Planet Mozilla - Latest News

- The Mozilla Blog: Try Tab Notes in Firefox to leave a note on any page (2026/03/23 19:00)
  Don’t remember why you have all those webpages open? Now you can leave yourself a note for any tab. Tab Notes — our latest experimental feature in Firefox — are designed to help you remember, reflect, and pick up where you left off on the web by letting you attach a short note to a webpage. Indicated by a sticky note icon and visible when hovering over tabs, Tab Notes notes remain connected to the page’s URL until you delete them. Your notes are yours. They remain private and accessible only to you. Firefox stores them locally in your browser and doesn’t send them to Mozilla. Starting March 24, you can try Tab Notes by following these steps: Go to Settings. Navigate to Firefox Labs (or enter about:preferences#experimental in the address bar). Tick the box beside Tab notes. Now you’re all set! Just right-click or hover over a tab and choose “Add Note” to create your first tab note! This work is inspired by user research that we conducted last year, which explored how people resume tasks after interruptions. One key insight we learned is that when we are interrupted, even a small reminder or message can significantly improve our ability to resume a task. Many people use a variety of analog (e.g., sticky notes) and digital tools (e.g., note-taking apps) for these purposes as well, and Tab Notes are our exploration of that idea in a practical, lightweight way. These notes are easy to create, edit, and delete. This is an early experiment, part of the Firefox Labs program. We are eager for feedback, which you can share on Mozilla Connect or by filing a ticket in Bugzilla. Take control of your internet Download Firefox The post Try Tab Notes in Firefox to leave a note on any page appeared first on The Mozilla Blog.
- The Mozilla Blog: Split View in Firefox: Two tabs side by side, right where you need them (2026/03/23 19:00)
  Much of what we do on the web involves looking at more than one thing at a time – booking tickets while checking your calendar, taking notes as you go through a report, or comparing options before making a purchase. The web is inherently multidimensional. For years, browsing this way meant bouncing back and forth between multiple open tabs, or spinning up multiple windows and using other tools to organize them side-by-side. The new Split View feature makes these moments easier. It lets you place two tabs next to each other in the same Firefox window so you can see both at once and keep the context you need right in front of you. Split View is available to all Firefox users starting with Firefox 149, rolling out on March 24. If you’d like to give it a go: Make sure you’ve got the latest version of Firefox. Right-click a tab and choose Add Split View. You can also select two tabs, right-click, and choose Open in Split View. How the Firefox team uses Split View The team behind Split View has been using it actively over the past few months, and a few workflows quickly stood out. Here are some of the ways people on our team have been using it: Planning and comparing Sometimes, you just need two things visible at once. Gabriel: I’ve been using Split View to plan camping trips. I open a map on one side and a campsite booking page on the other. This makes it easy to explore locations and check availability without constantly switching tabs. Everyday tasks Split View is also helpful for small administrative tasks, the kind that involve copying information from one place to another. Jonathan: I used Split View while filing my taxes. All my documents – W-2s and other forms – were online, so I kept them open on one side while filling things out on the FreeTaxUSA site on the other. Having both visible made the process much easier. Note-taking Ania: I often use Split View when reading and writing at the same time. I’ll keep a PDF or article open on one side and take notes on the other as I go. Recently, I’ve been using this setup while preparing notes for my reading group. It helps me stay focused and quickly organize what I want to

share. What’s next for Split View We built Split View to support the way people naturally move through information on the web – comparing, referencing and writing along the way. This first version focuses on making the most common side-by-side workflows easy. If you try it, we’d love your feedback on how it fits into your day-to-day browsing and what would make it even more useful. Take control of your internet Download Firefox The post Split View in Firefox: Two tabs side by side, right where you need them appeared first on The Mozilla Blog.

- Mozilla Privacy Blog: Competition, Innovation, and the Future of the Web – Why Independent Browser Engines Matter (2026/03/23 15:13) Gecko matters because it ensures there’s an independent voice shaping how the internet evolves. Without Gecko, the landscape would be dominated by Apple and Google alone. From accessing information, communicating with others, shopping, working, learning, and entertainment, the vast majority of our time online is spent within a browser. While there are many browsers out there, there are only a few browser engines, the technology necessary to render the data that makes up the web as websites we can use. Browser engines are among the most complex and consequential pieces of infrastructure on the modern internet. They determine how web standards are implemented, how security and privacy protections are enforced, and which actors ultimately shape the evolution of the web. As the internet increasingly fragments into walled gardens, and as new technologies like artificial intelligence (AI) are integrated directly into browsers, the influence of browser engines is only growing. When innovation is built on a single dominant engine, it concentrates technical and economic power, narrows choice, and risks steering the web toward the priorities of a few large platforms rather than the public interest. Gecko is Mozilla’s browser engine that powers Firefox. It is one of only three widely used engines and the only independent browser engine. In other words, it is not governed by a company that also runs an operating system to distribute their own browser. Why Browser Engines Matter Browser engines (not to be confused with search engines) are the lesser-known technology powering your web browsers. As the core software layer responsible for interpreting and rendering web content, browser engines play the fundamental role of turning HTML, CSS, and JavaScript into webpages users can interact with. While browsers are user-facing products, engines are the layer where structural decisions about the web are made. Examples include privacy and security protections, performance characteristics, and the support of APIs. Browser engines are at the heart of the web. Gecko and the Browser Monoculture The browser engine landscape is highly concentrated. In 2013, there were five major browser engines. In 2026, there are only three left: Apple’s WebKit (which companies are required to use to build on iOS), Google’s Blink, and Mozilla’s Gecko. Gecko is the only remaining independent browser engine and it powers Firefox. When engine diversity declines, so does the practical ability to challenge dominant business models or introduce alternative implementations that can put users first through security, privacy, or other features. There are only three major browser engines left — Apple’s WebKit, Google’s Blink and Gecko from Mozilla. Apple’s WebKit mainly runs on Apple devices, making Gecko the only cross-platform challenger to Blink. This concentration increasingly risks hard-coding a single company’s technical assumptions into the future of the web. Market pressures often turn standards-compliant but differing implementation choices into “bugs” that need fixing. As both human and AI-driven browsing expand in use, choices about API implementation, data access, and security boundaries at the browser engine level become even more critical. A monoculture at the engine layer could extend to producing a monoculture in AI browsing experiences as well. Maintaining an Independent Browser Engine Allows Mozilla to be More User-centric Gecko, as an independent browser engine, tangibly allows Mozilla to build and operate in a way that is aligned with our mission: keeping the web open, secure, privacy-first, and accessible to everyone. It ensures that Mozilla is not only advocating for these principles but actively building the underlying infrastructure that makes them possible. Through Gecko, we have the freedom to design and ship features based on what is best for users, rather than what is easiest or most profitable within another company’s technology stack. In practice, this enables us to: Introduce privacy and security protections that go beyond industry defaults, such as

