, here are my blog posts that I made during the project: [Decoding Open Source: Vocabulary I've Learned on My Outreachy Journey](#) [Mid-Internship Progress Report: Achievements and Goals Ahead](#) [Navigating TC39 Proposals: From Error Handling to Iterator.range](#)

- [This Week In Rust: This Week in Rust 589](#) (2025/03/05 05:00)

Hello and welcome to another issue of This Week in Rust! Rust is a programming language empowering everyone to build reliable and efficient software. This is a weekly summary of its progress and community. Want something mentioned? Tag us at [@ThisWeekInRust](#) on X (formerly Twitter) or [@ThisWeekinRust](#) on mastodon.social, or send us a pull request. Want to get involved? We love contributions. This Week in Rust is openly developed on GitHub and archives can be viewed at [this-week-in-rust.org](#). If you find any errors in this week's issue, please submit a PR. Want TWIR in your inbox? [Subscribe here](#). Updates from Rust Community Official Rust participates in Google Summer of Code 2025 February

Project Goals Update Announcing Rustup 1.28.0 Newsletters This Month in Rust OSDev: February 2025 Rust Trends Issue #60 The Embedded Rustacean Issue #40 Project/Tooling Updates Announcing Wiremocket: Wiremock For Websockets A More Modular request memberlist 0.6 - gossip protocol for cluster membership management Maelstrom Clustered Test Runner v0.13: new watch mode and GitHub workflow support Bincode 2.0.0 Observations/Thoughts The problem with type aliases Take a break: Rust match has fallthrough Fast columnar JSON decoding with arrow-rs Some things that make Rust lifetimes hard to learn Performance optimization, and how to do it wrong Do not run any Cargo commands on untrusted projects Cargo's missing stability guarantees [video] Rust Under the Hood [video] 9 Rules for Porting Rust to the Browser Rust Walkthroughs The power of interning: making a time series database 2000x smaller in Rust Lowering Top Level Items Building a DNS Server in Rust: Part 1 of 2 Miscellaneous [video] Rust's Global Allocator [video] Vulkanised 2025: Shipping a Game with Vulkan and Rust in 100 Days [video] Creating a website on GitHub pages with mdBook EuroRust 2025 Paris announced Please nominate newer innovative projects for GOSIM Rust Spotlight ASAP! Crate of the Week This week's crate is wild, a pretty fast linker written in Rust. Thanks to Mateusz Miłkuła for the (sort of self-)suggestion! Please submit your suggestions and votes for next week! Calls for Testing An important step for RFC implementation is for people to experiment with the implementation and give feedback, especially before stabilization. If you are a feature implementer and would like your RFC to appear in this list, add a call-for-testing label to your RFC along with a comment providing testing instructions and/or guidance on which aspect(s) of the feature need testing. No calls for testing were issued this week by Rust, Rust language RFCs or Rustup. Let us know if you would like your feature to be tracked as a part of this list. Call for Participation; projects and speakers CFP - Projects Always wanted to contribute to open-source projects but did not know where to start? Every week we highlight some tasks from the Rust community for you to pick and get started! Some of these tasks may also have mentors available, visit the task page for more information. No Calls for participation were submitted this week. If you are a Rust project owner and are looking for contributors, please submit tasks here or through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! CFP - Events Are you a new or experienced speaker looking for a place to share something cool? This section highlights events that are being planned and are accepting submissions to join their event as a speaker. EuroRust 2025| 2025-05-15 | Paris | 2025-10-09-2025-10-10 If you are an event organizer hoping to expand the reach of your event, please submit a link to the website through a PR to TWiR or by reaching out on X (formerly Twitter) or Mastodon! Updates from the Rust Project 502 pull requests were merged in the last week Compiler introduce feature(generic_const_parameter_types) fix parsing of ranges after unary operators implement #[cfg] in where clauses optimize empty provenance range checks Library add IntoBounds::intersect and RangeBounds::is_empty fix Windows Command search path bug stabilize core::str::from_utf8_mut as const stabilize extract_if stabilize hash_extract_if Cargo cargo: add SBOM support (RFC #3553) cargo: cli: forward bash completions of third party subcommands cargo: add completions for --lockfile-path cargo: reset \$CARGO if the running program is real cargo[.exe] cargo: get all members as available targets even though default-members was specified cargo: implemented build.build-dir config option Rustdoc librustdoc: return impl fmt::Display in more places instead of writing to strings fully qualify Result in generated doctest code Rustfmt use semver to match required version Clippy new lints: manual_midpoint, add unnecessary_debug_formatting lint move comparison_chain from style to pedantic macro_use_import: Don't check is attribute comes from expansion manual_strip: use existing identifier instead of placeholder needless_collect: avoid warning if non-iterator methods are used check for MSRV attributes in late passes using the HIR configuration option to lint incompatible_msrv in test code extend {implicit,inverted}_saturating_sub to expressions fix ICE in doc_nested_refdefs check by checking range fix ICE in manual_map lint fix: map_entry false positive inside closure fix: map_entry suggest wrongly when key is not Copy lint more cases with ptr_eq split needless_lifetime ' _ suggestions into elidable_lifetime_names Rust-Analyzer rust-analyzer: add identifier to

pull diagnostic LSP capabilities rust-analyzer: add anchor for intra-doc links to associated items rust-analyzer: add flip or-pattern assist rust-analyzer: allow "package/feature" format feature flag rust-analyzer: allow rust-project.json to specify sysroot workspace rust-analyzer: allow unsetting default cfgs rust-analyzer: cofigure out ohos target to avoid compilation crashes rust-analyzer: completion-ref-matching rust-analyzer: doc tests rust-analyzer: doc: remove nit from setup.md rust-analyzer: fix prefix adjustment hints unnecessarily introducing parens rust-analyzer: fix sysroot crate-graph construction not mapping crate-ids for proc-macros rust-analyzer: have inline_local_variable use precedence calculation for parentheses rust-analyzer: remove syntax editing from parenthesis computation rust-analyzer: support tuple struct patterns for expand_rest_pattern assist rust-analyzer: warn when the used toolchain looks too old for rust-analyzer Rust Compiler Performance Triage A pretty quiet week, with minimal changes in performance (positive or negative). Triage done by @simulacrum. Revision range: f5729cfe..daf59857 1 Regressions, 4 Improvements, 1 Mixed; 2 of them in rollups 29 artifact comparisons made in total Full report here Approved RFCs Changes to Rust follow the Rust RFC (request for comments) process. These are the RFCs that were approved for implementation this week: No RFCs were approved this week. Final Comment Period Every week, the team announces the 'final comment period' for RFCs and key PRs which are reaching a decision. Express your opinions now. Tracking Issues & PRs Rust Denote ControlFlow as #[must_use] Turn order dependent trait objects future incompat warning into a hard error Stabilize const_vec_string_slice add a "future" edition Tracking Issue for const_sockaddr_setters Rust RFCs RFC: Deprecate the per-build-target edition field in Cargo.toml RFC: Demote i686-pc-windows-gnu to Tier 2 Cargo feat(package): add --exclude-lockfile flag Other Areas *No Items entered Final Comment Period this week for Language Team, Language Reference or Unsafe Code Guidelines. Let us know if you would like your PRs, Tracking Issues or RFCs to be tracked as a part of this list. New and Updated RFCs Local Default Bounds to assist Forget and other ?Trait. Forget marker trait RFC: Crate changelog field Upcoming Events Rusty Events between 2025-03-05 - 2025-04-02 Virtual 2025-03-05 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2025-03-06 | Virtual (Nürnberg, DE) | Rust Nurnberg DE Rust Nürnberg online 2025-03-06 | Virtual (Rotterdam, NL) | Bevy Game Development Bevy Meetup #9 2025-03-06 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - - Ratatui - Terminal User Interfaces in Rust with Orhun Parmaksız 2025-03-09 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - - Creating A Mock Blockchain in Rust with Sourav Mishra 2025-03-09 | Virtual (Tel Aviv-Yafo, IL) | Rust TLV Becoming a Rust Champion: Leading Your Team to Success with Yoni Peleg 2025-03-11 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2025-03-11 | Virtual (London, UK) | Women in Rust Community Catch Up 2025-03-13 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-03-18 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - - crum: Complex Numbers and Complex Matrices in Rust with Frans Slabber 2025-03-18 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2025-03-19 | Virtual (Vancouver, BC, CA) | Vancouver Rust Bacon & Performance Benchmarking 2025-03-20 | Virtual (Tel Aviv-Yafo, IL) | Code Mavens - - Rust and embedded programming with Leon Vak (online in English) 2025-03-25 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2025-03-25 | Virtual (London, UK) | Women in Rust Lunch & Learn: SKling into Rust - crafting a simple interpreter 2025-03-27 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2025-04-01 | Virtual (Buffalo, NY, US) | Buffalo Rust Meetup Buffalo Rust User Group 2025-04-02 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing Africa 2025-03-11 | Johannesburg, ZA | Johannesburg Rust Meetup The Stages of Error Management Asia 2025-03-15 | Beijing, CN | WebAssembly and Rust Meetup (Wasm Empowering AI) KCD Beijing 2025 2025-03-19 | Tel Aviv-Yafo, IL | Rust TLV In person Rust March 2025 at Jit in Tel Aviv 2025-03-28 | Kowloon Tong, HK | Rust Asia Rust Asia 2025 Europe 2025-03-05 | Barcelona, ES | BcnRust 17th BcnRust Meetup 2025-03-05 | Köln, DE | Rust Cologne Rust in March: Rust Edition 2024 2025-03-05 | Oxford, UK | Oxford Rust Meetup Group Hackathon 2025-03-07 | Prague, CZ | Rust Czech Republic Rust meetup in Braiins offices 2025-03-12 | Reading, UK | Reading Rust Workshop Reading Rust Meetup 2025-03-13 | Biel,

CH | Rust Bern 2025 Rust Talks Bern #2 @ BFH Biel 2025-03-14 | Paris, FR | Rust in Paris Rust in Paris 2025 2025-03-18 | Basel, CH | Rust Basel Rust Meetup #10 @ MDPI Basel 2025-03-18 | Leipzig, DE | Rust - Modern Systems Programming in Leipzig Topic TBD 2025-03-20 | Edinburgh, UK | Rust and Friends March Talks! (Two) 2025-03-20 | Prague, CZ | Rust Prague Rust/C++ Meetup Prague (March 2025) 2025-03-25 | Aarhus, DK | Rust Aarhus Hack Night - Robot Edition 2025-03-25 | Eindhoven, NL | RustNL Rust x Julia Meetup Eindhoven 2025-03-26 | Warsaw, PL | Rustikon Rustikon 2025-03-27 | Augsburg, DE | Rust Meetup Augsburg Rust Meetup #12: Testing in Rust 2025-04-02 | München, DE | Rust Munich Rust Munich 2025 / 1 - hybrid 2025-04-02 | Oxford, UK | Oxford Rust Meetup Group Oxford Rust and C++ social North America 2025-03-06 | Montréal, QC, CA | Rust Montréal March Monthly Social 2025-03-06 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2025-03-06 | Saint Louis, MO, US | STL Rust CRDTs in Rust 2025-03-10 | Boston, MA, US | Boston Rust Meetup Davis Square Rust Lunch, Mar 10 2025-03-13 | Chicago, IL, US | Chicago Rust Meetup Rust Happy Hour 2025-03-18 | San Francisco, CA, US | San Francisco Rust Study Group Rust Hacking in Person 2025-03-18 | Spokane, WA, US | Spokane Rust Monthly Meetup: Intro to Rust and Python; Using Rustup, Cargo, and Rust! 2025-03-20 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2025-03-20 | Redmond, WA, US | Seattle Rust User Group March, 2025 SRUG (Seattle Rust User Group) Meetup 2025-03-21 | México City, MX | Rust MX Rust y AWS Lambda. Preparando el camino para desplegar ML/AI 2025-03-26 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2025-03-27 | Atlanta, GA, US | Rust Atlanta We're going again! 2025-03-31 | Boulder, CO, US | Solid State Depot Boulder Rust: Bryan presents Rusted Hardware Oceania 2025-03-11 | Christchurch, NZ | Christchurch Rust Meetup Group Christchurch Rust Meetup South America 2025-03-15 | São Paulo, BR | Rust São Paulo Meetup Encontro do Rust-SP na CloudWalk If you are running a Rust event please add it to the calendar to get it mentioned here. Please remember to add a link to the event too. Email the Rust Community Team for access. Jobs Please see the latest Who's Hiring thread on r/rust Quote of the Week The performance impact of moving to Rust - and this is a common theme across everything done when we've moved from C/C++ to Rust - we saw a 5 to 15% performance Improvement. I'll say that one of the ways that you could attack that kind of stat is say well you rewrote it so whenever you rewrite something you're going to improve it and if you'd rewritten it in C or C++ you would have also seen an improvement like that but the fact is we did not intend to get a performance Improvement. This was purely a porting exercise and we saw this now. And the other aspect of this is that we never see performance regressions either when we're doing our ports [...] - Mark Russinovich at RustNationUK '25' Despite lacking suggestions, llogiq is quite pleased with his choice. Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell, llogiq, cdmistman, ericseppanen, extrawurst, U007D, joelmarcey, mariannegoldin, bennyvasquez, bdillo Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- [Firefox Nightly: High Profile Improvements - These Weeks in Firefox: Issue 176](#) (2025/03/05 02:09)

Highlights Multiple profiles are now enabled by default in Nightly! Prefred on in Nightly only - upcoming 0.5% rollout in Beta and Release will be done via Nimbus File bugs against Toolkit :: Startup and Profile System For WebExtension authors, failed downloads with a NETWORK_FAILED error can now be resumed through the downloads WebExtensions API - Bug 1694049 Thanks to kernp25 for contributing a fix for this longstanding bug ☐ Alexandre Poirot has enabled Service Worker debugging by default in the Firefox DevTools debugger Moritz fixed a bug where users switching between Fx versions using the same profile were unable to search and saw broken context menus. This affected a small number of users in release, and so his patch landed in a dot release. Friends of the Firefox team Resolved bugs (excluding employees) Volunteers that fixed more than one bug kernp25 New contributors (☐ = first patch) ☐ Arkadiy Tetelman added a patch to avoid skipping domains in DNR requestDomains/initiatorDomains Spencer renamed 'action' params in Actions.sys.mjs to avoid conflict with 'action' module Project Updates Add-

ons / Web Extensions WebExtensions Framework Tabs that have been redirected to a moz-extension urls (e.g. through webRequest or DNR) can now be reloaded or navigated back successfully to the moz-extension urls in the tab history - Bug 1826867 WebExtension APIs Starting from Firefox 136 menus.update and menus.remove API methods will return a rejected promise instead of ignoring non-existing menus - Bug 1688743 Fixed bug in DNR `initiatorDomains` and `requestDomains` conditions - Bug 1939981 Thanks to Arkadiy Tetelman for reporting and contributing a fix for this bug DevTools Alexandre Poirot worked on several fixes for webextensions debugging: Fixed a regression where webextensions creating iframes would lead to blank devtools (#1941006) Fixed a bug where content scripts could appear duplicated in the source tree, and would also result in awkward behavior when pausing / stepping (#1941681) Improved the handling of messages logged from content scripts to avoid duplicated log entries (#1941418) Show manifest error when reloading an addon from DevTools (#1940079) An error message is certainly better than silently failing in this case. Nicolas Chevobbe fixed a few accessibility and High Contrast mode issues: Fixed the color of the close button used in sourcemap warnings to be correct in dark mode (#1942227) Updated the netmonitor status codes to be visible in High Contrast mode (#1940749) Nicolas Chevobbe also made several improvements to autocompletes in DevTools: The CSS autocomplete will now provide suggestions for variables when used inside calc, for instance in width: calc(var(- (#1444772) The autocomplete in the inspector search is now handling correctly classnames containing numbers and special characters (#1220387) Nicolas Chevobbe addressed a long standing issue in the JSON viewer which would always display values after parsing them via JS, which could result in approximate values for big numbers or high precision decimals. The source is now always displayed, and we also show the JS parsed value next to it when relevant. (#1431808) Julian Descottes fixed a bug with the Debugger local script override which might incorrectly show scripts as being overridden if another DevTools Toolbox previously added an override (#1940998) Julian Descottes addressed a regression which would lead to blank devtools if any iframe on the page was showing an XML document with an invalid XSLT (#1934520) Julian Descottes fixed an issue with the Debugger pretty printing feature, which could fail if the file contained windows-style line-breaks (#1932023) Julian Descottes finalized an old patch to show details about the throttling profiles you can select in the network monitor and RDM (#1770932) In case you need a reminder of what 2G speed is like. Hubert Boma Manilla fixed a recent regression with the preview popup in the debugger which would not show the value of the hovered token (#1941269) Hubert Boma Manilla improved the edit and resend feature in the netmonitor to properly set the security flags for the resent requests and avoid bugs for instance with requests using CORS (#1883380) WebDriver BiDi Spencer Poor (:speneth) fixed all the linter warnings in Actions.sys.mjs. Henrik Skupin fixed an issue when another Firefox instance is using the Remote Agent and hasn't shut down properly, a new test starting Firefox on the same WebSocket port will trigger a 5-second wait for the port to free up. If it remains unavailable, Firefox will shut down. Lint, Docs and Workflow Gijs enabled the eslint-plugin-promise rule for valid-params which checks for functions on Promise being called with the correct amount of arguments. This will protect against cases such as foo.catch() which looks like it should do something, but doesn't. Standard8 added a linter for checking that we don't include package names in unpublished package.json files. New Tab Page The New Tab layout update has rolled out to 100% of our US population! We're aiming to do the rest of the world soon. Search and Navigation Address Bar Mandy fixed a bug so that we now correctly handle a question mark appearing at the end of a search string (1648956) James landed a patch related to urlbar interaction tracking that ends all interactions after one hour of inactivity (1936956) Emilio fixed a bug where, when Firefox Translations was used for a page, the address bar wasn't showing what language the page had been translated to (1938121) Marco fixed a bug where the search open tabs feature wasn't finding tabs where the page had loaded with a 4xx or 5xx server status error (1939597) Scotch Bonnet Yazan fixed an issue where search mode was defaulting to Google after the browser was restarted (1925235) Daisuke fixed an issue where

screenreaders were failing to narrate secondary actions buttons when in actions mode (1929515) Daisuke also fixed an issue related to using ctrl+shift+tab to navigate to the previous tab (1933243) Daisuke fixed an issue where the unified search button dropdown wasn't being dismissed when a new tab was opened using ctrl+T (1936545) Marco enabled the not secure label for http for Release 136 (1937136) Dale fixed an issue so that the search keyword is retained for contextual search mode when the engine is autofilled (1938036) Daisuke landed a patch so that contextual search works with multiple-word searches (1938040) Moritz fixed a bug where the unified search button icon wasn't being refreshed correctly in private browsing windows (1938567) Firefox Suggest Drew landed a significant number of telemetry-related patches (1940808, 1941100, 1941161, 1941690, 1941692, 1941694, 1941988, 1941989) Storybook/Reusable Components Hanna created the moz-input-text component Storybook Tim created the moz-input-search component (more changes to come) Storybook Anna added documentation for moz-button Storybook Jules added support for description and support-page to the moz-radio-group (for the group itself, it was already there for individual radios) Storybook Hanna added the moz-box-button component that will be used in the Settings redesign and some other settings throughout Firefox Storybook

- [Mozilla Thunderbird: Thunderbird Release Channel Update](#) (2025/03/04 20:17)

The monthly Release channel is ready to help you move from annual to monthly updates in Thunderbird. This update lets you know how to switch from the annual update (ESR) to monthly updates (Release), why you might have to wait, and what features you'll get first! How do I switch from annual to monthly updates (ESR to Release)? Right now, you can switch to the Release channel through manual installs only from the Thunderbird website Downloads page. Other installation sources will have the Release version in the future such as Windows Store, 3rd-party sites and various Linux packages such as Snap and Flatpak. If you use Add-ons, however, we recommend checking if your installed Add-ons work in the Release channel with versions 136.0 or higher. First, back up your profile data, as you should always do before making major changes. And check that your computer meets the System Requirements for version 136. Then go to the Downloads page of the website. If Release Channel does not show "Thunderbird Release" then correct it. Click the 'Download' button. For Windows and macOS, run the downloaded file to install the monthly release into the same directory where the ESR is currently installed. (If you have installed Thunderbird ESR into a directory that is different from the default location, then you must do a custom installation to that directory.) For Linux, consult the Linux installation knowledge base (KB) article. I switched to Release but I want to switch back to ESR. How do I do this? If you switched to Release but want to switch back, for example, because of Add-ons, follow the steps below. Please note, this is valid for the current Release and ESR channels, and we will update here in the event of an underlying database change in ESR that would not make this possible: Backup the Thunderbird profile Download ESR and install ESR version into "Mozilla Thunderbird" directory where the release version is installed, but ... You must use command line for the first start up, to override downgrade protection, per

https://support.mozilla.org/kb/dedicated-profile-thunderbird-installation#w_what-happens-to-my-profile-if-i-downgrade-to-a-previous-version-of-thunderbird. Otherwise you will see a "you are using an old version" check If you have trouble, ask for help at

<https://support.mozilla.org/questions/thunderbird> What's new in 136.0? Now that you know how to make the switch, here's some reasons to make the change. Here are some of the key features you'll get as soon as you upgrade to the Release channel: Improved Dark Reader Enable dark reader for the message pane with `mail.dark-reader.enabled` preference Bug: 1715361 Improved Dark Mode Messages are automatically adapted to dark mode with a quick toggle in the header Bug: 1942206 A Global Switch for Threading New "Appearance" Settings UI to globally control message threading/sorting order Bugs: 1829737, 1943724 Filters in the Folder Pane Message filters are now available in the Folder Pane

context menu Bug: 1845206 Horizontal Threadpane Scrolling Enable horizontal threadpane scrolling with ``mail.threadpane.table.horizontal_scroll`` preference Bug: 381821 Improved Calendar Setup Wizard Added checkbox to select/unselect all calendars in the calendar setup wizard Bug: 1733959 See all the changes in our Release Notes. The post Thunderbird Release Channel Update appeared first on The Thunderbird Blog.

- [Eitan Isaacson: New Contrast Control Settings](#) (2025/03/04 00:00)

What We Did In today's nightly build of Firefox we introduced a simple three-state toggle for color contrast. It allows you to choose between these options: Automatic (use system settings) Honor the system's high contrast preferences in web content. If this is enabled, and say you are using a Windows High Contrast theme, the web content's style will be overridden wherever needed to display text and links with the system's colors. Off Regardless of the system's settings the web content should be rendered as the content author intended. Colors will not be forced. Custom Always force user specified colors in web content. Colors for text and links foreground and background can be specified in the preferences UI. Who Would Use This and How Would They Benefit? There are many users with varying degrees of vision impairment that benefit from having control over the colors and intensity of the contrast between text and its background. We pride ourselves in Firefox for offering fine grained control to those who need it. We also try to do a good job anticipating the edge cases where forced high contrast might interfere with a user's experience, for example we put a solid background backplate under text that is superimposed on top of an image. But, the web is big and complex and there will always be cases where forced colors will not work. For example, take this Wordle puzzle: The gold tiles indicate letters that are in the hidden word, and gold tiles represent letters that are in the word and in the right place in the word. When a user is in forced colors mode, the background of the tiles disappears and the puzzle becomes unsolvable: What More Can We Expect? The work we did to add contrast control to the settings is a first step in allowing a user to quickly configure their web browser experience in order to adapt to their current uses. Just like text zoom, there is no one-size-fits-all. We will be working to allow users to easily adjust their color contrast settings as they are browsing and using the web.

- [The Rust Programming Language Blog: Announcing rustup 1.28.1](#) (2025/03/04 00:00)

The rustup team is happy to announce the release of rustup version 1.28.1. Rustup is the recommended tool to install Rust, a programming language that is empowering everyone to build reliable and efficient software. Challenges with rustup 1.28.0 rustup 1.28.0 was a significant release with many changes, and there was a quick response from many folks that this release broke their processes. While we considered yanking the release, we worried that this would cause problems for people who had already updated to adopt some of the changes. Instead, we are rolling forward with 1.28.1 today and potentially further bugfix releases to address the feedback that comes in. We value all constructive feedback -- please keep it coming in the issue tracker. In particular, the change with regard to implicit toolchain installation is being discussed in this issue. What's new in rustup 1.28.1 This release contains the following fixes: Automatic install is enabled by default but can be opted out by setting `RUSTUP_AUTO_INSTALL` environment variable to 0. [pr#4214](#) [pr#4227](#) rustup show active-toolchain will only print a single line, as it did in 1.27. [pr#4221](#) Fixed a bug in the request backend that would erroneously timeout downloads after 30s. [pr#4218](#) Use relative symlinks for proxies. [pr#4226](#) How to update If you have a previous version of rustup installed, getting rustup 1.28.1 is as easy as stopping any programs which may be using Rustup (e.g. closing your IDE) and running: `$ rustup self update` Rustup will also automatically update itself at the end of a normal toolchain update: `$ rustup update` If you don't have it already, you can get rustup from the appropriate page on our website. Rustup's documentation is also available in the rustup book. Caveats Rustup releases can come with problems not caused by rustup itself but just due to

having a new release. As such, we recommend paying attention to the following potential issues in particular: Anti-malware scanners might be blocking rustup or stopping it from creating or copying files (especially when installing rust-docs, since it contains many small files). In your CI environment, rustup might fail when trying to perform a self-update. This is a known issue, and in the case where this issue does occur, we recommend applying the following workaround at the beginning of your workflow: `$ rustup set auto-self-update disable` Also, starting from 1.28.0, rustup will no longer attempt to self-update in CI environments, so this workaround should not be necessary in the future. These issues should be automatically resolved in a few weeks when the anti-malware scanners are updated to be aware of the new rustup release, and the hosted version is updated across all CI runners. Thanks Thanks to the rustup and t-release team members who came together to quickly address these issues.

- [Mozilla Addons Blog: Styling your listing page on AMO with Markdown](#) (2025/03/03 19:41)

The Mozilla Add-ons team is excited to announce that developers can now style content on addons.mozilla.org (AMO) using Markdown. From the early days of AMO developers have been able to style parts of their add-ons' listings using basic HTML tags like ``, `<code>`, and `<abbr>`. This has been a great way for developers to emphasize information or add character to their listings. This feature has, unfortunately, also been a common source of mistakes. It's not unusual to see HTML tags in listings due to closing tags, typos, or developers trying to use unsupported HTML elements. To address this common source of errors and to better align with other tools that developers use, Mozilla has replaced AMO's limited HTML support with a limited set of Markdown. Our flavor of Markdown AMO supports about a dozen Markdown rules: bold, italic, monospace, links, abbreviations, code blocks, blockquotes, ordered lists, and unordered lists. HTML is not supported. The full list of supported syntax and examples can be found in our documentation. Currently, Markdown can be used in four areas: an add-on's description, a custom license, custom privacy policy, and in review replies posted by the developer. As before, links are automatically replaced by a version that is routed through AMO's link bouncer. How this change affects HTML content AMO's Markdown does not support HTML content. As a result, any HTML entered in a field that supports Markdown will display as plain text. This change is not retroactive. Any content authored while HTML was supported will continue to display as originally intended. Only content updated after Markdown support was added will be treated as Markdown. The post [Styling your listing page on AMO with Markdown](#) appeared first on Mozilla Add-ons Community Blog.