strong cross-site tracking protections and anti-fingerprinting measures. Experiment with new user interface designs and customization options that give people more control over how they use the web. Build features that reflect Mozilla's mission-driven priorities, even when they diverge from dominant commercial models. If a small number of vertically integrated companies (AI assistants, search, operating systems, ads) completely control browser engines, then competition, transparency, and user choice on the open web will be much harder to achieve. They will have strong incentives to favour their own services, limit interoperability, and steer defaults and standards to their advantage. Maintaining an independent engine also lowers barriers for others. Newer entrants to the browser space can rely on interoperability as defined in specifications. If they are not building their own engine, building on Gecko can help sustain a more competitive browser ecosystem. Engine diversity at this foundational layer enables innovation, which is shaped by multiple actors and multiple visions, rather than it being dictated by a single dominant platform. Browser Engine Plurality Ensures Tech is Built For People, Not Shareholders In an era defined by platform consolidation and AI-driven change, browser engines can't be treated as invisible infrastructure. Independent engines like Gecko provide a structural counterbalance. Browser engine plurality is needed to ensure competition, transparency, and technology built for people, not shareholders. As governments increasingly focus on security, resilience and sustainable growth, browser engine competition has a central role to play in avoiding single points of vulnerability or failure. Meaningful competition and a focus on open source approaches help ensure that economies are not locked into a single company's infrastructure and that governments, companies, and people retain real choice over where to build and how to optimize for their needs. Mozilla has long engaged with policymakers and regulators on the importance of competition and openness at the browser and engine layer. As the web and broader technology landscape continue to evolve, especially in the face of AI, we will continue to advance policies that protect engine diversity, promote fair competition, and ensure the web evolves in the public interest. The post Competition, Innovation, and the Future of the Web – Why Independent Browser Engines Matter appeared first on Open Policy & Advocacy.

- Niko Matsakis: Maximally minimal view types, a follow-up (2026/03/22 16:52)
  A short post to catalog two interesting suggestions that came in from my previous post, and some other related musings. Syntax with . It was suggested to me via email that we could use . to eliminate the syntax ambiguity: let place = &mut self.{statistics}; Conceivably we could do this for the type, like: fn method(mp: &mut MessageProcessor.{statistics}, ...) and in self position: fn foo(&mut self.{statistics}) {} I have to sit with it but...I kinda like it? I'll use it in the next example to try it on for size. Coercion for calling public methods that name private types In my post I said that if you hvae a public method whose self type references private fields, you would not be able to call it from another scope: mod module { #[derive(Default)] pub struct MessageProcessor { messages: Vec<String>, statistics: Statistics, } pub struct Statistics { .. } impl MessageProcessor { pub fn push_message(&mut self.{messages}, // -------- private field message: String,) {} } } pub fn main() { let mp = MessageProcessor::default(); mp.push_message(format!("Hi")); // ------------ Error! } The error arises from desugaring push_message to a call that references private fields: MessageProcessor::push_message(&mut mp.{messages}, // -------- not nameable here format!("Hi"),) I proposed we could lint to avoid this situation. But an alternative was proposed where we would say that, when we introduce an auto-ref, if the callee references local variables not visible from this point in the program, we just borrow the entire struct rather than borrowing specific fields. So then we would desugar to: MessageProcessor::push_message(&mut mp, // -- borrow the whole struct format!("Hi"),) If we then say that &mut MessageProcessor is coercable to a &mut MessageProcessor.{messages}, then the call would be legal. Interestingly, the autoderef loop already considers visibility: if you do a.foo, we will deref until we see a foo field visible to you at the current point. Oh and a side note, assigning etc This raises an interesting question I did not discuss. What happens when you write a value of a type like MessageProcessor.{messages}? For

example, what if I do this: fn swap_fields(mp1: &mut MessageProcessor.{messages}, mp2: &mut MessageProcessor.{messages},) { std::mem::swap(mp1, mp2); } What I expect is that this would just swap the selected fields (messages, in this case) and leave the other fields untouched. The basic idea is that a type MessageProcessor.{messages} indicates that the messages field is initialized and accessible and the other fields must be completely ignored. Another possible future extension: moved values This represents another possible future extension. Today if you move out of a field in a struct, then you can no longer work with the value as a whole: impl MessageProcessor { fn example(mut self) { // move from self.statistics std::mem::drop(self.statistics); // now I cannot call this method, // because I can't borrow `self`: self.push_message(format!("Hi again")); } } But with selective borrowing, we could allow this, and you could even return “partially initialized” values: impl MessageProcessor { fn take_statistics(mut self,) -> MessageProcessor.{messages} { std::mem::drop(self.statistics); self } } That’d be neat.

- Jonathan Almeida: Create new revisions in Jujutsu with multiple heads (2026/03/22 00:00)
  It was one of those "ah ha!" moments for me when I finally used it. Chris Krycho covers the concept of megamerges with this diagram: m --- n / \ a -- b -- c -- [merge] -- [wip] \ / w --- x I've found a more realistic example that best relates to my natural workflow: implementing feature (A) benefitted from having the changes of another tooling patch upgrade (B), that lead to discovering and fixing a bug (C). (B) m ----- n / \ (A) a -- b ---------- [merge] --- y -- z \ / \ (C) ------------------- ----- [merge] -- w -- x \ / ------------------------------ In this case, trying to separate these into distinct streams of work is quite logically, but we also don't need to leave them unlinked so that they can benefit from each other. This is what my jj log ended up looking like: @ oppmsuvz jxxxxxxxxxxx@gmail.com 2026-03-22 00:34:10 firefox@ 05259417 │ Bug xxxxxxx - Simplify the tests ○ ultowtnr jxxxxxxxxxxx@gmail.com 2026-03-22 00:34:04 100c4cce │ Bug xxxxxxx - Include private flag in ShareData ○ lorusmuo jxxxxxxxxxxx@gmail.com 2026-03-21 20:19:30 905b0460 ├┐ (empty) (no description set) │ ○ sumqskuu jxxxxxxxxxxx@gmail.com 2026-03-21 04:22:00 92f6028b │ │ Add a new secret settings fragment │ ○ oylmprpu jxxxxxxxxxxx@gmail.com 2026-03-21 04:22:00 18931825 │ │ Create a new feature for receiving and sending commands. │ ○ xrnnoonu jxxxxxxxxxxx@gmail.com 2026-03-21 04:21:48 618020c7 ┌┤ (empty) (no description set) │ ○ rqlyqqzx jxxxxxxxxxxx@gmail.com 2026-03-19 17:20:20 c9b5323c │ │ Bug xxxxxxx - Part 2: Create new android gradle module skill │ ○ txvozpwz jxxxxxxxxxxx@gmail.com 2026-03-19 17:20:13 cee18510 ├┘ Bug xxxxxxx - Part 1: Add new gradle example module ◆ pwsnmryn vxxxxxxxxxxx@gmail.com 2026-03-18 13:21:47 main@origin fa20ce29 │ Bug xxxxxxx - Make my feature work for everyone ~ When I need to submit these, [moz-phab][1 has support for specifying revset ranges with moz-phab start_rev end_rev. However, I can also use jj rebase -s <rev> -d main@origin to put out some try pushes to validate they still work separately - so far, no conflicts in this step.
- Jonathan Almeida: Use |mach try --no-push| for a configuration dry run (2026/03/22 00:00)
  I wanted to see what the generated try configuration would be for a new preset I made and did this by submitting real try pushes (with empty so they don't execute resources). What I was looking for was "dry run" in the help files, but I recently discovered it to be --no-push. $ jj try-push --preset fenix --no-push # 'fenix' as an example preset Artifact builds enabled, pass --no-artifact to disable Commit message: Fuzzy (preset: fenix) query='build-apk-fenix-debug&query='signing-apk-fenix-debug&query='build-apk-fenix-android-test-debug&query='signing-apk-fenix-android-test-debug&query='test-apk-fenix-debug&query='ui-test-apk-fenix-arm-debug&query=^source-test 'fenix&query='generate-baseline-profile-firebase-fenix mach try command: `./mach try --preset fenix --no-push` Pushed via `mach try fuzzy` Calculated try_task_config.json: { "parameters": { "optimize_target_tasks": false, "try_task_config": { "disable-pgo": true, "env": { "TRY_SELECTOR": "fuzzy" }, "tasks": ["build-apk-fenix-android-test-debug", "build-apk-fenix-debug", "generate-baseline-profile-firebase-fenix", "source-test-android-detekt-detekt-fenix", "source-

test-android-l10n-lint-l10n-lint-fenix", "source-test-android-lint-fenix", "source-test-buildconfig-buildconfig-fenix", "source-test-ktlint-fenix", "source-test-mozlint-android-fenix", "test-apk-fenix-debug", "ui-test-apk-fenix-arm-debug", "ui-test-apk-fenix-arm-debug-smoke"], "use-artifact-builds": true } }, "version": 2 } Here, jj try-push is my quick alias around ./mach try for personal simplicity with my workflow.

- Niko Matsakis: Maximally minimal view types (2026/03/21 16:37)
  This blog post describes a maximally minimal proposal for view types. It comes out of a converastion at RustNation I had with lcnr and Jack Huey, where we talking about various improvements to the language that are “in the ether”, that basically everybody wants to do, and what it would take to get them over the line. Example: MessageProcessor Let’s start with a simple example. Suppose we have a struct MessageProcessor which gets created with a set of messages. It will process them and, along the way, gather up some simple statistics: pub struct MessageProcessor { messages: Vec<String>, statistics: Statistics, } #[non_exhaustive] // Not relevant to the example, just good practice! pub struct Statistics { pub message_count: usize, pub total_bytes: usize, } The basic workflow for a message processor is that you accumulate messages by pushing them into the self.messages vector drain the accumulate messages and process them reuse the backing buffer to push future messages Accumulating messages Accumulating messages is easy: impl MessageProcessor { pub fn push_message(&mut self, message: String) { self.messages.push(message); } } Processing a single message The function to process a single message takes ownership of the message string because it will send it to another thread. Before doing so, it updates the statistics: impl MessageProcessor { fn process_message(&mut self, message: String) { self.statistics.message_count += 1; self.statistics.total_bytes += message.len(); // ... plus something to send the message somewhere } } Draining the accumulated messages The final function you need is one that will drain the accumulated messages and process them. Writing this ought to be straightforward, but it isn’t: impl MessageProcessor { pub fn process_pushed_messages(&mut self) { for message in self.messages.drain(..) { self.process_message(message); // <-- ERROR: `self` is borrowed } } } The problem is that self.messages.drain(..) takes a mutable borrow on self.messages. When you call self.process_message, the compiler assumes you might modify any field, including self.messages. It therefore reports an error. This is logical, but frustrating. Experienced Rust programmers know a number of workarounds. For example, you could swap the messages field for an empty vector. Or you could invoke self.messages.pop(). Or you could rewrite process_message to be a method on the Statistics type. But all of them are, let’s be honest, suboptimal. The code above is really quite reasonable, it would be nice if you could make it work in a straightforward way, without needing to restructure it. What’s needed: a way for the borrow checker to know what fields a method may access The core problem is that the borrow checker does not know that process_message will only access the statistics field. In this post, I’m going to focus on an explicit, and rather limited, notation, but I’ll also talk about how we might extend it in the future. View types extend struct types with a list of fields The basic idea of a view type is to extend the grammar of a struct type to optionally include a list of accessible fields: RustType := StructName<...> | StructName<...> { .. } // <-- what we are adding | StructName<...> { (fields),* } // <-- what we are adding A type like MessageProcessor { statistics } would mean “a MessageProcessor struct where only the statistics field can be accessed”. You could also include a .., like MessageProcessor { .. }, which would mean that all fields can be accessed, which is equivalent to today’s struct type MessageProcessor. View types respect privacy View types would respect privacy, which means you could only write MessageProcessor { messages } in a context where you can name the field messages in the first place. View types can be named on self arguments and elsewhere You could use this to define that process_message only needs to access the field statistics: impl MessageProcessor { fn process_message(&mut self {statistics}, message: String) { // ---------- // Shorthand for: `self: &mut MessageProcessor {statistics}` // ... as before ... } } Of course you could use this notation in other arguments as well: fn silly_example(.., mp:

&mut MessageProcessor {statistics}, ..) { } Explicit view-limited borrows We would also extend borrow expressions so that it is possible to specify precisely which fields will be accessible from the borrow: let messages = &mut some_variable {messages}; // Ambiguous grammar? See below. When you do this, the borrow checker produces a value of type &mut MessageProcessor {messages}. Sharp-eyed readers will note that this is ambiguous. The above could be parsed today as a borrow of a struct expression like some_variable { messages } or, more verbosely, some_variable { messages: messages }. I'm not sure what to do about that. I'll note some alternative syntaxes below, but I'll also note that it would be possible for the compiler to parse the AST in an ambiguous fashion and disambiguate later on once name resolution results are known. We automatically introduce view borrows in an auto-ref In our example, though, the user never writes the &mut borrow explicitly. It results from the auto-ref added by the compiler as part of the method call: pub fn process_pushed_messages(&mut self) { for message in self.messages.drain(..) { self.process_message(message); // <-- auto-ref occurs here } } The compiler internally rewrites method calls like self.process_message(message) to fully qualified form based on the signature declared in process_message. Today that results in code like this: MessageProcessor::process_message(&mut *self, message) But because process_message would now declare &mut self { statistics }, we can instead desugar to a borrow that specifies a field set: MessageProcessor::process_message(&mut *self { statistics }, message) The borrow checker would respect views Integrating views into the borrow checker is fairly trivial. The way the borrow checker works is that, when it sees a borrow expression, it records a "loan" internally that tracks the place that was borrowed, the way it was borrowed (mut, shared), and the lifetime for which it was borrowed. All we have to do is to record, for each borrow using a view, multiple loans instead of a single loan. For example, if we have &mut self, we would record one mut-loan of self. But if we have &mut self {field1, field2}, we would two mut-loans, one of self.field1 and one of self.field2. Example: putting it all together OK, let's put it all together. This was our original example, collected: pub struct MessageProcessor { messages: Vec<String>, statistics: Statistics, } #[non_exhaustive] pub struct Statistics { pub message_count: usize, pub total_bytes: usize, } impl MessageProcessor { pub fn push_message(&mut self, message: String) { self.messages.push(message); } pub fn process_pushed_messages(&mut self) { for message in self.messages.drain(..) { self.process_message(message); // <-- ERROR: `self` is borrowed } } fn process_message(&mut self, message: String) { self.statistics.message_count += 1; self.statistics.total_bytes += message.len(); // ... plus something to send the message somewhere } } Today, process_pushed_messages results in an error: pub fn process_pushed_messages(&mut self) { for message in self.messages.drain(..) { // ---------- borrows `self.messages` self.process_message(message); // <-- ERROR! // ---------- borrows `self` } } The error arises from a conflict between two borrows: self.messages.drain(..) desugars to Iterator::drain(&mut self.messages, ..) which, as you can see, mut-borrows self.messages; then self.process_message(..) desugars to MessageProcessor::process_message(&mut self, ..) which, as you can see, mut-borrows all of self, which overlaps self.messages. But in the "brave new world", we'll modify the program in one place: - fn process_message(&mut self, message: String) { + fn process_message(&mut self {statistics}, message: String) { and as a result, the process_pushed_messages function will now borrow check successfully. This is because the two loans are now issued for different places: as before, self.messages.drain(..) desugars to Iterator::drain(&mut self.messages, ..) which mut-borrows self.messages; but now, self.process_message(..) desugars to MessageProcessor::process_message(&mut self {statistics}, ..) which mut-borrows self.statistics, which doesn't overlap self.messages. At runtime, this is still just a pointer One thing I want to emphasize is that "view types" are a purely static construct and do not change how things are compiled. They simply give the borrow checker more information about what data will be accessed through which references. The process_message method, for example, still takes a single pointer to self. This is in contrast with the workarounds that exist today. For example, if I were writing the above code, I might well rewrite