- [Jan-Erik Rediger: Seven-year Moziversary](#) (2025/03/03 09:50)

In March 2018 I started a job as a Telemetry engineer at Mozilla. Seven years later I'm still here on my Moziversary, as I was in 2019, 2020, 2021, 2022, 2023 and 2024. Mozilla is not the same company it was when I joined. No one on the C-level is here for as long as I am. There have been 5 larger reorgs in the past year and two layoffs in different departments, plus the big round of layoffs at the Mozilla Foundation. My part of the organization was moved around again and for the moment Data Engineering is nested under the Infrastructure Org. Good colleagues left or were laid off. Just recently my team shrunk again. Notably I haven't posted anything under the mozilla tag since 2022, other than my Moziversary blog posts. All-Hands MozWeek1 Dublin happened. The next one will be in Washington, D.C. Probably one of the worst choices given the state of the USA right now, and I might just skip it. I do hope for a team work week, but that's not decided yet. The one constant over the past years is my work on Glean, which hasn't stopped. We're finally putting dedicated effort to move legacy telemetry in Firefox Desktop over to Glean (Are we Glean yet?). But still Glean isn't where I'd like it to be. Some of the early decisions in its design are coming back to bite us, the codebase grew significantly and could use a bit of cleanup, we never got the time to work on performance and memory improvements and our chosen local data storage desperately needs an update. Some of that was on my list last year already. If only there were less distractions that pull me of this work

again and again. It's on my list of goals again this year, so maybe it works out better this time. But other than that I don't know where Mozilla will be a year from now. It's going to be a challenging year. Thank you I'm in this job for seven years not only because I like the tech I get to work on, but also because my colleagues make it a delight to come back to work every day. Thanks to Alessio, Chris, Travis and Abhishek for being the Data Collection Tools team. Also thanks to Bruno, who was part of the team until recently. There's countless other people at Mozilla, inside Data Engineering but also across other parts of the organization, that I got to connect, chat and work with. Thank you! Footnotes: 1 Change everywhere. Those Mozilla work weeks were renamed by Marketing.

- [The Rust Programming Language Blog: February Project Goals Update](#) (2025/03/03 00:00)

This is the first Project Goals update for the new 2025h1 period. For the first 6 months of 2025, the Rust project will work towards a slate of 39 project goals, with 3 of them designed as Flagship Goals. This post provides selected updates on our progress towards these goals (or, in some cases, lack thereof). The full details for any particular goal are available in its associated tracking issue on the rust-project-goals repository. Flagship goals Bring the Async Rust experience closer to parity with sync Rust Why this goal? This work continues our drive to improve support for async programming in Rust. In 2024H2 we stabilized async closures; explored the generator design space; and began work on the dinosaur crate, an experimental proc-macro to provide dynamic dispatch for async functions in traits. In 2025H1 our plan is to deliver (1) improved support for async-fn-in-traits, completely subsuming the functionality of the async-trait crate; (2) progress towards sync and async generators, simplifying the creation of iterators and async data streams; (3) and improve the ergonomics of Pin, making lower-level async coding more approachable. These items together start to unblock the creation of the next generation of async libraries in the wider ecosystem, as progress there has been blocked on a stable solution for async traits and streams. What has happened? The biggest news is that Rust 1.85 is stable and includes two major features that impact Async Rust. The first is async closures, which has been on many people's wish lists for a long time and was expertly moved forward by @compiler-errors over the last year. The second feature included in 1.85 is the new lifetime capture rules as part of Rust 2024 edition. This should substantially improve the experience of using async Rust anytime a user writes `-> impl Future`, as it removes the need for `+ '_` or similar bounds in most cases. It will also lead to an easier to understand language, since those bounds only worked by exploiting the more subtle rules of `impl Trait` in a way that runs contrary to their actual semantic role in the language. In the 2024 Edition, the subtle rule is gone and we capture all input lifetimes by default, with the ability to use `+ use<>` syntax to opt out. See this blog post for more. Generators. The lang team also held a design meeting to review the design for generators, with the outcome of the last one being that we will implement a `std::iter::iter!` macro (exact path TBD) in the compiler, as a lang team experiment that allows the use of the yield syntax. We decided to go in this direction because we want to reserve `gen` for self-borrowing and perhaps lending generators, and aren't yet decided on which subset of features to expose under that syntax. This decision interacts with ongoing compiler development that isn't ready yet to enable experimentation with lending. Our hope is that in the meantime, by shipping `iter!` we will give people the chance to start using generators in their own code and better understand which limitations people hit in practice. As you may have noticed, I'm not talking about async generators here. Those are the ultimate goal for the async initiative, but we felt the first step should be clarifying the state of synchronous generators so we can build on that when talking about async ones. Dinosaur. dinosaur v0.1.3 was released, with another release in the works. We think we are approaching a 1.0 release real soon now (tm). At this point you should be able to try it on your crate to enable dyn dispatch for traits with async fn and other `-> impl Trait` methods. If you need to use it together with `#[trait_variant]`, you may need to wait until the next release when `#55` is fixed. 1 detailed update available. Comment by @tmandry posted on 2025-02-26: Rust 1.85 The first update is the release of Rust 1.85, which

had at least two major features that impact Async Rust. The first is async closures, which has been on many people's wish lists for a long time and was expertly moved forward by @compiler-errors over the last year. The second is the new lifetime capture rules as part of Rust 2024 edition. This should substantially improve the experience of using async Rust anytime a user writes `-> impl Future`, as it removes the need for `+'_` or similar bounds in most cases. It will also lead to an easier to understand language, since those bounds only worked by exploiting the more subtle rules of `impl Trait` in a way that runs contrary to their actual semantic role in the language. In the 2024 Edition, the subtle rule is gone and we capture all input lifetimes by default, with the ability to use `+ use<>` syntax to opt out. See this blog post for more. Generators The lang team held two design meetings on generators, with the outcome of the last one being that we will implement a `std::iter::iter!` macro (exact path TBD) in the compiler, as a lang team experiment that allows the use of the `yield` syntax. We decided to go in this direction because we want to reserve `gen` for self-borrowing and perhaps lending generators, and aren't yet decided on which subset of features to expose under that syntax. This decision interacts with ongoing compiler development that isn't ready yet to enable experimentation with lending. Our hope is that in the meantime, by shipping `iter!` we will give people the chance to start using generators in their own code and better understand which limitations people hit in practice. As you may have noticed, I'm not talking about async generators here. Those are the ultimate goal for the async initiative, but we felt the first step should be clarifying the state of synchronous generators so we can build on that when talking about async ones.

`dynosaur` `dynosaur v0.1.3` was released, with another release in the works. We think we are approaching a 1.0 release real soon now (tm). At this point you should be able to try it on your crate to enable `dyn dispatch` for traits with `async fn` and other `-> impl Trait` methods. If you need to use it together with `#[trait_variant]`, you may need to wait until the next release when #55 is fixed. Other The async project goal was accepted for 2025H1! Organize Rust All-Hands 2025 Why this goal? May 15, 2025 marks the 10-year anniversary of Rust's 1.0 release; it also marks 10 years since the creation of the Rust subteams. At the time there were 6 Rust teams with 24 people in total. There are now 57 teams with 166 people. In-person All Hands meetings are an effective way to help these maintainers get to know one another with high-bandwidth discussions. This year, the Rust project will be coming together for RustWeek 2025, a joint event organized with RustNL. Participating project teams will use the time to share knowledge, make plans, or just get to know one another better. One particular goal for the All Hands is reviewing a draft of the Rust Vision Doc, a document that aims to take stock of where Rust is and lay out high-level goals for the next few years. What has happened? Planning is proceeding well. In addition to Rust maintainers, we are inviting all project goal owners to attend the All Hands (note that the accompanying RustWeek conference is open to the public, it's just the All Hands portion that is invitation only). There are currently over 100 project members signed up to attend. For those invited to the All Hands: Travel funds may be available if you are unable to finance travel through your employer. Get in touch for details. Please participate in the brainstorm for how best to use the All Hands time in the `all-hands-2025 Zulip stream`. If you do plan to attend, please purchase your travel + hotels in a timely fashion as the discount rates will expire. 1 detailed update available. Comment by @m-ou-se posted on 2025-02-28: What happened so far: Allocated a budget for the event. Funds have been transferred from the Rust Foundation to RustNL. Booked the venue, including lunch and snacks. Remaining budget from last year's travel grant programme added to this year's, to help cover the travel costs. Announcement published: <https://blog.rust-lang.org/inside-rust/2024/09/02/all-hands.html> After sending many reminders to teams and individuals, about 110 project members signed up. (And a few cancelled.) Formal invitations sent out to all those who registered. Information on the all-hands: <https://rustweek.org/all-hands/> Hotel reservations available: <https://rustweek.org/hotels-all-hands/> Created a public and a private zulip channel. About 65 people confirmed they booked their hotel. Opened up discussion on what to discuss at the all-hands. Invited guests: project goal (task) owners who aren't a project member (12 people in total). 4 of those signed up so far. Acquired 150

secret gifts for the pre-all-hands day. Next up: Remind folks to get a ticket for the RustWeek conference (tuesday+wednesday) if they want to join that as well. Invite more guests, after deciding on who else to invite. (To be discussed today in the council meeting.) Figure out if we can fund the travel+hotel costs for guests too. (To be discussed today in the council meeting.) Organise all the ideas for topics at the all-hands, so we can turn them into a consistent schedule later. Draft an allocation of the rooms depending on the teams and topics. Open the call for proposals for talks for the Project Track (on wednesday) as part of the RustWeek conference. Stabilize tooling needed by Rust for Linux Why this goal? This goal continues our work from 2024H2 in supporting the experimental support for Rust development in the Linux kernel. Whereas in 2024H2 we were focused on stabilizing required language features, our focus in 2025H1 is stabilizing compiler flags and tooling options. We will (1) implement RFC #3716 which lays out a design for ABI-modifying flags; (2) take the first step towards stabilizing build-std by creating a stable way to rebuild core with specific compiler options; (3) extending rustdoc, clippy, and the compiler with features that extract metadata for integration into other build systems (in this case, the kernel's build system). What has happened? We established the precise set of 2025H1 deliverables and we have been tracking them and have begun making progress towards them. Rustdoc has been updated to support extracting doc tests so that the Kernel can execute them in a special environment (this was previously done with a big hack) and RFL is in the process of trying to use that new support. The first PR towards the implementation of RFC #3716 has landed and the ARM team has begun reading early drafts of the design doc for -Zbuild-core with the cargo team. We are also working to finalize the stabilization of the language features that were developed in 2024H2, as two late-breaking complications arose. The first (an interaction between casting of raw pointers and arbitrary self types) is expected to be resolved by limiting the casts of raw pointers, which previously accepted some surprising code. We identified that only a very small set of crates relied on this bug/misfeature; we expect nonetheless to issue a forwards compatibility warning. We are also resolving an issue where derive(CoercePointee) was found to reveal the existence of some unstable impls in the stdlib. 3 detailed updates available. Comment by @nikomatsakis posted on 2025-01-15: In our meeting today we reviewed the plans for the 2025H1 project goal... "Almost done" stuff from before re-stabilize CoercePointee -- Alice is looking at this, it's a good opportunity to try out the new template that is being discussed stabilize arbitrary self types v2 -- @adetaylor left a comment 3 weeks ago indicating that everything was more-or-less landed. RFL is already using it, providing evidence that it works reasonably well. Next steps are then sorting out documentation and moving to stabilize. asm-goto -- ready for stabilization, not been merged yet, still doing some work on the Rust reference (PRs <https://github.com/rust-lang/rust/pull/133870>, <https://github.com/rust-lang/reference/pull/1693>) ABI-modifying flags The rust-lang/rfcs#3716 is now in Final Comment Period (FCP). There is a preliminary implementation in #133138 that @petrochenkov is going to be reviewing. Some more work will be needed to test, cleanup, etc after that lands. Other flags from RFL#2 We went through a series of flags from that RFL uses and looking into what might be blocking each of them. The process to stabilize one of these is basically to prepare the stabilization PR (minimal, but we need to rename the flag from -Z to -C) with a stabilization report and proper docs and then cc @davidtwco or @wesleywiser to prepare stabilization. In most cases we need to document how the flags can be misused, most commonly by linking against std or other crates not built with the same flags. If there are major correctness concerns as a result we likely want to consider the flag as "ABI-modifying". ability to extract dependency info, currently using -Zbinary_dep_depinfo=y -- basically ready to stabilize * tmandry: Do you want toolchain runtimes (libstd, compiler-rt) in your dep info? In my experience this features does 90% of what I want it to do, but removing the inclusion of runtimes is the only question I have before stabilizing. -Zcrate-attr, used to configure no-std without requiring it in the source file -- no real concerns -Zunpretty=expanded -- have to clearly document that the output is not stable, much like we did for emitting MIR. This is already "de facto" stable because in practice cargo expand uses

RUSTC_BOOTSTRAP=1 and everybody uses it. -Zno-jump-tables -- this should be considered an ABI-modifying flag because we believe it is needed for CFI and therefore there is a risk if incompatible modules are linked. -Zdwarf-version, -Zdebuginfo-compression -- this should be ready to stabilize, so long as we document that you should expect a mix of debuginfo etc when mixing things compiled with different versions (notably libstd, which uses DWARF4). Debuggers are already prepared for this scenario. zstd compression is supported as of Rust 1.82.0. stable rustdoc features allowing the RFL project to extract and customize rustdoc tests (--extract-doctests) @imperio authored <https://github.com/rust-lang/rust/pull/134531>, which is now up for review. Once PR lands, RFL will validate the design, and it can proceed to stabilization. clippy configuration (possibly .clippy.toml and CLIPPY_CONF_DIR) We discussed with clippy team, seems like this is not a big deal, mostly a doc change, one concern was whether clippy should accept options it doesn't recognize (because they may come from some future version of clippy). Not a big deal as of now, RFL only uses (msrv, check-private-items=true, disallowed-macros). rebuild libcore ARM team is working on this as part of this project goal, expect updates. □ Comment by @nikomatsakis posted on 2025-01-30: Updates: 2024H2 cleanup Arbitrary self types v2: Stabilization PR is open (rust-lang/rust#135881). Stabilize CoercePointee: RFL is using it now; we are making progress towards completing the stabilization. 2025H1 ABI-modifying flags RFC (rust-lang/rfcs#3716) has completed FCP and needs to be merged. Implementation PR #133138 authored by @azhagin remains under review. Other compiler flags, we made a prioritized list. One easy next step would be to rename all -Z flags in this list to something stabilizable (e.g., -C) that requires -Zunstable-features. [] -Zdwarf-version -- wesley wiser [] -Zdebuginfo-compression, unblocked [] -Zcrate-attr, used to configure no-std without requiring it in the source file, no real concerns [] -Zunpretty=expanded, unblocked, maybe needs a PR that says "don't rely on this", Linux only uses it for debugging macros (i.e. not in the normal build, so it is less critical). Needs a stable name, e.g., --emit macro-expanded, and we should make sure output cannot be piped to rustc. rustfmt told us (Rust for Linux) that they will try their best to keep rustfmt able to format the output of the expansion. [] -Zno-jump-tables, considered an ABI-modifying flag [] -Zbinary_dep_depinfo=y -- basically ready to stabilize (@tmandry asked "Do you want toolchain runtimes (libstd, compiler-rt) in your dep info? In my experience this features does 90% of what I want it to do, but removing the inclusion of runtimes is the only question I have before stabilizing", but we don't understand this point yet, as they were not present in the meeting). stable rustdoc: PR rust-lang/rust#134531 is under review, should land soon, then next step will be for RFL folks to try it out. Clippy configuration: no progress, discussed some new options and posted in thread, very minimal work required here. rebuild libcore: @davidtwco wasn't present so no update, but we know progress is being made Publicizing this work We discussed briefly how to better get the word out about this collaboration. Some points: @Darksonn will be speaking at Rust Nation We could author a Rust-lang blog post, perhaps once the language items are done done done? LWN article might be an option general-regs-only We discussed the possibility of a flag to avoid use of floating point registers, no firm conclusion yet reached. Comment by @nikomatsakis posted on 2025-02-20: Updates from our 2025-02-12 meeting: Given the recent controversy about Rust usage in the Kernel, the RFL group wrote up a policy document explainer to explain the policy, and there was a write-up on LWN. Regarding arbitrary self types and coerce pointee, we are waiting on rust-lang/rust#136764 and rust-lang/rust#136776. The former is on lang team FCP. The latter has received approval from lang team and is awaiting further impl work by @BoxyUwU. @ojeda is looking into how to manage dependency information and configure no-std externally. @GuillaumeGomez's impl of rustdoc features has landed and we are waiting on RFL to experiment with it. @davidtwco's team at ARM has authored a document regarding a blessed way to build-std and are collecting feedback. @wesleywiser is preparing a PR to add -Zdwarf-version to help advance compiler flags. There is an annoying issue related to cfg(no_global_oom_handling), which is no longer used by RFL but which remains in an older branch of the kernel (6.12, LTS). As a set of "leaf crates" that evolve together in a mono-

repo-like fashion, RFL would like to have a solution for disabling the orphan rule. Goals looking for help Implement Open API Namespace Support Help wanted: this project goal needs a compiler developer to move forward. If you'd like to help, please post in this goal's dedicated zulip topic. 1 detailed update available. Comment by @epage posted on 2025-02-20: Help wanted: this project goal needs a compiler developer to move forward. Prototype a new set of Cargo "plumbing" commands Help wanted: this project goal needs someone to work on the implementation. If you'd like to help, please post in this goal's dedicated zulip topic. 2 detailed updates available. Comment by @epage posted on 2025-02-20: Help wanted: this project goal needs someone to work on the implementation Comment by @ashiskumarnaik posted on 2025-02-23: Hi @epage I am very interested to collaborate in this implementation work. Let's talk on Zulip, Check DM. Publish first version of StableMIR on crates.io Help wanted: looking for people that want to help do a bit of refactoring. Please reach out through the project-stable-mir zulip channel or repository. 1 detailed update available. Comment by @celinval posted on 2025-02-25: No progress yet. Help Wanted: Looking for people that want to help do a bit of refactoring. Please reach out through the project-stable-mir zulip channel or repository. Stabilize public/private dependencies Help wanted: this project goal needs a compiler developer to move forward. If you'd like to help, please post in this goal's dedicated zulip topic. 1 detailed update available. Comment by @epage posted on 2025-02-20: Help wanted: this project goal needs a compiler developer to move forward. Other goal updates "Stabilizable" prototype for expanded const generics 1 detailed update available. Comment by @BoxyUwU posted on 2025-02-27: camelid has a PR up which is ~fully finished + reviewed which enables resolving and lowering all paths under `min_generic_const_args`. It's taken a while to get this bit finished as we had to take care not to make parts of the compiler unmaintainable by duplicating all the logic for type and const path lowering. build-std 1 detailed update available. Comment by @davidtwco posted on 2025-02-19: An initial update on what we've been up to and some background: This goal is submitted on behalf of the Rust team at Arm, but primarily worked on by @AdamGemmell. Anyone interested can always contact me for updates and I'll keep this issue up-to-date. Our team has been trying to make progress on build-std by completing the issues in rust-lang/wg-cargo-std-aware but found this wasn't especially effective as there wasn't a clearly defined scope or desired outcome for most issues and the relevant teams were lacking in the necessary context to evaluate any proposals. We've since had discussions with the Cargo team and agreed to draft a document describing the use cases, motivations and prior art for build-std such that the Cargo team can feel confident in reviewing any further proposals. @AdamGemmell shared an initial draft of this in #t-cargo on Zulip and it is undergoing further revisions following feedback. Following reaching a shared understanding of the context of the feature, @AdamGemmell will then draft a complete proposal for build-std that could feasibly be stabilised. It will describe the use cases which are and are not considered in-scope for build-std, and which will feature in an initial implementation. @davidtwco is ensuring that whatever mechanism that is eventually proposed to enable build-std to work on stable toolchains will also be suitable for the Rust for Linux project to use when building core themselves. Continue resolving `cargo-semver-checks` blockers for merging into cargo 1 detailed update available. Comment by @obi1kenobi posted on 2025-02-19: Thanks, Niko! Copying in the new tasks from the 2025h1 period: [] Prototype cross-crate linting using workarounds (@obi1kenobi) [] Allow linting generic types, lifetimes, bounds (@obi1kenobi) [] Handle "special cases" like 'static and ?Sized (@obi1kenobi) [] Handle #[doc(hidden)] in sealed trait analysis (@obi1kenobi) [x] Discussion and moral support (cargo, [rustdoc] Declarative (`macro_rules!`) macro improvements No detailed updates available. Evaluate approaches for seamless interop between C++ and Rust 1 detailed update available. Comment by @tmandry posted on 2025-02-26: We had a lang team design meeting about C++ interop today. The outcome was very positive, with enthusiasm for supporting an ambitious vision of C++ interop: One where a large majority of real-world C++ code can have automated bindings to Rust and vice-versa. At the same time, the team expressed a desire to do so in a way that remains in line

with Rust's values. In particular, we would not make Rust a superset of Rust+C++, but instead would define extension points that can be used to express language interop boundaries that go beyond what Rust itself allows. As an example, we could allow template instantiation via a Rust "plugin" without adding templates to Rust itself. Similarly, we could allow calling overloaded C++ methods without adding function overloading to Rust itself. Other interop needs are more natural to enable with features in the Rust language, like custom reference types. In either case, anything we do to support interop will need to be considered on its merits. Interop is a reason to support a feature, but it is never a "blank check" to add anything we might consider useful. The discussion so far has been at a high level. Next steps will be: Discuss what significant-but-realistic milestones we might pursue as part of upcoming project goals, and what it would take to make them happen. Whether this happens as part of another lang team meeting or a more dedicated kickoff meeting for interested parties, I'll be sure to keep the lang team in the loop and will continue posting updates here. Dive into more specific proposals for use cases we would like to enable in meetings with the language, library, and compiler teams. Notes: https://hackmd.io/2Ar_7CNoRkeXk1AARyOL7A?view Experiment with ergonomic ref-counting 1 detailed update available. Comment by @spastorino posted on 2025-02-26: There's a PR up <https://github.com/rust-lang/rust/pull/134797> which implements the proposed RFC without the optimizations. The PR is not yet merged and we need to continue now working on addressing comments to the PR until it is merged and then start implementing optimizations. Expose experimental LLVM features for GPU offloading 1 detailed update available. Comment by @ZuseZ4 posted on 2025-01-03: Happy New Year everyone! After a few more rounds of feedback, the next autodiff PR recently got merged: <https://github.com/rust-lang/rust/pull/130060> With that, I only have one last PR open to have a fully working autodiff MVP upstream. A few features had to be removed during upstreaming to simplify the reviewing process, but they should be easier to bring back as single PRs. Beginning next week, I will also work on an MVP for the batching feature of LLVM/Enzyme, which enables some AoS and SoA vectorization. It mostly re-uses the existing autodiff infrastructure, so I expect the PRs for it to be much smaller. On the GPU side, there has been a recent push by another developer to add a new AMD GPU target to the Rust compiler. This is something that I would have needed for the llvm offload project anyway, so I'm very happy to see movement here: <https://github.com/rust-lang/compiler-team/issues/823> Extend pubgrub to match cargo's dependency resolution 1 detailed update available. Comment by @Eh2406 posted on 2025-02-26: The major update so far is the release of PubGrub 0.3. This makes available the critical improvements made to allow the functionality and performance demanded by Cargo and UV. The other production users can now take advantage of the past few years of improvements. Big thanks to @konstin for making the release happen. Other progress is being stymied by being sick with Covid for a week in January and the resulting brain fog, followed by several high-priority projects for \$DAY_JOB. It is unclear when the demands of work will allow me to return focus to this project. Externally Implementable Items 1 detailed update available. Comment by @m-ou-se posted on 2025-02-28: Now that <https://github.com/rust-lang/rust/pull/135726> is merged, @jdonzelmann and I will be working on implementing EII. We have the design for the implementation worked out on our whiteboard. We don't expect any significant blockers at this point. We'll know more once we start writing the code next week. Finish the libtest json output experiment 1 detailed update available. Comment by @epage posted on 2025-02-20: This is on pause until the implementation for #92 is finished. The rustc side of #92 is under review and then some additional r-a and cargo work before implementation is done and its waiting on testing and stabilization. Implement restrictions, prepare for stabilization 1 detailed update available. Comment by @jhpratt posted on 2025-02-26: First status update: No progress. I will be reviewing the existing PR this weekend to see the feasibility of rebasing it versus reapplying patches by hand. My suspicion is that the latter will be preferable. Improve state machine codegen 1 detailed update available. Comment by @traviscross posted on 2025-02-26: This is, I believe, mostly waiting on us on the lang team to have a look, probably in a design meeting, to feel out what's in

the realm of possibility for us to accept. Instrument the Rust standard library with safety contracts 1 detailed update available. Comment by @celinval posted on 2025-01-03: Key developments: We have written and verified around 220 safety contracts in the verify-rust-std fork. 3 out of 14 challenges have been solved. We have successfully integrated Kani in the repository CI, and we are working on the integration of 2 other verification tools: VeriFast and Goto-transcoder (ESBMC) Making compiletest more maintainable: reworking directive handling 2 detailed updates available. Comment by @jiejouyoux posted on 2025-02-19: Update (2025-02-19): To make it easier to follow bootstrap's test flow going into running compiletest-managed test suites, I've added more tracing to bootstrap in <https://github.com/rust-lang/rust/pull/137080>. There are some prerequisite cleanups/improvements that I'm working on first to make it easier to read bootstrap + compiletest's codebase for reference: <https://github.com/rust-lang/rust/pull/137224>, <https://github.com/rust-lang/rust/pull/136474>, <https://github.com/rust-lang/rust/pull/136542> I'm thinking for the prototype I'm going to experiment with a branch off of rust-lang/rust instead of completely separately, so I can experiment under the context of bootstrap and tests that we actually have, instead of trying to experiment with it in a complete vacuum (esp. with respect to staging and dependency licenses). Onur is working on stabilize stage management for rustc tools #137215 which I will wait on to match ToolStd behavior as used by compiletest. Comment by @jiejouyoux posted on 2025-02-27: Update (2025-02-27): Cleanups still waiting to be merged (some PRs are blocked on changes from others making this slow). Metrics Initiative 2 detailed updates available. Comment by @yaahc posted on 2025-02-25: I'm very excited to see that this got accepted as a project goal ☐ ☐ Let me go ahead and start by giving an initial status update of where I'm at right now. We've already landed the initial implementation for configuring the directory where metrics should be stored which also acts as the enable flag for a default set of metrics, right now that includes ICE reports and unstable feature usage metrics Implemented basic unstable feature usage metrics, which currently dumps a json file per crate that is compiled showing which metrics are enabled. (example below) Implemented rust-lang/rust/src/tools/features-status-dump which dumps the status information for all unstable, stable, and removed features as a json file setup an extremely minimal initial Proof of Concept implementation locally on my laptop using InfluxDB 3.0 Alpha and Graphana (image below) I have a small program I wrote that converts the json files into influxdb's line protocol for both the usage info and the status info (snippets shown below) The timestamps are made up, since they all need to be distinct or else influxdb will treat them all as updates to the same row, I'd like to preserve this information from when the metrics were originally dumped, either in the json or by changing rustc to dump via influxdb's line format directly, or some equivalent protocol. (note this is probably only necessary for the usage metrics, but for the status metrics I'd have to change the line format schema from the example below to avoid the same problem, this has to do with how influxdb treats tags vs fields) I gathered a minimal dataset by compiling rustc with RUSTFLAGS_NOT_BOOTSTRAP="-Zmetrics-dir=\$HOME/tmp/metrics" ./x build --stage 2 and ./x run src/tools/features-status-dump/, save the output to the filesystem, and convert the output to the line protocol with the aforementioned program Write the two resulting files to influxdb I then setup the table two different ways, once by directly querying the database using influxdb's cli (query shown below), then again by trying to setup an equivalent query in graphana (there's definitely some kinks to work out here, I'm not an expert on graphana by any means.) from unstable_feature_usage_metrics-rustc_hir-3bc1eef297abaa83.json {"lib_features":[{"symbol":"variant_count"}],"lang_features":[{"symbol":"associated_type_defaults","since":null}, {"symbol":"closure_track_caller","since":null}, {"symbol":"let_chains","since":null}, {"symbol":"never_type","since":null}, {"symbol":"rustc_attrs","since":null}]} Snippet of unstable feature usage metrics post conversion to line protocol featureUsage,crateID="bc8fb5c22ba7eba3" feature="let_chains" 1739997597429030911 featureUsage,crateID="439ccecea0122a52" feature="assert_matches" 1739997597429867374 featureUsage,crateID="439ccecea0122a52" feature="extract_if" 1739997597429870052 featureUsage,crateID="439ccecea0122a52"

```
feature="iter_intersperse" 1739997597429870855 featureUsage,crateID="439ccecea0122a52" feature="box_patterns" 1739997597429871639
Snippet of feature status metrics post conversion to line protocol featureStatus,kind=lang
status="unstable",since="1.5.0",has_gate_test=false,file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/unstable.rs",line=228,name="omit_gdb_pretty_printer_section" 1739478396884006508
featureStatus,kind=lang status="accepted",since="1.83.0",has_gate_test=false,tracking_issue="123742",file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/accepted.rs",line=197,name="expr_fragment_specifier_2024" 1739478396884040564
featureStatus,kind=lang status="accepted",since="1.0.0",has_gate_test=false,file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/accepted.rs",line=72,name="associated_types" 1739478396884042777 featureStatus,kind=lang
status="unstable",since="1.79.0",has_gate_test=false,tracking_issue="123646",file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/unstable.rs",line=232,name="pattern_types" 1739478396884043914 featureStatus,kind=lang
status="accepted",since="1.27.0",has_gate_test=false,tracking_issue="48848",file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/accepted.rs",line=223,name="generic_param_attrs" 1739478396884045054 featureStatus,kind=lang
status="removed",since="1.81.0",has_gate_test=false,tracking_issue="83788",file="/home/jlusby/git/rust-
lang/rust/compiler/rustc_feature/src/removed.rs",line=245,name="wasm_abi" 1739478396884046179 Run with influxdb3 query --
database=unstable-feature-metrics --file query.sql SELECT COUNT(*) TotalCount, "featureStatus".name FROM "featureStatus" INNER JOIN
"featureUsage" ON "featureUsage".feature = "featureStatus".name GROUP BY "featureStatus".name ORDER BY TotalCount DESC Comment by
@yaahc posted on 2025-02-25: My next step is to revisit the output format, currently a direct json serialization of the data as it is represented
internally within the compiler. This is already proven to be inadequate by personal experience, given the need for additional ad-hoc conversion
into another format with faked timestamp data that wasn't present in the original dump, and by conversation with @badboy (Jan-Erik), where he
recommended we explicitly avoid ad-hoc definitions of telemetry schemas which can lead to difficult to manage chaos. I'm currently evaluating
what options are available to me, such as a custom system built around influxdb's line format, or opentelemetry's metrics API. Either way I want
to use firefox's telemetry system as inspiration / a basis for requirements when evaluating the output format options Relevant notes from my
conversation w/ Jan-Erik firefox telemetry starts w/ the question, what is it we want to answer by collecting this data? has to be explicitly noted
by whoever added new telemetry, there's a whole process around adding new telemetry defining metric description owner term limit (expires
automatically, needs manual extension) data stewards do data review checks that telemetry makes sense check that everything adheres to
standards can have downside of increasing overhead to add metrics helps w/ tooling, we can show all this info as documentation schema is
generated from definitions Model coherence in a-mir-formality 1 detailed update available. Comment by @nikomatsakis posted on 2025-02-26: I
have established some regular "office hour" time slots where I am available on jitsi. We landed a few minor PRs and improvements to the parser
and oli-obk and tif have been working on modeling of effects for the const generics work. I'm planning to start digging more into the modeling of
coherence now that the basic merging is done. Next-generation trait solver 1 detailed update available. Comment by @lcnr posted on
2025-02-27: We've stabilized -Znext-solver=coherence in version 1.84 and started to track the remaining issues in a project board. Fixing the
opaque types issue breaking wg-grammar is difficult and requires a more in-depth change for which there is now an accepted Types MCP. This
likely also unblocks the TAIT stabilization using the old solver. While waiting on the MCP I've started to look into the errors when compiling
nalgebra. @lqd minimized the failures. They have been caused by insufficiencies in our cycle handling. With
```

<https://github.com/rust-lang/rust/pull/136824> and <https://github.com/rust-lang/rust/pull/137314> cycles should now be fully supported. We also fully triaged all UI tests with the new solver enabled with @compiler-errors, @BoxyUwU, and myself fixing multiple less involved issues. Nightly support for ergonomic SIMD multiversioning 1 detailed update available. Comment by @veluca93 posted on 2025-02-25: Key developments: Further discussions on implementation details of the three major proposed ways forward. Requested a design meeting in <https://github.com/rust-lang/lang-team/issues/309>. Null and enum-discriminant runtime checks in debug builds 1 detailed update available. Comment by @1c3t3a posted on 2025-02-19: Status update: The null-checks landed in rust#134424. Next up are the enum discriminant checks. Optimizing Clippy & linting 1 detailed update available. Comment by @blyxyas posted on 2025-02-26: As a monthly update (previously posted on Zulip): We have some big progress! rust-clippy#13821 has been open and is currently being reviewed. It moves the MSRV logic out of lint-individualistic attribute extraction and into Clippy-wide MSRV (with a very good optimization, taking into account that only a small percentage of crates use. A Clippy-exclusive benchmarker has arrived, powered by the existing lintcheck infrastructure and perf. So it's compatible with flamegraph and other such tools. rust-clippy#14194 We can later expand this into CI or a dedicated bot. As you probably know, rust#125116 has been merged, just as a reminder of how that big goal was slayed like a dragon :dragon:. We now know what functions to optimize (or, at least have a basic knowledge of where Clippy spends its time). As some future functionality, I'd love to have a functionality to build cargo and rust with debug symbols and hook it up to Clippy, but that may be harder. It's not perfect, but it's a good start! Prepare const traits for stabilization No detailed updates available. Promoting Parallel Front End No detailed updates available. Publish first rust-lang-owned release of "FLS" 1 detailed update available. Comment by @JoelMarcey posted on 2025-02-24: Last week at the Safety Critical Rust Consortium meeting in London, Ferrous systems publicly announced to consortium members that they have committed to contributing the FLS to the Rust Project. We are finalizing the details of that process, but we can start FLS integration testing in parallel, in anticipation. Research: How to achieve safety when linking separately compiled code 2 detailed updates available. Comment by @m-ou-se posted on 2025-02-28: We started the collaboration with the Delft University of Technology. We assembled a small research team with a professor and a MSc student who will be working on this as part of his MSc thesis. We meet weekly in person. The project kicked off two weeks ago and is now in the literature research phase. Comment by @m-ou-se posted on 2025-02-28: And related to this, someone else is working on an implementation of my #[export] rfc: <https://github.com/rust-lang/rust/pull/134767> This will hopefully provide meaningful input for the research project. Run the 2025H1 project goal program 1 detailed update available. Comment by @nikomatsakis posted on 2025-02-19: Update: We are running behind schedule, but we are up and running nonetheless! Bear with us. The goals RFC finished FCP on Feb 18. The new project goals team has been approved and we've updated the tracking issues for the new milestone. Project goal owners are encouraged to update the issue body to reflect the actual tasks as they go with github checkboxes or other notation as described here. We're going to start pinging for our first status update soon! Rust Vision Document 1 detailed update available. Comment by @nikomatsakis posted on 2025-02-19: Update so far: I put out a call for volunteers on Zulip (hackmd) and a number of folks responded. We had an initial meeting on Jan 30 (notes here). We have created a Zulip stream for this project goal (vision-doc-2025) and I also did some experimenting in the <https://github.com/nikomatsakis/farsight> repository for what the table of contents might look like. Next milestone is to layout a plan. We are somewhat behind schedule but not impossibly so! rustc-perf improvements 1 detailed update available. Comment by @davidtwco posted on 2025-02-19: An initial update on what we've been up to and some background: This goal is submitted on behalf of the Rust team at Arm, but primarily worked on by @Jamesbarford. Anyone interested can always contact me for updates and I'll keep this issue up-to-date. We've scheduled a regular call with @Kobzol to discuss the constraints and requirements of any changes to

rust-perf (see the t-infra calendar) and have drafted a document describing a proposed high-level architecture for the service following our changes. This has been shared in the #project-goals/2025h1/rustc-perf-improvements Zulip channel to collect feedback. Once we've reached an agreement on the high-level architecture, we'll prepare a more detailed plan with details like proposed changes to the database schema, before proceeding with the implementation. Scalable Polonius support on nightly 1 detailed update available. Comment by @lqd posted on 2025-01-31: Key developments from this month: @amandasystems has continued working on the Sisyphean <https://github.com/rust-lang/rust/pull/130227> and has made progress on rewriting type tests, diagnostics issues, fixing bugs, kept up with changes on master, and more. big thanks to @jackh726 and @matthewjasper on reviews: with their help, all the PRs from the previous update have landed on nightly. I've opened a couple of PRs on the analysis itself (<https://github.com/rust-lang/rust/pull/135290>, <https://github.com/rust-lang/rust/pull/136299>) as well as a few cleanups. With these, there are only around 4 failing tests that still need investigation, and 8-10 diagnostics differences to iron out. This is my current focus, but we'll also need to expand test coverage. I've also opened a handful of PRs gradually expanding the polonius MIR dump with visualizations. I'll next add the interactive "tool" I've been using to help debug the test failures. on organization and collaboration: we've met with one of Amanda's students for a possible Master's thesis on the more engineer-y side of polonius (perf <3) and have also discussed, with @ralfjung's team, the related topic of modeling the borrowck in a-mir-formality Secure quorum-based cryptographic verification and mirroring for crates.io No detailed updates available. SVE and SME on AArch64 1 detailed update available. Comment by @davidtwco posted on 2025-02-19: An initial update on what we've been up to and some background: This goal is submitted on behalf of the Rust team at Arm, but primarily worked on by myself (@davidtwco) and @JamieCunliffe. Anyone interested can always contact me for updates and I'll keep this issue up-to-date. @JamieCunliffe been working on supporting Arm's scalable vector extension (SVE) for a couple years - primarily in rust-lang/rfcs#3268 and its implementation rust-lang/rust#118917. Through this work, we've discovered other changes to the language necessary to be able to support these types without special cases in the type system, which we're also working on (see below). Jamie is still resolving feedback on this RFC and its implementation, and keeping it rebased. We hope that it can be landed experimentally now that there's a feasible path to remove the special cases in the type system (see below). The next steps for this RFC and implementation are.. ..to continue to respond to feedback on the RFC and implementation. I've (@davidtwco) been working on rust-lang/rfcs#3729 which improves Rust's support for exotically sized types, and would allow scalable vectors to be represented in the type system without special cases. We've had two design meetings with the language team about the RFC and had a broadly positive reception. There is a non-strict dependency on const traits (rust-lang/rfcs#3762) which has created uncertainty as to whether this RFC could be accepted without the specifics of const traits being nailed down. I've been working on implementing the RFC: an initial implementation of the non-const traits has been completed and adding the const traits is in-progress. The language team have indicated interest in seeing this land experimentally, but this will depend on whether the implementors of const traits are okay with this, as it would add to the work they need to do to make any syntactic changes requested by the language team in rust-lang/rfcs#3762. I'm continuing to respond to feedback on the RFC, but as this has largely trailed off, the next steps for this RFC are.. ..for the language team to decide to accept, reject, or request further changes to the RFC. ..for progress on the implementation to continue. Unsafe Fields 1 detailed update available. Comment by @jswrenn posted on 2025-02-26: Key developments: In a Feb 19 Lang Team Design Meeting, we reached consensus that the MVP for unsafe fields should be limited to additive invariants. Use annotate-snippets for rustc diagnostic output No detailed updates available.

- [The Rust Programming Language Blog: Rust participates in Google Summer of Code 2025](#) (2025/03/03 00:00)

We are happy to announce that the Rust Project will again be participating in Google Summer of Code (GSoc) 2025, same as last year. If you're

not eligible or interested in participating in GSoC, then most of this post likely isn't relevant to you; if you are, this should contain some useful information and links. Google Summer of Code (GSoC) is an annual global program organized by Google that aims to bring new contributors to the world of open-source. The program pairs organizations (such as the Rust Project) with contributors (usually students), with the goal of helping the participants make meaningful open-source contributions under the guidance of experienced mentors. The organizations that have been accepted into the program have been announced by Google. The GSoC applicants now have several weeks to discuss project ideas with mentors. Later, they will send project proposals for the projects that they found the most interesting. If their project proposal is accepted, they will embark on a several month journey during which they will try to complete their proposed project under the guidance of an assigned mentor. We have prepared a list of project ideas that can serve as inspiration for potential GSoC contributors that would like to send a project proposal to the Rust organization. However, applicants can also come up with their own project ideas. You can discuss project ideas or try to find mentors in the #gsoc Zulip stream. We have also prepared a proposal guide that should help you with preparing your project proposals. You can start discussing the project ideas with Rust Project mentors and maintainers immediately, but you might want to keep the following important dates in mind: The project proposal application period starts on March 24, 2025. From that date you can submit project proposals into the GSoC dashboard. The project proposal application period ends on April 8, 2025 at 18:00 UTC. Take note of that deadline, as there will be no extensions! If you are interested in contributing to the Rust Project, we encourage you to check out our project idea list and send us a GSoC project proposal! Of course, you are also free to discuss these projects and/or try to move them forward even if you do not intend to (or cannot) participate in GSoC. We welcome all contributors to Rust, as there is always enough work to do. Last year was our first time participating in GSoC, and it was a success! This year we are very excited to participate again. We hope that participants in the program can improve their skills, but also would love for this to bring new contributors to the Project and increase the awareness of Rust in general. Like last year, we expect to publish blog posts in the future with updates about our participation in the program.

- [The Rust Programming Language Blog: Announcing Rustup 1.28.0](#) (2025/03/02 00:00)

The rustup team is happy to announce the release of rustup version 1.28.0. Rustup is the recommended tool to install Rust, a programming language that is empowering everyone to build reliable and efficient software. What's new in rustup 1.28.0 This new release of rustup has been a long time in the making and comes with substantial changes. Before digging into the details, it is worth mentioning that Chris Denton has joined the team. Chris has a lot of experience contributing to Windows-related parts of the Rust Project -- expertise we were previously lacking -- so we're happy to have him on board to help address Windows-specific issues. The following improvements might require changes to how you use rustup: rustup will no longer automatically install the active toolchain if it is not installed. To ensure its installation, run `rustup toolchain install` with no arguments. The following command installs the active toolchain both before and after this change: `rustup show active-toolchain || rustup toolchain install # Or, on older versions of PowerShell: rustup show active-toolchain; if ($LASTEXITCODE -ne 0) { rustup toolchain install }` Installing a host-incompatible toolchain via `rustup toolchain install` or `rustup default` will now be rejected unless you explicitly add the `--force-non-host` flag. Rustup now officially supports the following host platforms: `aarch64-pc-windows-msvc loongarch64-unknown-linux-musl` This release also comes with various quality-of-life improvements, to name a few: `rustup show`'s output format has been cleaned up, making it easier to find out about your toolchains' status. `rustup doc` now accepts a flag and a topic at the same time, enabling quick navigation to specific parts of more books. `rustup`'s `remove` subcommands now support more aliases such as `rm` and `del`. Basic support for `nushell` has been added. We have additionally made the following internal changes: The default download backend has been changed from `reqwest` with `native-tls` to `reqwest` with

rustls. RUSTUP_USE_CURL and RUSTUP_USE_RUSTLS can still be used to change the download backend if the new backend causes issues. If issues do happen, please let us know. The default backend now uses rustls-platform-verifier to verify server certificates, taking advantage of the platform's certificate store on platforms that support it. When creating proxy links, rustup will now try symlinks first and fall back to hardlinks, as opposed to trying hardlinks first. A new RUSTUP_LOG environment variable can be used to control tracing-based logging in rustup binaries. See the dev guide for more details. Finally, there are some notable changes to our official website as well: The overall design of the website has been updated to better align with the Rust Project's branding. It is now possible to download the prebuilt rustup-init.sh installer for the aarch64-pc-windows-msvc host platform via <https://win.rustup.rs/aarch64>. Further details are available in the changelog! How to update If you have a previous version of rustup installed, getting rustup 1.28.0 is as easy as stopping any programs which may be using Rustup (e.g. closing your IDE) and running: `$ rustup self update` Rustup will also automatically update itself at the end of a normal toolchain update: `$ rustup update` If you don't have it already, you can get rustup from the appropriate page on our website. Rustup's documentation is also available in the rustup book. Caveats Rustup releases can come with problems not caused by rustup itself but just due to having a new release. As such, we recommend paying attention to the following potential issues in particular: Anti-malware scanners might be blocking rustup or stopping it from creating or copying files (especially when installing rust-docs, since it contains many small files). In your CI environment, rustup might fail when trying to perform a self-update. This is a known issue, and in the case where this issue does occur, we recommend applying the following workaround at the beginning of your workflow: `$ rustup set auto-self-update disable` Also, starting from 1.28.0, rustup will no longer attempt to self-update in CI environments, so this workaround should not be necessary in the future. These issues should be automatically resolved in a few weeks when the anti-malware scanners are updated to be aware of the new rustup release, and the hosted version is updated across all CI runners. Thanks Thanks again to all the contributors who made rustup 1.28.0 possible!

- [Don Marti: two open source stories](#) (2025/03/02 00:00)

First, I know that pretty much everyone is (understandably) freaking out about stuff that is getting worse, but I just wanted to share some good news in the form of an old-fashioned open-source success story. I'm a fairly boring person and developed most of my software habits in the late 1990s and early 2000s, so it's pretty rare that I actually hit a bug. But so far this blog has hit two: one browser compatibility issue and this one. The script for rebuilding when a file changes depends on the inotifywait utility, and it turned out that until recently it breaks when you ask it to watch more than 1024 files. I filed a bug A helpful developer, Jan Kratochvil, wrote a fix and put in a pull request. A bot made test packages and commented with instructions for me on how to test the fix. I commented that the new version works for me The fix just went into Fedora. Pretty damn slick. This is a great improvement over how this kind of thing used to work. I hardly had to do anything. These kids today don't know how good they have it. story number 2: why support the Linux desktop? Amazon Chime is shutting down. Did anyone use it? I get invited to a lot of video conferences, and I never got invited to an Amazon Chime meeting. Even though Amazon.com is normally really good at SaaS, this one didn't take off. What happened? It looks like Amazon Chime was an interesting example of Nassim Nicholas Taleb's intransigent minority effect. The system requirements for Amazon Chime look pretty reasonable, right? Should get 95% of the client systems out there. The number of desktop Linux users is pretty small. But if you have 20 meetings a week, at 95% compatibility you're going to average a compatibility issue every week. Even worse, the people you most want to make a good first impression on are the people whose client platform you're least likely to know. And if you do IT support for a company with 100 people organizing meetings, Amazon Chime is going to cause way too many support issues to put up with. Taleb uses the examples of kosher and halal food—only a small fraction of the population will only eat kosher or halal, but when

planning food for a large group, the most practical choice is to satisfy the minority. The minority rule will show us how it all it takes is a small number of intolerant virtuous people with skin in the game, in the form of courage, for society to function properly. Anyway, something to keep in mind in the future for anyone considering moving the support desktop Linux or support Firefox tickets to backlog. None of the successful video conferencing platforms give me any grief for my Linux/Firefox/privacy nerdery client-side setup. Bonus links Liam Proven and Thomas Claburn cover the latest web browser surveillance drama: Mozilla flamed by Firefox fans after renegeing on promises to not sell their data Mozilla doesn't sell data about you (in the way that most people think about selling data), and we don't buy data about you, he said. We changed our language because some jurisdictions define sell more broadly than most people would usually understand that word. (Don't forget to turn off advertising features in Firefox.) David Roberts interviews Mustafa Amjad and Waqas Moosa about Pakistan's solar boom. What has prompted this explosion of distributed solar is some combination of punishingly high prices for grid power and solar panels getting very, very, very cheap. A glut of Chinese overcapacity means that the price of panels in Pakistan has gone from 24 cents a watt to 10 cents a watt in just the past year or two. Distributed solar is breaking over Pakistan like a tidal wave, despite utilities and a grid that do not seem entirely prepared for it. AI and Esoteric Fascism by Baldur Bjarnason. When I first began to look into Large Language Models (LLMs) and Diffusion Model back in 2022, it quickly became obvious that much of the rhetoric around LLMs was... weird. Or, if we're being plain-spoken, much of what the executives and engineers at the organisations making these systems were saying was outright weirdo cult shit... Poll: Small business owners feel more uncertain about the future The NFIB said small business owners are feeling less confident about investing in their business due to uncertain business conditions in the coming months. It is no longer safe to move our governments and societies to US clouds by Bret Hubert. With all sorts of magic legal spells like DPIAs and DTIAs, organizations attempt to justify transferring our data and processes to the US. There is a whole industry that has been aiding and abetting this process for years. People also fool themselves that special keys and "servers in the EU" will get you "a safe space" within the American cloud. It won't. Max Murphy says Goodbye Surveillance Capitalism, Hello Surveillance Fascism Surveillance fascism thrives where corporate greed and state power merge. This came out in 2020 but worth re-reading today: Puncturing the Paradox: Group Cohesion and the Generational Myth by Harry Guild. The highest group cohesion by profession is in Marketing. This is advertising's biggest problem in a single chart. This is the monoculture. How can we possibly understand, represent and sell to an entire country when we exist in such a bubble? We like to style ourselves as free thinkers, mavericks and crazies, but the grim truth is that we're a more insular profession than farming and boast more conformists than the military. Tech continues to be political by Miriam Eric Suzanne. Maybe we should consider the beliefs and assumptions that have been built into a technology before we embrace it? But we often prefer to treat each new toy as as an abstract and unmotivated opportunity. If only the good people like ourselves would get involved early, we can surely teach everyone else to use it ethically! Important reminder at Sauropod Vertebra Picture of the Week. If you believe in "Artificial Intelligence", take five minutes to ask it about stuff you know well Because LLMs get catastrophically wrong answers on topics I know well, I do not trust them at all on topics I don't already know. And if you do trust them, I urge you to spend five minutes asking your favourite one about something you know in detail. (This is a big part of the reason I don't use LLMs for search or research. A lot of the material isn't just wrong, but reflects old, wrong, but often repeated assumptions that you need to know a lot about a field to know not to apply.) Wendy Davis covers Meta Sued Over Discriminatory College Ads. A civil rights group has sued Meta Platforms over ad algorithms that allegedly discriminate by disproportionately serving ads for for-profit colleges to Black users and ads for public colleges to white users. Situations like this are a big part of the reason why people should stop putting privacy-enhancing advertising technologies in web browsers—they mainly obfuscate discrimination and fraud.) Cities Can Cost Effectively Start Their Own Utilities Now by Kevin

Burke. Most PG&E ratepayers don't understand how much higher the rates they pay are than what it actually costs PG&E to generate and transmit the electricity to their house. When I looked into this recently I was shocked. The average PG&E electricity charge now starts at 40 cents per kilowatt hour and goes up from there. Silicon Valley Power, Santa Clara's utility company, is getting power to customers for 17 cents per kilowatt hour. Sacramento's utility company charges about the same. Pluralistic: MLMs are the mirror-world version of community organizing (04 Feb 2025) by Cory Doctorow. The rise of the far right can't be separated from the history of MLMs. Jeffrey Emanuel covers A bear case for Nvidia: competition from hardware startups, inference-heavy "reasoning" models, DeepSeek's training and inference efficiency breakthroughs, more. There are certainly many historical cases where a very important new technology changed the world, but the main winners were not the companies that seemed the most promising during the initial stages of the process. Three years on, Europe looks to Ukraine for the future of defense tech by Mike Butcher. But in order to invest in the right technology, Europe will have to look to Ukraine, because that is where future wars are being fought right now. TechCrunch recently put a call out for Ukrainian dual-use and defense tech startups to update us on what they are working on. Below is what they sent us, in their own words. Related: Ukrainian Drones Flew 500 Miles And, In A Single Strike, Damaged 5% Of Russia's Oil Refining Capacity by David Axe. (Drones are getting longer ranges and better autonomy, fast. Fossil fuel infrastructure is not getting any better protected or faster to repair. In the near future, you're only going to be in the oil or gas business if nobody who is good at ML and model airplanes has a strong objection to you being in the oil or gas business.)

- [The Mozilla Blog: An update on our Terms of Use](#) (2025/02/28 23:09)

On Wednesday we shared that we're introducing a new Terms of Use (TOU) and Privacy Notice for Firefox. Since then, we've been listening to some of our community's concerns with parts of the TOU, specifically about licensing. Our intent was just to be as clear as possible about how we make Firefox work, but in doing so we also created some confusion and concern. With that in mind, we're updating the language to more clearly reflect the limited scope of how Mozilla interacts with user data. Here's what the new language will say: You give Mozilla the rights necessary to operate Firefox. This includes processing your data as we describe in the Firefox Privacy Notice. It also includes a nonexclusive, royalty-free, worldwide license for the purpose of doing as you request with the content you input in Firefox. This does not give Mozilla any ownership in that content. In addition, we've removed the reference to the Acceptable Use Policy because it seems to be causing more confusion than clarity. Privacy FAQ We also updated our Privacy FAQ to better address legal minutia around terms like "sells." While we're not reverting the FAQ, we want to provide more detail about why we made the change in the first place. TL;DR Mozilla doesn't sell data about you (in the way that most people think about "selling data"), and we don't buy data about you. We changed our language because some jurisdictions define "sell" more broadly than most people would usually understand that word. Firefox has built-in privacy and security features, plus options that let you fine-tune your data settings. The reason we've stepped away from making blanket claims that "We never sell your data" is because, in some places, the LEGAL definition of "sale of data" is broad and evolving. As an example, the California Consumer Privacy Act (CCPA) defines "sale" as the "selling, renting, releasing, disclosing, disseminating, making available, transferring, or otherwise communicating orally, in writing, or by electronic or other means, a consumer's personal information by [a] business to another business or a third party" in exchange for "monetary" or "other valuable consideration." Similar privacy laws exist in other US states, including in Virginia and Colorado. And that's a good thing — Mozilla has long been a supporter of data privacy laws that empower people — but the competing interpretations of do-not-sell requirements does leave many businesses uncertain about their exact obligations and whether or not they're considered to be "selling data." In order to make Firefox commercially viable, there are a number of places where we collect and share some data with our partners, including our optional ads on

New Tab and providing sponsored suggestions in the search bar. We set all of this out in our Privacy Notice. Whenever we share data with our partners, we put a lot of work into making sure that the data that we share is stripped of potentially identifying information, or shared only in the aggregate, or is put through our privacy preserving technologies (like OHTTP). We're continuing to make sure that Firefox provides you with sensible default settings that you can review during onboarding or adjust at any time. The post [An update on our Terms of Use](#) appeared first on [The Mozilla Blog](#).