process_message into an associated fn that takes a &mut Statistics: impl MessageProcessor { fn process_message(statistics: &mut Statistics, message: String) { statistics.message_count += 1; statistics.total_bytes += message.len(); // ... plus something to send the message somewhere } } This would be annoying, of course, since I'd have to write Self::process_message(&mut self.statistics, ..) instead of self.process_message(), but it would avoid the borrow check error. Beyond being annoying, it would change the way the code is compiled. Instead of taking a reference to the MessageProcessor it now takes a reference to the Statistics. In this example, the change from one type to another is harmless, but there are other examples where you need access to mulitple fields, in which case it is less efficient to pass them individually. Frequently asked questions How hard would this be to implement? Honestly, not very hard. I think we could ship it this year if we found a good contributor who wanted to take it on. What about privacy? I would require that the fields that appear in view types are 'visible' to the code that is naming them (this includes in view types that are inserted via auto-ref). So the following would be an error: mod m { #[derive(Default)] pub struct MessageProcessor { messages: Vec<String>, ... } impl MessageProcessor { pub fn process_message(&mut self {messages}, message: String) { // ---------- // It's *legal* to reference a private field here, but it // results in a lint, just as it is currently *legal* // (but linted) for a public method to take an argument of // private type. The lint is because doing this is effectively // going to make the method uncallable from outside this module. self.messages.push(message); } } } fn main() { let mut mp = m::MessageProcessor::default(); mp.process_message(format!("Hello, world!")); // ---------- ERROR: field `messages` is not accessible here // // This desugars to: // // ``` // MessageProcessor::process_message(// &mut mp {messages}, // <-- names a private field! // format!("Hello, world!"), //) // ``` // // which names the private field `messages`. That is an error. } Does this mean that view types can't be used in public methods? More-or-less. You can use them if the view types reference public fields: #[non_exhaustive] pub Statistics { pub message_count: usize, pub average_bytes: usize, // ... maybe more fields will be added later ... } impl Statistics { pub fn total_bytes(&self {message_count, average_bytes}) -> usize { // ---------- // Declare that we only read these two fields. self.message_count * self.average_bytes } } Won't it be limited that view types more-or-less only work for private methods? Yes! But it's a good starting point. And my experience is that this problem occurs most often with private helper methods like the one I showed here. It can occur in public contexts, but much more rarely, and in those circumstances it's often more acceptable to refactor the types to better expose the groupings to the user. This doesn't mean I don't want to fix the public case too, it just means it's a good use-case to cut from the MVP. In the future I would address public fields via abstract fields, as I described in the past. What if I am borrowing the same sets of fields over and over? That sounds repititive! That's true! It will be! I think in the future I'd like to see some kind of 'ghost' or 'abstract' fields, like I described in my abstract fields blog post. But again, that seems like a "post-MVP" sort of problem to me. Must we specify the field sets being borrowed explicitly? Can't they be inferred? In the syntax I described, you have to write &mut place {field1, field2} explicitly. But there are many approaches in the literature to inferring this sort of thing, with row polymorphism perhaps being the most directly applicable. I think we could absolutely introduce this sort of inference, and in fact I'd probably make it the default, so that &mut place always introduces a view type, but it is typically inferred to "all fields" in practice. But that is a non-trivial extension to Rust's inference system, introducing a new kind of inference we don't do today. For the MVP, I think I would just lean on auto-ref covering by far the most common case, and have explicit syntax for the rest. Man, I have to write the fields that my method uses in the signature? That sucks! It should be automatic! I get that for many applications, particularly with private methods, writing out the list of fields that will be accessed seems a bit silly: the compiler ought to be able to figure it out. On the flip side, this is the kind of inter-procedural inference we try to avoid in Rust, for a number of reasons: it introduces dependecies between methods which makes inference more difficult (even undecidable, in extreme cases); it makes for 'non-local errors' that can be really confusing as a user, where

modifying the body of one method causes errors in another (think of the confusion we get around futures and Send, for example); it makes the compiler more complex, we would not be able to parallelize as easily (not that we parallelize today, but that work is underway!) The bottom line for me is one of staging: whatever we do, I think we will want a way to be explicit about exactly what fields are being accessed and where. Therefore, we should add that first. We can add the inference later on. Why does this need to be added to the borrow checker? Why not desugar? Another common alternative (and one I considered for a while…) is to add some kind of “desugaring” that passes references to fields instead of a single reference. I don’t like this for two reasons. One, I think it’s frankly more complex! This is a fairly straightforward change to the borrow checker, but that desugaring would leave code all over the compiler, and it would make diagnostics etc much more complex. But second, it would require changes to what happens at runtime, and I don’t see why that is needed in this example. Passing a single reference feels right to me. What about the ambiguous grammar? What other syntax options are there? Oh, right, the ambiguous grammar. To be honest I’ve not thought too deeply about the syntax. I was trying to have the type Struct { field1, field 2 } reflect struct constructor syntax, since we generally try to make types reflect expressions, but of course that leads to the ambiguity in borrow expressions that causes the problem: let foo = &mut some_variable { field1 }; // ---------- is this a variable or a field name? Options I see: Make it work. It’s not truly ambiguous, but it does require some semantic diambiguation, i.e., in at least some cases, we have to delay resolving this until name resolution can complete. That’s unusual for Rust. We do it in some small areas, most notably around the interpretation of a pattern like None (is it a binding to a variable None or an enum variant?). New syntax for borrows only. We could keep the type syntax but make the borrow syntax different, maybe &mut {field1} in some_variable or something. Given that you would rarely type the explicit borrow form, that seems good? Some new syntax altogether. Perhaps we want to try something different, or introduce a keyword everywhere? I’d be curious to hear options there. The current one feels nice to me but it occupies a “crowded syntactic space”, so I can see it being confusing to readers who won’t be sure how to interpret it. Conclusion: this is a good MVP, let’s ship it! In short, I don’t really see anything blocking us from moving forward here, at least with a lang experiment.