- [The Mozilla Blog: Introducing a terms of use and updated privacy notice for Firefox](#) (2025/02/26 17:00)

UPDATE: We've seen a little confusion about the language regarding licenses, so we want to clear that up. We need a license to allow us to make some of the basic functionality of Firefox possible. Without it, we couldn't use information typed into Firefox, for example. It does NOT give us ownership of your data or a right to use it for anything other than what is described in the Privacy Notice. We're introducing a Terms of Use for Firefox for the first time, along with an updated Privacy Notice. Why now? Although we've historically relied on our open source license for Firefox and public commitments to you, we are building in a much different technology landscape today. We want to make these commitments abundantly clear and accessible. While for most companies these are pretty standard legal documents, at Mozilla we look at things differently. We lay out our principles in our Manifesto: Your security and privacy on the internet are fundamental and must not be treated as optional. You deserve the ability to shape the internet and your own experiences on it — including how your data is used. We believe that practicing transparency creates accountability and trust. Firefox will always continue to add new features, improve existing ones, and test new ideas. We remain dedicated to making Firefox open source, but we believe that doing so along with an official Terms of Use will give you more transparency over your rights and permissions as you use Firefox. And actually asking you to acknowledge it is an important step, so we're making it a part of the standard product experience starting in early March for new users and later this year for existing ones. In addition to the Terms of Use, we are providing a more detailed explanation of our data practices in our updated Privacy Notice. We tried to make these easy to read and understand — there shouldn't be any surprises in how we operate or how our product works. We have always prioritized user privacy and will continue to do so. We use data to make Firefox functional and sustainable, improve your experience, and keep you safe. Some optional Firefox features or services may require us to collect additional data to make them work, and when they do, your privacy remains our priority. We intend to be clear about what data we collect and how we use it. Finally, you are in control. We've set responsible defaults that you can review during onboarding or adjust in your settings at any time: These simple, yet powerful tools let you manage your data the way you want. You deserve that choice, and we hope all technology companies will start to provide it. It's standard operating procedure for us. The post [Introducing a terms of use and updated privacy notice for Firefox](#) appeared first on [The Mozilla Blog](#).

- [Firefox Developer Experience: WebDriver BiDi Becomes the Default for Cypress with Firefox](#) (2025/02/26 13:06)

At Mozilla, we know that creating compelling web experiences depends on the ability to test sites across all browsers. This means that enabling excellent cross-browser automation is a key part of how we deliver our vision for the open web. Today, we are excited to share a significant milestone: Cypress will use WebDriver BiDi as its default automation protocol for Firefox 135 and later, starting with Cypress 14.1.0. Why WebDriver BiDi? WebDriver BiDi is a cross-browser automation protocol undergoing standardisation at W3C. It is designed as an update to the existing WebDriver standard that provides the two-way communication and rich feature set required for modern testing tools. Prior to WebDriver BiDi, browser testing tools such as Cypress often used the Chrome Devtools Protocol (CDP) to gain access to functionality not available in WebDriver. However this comes with a significant disadvantage: as the name suggests the Chrome Devtools Protocol is only available in

Chromium-based browsers, significantly limiting its suitability for cross-browser testing. For more information on the relationship between CDP and WebDriver BiDi, see our posts on the hacks blog. To help bridge this gap, back in 2019 Firefox started to implement a subset of CDP directly in Gecko. This has been used by Cypress as part of their Firefox backend since version 4.0. However with WebDriver BiDi now offering a richer feature set and undergoing continual improvements, we believe that switching to the standard protocol will better serve the needs of Cypress users. If you use Cypress and run into difficulties, it is temporarily possible to switch back to the CDP backend by setting the environment variable `FORCE_FIREFOX_CDP=1`. If you need to do this please file a bug. Cypress are planning to deprecate support for this in version 15 and the CDP support in Firefox will be removed after the next ESR release. You can also check out the Cypress team's blog post about the change. [Close Collaboration for a Seamless Transition](#) This achievement would not have been possible without the close collaboration between Mozilla and Cypress. From initial planning to implementation and testing, we worked together to ensure WebDriver BiDi meets the needs of Cypress users. This underscores our commitment to providing developers with powerful and reliable tools for cross-browser testing. [CDP Support Ends with Firefox 141](#) As we continue transitioning to WebDriver BiDi, support for the experimental Chrome DevTools Protocol in Firefox will officially end with Firefox 141, set for release on July 22nd. The 140 ESR release will continue to have CDP support for its full support lifecycle. Removing the CDP implementation will allow us to focus our efforts entirely on improving WebDriver BiDi. [What's Next?](#) We believe that WebDriver BiDi provides a solid foundation for the future of browser testing and automation. We will continue to work closely with automation tools to ensure that it meets their needs for a robust, feature-rich, cross-browser automation protocol. Stay tuned for further updates as we continue improving the Firefox automation experience. The future of automation is here, and WebDriver BiDi is leading the way!

- [The Servo Blog: Servo Security Report: findings and solutions](#) (2025/02/26 00:00)

The Servo project has received several grants from NLnet Foundation, and as part of these grants, NLnet offers different support services. These services include security audits from Radically Open Security. In one of our projects with NLnet, we were working on adding support for CSS floats and tables in Servo. Once the project was completed, we reached out to Radically Open Security to run a security audit. The focus of the audit was in the code related to that project, so the main components investigated were the CSS code paths in the layout engine and Stylo. As part of this audit, four vulnerabilities were identified: ID Type Description Threat level CLN-009 Third-Party Library Vulnerability Servo uses an outdated version of the time crate that is vulnerable to a known issue. Moderate CLN-004 Arithmetic underflow By calling methods on a TableBuilder object in a specific order, an integer underflow can be triggered, followed by an attempted out-of-bounds access, which is caught by Rust, resulting in a panic. Low CLN-002 Arithmetic underflow An arithmetic underflow condition is currently impossible to trigger, but may become accessible as its surrounding logic evolves. N/A CLN-007 Unguarded casting Casting from an unsigned platform pointer-width integer into a fixed-width signed integer may result in erroneous layouts. N/A If you want to know more details you can read the full report. The first issue (CLN-009) was related to a known vulnerability in version 0.1 of the time crate that Servo depended on (RUSTSEC-2020-0071). We had removed this in most of Servo, but there was one remaining dependency in WebRender. We have since addressed this vulnerability by removing it from the version of WebRender used in Servo (@mrobinson, #35325), and we will also try to upstream our changes so that Firefox is unaffected. We have also fixed the second (CLN-004) and third (CLN-002) issues by making the affected code more robust (@Loirooriol, #34247, #35437), in the event web content can trigger them. The fourth issue (CLN-007) is not urgent, since the values are constrained elsewhere and it would otherwise only affect layout correctness and not security, but it's worth fixing at some point too. Servo has long been highlighted for being a memory-safe and concurrency-safe web rendering engine, thanks to the guarantees provided by the Rust programming language, including its ownership system,

borrow checker, and built-in data structures that enable safe concurrent programming. These features help prevent memory and concurrency vulnerabilities, such as use-after-free bugs and data races. We find it promising that this security audit, although smaller and of limited scope, identified no severe vulnerabilities, especially none of that nature, and that we were able to address any vulnerabilities identified. This was a positive experience for Servo and the web, and we're keen to explore more security auditing for Servo in the future. Thanks to the folks at Radically Open Security for their work on this audit, and NLnet Foundation for continuously supporting the Servo project.

- [Niko Matsakis: View types redux and abstract fields](#) (2025/02/25 16:04)

A few years back I proposed view types as an extension to Rust's type system to let us address the problem of (false) inter-procedural borrow conflicts. The basic idea is to introduce a "view type" `{f1, f2} Type1`, meaning "an instance of Type where you can only access the fields f1 or f2". The main purpose is to let you write function signatures like `& {f1, f2} self` or `&mut {f1, f2} self` that define what fields a given type might access. I was thinking about this idea again and I wanted to try and explore it a bit more deeply, to see how it could actually work, and to address the common question of how to have places in types without exposing the names of private fields. Example: the Data type

The Data type is going to be our running example. The Data type collects experiments, each of which has a name and a set of f32 values. In addition to the experimental data, it has a counter, successful, which indicates how many measurements were successful.

```
struct Data {
    experiments: HashMap<String, Vec<f32>>,
    successful: u32,
}
```

There are some helper functions you can use to iterate over the list of experiments and read their data. All of these return data borrowed from self. Today in Rust I would typically leverage lifetime elision, where the `&` in the return type is automatically linked to the `&self` argument:

```
impl Data {
    pub fn experiment_names( &self, ) -> impl Iterator<Item = &String> {
        self.experiments.keys()
    }
    pub fn for_experiment( &self, experiment: &str, ) -> &[f32] {
        experiments.get(experiment).unwrap_or(&[])
    }
}
```

Tracking successful experiments

Now imagine that Data has methods for reading and modifying the counter of successful experiments:

```
impl Data {
    pub fn successful(&self) -> u32 { self.successful }
    pub fn add_successful(&mut self) { self.successful += 1; }
}
```

Today, "aggregate" types like Data present a composition hazard

The Data type as presented thus far is pretty sensible, but it can actually be a pain to use. Suppose you wanted to iterate over the experiments, analyze their data, and adjust the successful counter as a result. You might try writing the following:

```
fn count_successful_experiments(data: &mut Data) {
    for n in data.experiment_names() {
        if is_successful(data.for_experiment(n)) {
            data.add_successful();
            // ERROR: data is borrowed here
        }
    }
}
```

Experienced Rustaceans are likely shaking their head at this point—in fact, the previous code will not compile. What's wrong? Well, the problem is that `experiment_names` returns data borrowed from self which then persists for the duration of the loop. Invoking `add_successful` then requires an `&mut Data` argument, which causes a conflict. The compiler is indeed flagging a reasonable concern here. The risk is that `add_successful` could mutate the experiments map while `experiment_names` is still iterating over it. Now, we as code authors know that this is unlikely — but let's be honest, it may be unlikely now, but it's not impossible that as Data evolves somebody might add some kind of logic into `add_successful` that would mutate the experiments map. This is precisely the kind of subtle interdependency that can make an innocuous "but it's just one line!" PR cause a massive security breach. That's all well and good, but it's also very annoying that I can't write this code. Using view types to flag what is happening

The right fix here is to have a way to express what fields may be accessed in the type system. If we do this, then we can get the code to compile today and prevent future PRs from introducing bugs. This is hard to do with Rust's current system, though, as types do not have any way of talking about fields, only spans of execution-time ("lifetimes"). With view types, though, we can change the signature from `&self` to `&{experiments} self`. Just as `&self` is shorthand for `self: &Data`, this is actually shorthand for `self: & {experiments} Data`.

```
impl Data {
    pub fn experiment_names( & {experiments} self, ) -> impl Iterator<Item =
```

```
&String> { self.experiments.keys() } pub fn for_experiment( & {experiments} self, experiment: &str, ) -> &[f32] {
self.experiments.get(experiment).unwrap_or(&[]) } } We would also modify the add_successful method to flag what field it needs: impl Data {
pub fn add_successful( self: &mut {successful} Self, ) { self.successful += 1; } } Getting a bit more formal The idea of this post was to sketch
out how view types could work in a slightly more detailed way. The basic idea is to extend Rust's type grammar with a new type... T = &'a mut? T
| [T] | Struct<...> | ... | {field-list} T // ← view types We would also have some kind of expression for defining a view onto a place. This would be
a place expression. For now I will write E = {f1, f2} E to define this expression, but that's obviously ambiguous with Rust blocks. So for example
you could write... let mut x: (String, String) = (String::new(), String::new()); let p: &{0} (String, String) = & {0} x; let q: &mut {1} (String, String)
= &mut {1} x; ...to get a reference p that can only access the field 0 of the tuple and a reference q that can only access field 1. Note the
difference between &{0}x, which creates a reference to the entire tuple but with limited access, and &x.0, which creates a reference to the field
itself. Both have their place. Checking field accesses against view types Consider this function from our example: impl Data { pub fn
add_successful( self: &mut {successful} Self, ) { self.successful += 1; } } How would we type check the self.successful += 1 statement? Today,
without view types, typing an expression like self.successful begins by getting the type of self, which is something like &mut Data. We then
“auto-deref”, looking for the struct type within. That would bring us to Data, at which point we would check to see if Data defines a field
successful. To integrate view types, we have to track both the type of data being accessed and the set of allowed fields. Initially we have variable
self with type &mut {successful} Data and allow set *. The deref would bring us to {successful} Data (allow-set remains *). Traversing a view
type modifies the allow-set, so we go from * to {successful} (to be legal, every field in the view must be allowed). We now have the type Data.
We would then identify the field successful as both a member of Data and a member of the allow-set, and so this code would be successful. If
however you tried to modify a function to access a field not declared as part of its view, e.g., impl Data { pub fn add_successful( self: &mut
{successful} Self, ) { assert!(!self.experiments.is_empty()); // ← modified to include this self.successful += 1; } } the self.experiments type-
checking would now fail, because the field experiments would not be a member of the allow-set. We need to infer allow sets A more interesting
problem comes when we type-check a call to add_successful(). We had the following code: fn count_successful_experiments(data: &mut Data) {
for n in data.experiment_names() { if is_successful(data.for_experiment(n)) { data.add_successful(); // Was error, now ok. } } } Consider the call
to data.experiment_names(). In the compiler today, method lookup begins by examining data, of type &mut Data, auto-deref'ing by one step to
yield Data, and then auto-ref'ing to yield &Data. The result is this method call is desugared to a call like Data::experiment_names(&*data). With
view types, when introducing the auto-ref, we would also introduce a view operation. So we would get Data::experiment_names(& {?X} *data).
What is this {?X}? That indicates that the set of allowed fields has to be inferred. A place-set variable ?X can be inferred to a set of fields or to *
(all fields). We would integrate these place-set variables into inference, so that {?A} Ta <: {?B} Tb if ?B is a subset of ?A and Ta <: Tb (e.g., [x, y]
Foo <: [x] Foo). We would also for dropping view types from subtypes, e.g., {*} Ta <: Tb if Ta <: Tb. Place-set variables only appear as an
internal inference detail, so users can't (e.g.) write a function that is generic over a place-set, and the only kind of constraints you can get are
subset (P1 <= P2) and inclusion (f in P1). I think it should be relatively straightforward to integrate these into HIR type check inference. When
generalizing, we can replace each specific view set with a variable, just as we do for lifetimes. When we go to construct MIR, we would always
know the precise set of fields we wish to include in the view. In the case where the set of fields is * we can also omit the view from the MIR.
Abstract fields So, view types allow us to address these sorts of conflicts by making it more explicit what sets of types we are going to access, but
they introduce a new problem — does this mean that the names of our private fields become part of our interface? That seems obviously
```


undesirable. The solution is to introduce the idea of abstract2 fields. An abstract field is a kind of pretend field, one that doesn't really exist, but which you can talk about "as if" it existed. It lets us give symbolic names to data. Abstract fields would be defined as aliases for a set of fields, like `pub abstract field_name = (list-of-fields)`. An alias defines a public symbolic names for a set of fields. We could therefore define two aliases for `Data`, one for the set of experiments and one for the count of successful experiments. I think it be useful to allow these names to alias actual field names, as I think that in practice the compiler can always tell which set to use, but I would require that if there is an alias, then the abstract field is aliased to the actual field with the same name.

```
struct Data { pub abstract experiments = experiments, experiments: HashMap<String, Vec<f32>>, pub abstract successful = successful, successful: u32, }
```

Now the view types we wrote earlier (`& {experiments} self`, etc) are legal but they refer to the abstract fields and not the actual fields. Abstract fields permit refactoring. One nice property of abstract fields is that they permit refactoring. Imagine that we decide to change `Data` so that instead of storing experiments as a `Map<String, Vec<f32>>`, we put all the experimental data in one big vector and store a range of indices in the map, like `Map<String, (usize, usize)>`. We can do that no problem:

```
struct Data { pub abstract experiments = (experiment_names, experiment_data), experiment_indices: Map<String, (usize, usize)>, experiment_data: Vec<f32>, // ... }
```

We would still declare methods like `&mut {experiments} self`, but the compiler now understands that the abstract field `experiments` can be expanded to the set of private fields. Frequently asked questions

Can abstract fields be mapped to an empty set of fields? Yes, I think it should be possible to define `pub abstract foo;` to indicate the empty set of fields.

How do view types interact with traits and impls? Good question. There is no necessary interaction, we could leave view types as simply a kind of type. You might do interesting things like implement `Deref` for a view on your struct:

```
struct AugmentedData { data: Vec<u32>, summary: u32, } impl Deref for {data} AugmentedData { type Target = [u32]; fn deref(&self) -> &[u32] { // type of `self` is `&{data} AugmentedData` &self.data } }
```

OK, you don't need to integrate abstract fields with traits, but could you? Yes! And it'd be interesting. You could imagine declaring abstract fields as trait members that can appear in its interface:

```
trait Interface { abstract data1; abstract data2; fn get_data1(&{data1} self) -> u32; fn get_data2(&{data2} self) -> u32; }
```

You could then define those fields in an impl. You can even map some of them to real fields and leave some as purely abstract:

```
struct OneCounter { counter: u32, } impl Interface for OneCounter { abstract data1 = counter; abstract data2; fn get_data1(&{counter} self) -> u32 { self.counter } fn get_data2(&{data2} self) -> u32 { 0 // no fields needed } }
```

Could view types include more complex paths than just fields? Although I wouldn't want to at first, I think you could permit something like `{foo.bar}` `Baz` and then, given something like `&foo.bar`, you'd get the type `&{bar} Baz`, but I've not really thought it more deeply than that.

Can view types be involved in moves? Yes! You should be able to do something like

```
struct Strings { a: String, b: String, c: String, } fn play_games(s: Strings) { // Moves the struct `s` but only the fields `a` and `c` let t: {a, c} Strings = {a, c} s; println!("{}", s.a); // ERROR: s.a has been moved println!("{}", s.b); // OK. println!("{}", s.c); // ERROR: s.a has been moved println!("{}", t.a); // OK. println!("{}", t.b); // ERROR: no access to field `b`. println!("{}", t.c); // OK. }
```

Why did you have a subtyping rules to drop view types from sub- but not super-types? I described the view type subtyping rules as two rules: `{?A} Ta <: {?B} Tb` if `?B` is a subset of `?A` and `Ta <: Tb {*}` `Ta <: Tb` if `Ta <: Tb` In principle we could have a rule like `Ta <: {*} Tb` if `Ta <: Tb` — this rule would allow "introducing" a view type into the supertype. We may wind up needing such a rule but I didn't want it because it meant that code like this really ought to compile (using the `Strings` type from the previous question):

```
fn play_games(s: Strings) { let t: {a, c} Strings = s; // ← just `= s`, not `= {a, c} s`. }
```

I would expect this to compile because `{a, c} Strings <: {*} Strings <: Strings` but I kind of don't want it to compile. Are there other uses for abstract fields? Yes! I think abstract fields would also be useful in two other ways (though we have to stretch their definition a bit). I believe it's important for Rust to grow stronger integration with theorem provers; I don't expect these to be widely used, but for certain key libraries (stdlib,

zerocopy, maybe even tokio) it'd be great to be able to mathematically prove type safety. But mathematical proof systems often require a notion of ghost fields — basically logical state that doesn't really exist at runtime but which you can talk about in a proof. A ghost field is essentially an abstract field that is mapped to an empty set of fields and which has a type. For example you might declare a BeanCounter struct with two abstract fields (a, b) and one real field that stores their sum: `struct BeanCounter { pub abstract a: u32, pub abstract b: u32, sum: u32, // ← at runtime, we only store the sum }` then when you create BeanCounter you would specify a value for those fields. The value would perhaps be written using something like an abstract block, indicating that in fact the code within will not be executed (but must still be type checkable): `impl BeanCounter { pub fn new(a: u32, b: u32) -> Self { Self { a: abstract { a }, b: abstract { b }, sum: a + b } }` Providing abstract values is useful because it lets the theorem prover act “as if” the code was there for the purpose of checking pre- and post-conditions and other kinds of contracts. Could we use abstract fields to replace phantom data? Yes! I imagine that instead of `a: PhantomData<T>` you could do `abstract a: T`, but that would mean we'd have to have some abstract initializer. So perhaps we permit an anonymous field `abstract _: T`, in which case you wouldn't be required to provide an initializer, but you also couldn't name it in contracts. So what are all the parts to an abstract field? I would start with just the simplest form of abstract fields, which is an alias for a set of real fields. But to extend to cover ghost fields or PhantomData, you want to support the ability to declare a type for abstract fields (we could say that the default is `()`). For fields with non-`()` types, you would be expected to provide an abstract value in the struct constructor. To conveniently handle PhantomData, we could add anonymous abstract fields where no type is needed. Should we permit view types on other types? I've shown view types attached to structs and tuples. Conceivably we could permit them elsewhere, e.g., `{0} &(String, String)` might be equivalent to `&{0} (String, String)`. I don't think that's needed for now and I'd make it ill-formed, but it could be reasonable to support at some point. Conclusion This concludes my exploration through view types. The post actually changed as I wrote it — initially I expected to include place-based borrows, but it turns out we didn't really need those. I also initially expected view types to be a special case of struct types, and that indeed might simplify things, but I wound up concluding that they are a useful type constructor on their own. In particular if we want to integrate them into traits it will be necessary for them to be applied to generics and the rest.≈g In terms of next steps, I'm not sure, I want to think about this idea, but I do feel we need to address this gap in Rust, and so far view types seem like the most natural. I think what could be interesting is to prototype them in a-mir-formality as it evolves to see if there are other surprises that arise. I'm not really proposing this syntax—among other things, it is ambiguous in expression position. I'm not sure what the best syntax is, though! It's an important question, but not one I will think hard about here. ← I prefer the name ghost fields, because it's spooky, but abstract is already a reserved keyword. ←

- [The Mozilla Blog: Mozilla's approach to Manifest V3: What's different and why it matters for extension users](#) (2025/02/25 14:00)

Extensions are like apps for your browser, letting you customize and enhance your online experience. Nearly half of all Firefox users have installed at least one extension, from privacy tools to productivity boosters. To build these extensions, developers rely on a platform called WebExtensions, which provides APIs — the tools that allow extensions to interact with web pages and browser features. Right now, all major browsers — including Firefox, Chrome and Safari — are implementing the latest version of this platform, Manifest V3. But different browsers are taking different approaches, and those differences affect which extensions you can use. Firefox's approach to Manifest V3 is shaped by our mission Principle 5 of the Mozilla Manifesto states: Individuals must have the ability to shape the internet and their own experiences on it. That philosophy drives our approach to Manifest V3. More creative possibilities for developers — We've introduced a broader range of APIs, including new AI functionality that allows extensions to run offline machine learning tasks directly in the browser. Support for both Manifest V2 and V3 —

While some browsers are phasing out Manifest V2 entirely, Firefox is keeping it alongside Manifest V3. More tools for developers means more choice and innovation for users. Giving people choice and control on the internet has always been core to Mozilla. It's all about making sure users have the freedom to shape their own experiences online. No limits on your extensions with Firefox Google began phasing out Manifest V2 last year and plans to end support for extensions built on it by mid-2025. That change has real consequences: Chrome users are already losing access to uBlock Origin, one of the most popular ad blockers, because it relies on a Manifest V2 feature called `blockingWebRequest`. Google's approach replaces `blockingWebRequest` with `declarativeNetRequest`, which limits how extensions can filter content. Since APIs define what extensions can and can't do inside a browser, restricting certain APIs can limit what types of extensions are possible. Firefox, however, will continue supporting both `blockingWebRequest` and `declarativeNetRequest` — giving developers more flexibility and keeping powerful privacy tools available to users. We'll keep you updated on what's next for extensions in Firefox. In the meantime, check out addons.mozilla.org to explore thousands of ways to customize your Firefox. Download Firefox Get the browser that protects what's important The post Mozilla's approach to Manifest V3: What's different and why it matters for extension users appeared first on The Mozilla Blog.

- [Firefox Developer Experience: GeckoDriver 0.36.0 Released](#) (2025/02/25 13:49)

We are proud to announce the next major release of `geckodriver` 0.36.0. It ships with some new features and fixes to improve your `WebDriver` experience. Contributions With `geckodriver` being an open source project, we are grateful to get contributions from people outside of Mozilla: Gatlin Newhouse added support for searching the Firefox Developer Edition's default path on macOS. `Geckodriver` code is written in Rust so any web developer can contribute! Read how to setup the work environment and check the list of mentored issues for `geckodriver`. Added Support for searching the Firefox Developer Edition's default path on macOS. Implemented by Gatlin Newhouse. Ability to push a `WebExtension` archive as created from a base64 encoded string to an Android device. Added an `allowPrivateBrowsing` field for `POST /session/{session id}/moz/addon/install` to allow the installation of a `WebExtension` that is enabled in Private Browsing mode. Introduced the `--allow-system-access` command line argument for `geckodriver`, which will be required for future versions of Firefox (potentially starting with 138.0) to allow testing in the chrome context. Added support for preserving crash dumps for crash report analysis when Firefox crashes. If the `MINIDUMP_SAVE_PATH` environment variable is set to an existing folder, crash dumps will be saved accordingly. For mobile devices, the generated `minidump` files will be automatically transferred to the host machine. For more details see the documentation of how to handle crash reports. Changed Updated the type of the `x` and `y` fields of pointer move actions (mouse and touch) from integer to fractional numbers to ensure a more precise input control. Replaced `serde_yaml` with `yaml-rust` because it's no longer officially supported. The `--enable-crash-reporter` command line argument has been deprecated to prevent crash reports from being submitted to Socorro. This argument will be completely removed in the next version. Instead, use the `MINIDUMP_SAVE_PATH` environment variable to get `minidump` files saved to a specified location. Fixed Fixed route registration for `WebAuthn` commands, which were introduced in `geckodriver` 0.34.0 but mistakenly registered under `/sessions/` instead of `/session/`, causing them to be non-functional. Removed Removed the `-no-remote` command-line argument usage for Firefox, which does no longer exist. Downloads `Geckodriver` 0.36.0 can be downloaded for all supported platforms as usual from our [GitHub release page](#).