- The Rust Programming Language Blog: Security advisory for Cargo (2026/03/21 00:00)
  The Rust Security Response Team was notified of a vulnerability in the third-party crate tar, used by Cargo to extract packages during a build. The vulnerability, tracked as CVE-2026-33056, allows a malicious crate to change the permissions on arbitrary directories on the filesystem when Cargo extracts it during a build. For users of the public crates.io registry, we deployed a change on March 13th to prevent uploading crates exploiting this vulnerability, and we audited all crates ever published. We can confirm that no crates on crates.io are exploiting this. For users of alternate registries, please contact the vendor of your registry to verify whether you are affected by this. The Rust team will release Rust 1.94.1 on March 26th, 2026, updating to a patched version of the tar crate (along with other non-security fixes for the Rust toolchain), but that won't protect users of older versions of Cargo using alternate registries. We'd like to thank Sergei Zimmerman for discovering the underlying tar crate vulnerability and notifying the Rust project ahead of time, and William Woodruff for directly assisting the crates.io team with the mitigations. We'd also like to thank the Rust project members involved in this advisory: Eric Huss for patching Cargo; Tobias Bieniek, Adam Harvey and Walter Pearce for patching crates.io and analyzing existing crates; Emily Albini and Josh Stone for coordinating the response; and Emily Albini for writing this advisory.
- Mozilla Addons Blog: How Your Feedback Shapes the Firefox Add-ons Experience (2026/03/20 21:44)
  Hey everyone! I’m Christos Bacharakis, and I’ve recently joined Mozilla as the new Add-ons Senior Developer Relations Engineer. I’m here to help developers overcome technical challenges so your extension visions may become reality. My goal is to keep things moving smoothly so you can

focus on what you do best: building awesome add-ons. We've been listening closely to the conversations happening over on Mozilla Connect, and I'm excited to share that your feedback is directly driving our latest updates. Here are two major changes we've rolled out because of your suggestions: Hiding the Extensions Button: We heard the UI was getting a bit crowded. Based on community requests, we've implemented the option to hide the Extensions button, giving back control over your browser's real estate. Version Roll-back Capability: Mistakes happen, and sometimes a new update doesn't go as planned. You asked for a safety net, and we delivered. Developers can now roll back to a previous version of their extension if an issue arises with a new release. These updates aren't just random features; they are the direct result of developers like you speaking up on Connect. Whether it's a UI tweak or a policy change like we've seen recently with third-party library usage and remote code execution (RCE) reviews, your perspective helps us refine our plans. So, keep those ideas coming! Head over to the Add-ons section of Mozilla Connect and tell us what's on your mind. I'm looking forward to working with all of you to make the Add-ons developer experience better than ever. The post How Your Feedback Shapes the Firefox Add-ons Experience appeared first on Mozilla Add-ons Community Blog.

- Firefox Nightly: Extension Theming Improvements and More – These Weeks in Firefox: Issue 198 (2026/03/20 17:34)
  Highlights Starting with Firefox 150, Firefox Desktop's theming internals have improved, allowing custom background images provided by built-in and custom themes to extend over the sidebar UI background – Bug 1952602. Thanks to Emilio for his work on these Firefox theme improvements! Alpenglow theme – current (left, Firefox = 150) behavior. We deprecated the built-in CSS filter for WebExtensions and changed how we theme extension icons – Bug 2001318. If you spot any issues with extension icons, please file a bug here. Extensions can now rely on SVG's native prefers-color-scheme media query support; see this comment for more information. The built-in CSS filter was implicitly applied to all extensions' page action SVG icons in dark mode. It did not provide an official opt-out mechanism, resulting in poor contrast for some extension icons. Niklas fixed broken and missing Picture-in-Picture video controls on Disney+. Found a Picture-in-Picture bug? Please file one here! Friends of the Firefox team Resolved bugs (excluding employees) Volunteers that fixed more than one bug Chris Vander Linden Justin Peter New contributors (= first patch) Jocsan: Profile delete page title should enclose profile name in quotes karan68: [dialog] 'X' icon buttons are missing labels Knot False: Replace KeywordUtils use of nsIScriptableUnicodeConverter with Services.textToSubURI.ConvertAndEscape Pushkar Singh: Remove unused devtools.inspector.remote pref MUTHUSRIHEMADHARSHINI: Card images do not have an alt attribute Sukhmeet[:sukh]: firefox forgets open tabs depending on window close order Alex Reisner: Create a preference to disable the drag-and-drop method of creating tab groups torrenceb90: Remove unused devtools.target-switching.server.enabled pref Project Updates Add-ons / Web Extensions Addon Manager & about:addons As part of the work for the SmartWindow mode, a new dismissible info messagebar will be shown in about:addons themes list view to users opting into SmartWindow that Firefox windows, the SmartWindow theming notice will be highlighting for the user that the SmartWindow mode will use their own custom theming – Bug 2010685 Special thanks to Maile Lucks and Chloe Zhou for gathering the clarifications needed and working with the SmartWindow team to reach the final agreement on the desired behaviors. Fixed "Available Updates" about:addons view not refreshing automatically when the "Check for updates" action is clicked while the "Available updates" is already selected until manually switching about:addons view or reloading the about:addons tab – Bug 1578182 WebExtensions Framework As part of hardening work on the WebExtensions internals, a new option has been added to Cu.Sandbox constructor to opt-in/opt-out from freezing all the built-ins available in the Sandbox global – Bug 2017957 WebExtension APIs As part of the work to support Firefox Split View, additional changes have been applied to make sure that Split View mode is preserved when extensions are moving all tabs part of a Split View into a new window – Bug 2017148 / Bug 2017768 / Bug 2016754 Fixed a regression (introduced in Firefox 149 by Bug 2013389 changes) which was reported to be breaking the