- [Mozilla Privacy Blog: Mozilla Joins Amicus Brief in Support of Law That Protects Your Private Messages](#) (2025/02/25 02:21)

Today Mozilla has joined an amicus brief in the California Supreme Court defending statutory privacy protections for messages on services such as Snapchat or Facebook. The amicus brief asks the court to overrule a lower court opinion that would significantly reduce the legal privacy protections for users of these widely used services. Mozilla is joined on the brief by the Electronic Frontier Foundation and the Center for

Democracy and Technology. Back in 1986, Congress passed a law called the Stored Communications Act (SCA) to provide privacy protections for stored electronic communications such as email. The SCA prohibits service providers from sharing private messages with the government or other third-parties without authorization. For example, it requires that the government must get a warrant to access recent communications (or at least a subpoena in other circumstances). In the years since 1986, it is fair to say we have developed many new forms of digital communication. Fortunately, the language of the SCA is sufficiently general (it uses the term “electronic communication service”) that courts have applied it to a large array of new products. Unfortunately, a California court recently narrowed the scope of the SCA. In the case of *Snap v. The Superior Court of San Diego County*, the California Court of Appeal ruled that the SCA does not protect users of Snapchat and Facebook. The court concluded that the SCA does not apply because, in addition to facilitating transmission of messages and storing backups, these companies also maintain that content for their own business purposes such as targeted advertising. If upheld, this ruling would remove the SCA’s protection not just for users of Snap and Facebook, but for many other modern forms of communication. While we may criticize some of Snap or Meta’s data practices, it would only compound the privacy harm to their users to hold that their privacy policies take them outside the scope of the SCA, with potential ramifications for the users of other services in the future. Our brief argues that this is both wrong on the law and bad policy. We hope the California Supreme Court will fix the lower court’s error and restore key statutory privacy protections to modern messaging services. The post *Mozilla Joins Amicus Brief in Support of Law That Protects Your Private Messages* appeared first on *Open Policy & Advocacy*.

- [Tantek Çelik: CSF_01: Three Steps for IndieWeb Cybersecurity](#) (2025/02/21 21:37)

Welcome to my first Cybersecurity Friday (CSF) post. Almost exactly one week ago I experienced (and had to fight & recover from) a cybersecurity incident. While that’s a much longer story, this post series is focused on sharing tips and incident learnings from an #indieweb-centric perspective. Steps for Cybersecurity Here are the top three steps in order of importance, that you should take ASAP to secure your online presence. Email MFA/2FA. Add multi-factor authentication (MFA) using an actual Authenticator application to all places where you store or check email. Some services call this second factor or two factor authentication (2FA). While checking your email security settings, verify recovery settings: Do not cross-link your emails as recovery methods for each other, and do not use a mobile/cell number for recovery at all. Domain Registrar MFA. Add MFA to your Domain Registrar(s) if you have any. Optionally disable password reset emails if possible (some registrars may allow this). Web Host MFA. Add MFA to your web hosting service(s) if you have any. This includes both website hosting and any content delivery network (CDN) services you are using for your domains. Do not use a mobile number for MFA, nor a physical/hardware key if you travel internationally. There are very good reasons to avoid doing so. I’ll blog the reasons in another post. Those are my top three recommended cybersecurity steps for protecting your internet presence. That’s it for this week. These are the bare minimum steps to take. There are many more steps you can take to strengthen your personal cybersecurity. I will leave you with this for now: Entropy is your friend in security. Glossary Glossary for various terms, phrases, and further reading on each. content delivery network https://indieweb.org/content_delivery_network cybersecurity <https://en.wikipedia.org/wiki/cybersecurity> domain registrar https://indieweb.org/domain_registrar email recovery A method for recovering a service account password via the email account associated with that account. See also: https://en.wikipedia.org/wiki/Password_notification_email entropy [https://en.wikipedia.org/wiki/Entropy_\(information_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory)) MFA / 2FA https://indieweb.org/multi-factor_authentication sometimes called Two Factor Authentication or Second Factor Authentication mobile number for MFA <https://indieweb.org/SMS#Criticism> web host https://indieweb.org/web_hosting Syndicated to: IndieNews

- [Karl Dubost: Fixing rowspan=0 on tables on WebKit](#). (2025/02/21 12:36)

Last week, I mentioned there were easy ways to fix or help the WebKit project. Find The Bug In January, looking at the FIXME: mentions on the WebKit project, I found this piece of code: `unsigned HTMLTableCellElement::rowSpan() const { // FIXME: a rowSpan equal to 0 should be allowed, and mean that the cell is to span all the remaining rows in the row group. return std::max(1u, rowSpanForBindings()); }` Searching on bugs.webkit.org, I found this bug opened by Simon Fraser on May 5, 2018: `rowspan="0"` results in different table layout than Firefox/Chrome. Would I be able to solve it? Test The Bug The first task is very simple. Understand what are the different renderings in between browsers. Simon had already created a testcase and Ahmad had created a screenshot for it showing the results of the testcase in Safari, Firefox and Chrome. This work was already done. If they had been missing, that would have been my first step. Read The Specification For having a better understanding of the issue, it is useful to read the specification related to this bug. In this case, the relevant information was in the HTML specification, where `rowspan` attribute on `td/th` elements is described. This is the text we need: The `td` and `th` elements may also have a `rowspan` content attribute specified, whose value must be a valid non-negative integer less than or equal to 65534. For this attribute, the value zero means that the cell is to span all the remaining rows in the row group. Create More Tests Let's take a normal simple table which is 3 by 3. `<table border="1">`
`<tr><td>A1</td><td>B1</td><td>C1</td></tr>` `<tr><td>A2</td><td>B2</td><td>C2</td></tr>`
`<tr><td>A3</td><td>B3</td><td>C3</td></tr>` `</table>` We might want to make the first cell overlapping the 3 rows of the tables. A way to do that is to set `rowspan="3"` because there are 3 rows. `<table border="1">` `<tr><td rowspan="3">A1</td><td>B1</td><td>C1</td></tr>`
`<tr>` `<td>B2</td><td>C2</td></tr>` `<tr>` `<td>B3</td><td>C3</td></tr>` `</table>` This will create a table were the first column will overlap the 3 rows. This is already working as expected in all rendering engines : WebKit, Gecko and Blink. So far, so good. Think About The Logic I learned from reading the specification that `rowspan` had a maximum value: 65534. My initial train of thoughts was: compute the number of rows the table. parse the value `rowspan` value when the value is 0, replace it with the number of rows. It seemed too convoluted. Would it be possible to use the maximum value for `rowspan`? The specification was saying "span all the remaining rows in the row group". I experimented with a bigger `rowspan` value than the number of rows. For example, put the value 30 on a 3 rows table. `<table border="1">` `<tr><td rowspan="30">A1</td><td>B1</td><td>C1</td></tr>` `<tr>` `<td>B2</td><td>C2</td></tr>` `<tr>` `<td>B3</td><td>C3</td></tr>` `</table>` I checked in Firefox, Chrome, and Safari. I got the same rendering. We were on the right track. Let's use the maximum value for `rowspan`. I made a test case with additional examples to be able to check in different browsers the behavior: Fixing The Code We just had to try to change the C++ code. My patch was `diff --git a/Source/WebCore/html/HTMLTableCellElement.cpp b/Source/WebCore/html/HTMLTableCellElement.cpp index 256c816acc37b..65450c01e369a 100644 --- a/Source/WebCore/html/HTMLTableCellElement.cpp +++ b/Source/WebCore/html/HTMLTableCellElement.cpp @@ -59,8 +59,14 @@ unsigned HTMLTableCellElement::colSpan() const unsigned HTMLTableCellElement::rowSpan() const { - // FIXME: a rowSpan equal to 0 should be allowed, and mean that the cell is to span all the remaining rows in the row group. - return std::max(1u, rowSpanForBindings()); + unsigned rowspanValue = rowSpanForBindings(); + // when rowspan=0, the HTML spec says it should apply to the full remaining rows. + // In https://html.spec.whatwg.org/multipage/tables.html#attr-tdth-rowspan + // > For this attribute, the value zero means that the cell is + // > to span all the remaining rows in the row group. + if (!rowSpanValue) + return maxRowspan; + return std::max(1u, rowSpanValue); } unsigned HTMLTableCellElement::rowSpanForBindings() const If rowspan was 0, just give the maximum value which is defined in HTMLTableCellElement.h. I compiled the code change and verified the results: (note for the careful reader the last table legend is wrong, it should be rowspan="3") This was fixed! A couple of tests needed to be rebaselined. I was ready to send a Pull Request for this bug. What Is Next? The fix is not yet available`

on the current version of Safari, but you can experiment it with Safari Technology Preview (STP 213 Release Notes). The biggest part of fixing the bugs is researching, testing different HTML scenario without even touching the C++ code, etc. I'm not a C++ programmer, but time to time I can find bugs that are easy enough to understand that I can fix them. I hope this makes it easier for you to understand and encourage you to look at other bugs. Note also, that it is not always necessary to fix until modifying everything. Sometimes, just creating testscases, screenshots, pointing to the right places in the specifications, creating the WPT test cases covering this bug are all super useful. PS: Doing all this work, I found also about the behavior of colspan which is interoperable (same behavior in all browsers), but which I find illogical comparing to the behavior of rowspan. Otsukare!

- [Niko Matsakis: Rust 2024 Is Coming](#) (2025/02/20 10:37)

So, a little bird told me that Rust 2024 is going to become stable today, along with Rust 1.85.0. In honor of this momentous event, I have penned a little ditty that I'd like to share with you all. Unfortunately, for those of you who remember Rust 2021's "Edition: The song", in the 3 years between Rust 2021 and now, my daughter has realized that her father is deeply uncool¹ and so I had to take this one on solo². Anyway, enjoy! Or, you know, suffer. As the case may be. Video Watch the movie embedded here, or watch it on YouTube: Lyrics In ChordPro format, for those of you who are inspired to play along. {title: Rust 2024} {subtitle: } {key: C} [Verse 1] [C] When I got functions that never return I write an exclamation point [G] But use it for an error that could never be the compiler [C] will yell at me [Verse 2] [C] We Rust designers, we want that too [C7] But we had to make a [F] change [F] That will be [Fm]better [C] Oh so much [A]better [D] in Rust Twenty [G7]Twenty [C]Four [Bridge] [Am] ... [Am] But will my program [E] build? [Am] Yes ... oh that's [D7] for sure [F] edi-tions [G] are [C] opt in [Verse 3] [C] Usually when I return an `impl Trait` everything works out fine [G] but sometimes I need a tick underscore and I don't really [C] know what that's for [Verse 4] [C] We Rust designers we do agree [C7] That was con- [F] fusing [F] But that will be [Fm]better [C] Oh so much [A]better [D] in Rust Twenty [G7]Twenty [C]Four [Bridge 2] [Am] Cargo fix will make the changes automatically [G] Oh that sure sounds great... [Am] but wait... [Am] my de-pen-denc-[E]-ies [Am] Don't worry e-[D7]ditions [F] inter [G] oper [C] ate [Verse 5] [C] Whenever I match on an ampersand T The borrow [G] propagates But where do I put the ampersand when I want to [C] copy again? [Verse 6] [C] We Rust designers, we do agree [C7] That really had to [F] change [F] That will be [Fm]better [C] Oh so much [A]better [D] in Rust Twenty [G7]Twenty [C]Four [Outro] [F] That will be [Fm]better [C] Oh so much [A]better [D] in Rust Twenty [G7]Twenty [C]Four One more time! [Half speed] [F] That will be [Fm]better [C] Oh so much [A]better [D] in Rust Twenty [G7]Twenty [C]Four It was bound to happen eventually. ← Actually, I had a plan to make this a duet with somebody who shall remain nameless (they know who they are). But I was too lame to get everything done on time. In fact, I may or may not have realized "Oh, shit, I need to finish this recording!" while in the midst of a beer with Florian Gilcher last night. Anyway, sorry, would-be-collaborator-I -was-really-looking-forward-to-playing-with! Next time! ←

- [The Rust Programming Language Blog: Announcing Rust 1.85.0 and Rust 2024](#) (2025/02/20 00:00)

The Rust team is happy to announce a new version of Rust, 1.85.0. This stabilizes the 2024 edition as well. Rust is a programming language empowering everyone to build reliable and efficient software. If you have a previous version of Rust installed via rustup, you can get 1.85.0 with: \$ rustup update stable If you don't have it already, you can get rustup from the appropriate page on our website, and check out the detailed release notes for 1.85.0. If you'd like to help us out by testing future releases, you might consider updating locally to use the beta channel (rustup default beta) or the nightly channel (rustup default nightly). Please report any bugs you might come across! What's in 1.85.0 stable Rust 2024 We are excited to announce that the Rust 2024 Edition is now stable! Editions are a mechanism for opt-in changes that may otherwise pose

a backwards compatibility risk. See the edition guide for details on how this is achieved, and detailed instructions on how to migrate. This is the largest edition we have released. The edition guide contains detailed information about each change, but as a summary, here are all the changes:

- Language RPIT lifetime capture rules — Changes the default capturing of parameters by impl Trait types when use<..> is not present.
- if let temporary scope — Changes the scope of temporaries for if let expressions.
- Tail expression temporary scope — Changes the scope of temporaries for the tail expression in a block.
- Match ergonomics reservations — Disallow some pattern combinations to avoid confusion and allow for future improvements.
- Unsafe extern blocks — extern blocks now require the unsafe keyword.
- Unsafe attributes — The export_name, link_section, and no_mangle attributes must now be marked as unsafe.
- unsafe_op_in_unsafe_fn warning — The unsafe_op_in_unsafe_fn lint now warns by default, requiring explicit unsafe { } blocks in unsafe functions.
- Disallow references to static mut — References to static mut items now generate a deny-by-default error.
- Never type fallback change — Changes to how the never type ! coerces, and changes the never_type_fallback_flowng_into_unsafe lint level to "deny".
- Macro fragment specifiers — The expr macro fragment specifier in macro_rules! macros now also matches const and _ expressions.
- Missing macro fragment specifiers — The missing_fragment_specifier lint is now a hard error, rejecting macro meta variables without a fragment specifier kind.
- gen keyword — Reserves the gen keyword in anticipation of adding generator blocks in the future.
- Reserved syntax — Reserves # "foo" # style strings and ## tokens in anticipation of changing how guarded string literals may be parsed in the future.
- Standard library Changes to the prelude — Adds Future and IntoFuture to the prelude.
- Add Intolterator for Box<[T]> — Changes how iterators work with boxed slices.
- Newly unsafe functions — std::env::set_var, std::env::remove_var, and std::os::unix::process::CommandExt::before_exec are now unsafe functions.
- Cargo Cargo: Rust-version aware resolver — Changes the default dependency resolver behavior to consider the rust-version field.
- Cargo: Table and key name consistency — Removes some outdated Cargo.toml keys.
- Cargo: Reject unused inherited default-features — Changes how default-features = false works with inherited workspace dependencies.
- Rustdoc Rustdoc combined tests — Doctests are now combined into a single executable, significantly improving performance.
- Rustdoc nested include! change — Changes to the relative path behavior of nested include! files.
- Rustfmt Rustfmt: Style edition — Introduces the concept of "style editions", which allow you to independently control the formatting edition from the Rust edition.
- Rustfmt: Formatting fixes — A large number of fixes to formatting various situations.
- Rustfmt: Raw identifier sorting — Changes to how r#foo identifiers are sorted.
- Rustfmt: Version sorting — Changes to how identifiers that contain integers are sorted.

Migrating to 2024 The guide includes migration instructions for all new features, and in general transitioning an existing project to a new edition. In many cases cargo fix can automate the necessary changes. You may even find that no changes in your code are needed at all for 2024! Note that automatic fixes via cargo fix are very conservative to avoid ever changing the semantics of your code. In many cases you may wish to keep your code the same and use the new semantics of Rust 2024; for instance, continuing to use the expr macro matcher, and ignoring the conversions of conditionals because you want the new 2024 drop order semantics. The result of cargo fix should not be considered a recommendation, just a conservative conversion that preserves behavior. Many people came together to create this edition. We'd like to thank them all for their hard work!

async closures Rust now supports asynchronous closures like async || { } which return futures when called. This works like an async fn which can also capture values from the local environment, just like the difference between regular closures and functions. This also comes with 3 analogous traits in the standard library prelude: AsyncFn, AsyncFnMut, and AsyncFnOnce. In some cases, you could already approximate this with a regular closure and an asynchronous block, like || async { }. However, the future returned by such an inner block is not able to borrow from the closure captures, but this does work with async closures: let mut vec: Vec<String> = vec![]; let closure = async || { vec.push(ready(String::from("")).await); }; It also has not been possible to

properly express higher-ranked function signatures with the Fn traits returning a Future, but you can write this with the AsyncFn traits: use `core::future::Future`; `async fn f<Fut>(_: impl for<'a> Fn(&'a u8) -> Fut) where Fut: Future<Output = ()>, { todo!() } async fn f2(_: impl for<'a> AsyncFn(&'a u8)) { todo!() } async fn main() { async fn g(_: &u8) { todo!() } f(g).await; //~^ ERROR mismatched types //~| ERROR one type is more general than the other f2(g).await; // ok! }` So async closures provide first-class solutions to both of these problems! See RFC 3668 and the stabilization report for more details. Hiding trait implementations from diagnostics The new `#[diagnostic::do_not_recommend]` attribute is a hint to the compiler to not show the annotated trait implementation as part of a diagnostic message. For library authors, this is a way to keep the compiler from making suggestions that may be unhelpful or misleading. For example: `pub trait Foo {} pub trait Bar {} impl<T: Foo> Bar for T {} struct MyType; fn main() { let _object: &dyn Bar = &MyType; } error[E0277]: the trait bound `MyType: Bar` is not satisfied --> src/main.rs:9:29 | 9 | let _object: &dyn Bar = &MyType; | ^^^^ the trait `Foo` is not implemented for `MyType` | note: required for `MyType` to implement `Bar` -> src/main.rs:4:14 | 4 | impl<T: Foo> Bar for T {} | --- ^^^ ^ ||| unsatisfied trait bound introduced here = note: required for the cast from `MyType` to `&dyn Bar` For some APIs, it might make good sense for you to implement Foo, and get Bar indirectly by that blanket implementation. For others, it might be expected that most users should implement Bar directly, so that Foo suggestion is a red herring. In that case, adding the diagnostic hint will change the error message like so: #[diagnostic::do_not_recommend] impl<T: Foo> Bar for T {} error[E0277]: the trait bound `MyType: Bar` is not satisfied --> src/main.rs:10:29 | 10 | let _object: &dyn Bar = &MyType; | ^^^^ the trait `Bar` is not implemented for `MyType` | = note: required for the cast from `MyType` to `&dyn Bar` See RFC 2397 for the original motivation, and the current reference for more details. FromIterator and Extend for tuples Earlier versions of Rust implemented convenience traits for iterators of (T, U) tuple pairs to behave like Iterator::unzip, with Extend in 1.56 and FromIterator in 1.79. These have now been extended to more tuple lengths, from singleton (T,) through to 12 items long, (T1, T2, ..., T11, T12). For example, you can now use collect() to fanout into multiple collections at once: use std::collections::{LinkedList, VecDeque}; fn main() { let (squares, cubes, tesseract): (Vec<_>, VecDeque<_>, LinkedList<_>) = (0i32..10).map(|i| (i * i, i.pow(3), i.pow(4))).collect(); println!("{squares:?}"); println!("{cubes:?}"); println!("{tesseract:?}"); } [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] [0, 1, 8, 27, 64, 125, 216, 343, 512, 729] [0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561] Updates to std::env::home_dir() std::env::home_dir() has been deprecated for years, because it can give surprising results in some Windows configurations if the HOME environment variable is set (which is not the normal configuration on Windows). We had previously avoided changing its behavior, out of concern for compatibility with code depending on this non-standard configuration. Given how long this function has been deprecated, we're now updating its behavior as a bug fix, and a subsequent release will remove the deprecation for this function. Stabilized APIs BuildHasherDefault::new ptr::fn_addr_eq io::ErrorKind::QuotaExceeded io::ErrorKind::CrossesDevices {float}::midpoint Unsigned {integer}::midpoint NonZeroU* midpoint impl std::iter::Extend for tuples with arity 1 through 12 FromIterator<(A, ...)> for tuples with arity 1 through 12 std::task::Waker::noop These APIs are now stable in const contexts mem::size_of_val mem::align_of_val Layout::for_value Layout::align_to Layout::pad_to_align Layout::extend Layout::array std::mem::swap std::ptr::swap NonNull::new HashMap::with_hasher HashSet::with_hasher BuildHasherDefault::new <float>::recip <float>::to_degrees <float>::to_radians <float>::max <float>::min <float>::clamp <float>::abs <float>::signum <float>::copysign MaybeUninit::write Other changes Check out everything that changed in Rust, Cargo, and Clippy. Contributors to 1.85.0 Many people came together to create Rust 1.85.0. We couldn't have done it without all of you. Thanks!`

- [Spidermonkey Development Blog: Making Teleporting Smarter](#) (2025/02/19 18:00)

Recently I got to land a patch which touches a cool optimization, that I had to really make sure I understood deeply. As a result, I wrote a huge

commit message. I'd like to expand that message a touch here and turn it into a nice blog post. This post assumes roughly that you understand how Shapes work in the JavaScript object model, and how prototypical property lookup works in JavaScript. If you don't understand that just yet, this blog post by Matthias Bynens is a good start. This patch aims to mitigate a performance cliff that occurs when we have applications which shadow properties on the prototype chain or which mutate the prototype chain. The problem is that these actions currently break a property lookup optimization called "Shape Teleportation". What is Shape Teleporting? Suppose you're looking up some property *y* on an object *obj*, which has a prototype chain with 4 elements. Suppose *y* isn't stored on *obj*, but instead is stored on some prototype object *B*, in slot 1. In order to get the value of this property, officially you have to walk from *obj* up to *B* to find the value of *y*. Of course, this would be inefficient, so what we do instead is attach an inline cache to make this lookup more efficient. Now we have to guard against future mutation when creating an inline cache. A basic version of a cache for this lookup might look like: Check *obj* still has the same shape. Check *obj*'s prototype (*D*) still has the same shape. Check *D*'s prototype (*C*) still has the same shape. Check *C*'s prototype (*B*) still has the same shape. Load slot 1 out of *B*. This is less efficient than we would like though. Imagine if instead of having 3 intermediate prototypes, there were 13 or 30? You'd have this long chain of prototype shape checking, which takes a long time! Ideally, what you'd like is to be able to simply say Check *obj* still has the same shape. Check *B* still has the same shape. Load slot 1 out of *B*. The problem with doing this naively is "What if someone adds *y* as a property to *C*? With the faster guards, you'd totally miss that value, and as a result compute the wrong result. We don't like wrong results. Shape Teleporting is the existing optimization which says that so long as you actively force a change of shape on objects in the prototype chain when certain modifications occur, then you can guard in inline-caches only on the shape of the receiver object and the shape of the holder object. By forcing each shape to be changed, inline caches which have baked in assumptions about these objects will no longer succeed, and we'll take a slow path, potentially attaching a new IC if possible. We must reshape in the following situations: Adding a property to a prototype which shadows a property further up the prototype chain. In this circumstance, the object getting the new property will naturally reshape to account for the new property, but the old holder needs to be explicitly reshaped at this point, to avoid an inline cache jumping over the newly defined prototype. Modifying the prototype of an object which exists on the prototype chain. For this case we need to invalidate the shape of the object being mutated (natural reshape due to changed prototype), as well as the shapes of all objects on the mutated object's prototype chain. This is to invalidate all stubs which have teleported over the mutated object. Furthermore, we must avoid an "A-B-A" problem, where an object returns to a shape prior to prototype modification: for example, even if we re-shape *B*, what if code deleted and then re-added *y*, causing *B* to take on its old shape? Then the IC would start working again, even though the prototype chain may have been mutated! Prior to this patch, Watchtower watches for prototype mutation and shadowing, and marks the shapes of the prototype objects involved with these operations as InvalidatedTeleporting. This means that property access with the objects involved can never more rely on the shape teleporting optimization. This also avoids the A-B-A problem as new shapes will always carry along the InvalidatedTeleporting flag. This patch instead chooses to migrate an object shape to dictionary mode, or generate a new dictionary shape if it's already in dictionary mode. Using dictionary mode shapes works because all dictionary mode shapes are unique and never recycled. This ensures the ICs are no longer valid as expected, as well as handily avoiding the A-B-A problem. The patch does keep the InvalidatedTeleporting flag to catch potentially ill-behaved sites that do lots of mutation and shadowing, avoiding having to reshape proto objects forever. The patch also provides a preference to allow cross-comparison between old and new, however this patch defaults to dictionary mode teleportation. Performance testing on micro-benchmarks shows large impact by allowing ICs to attach where they couldn't before, however Speedometer3 shows no real movement.