Clipboard2File extension. The fix has been landed in Nightly 150 and uplifted to 149 – Bug 2017797 DevTools Sebastian Zartner [:sebo] added a button in the Storage panel to delete all the items of a given storage type (Local Storage, Cookies, …) (#1349533) Chris Vander Linden migrated devtools/client/shared/widgets/TableWidget.js to use proper private ES fields (#2015241) Torrence removed the unused devtools.target-switching.server.enabled pref (#2003192) Pushkar Singh removed the unused devtools.inspector.remote pref (#2003191) Andreas Farre [:farre] improved the Session History panel to easily differentiate same-document and cross-document navigation (#2016130, #2016126) Nicolas Chevobbe [:nchevobbe] added a button in Computed sidebar that link to the declaration in the Rules panel (#1418351) Alexandre Poirot [:ochameau] fixed an issue in the inactive CSS that was impacting declarations on rules with multiple pseudo elements (#1998357) Nicolas Chevobbe [:nchevobbe] made position-area marked as inactive when the element is not absolutely positioned or doesn't have a default anchor (#1895185) Nicolas Chevobbe [:nchevobbe] added proper autocomplete for color() in the Rules view so it suggests colorspaces (#1898277) Hubert Boma Manilla (:bomsy) added text in the Response panel for redirected request to explain why the panel is empty (#2016679) WebDriver Alexandra Borovova implemented the "browser.setDownloadBehavior" command, which allows clients to allow or prohibit the downloads and also set a custom download folder. This behavior can be configured per session or per user contexts. Julian Descottes implemented the "emulation.setNetworkConditions" command for "type: offline", which can be used to emulate offline network situations. This configuration can be applied to contexts, user contexts or globally. Julian Descottes updated the WebDriver classic getShadowRoot command to no longer return user agent shadow roots. Henrik Skupin fixed the mach test command which was not able to run Marionette tests anymore. Lint, Docs and Workflow FYI to developers that stylelint changes now go to #desktop-theme-reviewers Search Drew Willcoxon finished his work on Online Suggest over OHTTP. It has been enabled by default in Nightly since the 149 cycle, improving privacy for remote suggestions while we validate quality and latency at scale. Daisuke Akatsuka changed sponsored suggestions to no longer display the destination URL, reducing visual clutter and emphasizing brand/title. Can be tuned via remote config as we watch CTR and trust metrics. Voice Control users can once again focus the URL bar via "click" or numbered commands — fixed by Morgan Rae Reschenberg. Dale Harvey continued his work on the unified trust button and panel. The HTTP-only "Not Secure" shield now consistently shows the red slash even when tracking protection is off, avoiding misleading site trust signals. Justin Peter contributed a bunch of fixes: Unit conversion in the URL bar now handles the degree symbol (°) so temperature math returns results as expected. Imperial unit conversion precision is improved (e.g., inches/feet/miles rounding), yielding more accurate answers. Added more units and abbreviations to conversion, broadening coverage of everyday queries. View bugs in Bugzilla Tests Ever had to deal with a test failure like? leaked 1 window(s) until shutdown [url = chrome://browser/content/browser.xhtml]and tried to follow the instructions to debug it? It just became way easier with the retention path printed directly as part of the failure: If you would like to see how the flakiness rate of xpcshell and mochitest harness evolve over time, you can see it on https://tests.firefox.dev/ Dynamic chunking of test jobs landed: Most mochitest jobs now take 20 minutes + the setup time of the job. (charts) Per-manifest and job name run time info can be queried at https://tests.firefox.dev/manifests.html UX Fundamentals In progress: Update messaging on file:// not-found error pages. The intro will show the actual file path instead of "local file", and the "What can you do?" section gives file-oriented advice rather than website-oriented advice. Adding access keys to error pages. All buttons on the error pages will have keyboard shortcuts. Updating the netTimeout error to use correct l10n IDs and whole origin with subdomains.

- Firefox hacking in Fedora: Firefox & Gtk Emoji picker (2026/03/20 14:48)
  After nearly three years of development (it takes time to make up one's mind) Firefox for Linux users can now enjoy seamless emoji insertion

using the native GTK emoji chooser. This long-requested feature, implemented inFirefox 150 (recent Beta), brings a more integrated experience for Linux users who want to add emojis to their web content. Starting with Firefox 150, the native Gtk emoji picker can be invoked directly within Firefox. This means you can insert emojis into text fields, comment boxes, social media posts, and any other web input using the same interface you’re already familiar with from other GTK applications. How to use it Using the emoji picker is simple and follows standard GTK conventions: Click into any text input field on a webpage Press Ctrl+. (Control + period) or Ctrl+; (Control + semicolon) The native GTK emoji chooser will appear Select your emoji, and it will be inserted at your cursor position. The picker works seamlessly in both main browser windows and popup windows. How to disable it The feature leverages GTK’s built-in GtkEmojiChooser widget, ensuring that the look and feel matches other applications in your desktop environment. For users who prefer not to use this feature (perhaps due to conflicts with custom keybindings or specific workflows), Firefox provides a preference to disable it. Go to about:config page and set widget.gtk.native-emoji-dialog to false. Why it took too long? The native GTK emoji picker implementation uses the GtkEmojiChooser popover built into GTK3. Unlike other GTK dialogs (file picker, print dialog), it can’t be invoked directly in GTK3 – it must be triggered by a key-binding event or signal on GtkEntry or GtkTextView widgets, and the widget has to be visible from GTK’s perspective. This conflicts with Firefox’s GTK architecture, which doesn’t use GTK widgets directly but instead paints everything itself. But a solution was found. Firefox already has an invisible GtkEntry widget that’s not attached to any actual window. It’s an offscreen widget used to invokekey-binding events from GTK input events, like copy/paste and other edit commands. It also receives the ’emoji-insert’ signal after Ctrl+., but normally ignores it since the GtkEntry itself isn’t visible. I configured the GtkEntry to listen for the ’emoji-insert’ signal. When received, I create a new GtkEntry as a child of the recently focused GtkWindow and redirect the ’emoji-insert’ signal there. The GtkEntry has to be ‘shown’ but remains invisible to users because Firefox paints a wl_subsurface over it. It also needs to be correctly positioned to show the GtkEmojiChooser in the right location, and connected to other signals like ‘insert_text’ to retrieve the selected emoji. Thanks to Emilio Cobos Álvarez and Masayuki Nakano for their helpful hints on text processing and positioning!

- Firefox Tooling Announcements: Firefox Profiler Deployment (March 20, 2026) (2026/03/20 12:26)
  The latest version of the Firefox Profiler is now live! Check out the full changelog below to see what’s changed. Highlights: [fatadel] Make network markers in the network panel sticky on click (#5884) Other Changes: [Markus Stange] Start using const enum (#5879) [Markus Stange] Some performance improvements (#5878) [Nazım Can Altınova] Add startLine, startColumn, sourceMapURL and rename uuid to id in source table (#5882) [Markus Stange] Reduce repetition in profile compacting code (#5885) [Markus Stange] Some more activity graph drawing perf improvements (#5886) [Markus Stange] Improve SampleGraph and HeightGraph performance (#5897) [Nazım Can Altınova] Sync: l10n → main (March 20, 2026) (#5899) Big thanks to our amazing localizers for making this release possible: be: Andrei Mukamolau el: Jim Spentzos en-GB: Paul es-CL: ravmn ia: Melo46 sv-SE: Andreas Pettersson tr: Nazım Can Altınova Find out more about the Firefox Profiler on profiler.firefox.com! If you have any questions, join the discussion on our Matrix channel! 1 post - 1 participant Read full topic
- The Rust Programming Language Blog: What we heard about Rust's challenges (2026/03/20 00:00)
  Author's note The original version of this article has been retracted. I used an LLM to write the first draft, though this had come after many hours of planning and going through the data and analyses to identify the points to be made, as well as me going through the post line by line, editing into my voice and verifying the wording and scope of the text was accurate. However, many people still felt like the LLM-speak bled through in ways that felt uncomfortable. Given this, I and other members of the Rust Project have decided to retract the post in its entirety. I stand by the content of the post. As I said, the LLM did not decide the points to be made - those were done well in advance of even beginning to write the blog

post. And, admittedly, I did need to make edits to dampen the scope of them (in large part because I couldn't find specific quotes to substantiate them, even though I often "felt" that they were true given what I know as a Rust Project member), but in general I (and the Vision Doc team) defined the content, not an LLM. Many people thought that the blog post felt "empty", with no "real substance". While I see the point here, this is unfortunately just how the data played out and goal of this effort. The Vision Doc team conducted ~70 interviews (mostly 1:1), which were the basis for the conclusions in this blog post. This is a lot of data, it's hard to fully capture the essence of them in a single blog post. And yet, it is also not enough data to fully capture the nuance of differences across groups of different types. On top of this, it shouldn't be that unexpected the problems we heard about in these interviews are the same problems that we (and many others) mostly already knew existed. The insight these interviews give us is that they allow us to begin to capture for whom which issues are most prominent. The insights we identify and the conclusions we make are supported by the data we have gathered. When making these posts, the Vision Doc team has tried to stay as neutral as possible, doing our best to not exert bias by making any claims that cannot be supported as stated by the data itself. With drastically more time, I would have loved to pull in data from the ~5500 survey responses we got, which ultimately could help us make stronger claims or conclusions, but unfortunately that is time that I haven't had. That shouldn't diminish the substance of the insights and conclusions we have been able to make though. Wording matters though. And it's clear that to many people, the blog post as-written didn't meet the mark that they want. LLMs are a tool that many people use (including me, obviously) to varying degrees to help do things that they couldn't do before (either for lack of skill, lack of time, or lack of motivation). In this case, I used an LLM to compensate for the lack of time for me to dedicate to sifting through transcripts for the ~70 interviews we did, and the many analyses that followed, to find specific quotes and write an early draft. It certainly did not help that writing and editing of this post happened over the span of about 3 months - meaning that things that "worked" in early edits did not necessarily work in later edits. This all being said, I think that we as a Vision Doc team owe it to the Rust Project and the community to share (at least to some extent) what we have learned here. So, I have taken the original challenges identified by the team (without the recommendations or conclusions) and will provide a brief personal commentary on them. I've chosen to exclude any specific quotes - rather, just focus on the "high level" ideas. So, as a disclaimer, this will mean that the statements here will be much more biased than what we typically want to publish as part of the Vision Doc work. Across the ~70 interviews the Vision Doc team conducted, we heard a lot of complaints. Of course, we tried to keep these interviews pretty high-level, not focusing on any particular technical details. Rather, we wanted to get a general sense of what the difficulties were that people encountered, among the other topics discussed during these interviews. Here, we've identified a few common challenges to most people, and then a few challenges that are more domain-specific. Challenges that are universal We heard a number of things that basically everyone said was an issue for them, in some capacity. Doing things to address these issues could have a universal impact, but that is not to say that these issues necessarily block people from using Rust. The universal challenges, you've definitely heard before. If you write Rust, you've probably encountered them. That's what makes them universal. However, the point is that we share the data that we gather, and the fact that we have learned that these challenges do affect everyone is data in itself: we have sampled different domains, different experience levels, and different backgrounds; and we have found that these challenges exist for everyone. Compilation performance Everybody knows that "compile times" are a thing that Rust is known for. This is an ever-moving target: the Rust Project tracks performance of the compiler on every merged change to track regressions, many people have attempted many times to make substantial progress here, and yet there is always more that we want to or could do. The good news, is that among our interviews, nobody really told us that compilation time currently blocks them. We did hear things to the effect of "if we keep writing more and more Rust code, we may eventually get to a point that compile times are an issue", so that's

not to say that we're "in the clear" but it is important to think about how this matters on balance with other challenges that people face. Borrow checking and ownership Again, another thing that Rust is known for. Borrow checking and ownership is a hard topic that basically every beginner struggles with. However, we found that "Rust experts" don't really complain about the borrow checker anymore: it is a challenge that goes away with experience. That's not to say we can't do better for beginners, but it's not clear exactly what that means. Certainly learning materials and compiler error messages help, and these are areas that we've tried in the past and today to sincerely provide the best experience. Despite that, the borrow checker remains a difficult part of the Rust language. We have ongoing efforts to improve the borrow checker, but it's likely that there are (for example) language features that may make this better. (Or worse!) We found previously that what makes Rust great is the balance that we put on reliability, efficiency, and versatility. And, we need to be careful when adjusting something as core as the borrow checker to maintain this balance. Async When conducting our interviews, async was consistently something that many people had issues with. Beginners often said that they basically completely ignore async while learning. People who do use async often said that the choice wasn't always clear, and that even though using async feels like the right choice now, they still encounter issues. Fortunately, unlike performance and the borrow checker, we have a number of clear "next steps" for async (e.g. async fns in traits, async drop, async version of std traits) that will begin to solve these issues and close the gap. Of course, for other things (like the coloring problem), we don't have good "solutions" just quite yet. Ecosystem crates We previously talked about how crates.io creates a wealth of resources for people to turn to, but people still run into issues. For one, when there are crates that do the thing people want, they need to know: which crates do the things they need, which crates can they trust, and which crates are just overall the "best" for them. Further, in some domains and industries, there aren't crates that do what people need; Rust support for some industries are still too immature. Challenges that are domain-specific Though more challenging given the limited diversity in the interviews we conducted, we still were able to find some domain-specific challenges: at least, we were able to hear about some challenges that seem to disproportionately effect some domains over others. Embedded For developers programming for embedded systems, we heard most often about the difficulties that fundamentally boil down to constrained resource management. For example, embedded developers are often unable to use the vast majority of the crate ecosystem, they often have trouble using the standard library, and the debugging experience is generally harder. Things that are "normal" for most Rust developers are oftentimes "special" for embedded developers. Safety-critical We made an entire post about shipping Rust in safety critical systems. The biggest issue for safety-crticial developers with Rust is the lack of availability or maturity for tools to certify their Rust code. GUI development The biggest issue heard from GUI developers is compilation time but is slightly different from the general case, because GUI development is so heavily dependent on the visual changes - and so this is a slightly different workflow than just "check if the code compiles and passes tests".

- Thunderbird Blog: Introducing our Public Roadmaps (2026/03/19 19:26)
  At Thunderbird, we firmly believe in the strength of listening to our community’s needs and wants, and balancing it with our resources and capabilities. While this has always been part of our ethos, we want to start 2026 by making our goals easier to read and comprehend at roadmaps.thunderbird.net, where you will find our roadmaps for our Services and both the Thunderbird Desktop and Mobile products. Go to the Roadmaps To better serve our community, we are making several thoughtful updates to how we build and communicate our roadmap. These key changes include clearer estimation practices, stronger strategic framing, and more transparent updates when priorities evolve. Intentional Descriptions You may notice that the descriptions of each roadmap item is written in very common, non-technical, language as much as possible. This is done purposefully, so that someone from any technical level can understand what we are trying to achieve. We have also tried to not be