- [The Servo Blog: This month in Servo: new webview API, relative colors, canvas buffs, and more!](#) (2025/02/19 00:00)

Servo now supports several new web API features: `:host selector` (@simonwuelker, #34870) `<slot>` elements in the DOM (@simonwuelker, #35013, #35177, #35191, #35221, #35137, #35222), with layout support landing next month Relative CSS colors, like `color(from ...)` and `rgb(from ...)` (@Loirooriol, #34897) `ClipboardEvent` (@Gae24, #33576, #35146), with `--pref dom_clipboardevent_enabled` Detecting WGSLL extensions via `WGSLLanguageFeatures` (@sagudev, #34928) `pointer_composite_access` WGSLL extension (@sagudev, #35161) `blitFramebuffer()` on `WebGL2RenderingContext` (@jdm, Istvan, #26389) `media property` on `HTMLStyleElement` (@webbeef, #35148) We've landed a bunch of `HTMLCanvasElement` improvements: `toDataURL()` now supports `image/jpeg` and `image/webp` (@webbeef, #34861) `toBlob()` is now supported (@webbeef, #34938) `transferControlToOffscreen()` is now supported (@webbeef, #34959) Streams are a lot more useful now, with `ReadableStreamBYOBReader` now supporting `read()` (@Taym95, #35040), `cancel()`, `close()`, and `releaseLock()` (@Taym95, #34958). Servo now passes 40.6% (+7.5pp) of enabled Shadow DOM tests, thanks to our landing support for the `:host selector` (@simonwuelker, #34870) and the `<slot>` element (@simonwuelker, #35013, #35177, #35191, #35221, #35137, #35222), plus improvements to event handling (@simonwuelker, #34788, #34884), `script` (@willypuzzle, #34787), `style` (@simonwuelker, @jdm, #35198, #35132), and the DOM tree (@simonwuelker, @Taym95, #34803, #34834, #34863, #34909, #35076). Table layout is significantly better now, particularly in 'table-layout: fixed' (@Loirooriol, #35170), table sizing (@Loirooriol, @mrobinson, #34889, #34947, #35167), rowspan sizing (@mrobinson, @Loirooriol, #35095), interaction with floats (@Loirooriol, #35207), and 'border-collapse' layout (@Loirooriol, #34932, #34908, #35097, #35122, #35165) and painting (@Loirooriol, #34933, #35003, #35100, #35075, #35129, #35163). As a result, Servo now passes 90.2% (+11.5pp) of enabled CSS tables tests, and of the tests that are in CSS 2, we now pass more than Blink and WebKit! We literally stood on the shoulders of giants here, because this would not have been possible without Blink's robust table impl. Despite their age, tables are surprisingly underspecified, so we also needed to report several spec issues along the way (@Loirooriol). Embedding Servo aims to be an embeddable web engine, but so far it's been a lot harder to embed Servo than it should be. For one, configuring and starting Servo is complicated. We found that getting Servo running at all, even without wiring up input or handling resizes correctly, took over 200 lines of Rust code (@delan, @mrobinson, #35118). Embedders (apps) could only control Servo by sending and receiving a variety of "messages" and "events", and simple questions like "what's the current URL?" were impossible to answer without keeping track of extra state in the app. Contrast this with WebKitGTK, where you can write a minimal kiosk app with a fully-functional webview in under 50 lines of C. To close that gap, we've started reworking our embedding API towards something more idiomatic and ergonomic, starting with the concept embedders care about most: the webview. Our new webview API is controlled by calling methods on a `WebView` handle (@delan, @mrobinson, #35119, #35183, #35192), including navigation and user input. Handles will eventually represent the lifecycle of the webview itself; if you have one, the webview is valid, and if you drop them, the webview is destroyed. Servo needs to call into the embedder too, and here we've started replacing the old `EmbedderMsg` API with a webview delegate (@delan, @mrobinson, #35211), much like the delegates in Apple's WebKit API. In Rust, a delegate is a trait that the embedder can install its own impl for. Stay tuned for more on this next month! Embedders can now intercept any request, not just navigation (@zhuhaichao518, #34961), and you can now identify the webview that caused an HTTP credentials prompt (@pewsheen, @mrobinson, #34808). Other embedding improvements include: Adding a trait to allow for alternative rendering contexts that are better suited to certain applications (@dklassic, @wusyong, #35052, #34813, #34780) Simplifying types used by consumers of `libservo` (@delan, @mrobinson, #35156) Making it easier to build Servo without crown (@jdm, #35055) Other changes We've reworked Servo's preferences system, making all prefs optional with reasonable defaults (@mrobinson, #34966, #34999, #34994). As a result:

The names of all preferences have changed; see the Prefs docs for a list Embedders no longer need a prefs.json resource to get Servo running. Some debug options were converted to preferences (@mrobinson, #34998). Devtools now highlights console.log() arguments according to their types (@simonwuelker, #34810). Servo's networking is more efficient now, with the ability to cancel fetches for navigation that contain redirects (@mrobinson, #34919) and cancel fetches for <video> and <media> when the document is unloaded (@mrobinson, #34883). Those changes also eliminate per-request IPC channels for navigation and cancellation respectively, and in the same vein, we've eliminated them for image loading too (@mrobinson, #35041). We've continued splitting up our massive script crate (@jdm, #34359, #35157, #35169, #35172), which will eventually make Servo much faster to build. A few crashes have been fixed, including when exiting Servo (@mukilan, #34917), when using the internal memory profiler (@jdm, #35058), and when running ResizeObserver callbacks (@willypuzzle, #35168). For developers We now run CI smoketests on OpenHarmony using a real device (@jschwe, @mukilan, #35006), increasing confidence in your changes beyond compile-time errors. We've also tripled our self-hosted CI runner capacity (@delan, #34983, #35002), making concurrent Windows and macOS builds possible without falling back to the much slower GitHub-hosted runners. Servo can't yet run WebDriver-based tests on wpt.fyi, wpt.servo.org, or CI, because the servo executor for the Web Platform Tests does not support testdriver.js. servodriver does, though, so we've started fixing test regressions with that executor with the goal of eventually switching to it (@jdm, #34957, #34997). Donations Thanks again for your generous support! We are now receiving 3835 USD/month (-11.4% over December) in recurring donations. With this money, we've been able to expand our capacity for self-hosted CI runners on Windows, Linux, and macOS builds, halving mach try build times from over an hour to under 30 minutes! Servo is also on thanks.dev, and already 21 GitHub users (+5 over December) that depend on Servo are sponsoring us there. If you use Servo libraries like url, html5ever, selectors, or cssparser, signing up for thanks.dev could be a good way for you (or your employer) to give back to the community. 3835 USD/month 10000 As always, use of these funds will be decided transparently in the Technical Steering Committee. For more details, head to our Sponsorship page. Conference talks Why Build a New Browser Engine in Rust? (slides) — Martin Robinson spoke about Servo's unique benefits as a Rust-based browser and web engine, and how browser diversity can create a new renaissance for the web platform

- [Cameron Kaiser: February patch set for TenFourFox](#) (2025/02/15 00:19)

I was slack on doing the Firefox 128ESR platform rebase for TenFourFox, but I finally got around tuit, mostly because I've been doing a little more work on the Quad G5 and put some additional patches in to scratch my own itches. (See, this is what I mean by "hobby mode.") The big upgrade is a substantial overhaul of Reader Mode to pick up interval improvements in Readability. I remind folks that I went all-in on Reader Mode for a reason: it's lightweight, it makes little demands of our now antiquated machines (and TenFourFox's antiquated JavaScript runtime), and it renders very, very fast. That's why, for example, you can open a link directly in Reader Mode (right-click, it's there in the menu), the browser defaults to "sticky" Reader Mode where links you click in an article in Reader Mode stay in Reader Mode (like Las Vegas) until you turn it off from the book icon in the address bar, and you can designate certain sites to always open in Reader Mode, either every page or just subpages in case the front page doesn't render well — though that's improved too. (You can configure that from the TenFourFox preference pane. All of these features are unique to TenFourFox.) I also made some subtle changes to the CSS so that it lays out wider, which was really my only personal complaint; otherwise I'm an avid user. The improvements largely relate to better byline and "fluff" text detection as well as improved selection of article inline images. Try it. You'll like it. I should note that Readability as written no longer works directly on TenFourFox due to syntactic changes and I had to do some backporting. If a page suddenly snaps to the non-Reader view, there was an error. Look in the Browser console for the message and report it; it's possible there is some corner I didn't hit with my own testing. In addition, there are updates to the ATSUI font blacklist (and a

tweak to CFF font table support) and a few low-risk security patches that likely apply to us, as well as refreshed HSTS pins, TLS root certificates, EV roots, timezone data and TLDs. I have also started adding certain AI-related code to the nuisance JavaScript block list as well as some new adbot host aliases I found. Those probably can't run on TenFourFox anyway (properly if at all), but now they won't even be loaded or parsed. The source code can be downloaded from Github (at the command line you can also just do `git clone https://github.com/classilla/tenfourfox.git`) and built in the usual way. Remember that these platform updates require a clobber, so you must build from scratch. I was asked about making TenFourFox a bit friendlier with Github; that's a tall order and I'm still thinking about how, but at least the wiki is readable currently even if it isn't very pretty.

- [Firefox Add-on Reviews: Supercharge your productivity with a Firefox extension](#) (2025/02/14 19:42)

With more work and education happening online (and at home) you may find yourself needing new ways to juice your productivity. From time management to organizational tools and more, the right Firefox extension can give you an edge in the art of efficiency. I need help saving and organizing a lot of web content Gyazo Capture, save, and share anything you find on the web. Gyazo is a great tool for personal or collaborative record keeping and research. Clip entire pages or just pertinent portions. Save images or take screenshots. Gyazo makes it easy to perform any type of web clipping action by either right-clicking on the page element you want to save or using the extension's toolbar button. Everything gets saved to your Gyazo account, making it accessible across devices and collaborative teams. On your Gyazo homepage you can easily browse and sort everything you've clipped; and organize everything into shareable topics or collections. <figcaption class="wp-element-caption">With its minimalist pop-up interface, Gyazo makes it easy to clip elements, sections, or entire web pages. </figcaption> Evernote Web Clipper Similar to Gyazo, Evernote Web Clipper offers a kindred feature set—clip, save, and share web content—albeit with some nice user interface distinctions. Evernote places emphasis on making it easy to annotate images and articles for collaborative purposes. It also has a strong internal search feature, allowing you to search for specific words or phrases that might appear across scattered groupings of clipped content. Evernote also automatically strips out ads and social widgets on your saved pages. Print Edit WE If you need to save or print an important web page — but it's mucked up with a bunch of unnecessary clutter like ads, sidebars, and other peripheral content — Print Edit WE lets you easily remove those unwanted elements. Along with a host of great features like the option to save web pages as either HTML or PDF files, automatically delete graphics, and the ability to alter text or add notes, Print Edit WE also provides an array of productivity optimizations like keyboard shortcuts and mouse gestures. This is the ideal productivity extension for any type of work steeped in web research and cataloging. Focus! Focus! Focus! Anti-distraction and decluttering extensions can be a major boon for online workers and students... Block Site Do you struggle avoiding certain time-wasting, productivity-sucking websites? With Block Site you can enforce restrictions on sites that tempt you away from good work habits. Just list the websites you want to avoid for specified periods of time (certain hours of the day or some days entirely, etc.) and Block Site won't let you access them until you're out of the focus zone. There's also a fun redirection feature where you're automatically redirected to a more productive website anytime you try to visit a time waster <figcaption class="wp-element-caption">Give yourself a custom message of encouragement (or scolding?) whenever you try to visit a restricted site with Block Site</figcaption> LeechBlock NG Very similar in function to Block Site, LeechBlock NG offers a few intriguing twists beyond standard site-blocking features. In addition to blocking sites during specified times, LeechBlock NG offers an array of granular, website-specific blocking abilities—like blocking just portions of websites (e.g. you can't access the YouTube homepage but you can see video pages) to setting restrictions on predetermined days (e.g. no Twitter on weekends) to 60-second delayed access to certain websites to give you time to reconsider that potentially productivity killing decision. Tomato Clock A simple but highly effective

time management tool, Tomato Clock (based on the Pomodoro technique) helps you stay on task by tracking short, focused work intervals. The premise is simple: it assumes everyone's productive attention span is limited, so break up your work into manageable "tomato" chunks. Let's say you work best in 40-minute bursts. Set Tomato Clock and your browser will notify you when it's break time (which is also time customizable). It's a great way to stay focused via short sprints of productivity. The extension also keeps track of your completed tomato intervals so you can track your achieved results over time.

Tabby - Window & Tab Manager Are you overwhelmed by lots of open tabs and windows? Need an easy way to overcome desktop chaos? Tabby - Window & Tab Manager to the rescue. Regain control of your ever-sprawling open tabs and windows with an extension that lets you quickly reorganize everything. Tabby makes it easy to find what you need in a chaotic sea of open tabs — you can not only word/phrase search for the content your looking for, but Tabby also has a visual preview feature so you can get a look at each of your open tabs without actually navigating to them. And whenever you need a clean slate but want to save your work, you can save and close all of your open tabs with a single mouse click and return to them later.

Access all of Tabby's features in one convenient pop-up.

Tranquility Reader Imagine a world wide web where everything but the words are stripped away—no more distracting images, ads, tempting links to related stories, nothing—just the words you're there to read. That's Tranquility Reader. Simply hit the toolbar button and instantly streamline any web page. Tranquility Reader offers quite a few other nifty features as well, like the ability to save content offline for later reading, customizable font size and colors, add annotations to saved pages, and more. We hope some of these great extensions will give your productivity a serious boost! Fact is there are a vast number of extensions out there that could possibly help your productivity—everything from ways to organize tons of open tabs to translation tools to bookmark managers and more.

- [Karl Dubost: Some Ways To Contribute To WebKit and Web Interoperability](#) (2025/02/14 06:55)

Someone asked me recently how to contribute to the WebKit project and more specifically how to find the low hanging fruits. While some of these are specific to WebKit, they can be easily applied to other browsers. Every browser engines projects have more bugs than they can handle with their teams. In no specific orders, some ideas for contributing. Curate Old Bugs on the bug tracker Go through old bugs of bugs.webkit.org. Try to understand what the bug is about. Create simplified test case when there is none and add them as an attachment. If they show differences in between the browsers, take a screenshot when it's visual in Safari (WebKit), Firefox (Gecko), Chrome (Blink). If there is no difference in between browsers, CC: me on the bug, and probably we will be able to close it. This might help reveal some old fixable bug or make it easier to fix it for another engineer. Some of them might be easy enough that you can start fix them yourself. Find Out About Broken Stuff On WPT. Dive into all the bugs which fail in Safari, but pass in Firefox and/or Chrome. (You can do similar search for things failing in Chrome or failing in Firefox.) Understand what the tests is doing. You can check this with the WPT.live links and/or the associated commit. Check if the test is not broken and makes sense. Check if there is an associated bug on bugs.webkit.org. If not, open a new one. FIXME Hunt Inside WebKit Code List all the FIXME which are flagged in the WebKit Source code. Not all of them are easy to fix, but some might be low hanging fruit. That will require to dive in the source code and understand it. Open a new bug on bugs.webkit.org if not yet existing. Eventually propose a patch. Tests WebKit Quirks There are a number Quirks in the WebKit project. These are in place to hotfix websites not doing the right thing. Sometimes these Quirks are not needed anymore. The site has made a silent fix. They didn't tell us about it. They need to be retested and flagged when there are not necessary anymore. This can lead to patches on removing the quirk when it is not needed anymore. Some of these quirks do not have the remove quirk bug counterpart. It would be good to create the bug for them. Example of a Remove Quirk Bug. Triage Incoming Bugs On Webcompat.Com For Safari Time to time there are bugs reported on webcompat.com for Safari. They require to be analyzed and understood.

Sometimes, a new bug needs to be opened on bugs.webkit.org. Again, this is strongly explaining how to help from the side of WebKit. But these type of participation can be easily transposed for Gecko and Blink. If you have other ideas for fixing bugs, let me know. Otsukare!

- [Hacks.Mozilla.Org: Launching Interop 2025](#) (2025/02/13 16:59)

Launching Interop 2025 The Interop Project is a collaboration between browser vendors and other platform implementors to provide users and web developers with high quality implementations of the web platform. Each year we select a set of focus areas representing key areas where we want to improve interoperability. Encouraging all browser engines to prioritize common features ensures they become usable for web developers as quickly as possible. Progress in each engine and the overall Interop score are measured by tracking the pass rate of a set of web-platform tests for each focus area using the Interop dashboard. Interop 2024 Before introducing the new focus areas for this year, we should look at the successes of Interop 2024. The Interop score, measuring the percentage of tests that pass in all of the major browser engines, has reached 95% in latest browser releases, up from only 46% at the start of the year. In pre-release browsers it's even higher — over 97%. This is a huge win that shows how effective Interop can be at aligning browsers with the specifications and each other. Each browser engine individually achieved a test pass score of 98% in stable browser releases and 99% in pre-release, with Firefox finishing slightly ahead with 98.8% in release and 99.1% in Nightly. For users, this means features such as `requestVideoFrameCallback`, Declarative Shadow DOM, and Popover, which a year ago only had limited availability, are now implemented interoperably in all browsers. Interop 2025 Building on Interop 2024's success, we are excited to continue the project into 2025. This year we have 19 focus areas; 17 new and 2 from previous years. A full description of all the focus areas is available in the Interop repository. From 2024 we're carrying forward Layout (really "Flexbox and Grid"), and Pointer and Mouse Events. These are important platform primitives where the Interop project has already led to significant interoperability improvements. However, with technologies that are so fundamental to the modern web we think it's important to set ambitious goals and continue to prioritize these areas, creating rock solid foundations for developers to build on. The new focus areas represent a broad cross section of the platform. Many of them — like Anchor Positioning and View Transitions — have been identified from clear developer demand in surveys such as State of HTML and State of CSS. Inclusion in Interop will ensure they're usable as soon as possible. In addition to these high profile new features, we'd like to highlight some lesser-known focus areas and explain why we're pleased to see them in Interop. Storage Access At Mozilla user privacy is a core principle. One of the most common methods for tracking across the web is via third-party cookies. When sites request data from external services, the service can store data that's re-sent when another site uses the same service. Thus the service can follow the user's browsing across the web. To counter this, Firefox's "Total Cookie Protection" partitions storage so that third parties receive different cookie data per site and thus reduces tracking. Other browsers have similar policies, either by default or in private browsing modes. However, in some cases, non-tracking workflows such as SSO authentication depend on third party cookies. Storage partitioning can break these workflows, and browsers currently have to ship site-specific workarounds. The Storage Access API solves this by letting sites request access to the unpartitioned cookies. Interop here will allow browsers to advance privacy protections without breaking critical functionality. Web Compat The Web Compat focus area is unique in Interop. It isn't about one specific standard, but focuses on browser bugs known to break sites. These are often in older parts of the platform with long-standing inconsistencies. Addressing these requires either aligning implementations with the standard or, where that would break sites, updating the standard itself. One feature in the Web Compat focus area for 2025 is CSS Zoom. Originally a proprietary feature in Internet Explorer, it allowed scaling layout by adjusting the computed dimensions of elements at a time before CSS transforms. WebKit reverse-engineered it, bringing it into Blink, but Gecko never implemented it, due to the lack of a specification and the complexities it created in layout calculations.

Unfortunately, a feature not being standardised doesn't prevent developers from using it. Use of CSS Zoom led to layout issues on some sites in Firefox, especially on mobile. We tried various workarounds and have had success using interventions to replace zoom with CSS transforms on some affected sites, but an attempt to implement the same approach directly in Gecko broke more sites than it fixed and was abandoned. The situation seemed to be at an impasse until 2023 when Google investigated removing CSS Zoom from Chromium. Unfortunately, it turned out that some use cases, such as Microsoft Excel Online's worksheet zoom, depended on the specific behaviour of CSS Zoom, so removal was not feasible. However, having clarified the use cases, the Chromium team was able to propose a standardized model for CSS Zoom that was easier to implement without compromising compatibility. This proposal was accepted by the CSS WG and led to the first implementation of CSS Zoom in Firefox 126, 24 years after it was first released in Internet Explorer. With Interop 2025, we hope to bring the story of CSS Zoom to a close with all engines finally converging on the same behaviour, backed by a real open standard.

WebRTC Video conferencing is now an essential feature of modern life, and in-browser video conferencing offers both ease of use and high security, as users are not required to download a native binary. Most web-based video conferencing relies on the WebRTC API, which offers high level tools for implementing real time communications. However, WebRTC has long suffered from interoperability issues, with implementations deviating from the standards and requiring nonstandard extensions for key features. This resulted in confusion and frustration for users and undermined trust in the web as a reliable alternative to native apps. Given this history, we're excited to see WebRTC in Interop for the first time. The main part of the focus area is the RTCRtpScriptTransform API, which enables cross browser end-to-end encryption. Although there's more to be done in the future, we believe Interop 2025 will be a big step towards making WebRTC a truly interoperable web standard.

Removing Mutation Events The focus area for Removing Mutation Events is the first time Interop has been used to coordinate the removal of a feature. Mutation events fire when the DOM changes, meaning the event handlers run on the critical path for DOM manipulation, causing major performance issues, and significant implementation complexity. Despite the fact that they have been implemented in all engines, they're so problematic that they were never standardised. Instead, mutation observers were developed as a standard solution for the use cases of mutation events without their complexity or performance problems. Almost immediately after mutation observers were implemented, a Gecko bug was filed: "We now have mutation observers, and we'd really like to kill support for mutation events at some point in the future. Probably not for a while yet." That was in 2012. The difficulty is the web's core commitment to backwards compatibility. Removing features that people rely on is unacceptable. However, last year Chromium determined that use of mutation events had dropped low enough to allow a "deprecation trial", disabling mutation events by default, but allowing specific sites to re-enable them for a limited time. This is good news, but long-running deprecation trials can create problems for other browsers. Disabling the feature entirely can break sites that rely on the opt-out. On the other hand we know from experience that some sites actually function better in a browser with mutation events disabled (for example, because they are used for non-critical features, but impact performance). By including this removal in Interop 2025, we can ensure that mutation events are fully removed in 2025 and end the year with reduced platform complexity and improved web performance.

Interop Investigations As well as focus areas, the Interop project also runs investigations aimed at long-term interoperability improvements to areas where we can't measure progress using test pass rates. For example Interop investigations can be looking to add new test capabilities, or increase the test coverage of platform features.

Accessibility Investigation The accessibility testing started as part of Interop 2023. It has added APIs for testing accessible name and computed role, as well as more than 1000 new tests. Those tests formed the Accessibility focus area in Interop 2024, which achieved an Interop score of 99.7%. In 2025 the focus will be expanding the testability of accessibility features. Mozilla is working on a prototype of AccessibleNode; an API that enables verifying the shape of the accessibility tree, along

with its states and properties. This will allow us to test the effect of features like CSS `display: contents` or `::before/::after` on the accessibility tree. Mobile Testing Investigation Today, all Interop focus areas are scored in desktop browsers. However, some features are mobile-specific or have interoperability challenges unique to mobile. Improving mobile testing has been part of Interop since 2023, and in that time we've made significant progress standing up mobile browsers in web-platform-tests CI systems. Today we have reliable runs of Chrome and Firefox Nightly on Android, and Safari runs on iOS are expected soon. However, some parts of our test framework were written with desktop-specific assumptions in the design, so the focus for 2025 will be on bringing mobile testing to parity with desktop. The goal is to allow mobile-specific focus areas in future Interop projects, helping improve interoperability across all device types. Driving the Web Forward The unique and distinguishing feature of the web platform is its basis in open standards, providing multiple implementations and user choice. Through the Interop project, web platform implementors collaborate to ensure that these core strengths are matched by a seamless user experience across browsers. With focus areas covering some of the most important new and existing areas of the modern web, Interop 2025 is set to deliver some of the biggest interoperability wins of the project so far. We are confident that Firefox and other browsers will rise to the challenge, providing users and developers with a more consistent and reliable web platform. Partner Announcements Apple Bocoup Google Igalia Microsoft The post Launching Interop 2025 appeared first on Mozilla Hacks - the Web developer blog.

- [The Rust Programming Language Blog: 2024 State of Rust Survey Results](#) (2025/02/13 00:00)

Hello, Rustaceans! The Rust Survey Team is excited to share the results of our 2024 survey on the Rust Programming language, conducted between December 5, 2024 and December 23, 2024. As in previous years, the 2024 State of Rust Survey was focused on gathering insights and feedback from Rust users, and all those who are interested in the future of Rust more generally. This ninth edition of the survey surfaced new insights and learning opportunities straight from the global Rust language community, which we will summarize below. In addition to this blog post, we have also prepared a report containing charts with aggregated results of all questions in the survey. Our sincerest thanks to every community member who took the time to express their opinions and experiences with Rust over the past year. Your participation will help us make Rust better for everyone. There's a lot of data to go through, so strap in and enjoy!

Participation	Survey Started	Completed	Completion rate
Views	2023 11 950	9 710	82.2%
Views	2024 9 450	7 310	77.4%

As shown above, in 2024, we have received fewer survey views than in the previous year. This was likely caused simply by the fact that the survey ran only for two weeks, while in the previous year it ran for almost a month. However, the completion rate has also dropped, which seems to suggest that the survey might be a bit too long. We will take this into consideration for the next edition of the survey. Community The State of Rust survey not only gives us excellent insight into how many Rust users around the world are using and experiencing the language but also gives us insight into the makeup of our global community. This information gives us a sense of where the language is being used and where access gaps might exist for us to address over time. We hope that this data and our related analysis help further important discussions about how we can continue to prioritize global access and inclusivity in the Rust community. Same as every year, we asked our respondents in which country they live in. The top 10 countries represented were, in order: United States (22%), Germany (14%), United Kingdom (6%), France (6%), China (5%), Canada (3%), Netherlands (3%), Russia (3%), Australia (2%), and Sweden (2%). We are happy to see that Rust is enjoyed by users from all around the world! You can try to find your country in the chart below: `<noscript> </noscript>` [PNG] [SVG] We also asked whether respondents consider themselves members of a marginalized community. Out of those who answered, 74.5% selected no, 15.5% selected yes,

and 10% preferred not to say. We have asked the group that selected “yes” which specific groups they identified as being a member of. The majority of those who consider themselves a member of an underrepresented or marginalized group in technology identify as lesbian, gay, bisexual, or otherwise non-heterosexual. The second most selected option was neurodivergent at 46% followed by trans at 35%. <noscript>

 </noscript> [PNG] [SVG] Each year, we must acknowledge the diversity, equity, and inclusivity (DEI) related gaps in the Rust community and open source as a whole. We believe that excellent work is underway at the Rust Foundation to advance global access to Rust community gatherings and distribute grants to a diverse pool of maintainers each cycle, which you can learn more about here. Even so, global inclusion and access is just one element of DEI, and the survey working group will continue to advocate for progress in this domain. Rust usage The number of respondents that self-identify as a Rust user was quite similar to last year, around 92%. This high number is not surprising, since we primarily target existing Rust developers with this survey. <noscript> </noscript> [PNG] [SVG] Similarly as last year, around 31% of those who did not identify as Rust users cited the perception of difficulty as the primary reason for not using Rust. The most common reason for not using Rust was that the respondents simply haven't had the chance to try it yet. <noscript> </noscript>

[PNG] [SVG] [Wordcloud of open answers] Of the former Rust users who participated in the 2024 survey, 36% cited factors outside their control as a reason why they no longer use Rust, which is a 10pp decrease from last year. This year, we also asked respondents if they would consider using Rust again if an opportunity comes up, which turns out to be true for a large fraction of the respondents (63%). That is good to hear!

<noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] Closed answers marked with N/A were not present in the previous version(s) of the survey. Those not using Rust anymore told us that it is because they don't really need it (or the goals of their company changed) or because it was not the right tool for the job. A few reported being overwhelmed by the language or its ecosystem in general or that switching to or introducing Rust would have been too expensive in terms of human effort. Of those who used Rust in 2024, 53% did so on a daily (or nearly daily) basis — an increase of 4pp from the previous year. We can observe an upward trend in the frequency of Rust usage over the past few years, which suggests that Rust is being increasingly used at work. This is also confirmed by other answers mentioned in the Rust at Work section later below. <noscript> </noscript>

[PNG] [SVG] Rust expertise is also continually increasing amongst our respondents! 20% of respondents can write (only) simple programs in Rust (a decrease of 3pp from 2023), while 53% consider themselves productive using Rust — up from 47% in 2023. While the survey is just one tool to measure the changes in Rust expertise overall, these numbers are heartening as they represent knowledge growth for many Rustaceans returning to the survey year over year. <noscript> </noscript> [PNG] [SVG]

Unsurprisingly, the most popular version of Rust is latest stable, either the most recent one or whichever comes with the users' Linux distribution. Almost a third of users also use the latest nightly release, due to various reasons (see below). However, it seems that the beta toolchain is not

used much, which is a bit unfortunate. We would like to encourage Rust users to use the beta toolchain more (e.g. in CI environments) to help test soon-to-be stabilized versions of Rust.                                    

[PNG] [SVG] [Wordcloud of open answers] People that use the nightly toolchain mostly do it to gain access to specific unstable language features. Several users have also mentioned that rustfmt works better for them on nightly or that they use the nightly compiler because of faster compilation times.                                

[PNG] [SVG] [Wordcloud of open answers] Learning Rust To use Rust, programmers first have to learn it, so we are always interested in finding out how do they approach that. Based on the survey results, it seems that most users learn from Rust documentation and also from The Rust Programming Language book, which has been a favourite learning resource of new Rustaceans for a long time. Many people also seem to learn by reading the source code of Rust crates. The fact that both the documentation and source code of tens of thousands of Rust crates is available on docs.rs and GitHub makes this easier.                          

[PNG] [SVG] [Wordcloud of open answers] In terms of answers belonging to the "Other" category, they can be clustered into three categories: people using LLM (large language model) assistants (Copilot, ChatGPT, Claude, etc.), reading the official Rust forums (Discord, URLO) or being mentored while contributing to Rust projects. We would like to extend a big thank you to those making our spaces friendly and welcoming for newcomers, as it is important work and it pays off. Interestingly, a non-trivial number of people "learned by doing" and used rustc error messages and clippy as a guide, which is a good indicator of the quality of Rust diagnostics. In terms of formal education, it seems that Rust has not yet penetrated university curriculums, as this is typically a very slowly moving area. Only a very small number of respondents (around 3%) have taken a university Rust course or used university learning materials.                  

[PNG] [SVG] Programming environment In terms of operating systems used by Rustaceans, Linux was the most popular choice, and it seems that it is getting increasingly popular year after year. It is followed by macOS and Windows, which have a very similar share of usage.               

</noscript> [PNG] [SVG] [Wordcloud of open answers] As you can see in the wordcloud, there are also a few users that prefer Arch, btw. Rust programmers target a diverse set of platforms with their Rust programs. We saw a slight uptick in users targeting embedded and mobile platforms, but otherwise the distribution of platforms stayed mostly the same as last year. Since the WebAssembly target is quite diverse, we have split it into two separate categories this time. Based on the results it is clear that when using WebAssembly, it is mostly in the context of browsers (23%) rather than other use-cases (7%).         

[PNG] [SVG] [Wordcloud of open answers] We cannot of course forget the favourite topic of many programmers: which IDE (developer environment) they use. Although Visual Studio Code still remains the most popular option, its share has dropped by 5pp this year. On the other hand, the Zed editor seems to have gained considerable traction recently. The small percentage of those who selected "Other" are using a wide range of different tools: from CursorAI to classics like Kate or Notepad++. Special mention to the 3 people using "ed", that's quite an achievement.     

alt="what-ide-do-you-use" height="500" src="https://blog.rust-lang.org/images/2025-02-13-rust-survey-2024/what-ide-do-you-use.png" />
</noscript> [PNG] [SVG] [Wordcloud of open answers] You can also take a look at the linked wordcloud that summarizes open answers to this question (the "Other" category), to see what other editors are also popular. Rust at Work We were excited to see that more and more people use Rust at work for the majority of their coding, 38% vs 34% from last year. There is a clear upward trend in this metric over the past few years.

</noscript> </noscript> [PNG] [SVG] The usage of Rust within companies also seems to be rising, as 45% of respondents answered that their organisation makes non-trivial use of Rust, which is a 7pp increase from 2023.

</noscript> </noscript> [PNG] [SVG] Once again, the top reason employers of our survey respondents invested in Rust was the ability to build relatively correct and bug-free software. The second most popular reason was Rust's performance characteristics. 21% of respondents that use Rust at work do so because they already know it, and it's thus their default choice, an uptick of 5pp from 2023. This seems to suggest that Rust is becoming one of the baseline languages of choice for more and more companies.

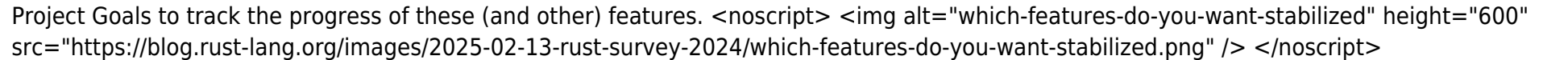
</noscript> </noscript> [PNG] [SVG] Similarly to the previous year, a large percentage of respondents (82%) report that Rust helped their company achieve its goals. In general, it seems that programmers and companies are quite happy with their usage of Rust, which is great!

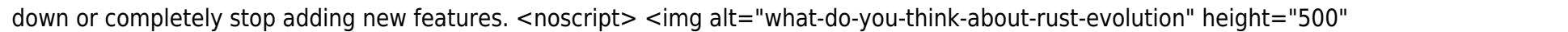
</noscript> </noscript> [PNG] [SVG] In terms of technology domains, the situation is quite similar to the previous year. Rust seems to be especially popular for creating server backends, web and networking services and cloud technologies. It also seems to be gaining more traction for embedded use-cases.

</noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] You can scroll the chart to the right to see more domains. Note that the Automotive domain was not offered as a closed answer in the 2023 survey (it was merely entered through open answers), which might explain the large jump. It is exciting to see the continued growth of professional Rust usage and the confidence so many users feel in its performance, control, security and safety, enjoyability, and more!

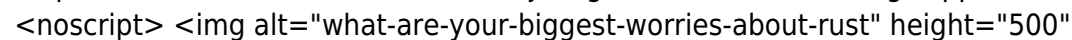
Challenges As always, one of the main goals of the State of Rust survey is to shed light on challenges, concerns, and priorities on Rustaceans' minds over the past year. We have asked our users about aspects of Rust that limit their productivity. Perhaps unsurprisingly, slow compilation was at the top of the list, as it seems to be a perennial concern of Rust users. As always, there are efforts underway to improve the speed of the compiler, such as enabling the parallel frontend or switching to a faster linker by default. We invite you to test these improvements and let us know if you encounter any issues. Other challenges included subpar support for debugging Rust and high disk usage of Rust compiler artifacts. On the other hand, most Rust users seem to be very happy with its runtime performance, the correctness and stability of the compiler and also Rust's documentation.

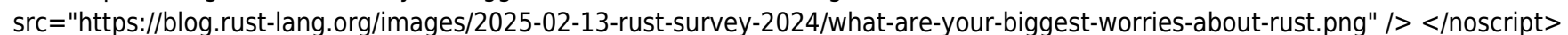
</noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] In terms of specific unstable (or missing) features that Rust users want to be stabilized (or implemented), the most desired ones were async closures and if/let while chains. Well, we have good news! Async closures will be stabilized in

the next version of Rust (1.85), and if/let while chains will hopefully follow soon after, once Edition 2024 is released (which will also happen in Rust 1.85). Other coveted features are generators (both sync and async) and more powerful generic const expressions. You can follow the Rust Project Goals to track the progress of these (and other) features. 

[PNG] [SVG] [Wordcloud of open answers] In the open answers to this question, people were really helpful and tried hard to describe the most notable issues limiting their productivity. We have seen mentions of struggles with async programming (an all-time favourite), debuggability of errors (which people generally love, but they are not perfect for everyone) or Rust tooling being slow or resource intensive (rust-analyzer and rustfmt). Some users also want a better IDE story and improved interoperability with other languages. This year, we have also included a new question about the speed of Rust's evolution. While most people seem to be content with the status quo, more than a quarter of people who responded to this question would like Rust to stabilize and/or add features more quickly, and only 7% of respondents would prefer Rust to slow down or completely stop adding new features. 

Interestingly, when we asked respondents about their main worries for the future of Rust, one of the top answers remained the worry that Rust will become too complex. This seems to be in contrast with the answers to the previous question. Perhaps Rust users still seem to consider the complexity of Rust to be manageable, but they worry that one day it might become too much. We are happy to see that the amount of respondents concerned about Rust Project governance and lacking support of the Rust Foundation has dropped by about 6pp from 2023.



src="https://blog.rust-lang.org/images/2025-02-13-rust-survey-2024/what-are-your-biggest-worries-about-rust.png" /> 

[PNG] [SVG] [Wordcloud of open answers] Looking ahead Each year, the results of the State of Rust survey help reveal the areas that need improvement in many areas across the Rust Project and ecosystem, as well as the aspects that are working well for our community. If you have any suggestions for the Rust Annual survey, please let us know! We are immensely grateful to those who participated in the 2024 State of Rust Survey and facilitated its creation. While there are always challenges associated with developing and maintaining a programming language, this year we were pleased to see a high level of survey participation and candid feedback that will truly help us make Rust work better for everyone. If you'd like to dig into more details, we recommend you to browse through the full survey report.

- [Andrew Halberstadt: Using Jujutsu With Mozilla Unified](#) (2025/02/12 20:19)

With Mozilla's migration from hg.mozilla.org to Github drawing near, the clock is ticking for developers still using Mercurial to find their new workflow. I previously blogged about how Jujutsu can help here, so please check that post out first if you aren't sure what Jujutsu is, or whether it's right for you. If you know you want to give it a shot, read on for a tutorial on how to get everything set up! We'll start with an existing Mercurial clone of mozilla-unified, convert it to use git-cinnabar and then set up Jujutsu using the co-located repo method. Finally I'll cover some tips and tricks for using some of the tooling that relies on version control.

- [Mozilla Thunderbird: Thunderbird Monthly Development Digest - January 2025](#) (2025/02/11 13:41)

Hello again Thunderbird Community! As January drew to a close, the team was closing in on the completion of some important milestones. Additionally, we had scoped work for our main Q1 priorities. Those efforts are now underway and it feels great to cross things off the list and start tackling new challenges. As always, you can catch up on all of our previous digests and updates. FOSDEM - Inspiration, collaboration and

education A modest contingent from the Thunderbird team joined our Mozilla counterparts for an educational and inspiring weekend at Fosdem recently. We talked about standards, problems, solutions and everything in between. However, the most satisfying part of the weekend being standing at the Thunderbird booth and hearing the gratitude, suggestions and support from so many users. With such important discussions among leading voices, we're keen to help in finding or implementing solutions to some of the meatier topics such as: OAuth 2.0 Dynamic Client Registration Protocol Support for unicode email addresses Support for OpenPGP certification authorities and trust delegation Exchange Web Services support in Rust With a reduction in team capacity for part of January, the team was able to complete work on the following tasks that form some of the final stages in our 0.2 release: Folder compaction Saving attachments to disk Download EWS messages in an nsIChannel Keep track of feature delivery here. Account Hub We completed the second and final milestone in the First Time User Experience for email configuration via the enhanced Account Hub over the course of January. Tasks included density and font awareness, refactoring of state management, OAuth prompts, enhanced error handling and more which can be followed via Meta bug & progress tracking. Watch out for this feature being unveiled in daily and beta in the coming weeks! Global Message Database With a significant number of the research and prototyping tasks now behind us, the project has taken shape over the course of January with milestones and tasks mapped out. Recent progress has been related to live view, sorting and support for Unicode server and folder names. Next up is to finally crack the problem of "non-unique unique IDs" mentioned previously, which is important preparatory groundwork required for a clean database migration. In-App Notifications Phase 2 is now complete, and almost ready for uplift to ESR, pending underlying Firefox dependencies scheduled in early March. Features and user stories in the latest milestone include a cache-control mechanism, a thorough accessibility review, schema changes and the addition of guard rails to limit notification frequency. Meta Bug & progress tracking. New Features Landing Soon Several requested features and fixes have reached our Daily users and include... More folder compaction fixes and performance improvements! Messages now adapt to dark mode and include a simple toggle in the header. Introduction of a new "Appearance" Settings UI to globally control message threading/sorting order and many more which are listed in release notes for beta. To see things as they land, and help squash early bugs, you can check the pushlog and try running daily. This would be immensely helpful for catching things early. — Toby Pilling Senior Manager, Desktop Engineering The post Thunderbird Monthly Development Digest - January 2025 appeared first on The Thunderbird Blog.

- [Niko Matsakis: How I learned to stop worrying and love the LLM \(2025/02/10 15:56\)](#)

I believe that AI-powered development tools can be a game changer for Rust—and vice versa. At its core, my argument is simple: AI's ability to explain and diagnose problems with rich context can help people get over the initial bump of learning Rust in a way that canned diagnostics never could, no matter how hard we try. At the same time, rich type systems like Rust's give AIs a lot to work with, which could be used to help them avoid hallucinations and validate their output. This post elaborates on this premise and sketches out some of the places where I think AI could be a powerful boost. Perceived learning curve is challenge #1 for Rust Is Rust good for every project? No, of course not. But it's absolutely great for some things—specifically, building reliable, robust software that performs well at scale. This is no accident. Rust's design is intended to surface important design questions (often in the form of type errors) and to give users the control to fix them in whatever way is best. But this same strength is also Rust's biggest challenge. Talking to people within Amazon about adopting Rust, perceived complexity and fear of its learning curve is the biggest hurdle. Most people will say, "Rust seems interesting, but I don't need it for this problem". And you know, they're right! They don't need it. But that doesn't mean they wouldn't benefit from it. One of Rust's big surprises is that, once you get used to it, it's "surprisingly decent" at very large number of things beyond what it was designed for. Simple business logic and scripts can be very pleasant in

Rust. But the phrase “once you get used to it” in that sentence is key, since most people’s initial experience with Rust is confusion and frustration. Rust likes to tell you no (but it’s for your own good) Some languages are geared to say yes—that is, given any program, they aim to run it and do something. JavaScript is of course the most extreme example (no semicolons? no problem!) but every language does this to some degree. It’s often quite elegant. Consider how, in Python, you write `vec[-1]` to get the last element in the list: super handy! Rust is not (usually) like this. Rust is geared to say no. The compiler is just itching for a reason to reject your program. It’s not that Rust is mean: Rust just wants your program to be as good as it can be. So we try to make sure that your program will do what you want (and not just what you asked for). This is why `vec[-1]`, in Rust, will panic: sure, giving you the last element might be convenient, but how do we know you didn’t have an off-by-one bug that resulted in that negative index?1 But that tendency to say no means that early learning can be pretty frustrating. For most people, the reward from programming comes from seeing their program run—and with Rust, there’s a lot of niggling details to get right before your program will run. What’s worse, while those details are often motivated by deep properties of your program (like data races), the way they are presented is as the violation of obscure rules, and the solution (“add a `*`”) can feel random. Once you get the hang of it, Rust feels great, but getting there can be a pain. I heard a great phrase from someone at Amazon to describe this: “Rust: the language where you get the hangover first”.3 AI today helps soften the learning curve My favorite thing about working at Amazon is getting the chance to talk to developers early in their Rust journey. Lately I’ve noticed an increasing trend—most are using Q Developer. Over the last year, Amazon has been doing a lot of internal promotion of Q Developer, so that in and of itself is no surprise, but what did surprise me a bit is hearing from developers the way that they use it. For most of them, the most valuable part of Q Dev is authoring code but rather explaining it. They ask it questions like “why does this function take an `&T` and not an `Arc<T>`?” or “what happens when I move a value from one place to another?”. Effectively, the LLM becomes an ever-present, ever-patient teacher.4 Scaling up the Rust expert Some time back I sat down with an engineer learning Rust at Amazon. They asked me about an error they were getting that they didn’t understand. “The compiler is telling me something about ‘static, what does that mean?” Their code looked something like this: `async fn log_request_in_background(message: &str) { tokio::spawn(async move { log_request(message); }); }` And the compiler was telling them: `error[E0521]: borrowed data escapes outside of function --> src/lib.rs:2:5 | 1 | async fn log_request_in_background(message: &str) { | ----- - let's call the lifetime of this reference `1` | | `message` is a reference that is only valid in the function body 2 | / tokio::spawn(async move { 3 | | log_request(message); 4 | | }); | | ^ | | | | | _____ `message` escapes the function body here | argument requires that `1` must outlive `static` This is a pretty good error message! And yet it requires significant context to understand it (not to mention scrolling horizontally, sheesh). For example, what is “borrowed data”? What does it mean for said data to “escape”? What is a “lifetime” and what does it mean that “1 must outlive 'static’”? Even assuming you get the basic point of the message, what should you do about it? The fix is easy... if you know what to do Ultimately, the answer to the engineer’s problem was just to insert a call to clone5. But deciding on that fix requires a surprisingly large amount of context. In order to figure out the right next step, I first explained to the engineer that this confusing error is, in fact, what it feels like when Rust saves your bacon, and talked them through how the ownership model works and what it means to free memory. We then discussed why they were spawning a task in the first place (the answer: to avoid the latency of logging)—after all, the right fix might be to just not spawn at all, or to use something like rayon to block the function until the work is done. Once we established that the task needed to run asynchronously from its parent, and hence had to own the data, we looked into changing the log_request_in_background function to take an Arc<String> so that it could avoid a deep clone. This would be more efficient, but only if the caller themselves could cache the Arc<String> somewhere. It turned out that the origin of this string was in another team’s code and that this code`

only returned an `&str`. Refactoring that code would probably be the best long term fix, but given that the strings were expected to be quite short, we opted to just clone the string. You can learn a lot from a Rust error. An error message is often your first and best chance to teach somebody something.—Esteban Küber (paraphrased) Working through this error was valuable. It gave me a chance to teach this engineer a number of concepts. I think it demonstrates a bit of Rust’s promise—the idea that learning Rust will make you a better programmer overall, regardless of whether you are using Rust or not. Despite all the work we have put into our compiler error messages, this kind of detailed discussion is clearly something that we could never achieve. It’s not because we don’t want to! The original concept for `--explain`, for example, was to present a customized explanation of each error was tailored to the user’s code. But we could never figure out how to implement that. And yet tailored, in-depth explanation is absolutely something an LLM could do. In fact, it’s something they already do, at least some of the time—though in my experience the existing code assistants don’t do nearly as good a job with Rust as they could. What makes a good AI opportunity? Emery Berger is a professor at UMass Amherst who has been exploring how LLMs can improve the software development experience. Emery emphasizes how AI can help close the gap from “tool to goal”. In short, today’s tools (error messages, debuggers, profilers) tell us things about our program, but they stop there. Except in simple cases, they can’t help us figure out what to do about it—and this is where AI comes in. When I say AI, I am not talking (just) about chatbots. I am talking about programs that weave LLMs into the process, using them to make heuristic choices or proffer explanations and guidance to the user. Modern LLMs can also do more than just rely on their training and the prompt: they can be given access to APIs that let them query and get up-to-date data. I think AI will be most useful in cases where solving the problem requires external context not available within the program itself. Think back to my explanation of the 'static error, where knowing the right answer depended on how easy/hard it would be to change other APIs. Where I think Rust should leverage AI I’ve thought about a lot of places I think AI could help make working in Rust more pleasant. Here is a selection. Deciding whether to change the function body or its signature Consider this code:

```
fn get_first_name(&self, alias: &str) -> &str { alias }
```

 This function will give a type error, because the signature (thanks to lifetime elision) promises to return a string borrowed from `self` but actually returns a string borrowed from `alias`. Now...what is the right fix? It’s very hard to tell in isolation! It may be that in fact the code was meant to be `&self.name` (in which case the current signature is correct). Or perhaps it was meant to be something that sometimes returns `&self.name` and sometimes returns `alias`, in which case the signature of the function was wrong. Today, we take our best guess. But AI could help us offer more nuanced guidance. Translating idioms from one language to another People often ask me questions like “how do I make a visitor in Rust?” The answer, of course, is “it depends on what you are trying to do”. Much of the time, a Java visitor is better implemented as a Rust `enum` and `match` statements, but there is a time and a place for something more like a visitor. Guiding folks through the decision tree for how to do non-trivial mappings is a great place for LLMs. Figuring out the right type structure When I start writing a Rust program, I start by authoring type declarations. As I do this, I tend to think ahead to how I expect the data to be accessed. Am I going to need to iterate over one data structure while writing to another? Will I want to move this data to another thread? The setup of my structures will depend on the answer to these questions. I think a lot of the frustration beginners feel comes from not having a “feel” yet for the right way to structure their programs. The structure they would use in Java or some other language often won’t work in Rust. I think an LLM-based assistant could help here by asking them some questions about the kinds of data they need and how it will be accessed. Based on this it could generate type definitions, or alter the definitions that exist. Complex refactorings like splitting structs A follow-on to the previous point is that, in Rust, when your data access patterns change as a result of refactorings, it often means you need to do more wholesale updates to your code.⁶ A common example for me is that I want to split out some of the fields of a struct into a substruct, so that they can be borrowed

separately.⁷ This can be quite non-local and sometimes involves some heuristic choices, like “should I move this method to be defined on the new substruct or keep it where it is?”. Migrating consumers over a breaking change When you run the cargo fix command today it will automatically apply various code suggestions to cleanup your code. With the upcoming Rust 2024 edition, cargo fix---edition will do the same but for edition-related changes. All of the logic for these changes is hardcoded in the compiler and it can get a bit tricky. For editions, we intentionally limit ourselves to local changes, so the coding for these migrations is usually not too bad, but there are some edge cases where it’d be really useful to have heuristics. For example, one of the changes we are making in Rust 2024 affects “temporary lifetimes”. It can affect when destructors run. This almost never matters (your vector will get freed a bit earlier or whatever) but it can matter quite a bit, if the destructor happens to be a lock guard or something with side effects. In practice when I as a human work with changes like this, I can usually tell at a glance whether something is likely to be a problem—but the heuristics I use to make that judgment are a combination of knowing the name of the types involved, knowing something about the way the program works, and perhaps skimming the destructor code itself. We could hand-code these heuristics, but an LLM could do it and better, and if could ask questions if it was feeling unsure. Now imagine you are releasing the 2.x version of your library. Maybe your API has changed in significant ways. Maybe one API call has been broken into two, and the right one to use depends a bit on what you are trying to do. Well, an LLM can help here, just like it can help in translating idioms from Java to Rust. I imagine the idea of having an LLM help you migrate makes some folks uncomfortable. I get that. There’s no reason it has to be mandatory—I expect we could always have a more limited, precise migration available.⁸ Optimize your Rust code to eliminate hot spots Premature optimization is the root of all evil, or so Donald Knuth is said to have said. I’m not sure about all evil, but I have definitely seen people rathole on microoptimizing a piece of code before they know if it’s even expensive (or, for that matter, correct). This is doubly true in Rust, where cloning a small data structure (or reference counting it) can often make your life a lot simpler. Llogiq’s great talks on Easy Mode Rust make exactly this point. But here’s a question, suppose you’ve been taking this advice to heart, inserting clones and the like, and you find that your program is running kind of slow? How do you make it faster? Or, even worse, suppose that you are trying to turn our network service. You are looking at the blizzard of available metrics and trying to figure out what changes to make. What do you do? To get some idea of what is possible, check out Scalene, a Python profiler that is also able to offer suggestions as well (from Emery Berger’s group at UMass, the professor I talked about earlier). Diagnose and explain miri and sanitizer errors Let’s look a bit to the future. I want us to get to a place where the “minimum bar” for writing unsafe code is that you test that unsafe code with some kind of sanitizer that checks for both C and Rust UB—something like miri today, except one that works “at scale” for code that invokes FFI or does other arbitrary things. I expect a smaller set of people will go further, leveraging automated reasoning tools like Kani or Verus to prove statically that their unsafe code is correct⁹. From my experience using miri today, I can tell you two things. (1) Every bit of unsafe code I write has some trivial bug or other. (2) If you enjoy puzzling out the occasionally inscrutable error messages you get from Rust, you’re gonna love miri! To be fair, miri has a much harder job—the (still experimental) rules that govern Rust aliasing are intended to be flexible enough to allow all the things people want to do that the borrow checker doesn’t permit. This means they are much more complex. It also means that explaining why you violated them (or may violate them) is that much more complicated. Just as an AI can help novices understand the borrow checker, it can help advanced Rustaceans understand tree borrows (or whatever aliasing model we wind up adopting). And just as it can make smarter suggestions for whether to modify the function body or its signature, it can likely help you puzzle out a good fix. Rust’s emphasis on “reliability” makes it a great target for AI Anyone who has used an LLM-based tool has encountered hallucinations, where the AI just makes up APIs that “seem like they ought to exist”.¹⁰ And yet anyone who has used Rust knows that “if it compiles, it works” is true may

more often than it has a right to be.¹¹ This suggests to me that any attempt to use the Rust compiler to validate AI-generated code or solutions is going to also help ensure that the code is correct. AI-based code assistants right now don't really have this property. I've noticed that I kind of have to pick between "shallow but correct" or "deep but hallucinating". A good example is match statements. I can use rust-analyzer to fill in the match arms and it will do a perfect job, but the body of each arm is `todo!`. Or I can let the LLM fill them in and it tends to cover most-but-not-all of the arms but it generates bodies. I would love to see us doing deeper integration, so that the tool is talking to the compiler to get perfect answers to questions like "what variants does this enum have" while leveraging the LLM for open-ended questions like "what is the body of this arm".¹²

Conclusion Overall AI reminds me a lot of the web around the year 2000. It's clearly overhyped. It's clearly being used for all kinds of things where it is not needed. And it's clearly going to change everything. If you want to see examples of what is possible, take a look at the ChatDBG videos published by Emery Berger's group. You can see how the AI sends commands to the debugger to explore the program state before explaining the root cause. I love the video debugging `bootstrap.py`, as it shows the AI applying domain knowledge about statistics to debug and explain the problem. My expectation is that compilers of the future will not contain nearly so much code geared around authoring diagnostics. They'll present the basic error, sure, but for more detailed explanations they'll turn to AI. It won't be just a plain old foundation model, they'll use RAG techniques and APIs to let the AI query the compiler state, digest what it finds, and explain it to users. Like a good human tutor, the AI will tailor its explanations to the user, leveraging the user's past experience and intuitions (oh, and in the user's chosen language). I am aware that AI has some serious downsides. The most serious to me is its prodigious energy use, but there are also good questions to be asked about the way that training works and the possibility of not respecting licenses. The issues are real but avoiding AI is not the way to solve them. Just in the course of writing this post, DeepSeek was announced, demonstrating that there is a lot of potential to lower the costs of training. As far as the ethics and legality, that is a very complex space. Agents are already doing a lot to get better there, but note also that most of the applications I am excited about do not involve writing code so much as helping people understand and alter the code they've written. We don't always get this right. For example, I find the zip combinator of iterators annoying because it takes the shortest of the two iterators, which is occasionally nice but far more often hides bugs. ← The irony, of course, is that AI can help you to improve your woeful lack of tests by auto-generating them based on code coverage and current behavior. ← I think they told me they heard it somewhere on the internet? Not sure the original source. ← Personally, the thing I find most annoying about LLMs is the way they are trained to respond like groveling serveants. "Oh, that's a good idea! Let me help you with that" or "I'm sorry, you're right I did make a mistake, here is a version that is better". Come on, I don't need flattery. The idea is fine but I'm aware it's not earth-shattering. Just help me already. ← Inserting a call to clone is actually a bit more subtle than you might think, given the interaction of the `async future` here. ← Garbage Collection allows you to make all kinds of refactorings in ownership structure without changing your interface at all. This is convenient, but—as we discussed early on—it can hide bugs. Overall I prefer having that information be explicit in the interface, but that comes with the downside that changes have to be refactored. ← I also think we should add a feature like `View Types` to make this less necessary. In this case instead of refactoring the type structure, AI could help by generating the correct type annotations, which might be non-obvious. ← My hot take here is that if the idea of an LLM doing migrations in your code makes you uncomfortable, you are likely (a) overestimating the quality of your code and (b) underinvesting in tests and QA infrastructure². I tend to view an LLM like a "inconsistently talented contributor", and I am perfectly happy having contributors hack away on projects I own. ← The student asks, "When unsafe code is proven free of UB, does that make it safe?" The master says, "Yes." The student asks, "And is it then still unsafe?" The master says, "Yes." Then, a minute later, "Well, sort of." (We may need new vocabulary.) ← My personal favorite story of this is when I asked ChatGPT to generate me a list

of “real words and their true definition along with 2 or 3 humorous fake definitions” for use in a birthday party game. I told it that “I know you like to hallucinate so please include links where I can verify the real definition”. It generated a great list of words along with plausible looking URLs for merriamwebster.com and so forth—but when I clicked the URLs, they turned out to all be 404s (the words, it turned out, were real—just not the URLs). ← This is not a unique property of Rust, it is shared by other languages with rich type systems, like Haskell or ML. Rust happens to be the most widespread such language. ← I’d also like it if the LLM could be a bit less interrupt-y sometimes. Especially when I’m writing type-system code or similar things, it can be distracting when it keeps trying to author stuff it clearly doesn’t understand. I expect this too will improve over time—and I’ve noticed that while, in the beginning, it tends to guess very wrong, over time it tends to guess better. I’m not sure what inputs and context are being fed by the LLM in the background but it’s evident that it can come to see patterns even for relatively subtle things. ←

- [Don Marti: the #Eurostack, so hot right now](#) (2025/02/08 00:00)

Is it just me or is it all about Europe right now? Put on some Kraftwerk and follow along I guess. Fedora Chooses Forgejo! This is GitHub-like project hosting software with version control, issues, pull requests, all the usual stuff. I have a couple of small projects on Codeberg, which is the (EU) hosted nonprofit instance and it works fine as far as I can tell. Also a meissa GmbH presentation at FOSDEM 2025 You know X, Facebook, Xing, SourceForge? What about GitHub? It is time to de-risk OpenSource engagement! Lots more Europe-hosted SaaS, too. Baldur Bjarnason has more info on Todo notes as a storm approaches The Sovereign Tech Agency is supporting some Linux plumbing: Arun Raghavan: PipeWire ♥ Sovereign Tech Agency. The northern German state of Schleswig-Holstein is moving 30,000 PCs from Microsoft Windows and Office to Linux and LibreOffice: LibreOffice at the Univention Summit 2025 I know, I know, government in Germany goes desktop Linux is the hey, Rocky, watch me pull a rabbit out of my hat of IT, but this time they’re not up against Microsoft in its prime, they’re up against a new generation that can’t open their old files, while LibreOffice can. They Said It Couldn’t Be Done by Pierre-Carl Langlais, Anastasia Stasenko, and Catherine Arnett. These represent the first ever models trained exclusively on open data, meaning data that are either non-copyrighted or are published under a permissible license. Trained on the Jean Zay supercomputer. Related: Pirate Libraries Are Forbidden Fruit for AI Companies. But at What Cost? Scott Locklin lists Examples of group madness in technology. One of the worst arguments I hear is that thing X is inevitable because the smart people are doing it. As I’ve extensively documented over the last 15 years on this blog, smart people in groups are not smart and are even more subject to crazes and mob behavior as everyone else. Not a European product: Framework Laptop’s RISC-V board for open source diehards is available for \$199 but there is a Europe angle here. European Union Seeks Chip Sovereignty Using RISC-V - EE Times, RISC-V Summit Europe. RISC-V holds significance for Europe due to its potential to foster innovation, enhance technological sovereignty, and stimulate economic growth within the region. By embracing RISC-V, European countries can reduce their dependency on foreign technologies and proprietary architectures, thereby enhancing their autonomy in critical sectors such as telecommunications, cybersecurity, and data processing. Also international, not Europe-specific: Postgres full-text search is Good Enough! by Rachid Belaid. (But there is a tech autonomy angle, and an active PostgreSQL Europe, so for practical purposes PostgreSQL is part of the Eurostack.) Good advice from tante/Jürgen Geuter: Innovation is a distraction The demand for more Innovation (and sometimes even the request for more research) has become a way to legitimize not doing anything. A way to say the unpleasant solutions we have are not perfect but in the future there might be a magic solution that doesn’t bother us and everyone gets a fucking unicorn. Marloes de Koning interviews Cristina Caffarra. ‘We have to get to work and put Europe first. But we are late. Terribly late’ You really don’t have to buy everything in Europe, says the competition expert, who is familiar with the criticism that the American supply is simply superior. But start with 30 percent of your procurement budget in Europe. That already makes a huge difference. (That seems like an easy target.

Not only are way more than 30 percent of the European Alternatives up to a servicable level by now, but unfortunately a lot of the legacy US vendors are having either quality or compliance problems, or both. The risks, technical and otherwise, keep going up. Greg Nojeim and Silvia Lorenzo Perez cover Trump's Sacking of PCLOB Members Threatens Data Privacy Aside from its importance in protecting civil liberties, the PCLOB cannot play its key role in enforcing U.S. obligations under the EU-U.S. Data Privacy Framework (DPF) while it lacks a quorum of members. The European Commission would lose a key oversight tool for which it bargained, and the adequacy decision that it issued to support the DPF could be struck down under review at the Court of Justice of the European Union (CJEU), which struck down two predecessor EU-U.S. data privacy arrangements, the Safe Harbor Agreement and the Privacy Shield. Karl Bode writes, Apple Has To Pull Its "AI" News Synopses Because They Were Routinely Full Of Shit (If the features unavailable in Europe are problematic anyway...) Sarah Perez covers Report: Majority of US teens have lost trust in Big Tech. Common Sense says that 64% of surveyed U.S. teens don't trust Big Tech companies to care about their mental health and well-being and 62% don't think the companies will protect their safety if it hurts profits. Over half of surveyed U.S. teens (53%) also don't think major tech companies make ethical and responsible design decisions (think: the growing use of dark patterns in user interface design meant to trick, confuse, and deceive. A further 52% don't think that Big Tech will keep their personal information safe and 51% don't think the companies are fair and inclusive when considering the needs of different users. (What if the Eurostack becomes the IT version of those European food brands that sell well in other countries too?) and more... Instead of F-35, Portugal turns to Europe in search of new fighter, In defense tech, Lithuania's sovereign VC fund is one step ahead, Ukraine prepares to showcase game-changing defense tech innovations at February forum in Kyiv, Open source LLMs hit Europe's digital sovereignty roadmap

- [Mozilla Thunderbird: Thunderbird Desktop Release Channel Will Become Default in March 2025](#) (2025/02/06 19:32)

UPDATE (March 4, 2025): The Release Channel is now default! See our update post on how to make the switch with a manual install and what's new in 136. We have an exciting announcement! Starting with the 136.0 release in March 2025, the Thunderbird Desktop Release channel will be the default download. If you're not already familiar with the Release channel, it will be a supported alternative to the ESR channel. It will provide monthly major releases instead of annual major releases. This provides several benefits to our users: Frequent Feature Updates: New features will potentially be available each month, versus the annual Extended Support Release (ESR). Smoother Transitions: Moving from one monthly release to the next will be less disruptive than updating between ESR versions. Consistent Bug Fixes: Users will receive all available bug fixes, rather than relying on patch uplifts, as is the case with ESR. We've been publishing monthly releases since 124.0. We added the Thunderbird Desktop Release Channel to the download page on Oct 1st, 2024. The next step is to make the release channel an officially supported channel and the default download. We don't expect this step alone to increase the population significantly. We're exploring additional methods to encourage adoption in the future, such as in-app notifications to invite ESR users to switch. One of our goals for 2025 is to increase daily active installations on the release channel to at least 20% of the total installations. At last check, we had 29,543 daily active installations on the release channel, compared to 20,918 on beta, and 5,941 on daily. The release channel installations currently account for 0.27% of the 10,784,551 total active installations tracked on stats.thunderbird.net. To support this transition and ensure stability for monthly releases, we're implementing several process improvements, including: Pre-merge freezes: A 4-day soft code freeze of comm-central before merging into comm-beta. We continue to bake the week-long post-merge freeze of the release channel into the schedule. Pre-merge reviews: We evaluate changes prior to both merges (central to beta and beta to release) where risky changes can be reverted. New uplift template: A new and more thorough uplift template. For more details on these release process details, please see the Release section of the developer docs. For more details on

scheduling, please see the Thunderbird Releases & Events calendar. Thank you for your support with this exciting step for Thunderbird. Let's work together to make the Release channel a success in 2025! Regards, Corey Bryant Manager, Release Operations | Mozilla Thunderbird
Note: This blog post was taken from Corey's original announcement at our Thunderbird Planning mailing list The post Thunderbird Desktop Release Channel Will Become Default in March 2025 appeared first on The Thunderbird Blog.

- [About:Community: FOSDEM 2025: A Celebration of Open Source Innovation](#) (2025/02/06 12:25)

Brussels came alive this weekend as Mozilla joined FOSDEM 2025, Europe's premier open-source conference. FOSDEM wasn't just another tech gathering. It is a representation of a vibrant community, open source innovation, and the spirit of collaboration. And we're proud of being part of this amazing event since its inception. This year, FOSDEM is celebrating its 25th anniversary. And unlike previous years' gloomy weather, this year, we were blessed with surprising sunshine, almost as if the universe was applauding a quarter-century of open-source achievements. As for Mozilla, our presence this year was extra special as we introduced our new brand. Over the weekend, we ran a bingo challenge in Mozilla's and Thunderbird's stands, where participants could play to win exclusive Mozilla t-shirts and many more special swag. It was a really fun way to introduce many projects from across pan-Mozilla. We also showcased a sneak peek of Firefox Nightly's new tab group feature in the Mozilla booth and gave away 2300 free cookies to participants on Saturday. Here are some more highlights from our presence this year: Highlights from Saturday Mozilla engineering manager Marco Casteluccio presented a talk about the usage of LLM's to support Firefox developers with code review in the main track. Firefox engineer Valentin Gosu also presented a talk in the DNS track about his journey on using the getaddrinfo API in Firefox. Another Firefox engineer who's working on Firefox Profiler, Nazim Can Altinova also presented a talk in the Web Performance track. It's also worth mentioning that the Web Performance devroom was co-run by some Mozillians. Danny Colin, one of Mozilla's active contributors, hosted a WebExtension BoF session featuring representatives from Mozilla Firefox (Rob Wu & Simeon Vincent) and Google Chrome's extensions team (Oliver Dunk). This was the first time the team ran a Birds Of a Feather session, and it's very likely that we're going to do the same next year. Danny Colin also hosted the Community Gathering where old and new contributors got together to discuss the future of Mozilla's community. It was really nice to have an interactive session like this where all of us can share our perspective, so thank you to all of you who attended the session! Highlights from Sunday Mitchell Baker kicked off Sunday with a keynote session that offered a thought-provoking exploration of Free/Libre Open Source Software (FLOSS) in the age of artificial intelligence and demonstrated how Mozilla plays a role in defining principled approach to AI that prioritizes transparency, ethics, and community-driven innovation. It was a perfect opening for the talks that we presented at the Mozilla devroom later that day. Around the same time as Mitchell's session, Mozilla engineer Max Inden also delivered a presentation in the Network devroom, showcasing various techniques the Firefox team uses to enhance Firefox performance. Then on the second half on Sunday, we also hosted the Mozilla devroom where we covered a wide range of Mozilla's latest innovations from Mythbusting to Mozilla's AI innovations and Firefox developments. Recordings will be available soon at FOSDEM's website and via our YouTube channel. So stay tuned! We're grateful for the enthusiasm, conversations, and curiosity of attendees at FOSDEM 2025. And big thanks to our amazing volunteers and Mozillians for co-hosting our booth and the Mozilla devroom this year. We sure had a blast, and we can't wait to see you again next year!

- [The Rust Programming Language Blog: crates.io: development update](#) (2025/02/05 00:00)

Back in July 2024, we published a blog post about the ongoing development of crates.io. Since then, we have made a lot of progress and shipped a few new features. In this blog post, we want to give you an update on the latest changes that we have made to crates.io. Crate deletions In RFC #3660 we proposed a new feature that allows crate owners to delete their crates from crates.io under certain conditions. This can be useful if

you have published a crate by mistake or if you want to remove a crate that is no longer maintained. After the RFC was accepted by all team members at the end of August, we began implementing the feature. We created a new API endpoint DELETE /api/v1/crates/:name that allows crate owners to delete their crates and then created the corresponding user interface. If you are the owner of a crate, you can now go to the crate page, open the "Settings" tab, and find the "Delete this crate" button at the bottom. Clicking this button will lead you to a confirmation page telling you about the potential impact of the deletion and requirements that need to be met in order to delete the crate: As you can see from the screenshot above, a crate can only be deleted if either: the crate has been published for less than 72 hours or the crate only has a single owner, and the crate has been downloaded less than 500 times for each month it has been published, and the crate is not depended upon by any other crate on crates.io. These requirements were put in place to prevent abuse of the deletion feature and to ensure that crates that are widely used by the community are not deleted accidentally. If you have any feedback on this feature, please let us know! OpenAPI description

Around the holiday season we started experimenting with generating an OpenAPI description for the crates.io API. This was a long-standing request from the community, and we are happy to announce that we now have an experimental OpenAPI description available at <https://crates.io/api/openapi.json>! Please note that this is still considered work-in-progress and e.g. the stability guarantees for the endpoints are not written down and the response schemas are also not fully documented yet. You can view the OpenAPI description in e.g. a Swagger UI at <https://petstore.swagger.io/> by putting <https://crates.io/api/openapi.json> in the top input field. We decided to not ship a viewer ourselves for now due to security concerns with running it on the same domain as crates.io itself. We may reconsider whether to offer it on a dedicated subdomain in the future if there is enough interest. The OpenAPI description is generated by the utoipa crate, which is a tool that can be integrated with the axum web framework to automatically generate OpenAPI descriptions for all of your endpoints. We would like to thank Juha Kukkonen for his great work on this tool! Support form and "Report Crate" button

Since the crates.io team is small and mostly consists of volunteers, we do not have the capacity to manually monitor all publishes. Instead, we rely on you, the Rust community, to help us catch malicious crates and users. To make it easier for you to report suspicious crates, we added a "Report Crate" button to all the crate pages. If you come across a crate that you think is malicious or violates the code of conduct or our usage policy, you can now click the "Report Crate" button and fill out the form that appears. This will send an email to the crates.io team, who will then review the crate and take appropriate action if necessary. Thank you to crates.io team member @eth3lbert who worked on the majority of this. If you have any issues with the support form or the "Report Crate" button, please let us know. You can also always email us directly at help@crates.io if you prefer not to use the form.

Publish notifications We have added a new feature that allows you to receive email notifications when a new version of your crate is published. This can be useful in detecting unauthorized publishes of your crate or simply to keep track of publishes from other members of your team. This feature was another long-standing feature request from our community, and we were happy to finally implement it. If you'd prefer not to receive publish notifications, then you can go to your account settings on crates.io and disable these notifications.

Miscellaneous These were some of the more visible changes to crates.io over the past couple of months, but a lot has happened "under the hood" as well. RFC #3691 was opened and accepted to implement "Trusted Publishing" support on crates.io, similar to other ecosystems that adopted it. This will allow you to specify on crates.io which repository/system is allowed to publish new releases of your crate, allowing you to publish crates from CI systems without having to deal with API tokens anymore. Slightly related to the above: API tokens created on crates.io now expire after 90 days by default. It is still possible to disable the expiry or choose other expiry durations though. The crates.io team was one of the first projects to use the diesel database access library, but since that only supported synchronous execution it was sometimes a little awkward to use in our codebase, which was increasingly moving into

an async direction after our migration to axum a while ago. The maintainer of diesel, Georg Semmler, did a lot of work to make it possible to use diesel in an async way, resulting in the diesel-async library. Over the past couple of months we incrementally ported crates.io over to diesel-async queries, which now allows us to take advantage of the internal query pipelining in diesel-async that resulted in some of our API endpoints getting a 10-15% performance boost. Thank you, Georg, for your work on these crates! Whenever you publish a new version or yank/unyank existing versions a couple of things need to be updated. Our internal database is immediately updated, and then we synchronize the sparse and git index in background worker jobs. Previously, yanking and unyanking a high number of versions would each queue up another synchronization background job. We have now implemented automatic deduplication of redundant background jobs, making our background worker a bit more efficient. The final big, internal change that was just merged last week is related to the testing of our frontend code. In the past we used a tool called Mirage to implement a mock version of our API, which allowed us to run our frontend test suite without having to spin up a full backend server. Unfortunately, the maintenance situation around Mirage had lately forced us to look into alternatives, and we are happy to report that we have now fully migrated to the "Industry standard API mocking" package msw. If you want to know more, you can find the details in the "small" migration pull request. Feedback We hope you enjoyed this update on the development of crates.io. If you have any feedback or questions, please let us know on Zulip or GitHub. We are always happy to hear from you and are looking forward to your feedback!

- [Firefox Developer Experience: Firefox WebDriver Newsletter 135](#) (2025/02/04 20:52)

WebDriver is a remote control interface that enables introspection and control of user agents. As such it can help developers to verify that their websites are working and performing well with all major browsers. The protocol is standardized by the W3C and consists of two separate specifications: WebDriver classic (HTTP) and the new WebDriver BiDi (Bi-Directional). This newsletter gives an overview of the work we've done as part of the Firefox 135 release cycle. Contributions Firefox is an open source project, and we are always happy to receive external code contributions to our WebDriver implementation. We want to give special thanks to everyone who filed issues, bugs and submitted patches. In Firefox 135, several contributors managed to land fixes and improvements in our codebase: Liam (ldebeasi) added a new argument to `browsingContext.captureScreenshot`. Patrick (pshannon) added internal logs to help us investigate incomplete network events. Spencer (spenneth) created a helper to check if a browsing context is a top level one and reduced duplication in our logic to list opened windows. WebDriver code is written in JavaScript, Python, and Rust so any web developer can contribute! Read how to setup the work environment and check the list of mentored issues for Marionette, or the list of mentored JavaScript bugs for WebDriver BiDi. Join our chatroom if you need any help to get started! General Improved user interactions simulation To make user events more realistic and better simulate real user interactions in the browser, we have moved the action sequence processing of the Perform Actions commands in both Marionette and WebDriver BiDi from the content process to the parent process. While events are still sent synchronously from the content process, they are now triggered asynchronously via IPC calls originating from the parent process. Due to this significant change, you might experience some regressions. If you encounter any issues, please file a bug for the Remote Agent. If the regressions block test execution, you can temporarily revert to the previous behavior by setting the Firefox preference `remote.events.async.enabled` to `false`. With the processing of actions now handled in the parent process the following issues were fixed as well: We now support proper queuing of action sequences without race conditions. This is particularly important for WebDriver BiDi's `input.performActions` command, which can be called multiple times in parallel and must execute the enqueued actions sequentially. When dispatching actions, the input cancel list is now correctly updated only after the action has been successfully dispatched. Previously, if an action failed to execute, a reverse action could be left in place, leading to unexpected side effects when resetting

the state of the input source. When performing actions, individual actions are now retried during dispatch, particularly in situations where a single action triggers a navigation that replaces the current browsing context. When performing actions, a `TypeError` error no longer occurs if an action (not the last one) in the action chain closed the window, and the remaining actions were still being dispatched. `WebDriver BiDi New`: format argument for `browsingContext.captureScreenshot` Thanks to Liam's work, the `browsingContext.captureScreenshot` command now supports the format argument. It allows clients to specify different file formats ("`image/png`" and "`image/jpeg`" are currently supported) and define the compression quality for screenshots. The argument should follow the `browsingContext.ImageFormat` type, with a "type" property which is expected to be a string, and an optional "quality" property which can be a float between 0 and 1. -> { "method": "`browsingContext.captureScreenshot`", "params": { "context": "6b1cd006-96f0-4f24-9c40-a96a0cf71e22", "origin": "document", "format": { "type": "image/jpeg", "quality": 0.1 } }, "id": 3 } <- { "type": "success", "id": 3, "result": { "data": "iVBORw0KGgoAAAANSUUhEUgAA[...].8AbxR064eNvgIAAAAASUVORK5CYII=" } } Bug Fixes Fixed a bug where some Marionette and `WebDriver BiDi` commands that rely internally on a `requestAnimationFrame` being emitted before returning would hang if the current browsing context was navigated during their execution.

- [Mozilla Privacy Blog: Navigating the Future of Openness and AI Governance: Insights from the Paris Openness Workshop](#) (2025/02/04 15:44) In December 2024, in the lead up to the AI Action Summit, Mozilla, Fondation Abeona, École Normale Supérieure (ENS) and the Columbia Institute of Global Politics gathered at ENS in Paris, bringing together a diverse group of AI experts, academics, civil society, regulators and business leaders to discuss a topic increasingly central to the future of AI: what does openness mean and how it can enable trustworthy, innovative, and equitable outcomes? The workshop followed the Columbia Convenings on Openness and AI, that Mozilla held in partnership with Columbia University's Institute of Global Politics. These gatherings, held over the course of 2024 in New York and San Francisco, have brought together over 40 experts to address what "openness" should mean in the AI era. Over the past two years, Mozilla has mounted a significant effort to promote and defend the role of openness in AI. Mozilla launched `Mozilla.ai`, an initiative focused on ethical, open-source AI tools, and supported small-scale, localized AI projects through its Builders accelerator program. Beyond technical investments, Mozilla has also been a vocal advocate for openness in AI policy, urging governments to adopt regulatory frameworks that foster competition and accountability while addressing risks. Through these initiatives, Mozilla is shaping a future where AI development aligns with public interest values. This Paris Openness workshop discussion — part of the official 'Road to the Paris AI Summit' taking place in February 2025 — looked to bring together the European AI community and form actionable recommendations for policymakers. While it embraced healthy debate and disagreement around issues such as definitions of openness in AI, there was nevertheless broad agreement on the urgency of crafting collective ideas to advance openness while navigating an increasingly complex commercial, political and regulatory landscape. The stakes could not be higher. As AI continues to shape our societies, economies, and governance systems, openness emerges as both an opportunity and a challenge. On one hand, open approaches can expand access to AI tools, foster innovation, and enhance transparency and accountability. On the other hand, they raise complex questions about safety and misuse. In Europe, these questions intersect with transformative regulatory frameworks like the EU AI Act, which seeks to ensure that AI systems are both safe and aligned with fundamental rights. As in software development, the goal of being 'open' in AI is a crucial one. At its heart, openness, we were reminded in the discussion, is a holistic outlook. For AI in particular it is a pathway to getting to a more pluralistic tool - one that can be more transparent, contextual, participatory and culturally appropriate. Each of these goals however contain natural tensions within them. A central question of this most recent dialogue challenged participants on the best ways to build with safety in

mind while also embracing openness. The day was broken down into two workshops that examined these questions from a technical and policy standpoint. Running through both of the workshops was the thread of a persistent challenge: the multifaceted nature of the term openness. In the policy context, the term “open-source” can be too narrow, and at times, it risks being seen as an ideological stance rather than a pragmatic tool for addressing specific issues. To address this, many participants felt openness should be framed as a set of components — including open models, data, and tools — each of which has specific benefits and risks. Examining Technical Perspectives on Openness and Safety A significant concern for many in the open-source community is getting access to the best existing safety tools. Despite the increasing importance of AI safety, many researchers can find it difficult or expensive to access tools to help identify and address AI risks. In particular the discussion surfaced an increasing tension between some researchers and startups who have found it difficult to access datasets of known CSAM (Child Sexual Abuse Material) hashtags. Accessing these data sets could help mitigate misuse or clean training datasets. The workshop called for broader sharing of safety tools and more support for those working at the cutting edge of AI development. More widely, some participants were frustrated by perceptions that open source AI development is not bothered by questions of safety. They pointed out that, especially when it comes to regulation, focusing on questions of safety makes them even more competitive. Discussing Policy Implications of Openness in AI Policy discussions during the workshop focused on the economic, societal, and regulatory dimensions of openness in AI. These ranged over several themes, including: Challenging perceptions of openness: There is a clear need to change the narrative around openness, especially in policymaking circles. The open-source community must both act as a community and present itself as knowledgeable and solution-oriented, demonstrating how openness can be a means to advancing the public interest — not an abstract ideal. As one participant pointed out, openness should be viewed as a tool for societal benefit, not as an end in itself. Tensions between regulation and innovation are misleading: As one of the first regulatory frameworks on AI to be drafted, many people view the EU’s AI Act as a test bed to get to smarter AI regulation. While there is a widespread characterisation of regulation obstructing innovation, some participants highlighted that this can be misleading — many new entrants seek out jurisdictions with favourable regulatory and competition policies that level the playing field. A changing U.S. Perspective: In the United States, the open-source AI agenda has gained significant traction, particularly in the wake of incidents like the Llama leaks, which showed that many of the feared risks associated with openness did not materialize. Significantly, the U.S. National Telecommunications and Information Administration emphasized the benefits of open source AI technology and introduced a nuanced view of safety concerns around open-weight AI models. Many participants also agreed that policymakers, many of whom are not deeply immersed in the technicalities of AI, need a clearer framework for understanding the value of openness. Considering the focus of the upcoming Paris AI Summit, some participants felt one solution could lie in focusing on public interest AI. This concept resonates more directly with broader societal goals while still acknowledging the risks and challenges that openness brings. Recommendations Embracing openness in AI is non-negotiable if we are to build trust and safety; it fosters transparency, accountability, and inclusive collaboration. Openness must extend beyond software to broader access to the full AI stack, including data and infrastructure, with a governance that safeguards public interest and prevents monopolization. It is clear that the open source community must make its voice louder. If AI is to advance competition, innovation, language, research, culture and creativity for the global majority of people, then an evidence-based approach to the benefits of openness, particularly when it comes to proven economic benefits, is essential for driving this agenda forward. Several recommendations for policymakers also emerged. Diversify AI Development: Policymakers should seek to diversify the AI ecosystem, ensuring that it is not dominated by a few large corporations in order to foster more equitable access to AI technologies and reduce monopolistic control. This should be approached holistically, looking at everything from procurement to compute

strategies. Support Infrastructure and Data Accessibility: There is an urgent need to invest in AI infrastructure, including access to data and compute power, in a way that does not exacerbate existing inequalities. Policymakers should prioritize distribution of resources to ensure that smaller actors, especially those outside major tech hubs, are not locked out of AI development. Understand openness as central to achieving AI that serves the public interest. One of the official tracks of the upcoming Paris AI Action Summit is Public Interest AI. Increasingly, openness should be deployed as a main route to truly publicly interested AI. Openness should be an explicit EU policy goal: As one of the furthest along in AI regulatory frameworks the EU will continue to be a testbed for many of the big questions in AI policy. The EU should adopt an explicit focus on promoting openness in AI as a policy goal. We will be raising all the issues discussed while at the AI Action Summit in Paris. The organizers hope to host another set of these discussions following the conclusion of the Summit in order to continue working with the community and to better inform governments and other stakeholders around the world. The list of participants at the Paris Openness Workshop is below: Linda Griffin – VP of Global Policy, Mozilla Udbhav Tiwari – Director, Global Product Policy, Mozilla Camille François – Researcher, Columbia University Tanya Perelmuter – Co-founder and Director of Strategy,, Fondation Abeona Yann Lechelle – CEO, Probabl Yann Guthmann – Head of Digital Economy, Department at the French Competition Authority Adrien Basdevant – Tech lawyer, Entropy Law Andrzej Neugebauer – AI Program Director, LINAGORA Thierry Poibeau – Director of Research, CNRS, ENS Nik Marda – Technical Lead for AI Governance, Mozilla Andrew Strait – Associate Director, Ada Lovelace Institute (UK) Paul Keller – Director of Policy, Open Future (Netherlands) Guillermo Hernandez – AI Policy Analyst, OECD Sandrine Elmi Hersi – Unit Chief of “Open Internet”, ARCEP The post Navigating the Future of Openness and AI Governance: Insights from the Paris Openness Workshop appeared first on Open Policy & Advocacy.

From:

<https://wiki.tromjaro.alexio.tf/> - **TROMjaro wiki**

Permanent link:

<https://wiki.tromjaro.alexio.tf/doku.php?id=news:planet:mozilla>

Last update: **2021/10/30 11:41**