too verbose so that you, the reader, can be informed without being bored. Regular Reviews and Updates At the end of each calendar quarter, we will hold internal roadmap review meetings with each of the Desktop, Mobile, and Services teams. We will review the estimated progress and priority of each item and adjust the roadmap as needed. Any changes to the roadmap will be clearly communicated to the tb-planning topicbox list. What the Roadmap is and is not We know that different companies and projects can approach the term “roadmap” differently so let’s be clear about what Thunderbird is providing. This roadmap reflects our current priorities and the work we believe will have the greatest impact in 2026. While priorities can evolve as new information arises, we’re committed to reviewing progress quarterly and communicating any adjustments clearly and transparently. Our public roadmap is focused on themes rather than individual tasks or bugs to fix. It is our directional plan that outlines the goals for this year. We view a roadmap as a plan to keep us on target, towards accomplishing broader goals, rather than a wishlist of bugs to fix. Balancing Ideas with Capacity Thunderbird thrives because of its community. Every year, we receive more great ideas than we have capacity to implement. Our responsibility is to focus on the initiatives that create the broadest impact while enduring the long-term health of the project. That doesn’t mean individual ideas don’t matter. In fact, community input directly influences our roadmap over time. The updated roadmap process helps us be more clear about what we can take on right now, and how new ideas can shape what comes next. If there’s something you’d love to see in Thunderbird, please share it. Momentum starts with voices like yours. Share your ideas on Thunderbird Desktop or Mobile Share your ideas on Thunderbird Pro The post Introducing our Public Roadmaps appeared first on The Thunderbird Blog.

- Firefox UX: Designing beyond the checklist (2026/03/19 17:27)
  Accessibility has always been close to my heart. I was introduced to it early in my career, when a client required our product to be AAA-compliant. As a team of two, we audited our entire app. We were barely AA. If those letters don’t mean much, they refer to the Web Content Accessibility Guidelines (WCAG), the international standard for making digital products usable for people with disabilities. AAA is the highest level of conformance. AA is typically considered the acceptable baseline. We weren’t even confidently there. That experience changed how I see design. Most products today aren’t perfect. Many aren’t even close. But accessibility isn’t just about passing audits or checking contrast ratios. Yes, color contrast matters. Alt text matters. Focus states matter. But those should be the floor, not the ceiling. Accessibility goes beyond a checklist. It’s a mindset. And that mindset deepened for me on the Firefox UX team. At Mozilla, I’ve had the opportunity to collaborate closely with our accessibility specialists and conduct research using Fable: a platform that connects us directly with people who use assistive technologies like screen readers, magnification software, and voice access. Instead of designing based on assumptions, we test with real users navigating Firefox in real ways. That difference is everything. In two recent moderated sessions I facilitated, I was reminded how much nuance lives in the margins of our designs. In one session, we worked with a participant who uses screen magnification software. We learned: Even the slightest pixelation can create ocular vibration and discomfort Swipe gestures, which often feel “modern” to designers, can be risky or disorienting for magnification users Thoughtful line height can meaningfully reduce cognitive load These aren’t things you catch in a checklist review. They’re things you learn by listening. In another session, an Android user who relies on Voice Access showed us something equally important: Holding and dragging interactions are difficult with voice control App-level voice commands can interfere with system-level voice access tools Features designed to feel innovative can unintentionally create barriers. That tension is where real accessibility work happens. This experience wasn’t just eye-opening for me. My teammate Nicole reflected: “Working with Fable was a great experience. One takeaway that stood out was how low vision users face similar challenges to anyone using their phone outside in bright sunlight, especially around contrast. It also made us think more about designing for an aging population… building in accessibility now sets us up to serve more users over time. Overall, it

broadened our perspective, helped us build empathy, and made the product better. This kind of research should be a standard part of the design process.” That last line stays with me: This kind of research should be standard. These sessions fundamentally changed how I approach design. Accessibility now shows up much earlier in my process: not as a final pass, but as a lens. When defining problems, I ask: What happens if this is heard instead of seen? How does this interaction work without touch? Could this increase cognitive load? Are we designing for someone unlike ourselves? It’s no longer a checklist I review at the end. It’s something I carry from the beginning. And it makes our critiques stronger, our conversations sharper, and our decisions more intentional. Accessibility also deeply aligns with Mozilla’s mission. We will always stand behind our belief that the internet should be open, usable, and available to everyone. Designing accessibly is just one of the ways we can achieve that. When we remove barriers, we’re not just improving usability; we’re protecting participation, agency, and inclusion on the web. Of course, no process is perfect. Deadlines are real. Technical constraints are real. Accessibility can feel overwhelming, especially if you’re new to it. But you don’t have to know everything to start. A lot of it is just learning as you go. Start with the basics Take advantage of resources you do have Advocate for testing with real users Most importantly: don’t treat accessibility as an afterthought. The more you practice it, the more it becomes instinctive. Accessibility extends beyond pixels. It’s not about compliance. It’s about empathy. And when we design with accessibility in mind, we’re not designing for a small group of users. We’re designing for everyone. Helpful resources Web Content Accessibility Guidelines (WCAG) Fable Stark for Figma (accessibility plugin) WebAIM Contrast Checker Book: Accessibility for Everyone by Laura Kalbag Originally published on Medium.com

- Olivier Mehani: Git: ignoring temporary changes (2026/03/18 06:53)
  One can use git add [-N|--intent-to-add] <PATH> to record that a new path will need to be considered in later additions to the index. But what about the other way round? How to tell git that a change SHOULD NOT be considered? tl;dr: git update-index --assume-unchanged <PATH> Fortunately, there is the git update-index(1) command. Amongst (many) other options, it supports --assume-unchanged <PATH>. This will tell git to ignore current and any further changes to this <PATH>, until --no-assume-unchanged is used. This abuses the initial purpose of the assume-unchanged logic, but it’s handy for when you want test changes to survive for a bit longer, without risking committing them when git adding in wild abandon. Also, a footgun for when you think your repository is clean, but it has some magic in an assumed-unchanged file you forgot about. You can use git ls-files -v to check the status of your files (h is assumed unchanged, H is normal behaviour). The post Git: ignoring temporary changes first appeared on Narf.
- This Week In Rust: This Week in Rust 643 (2026/03/18 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.bsky.social on Bluesky 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 1.29.0 Call for Testing: Build Dir Layout v2 Newsletters The Embedded Rustacean Issue #67 This Month in Rust OSDev: February 2026 Project/Tooling Updates loadgen-rs - h2load-compatible HTTP benchmark client written in Rust, supporting HTTP/1.1, HTTP/2, and HTTP/3 (QUIC) Introducing pgtui, a Postgres TUI client Avian Physics 0.6 Vite 8.0 is out! Building Rust Procedural Macros Without quote!: Introducing zyn bnum v0.14.0: a lot of big improvements! ClawShell: Secure the OpenClaw using OS-level primitives Giff v1.1.0: A terminal UI for git diffs with interactive rebase support mdterm v1.5.0: A terminal-

based Markdown browser flodl - A Rust-native deep learning framework built on libtorch Cot v0.6: Lazy Underneath Observations/Thoughts Summary - Rust Project Perspectives on AI How to use storytelling to fit inline assembly into Rust Why WebAssembly components yes, all longest regex matches in linear time is possible Accessing Hardware in Rust [audio] Netstack.FM episode 31 — Protocol Shorts: MITM Proxies and Transparent L4 Interception [video] Rust-powered SpacetimeDB is 1000x Faster? Founder Explains Rust Walkthroughs Building small and secure Docker images for Rust: scratch vs alpine vs debian Patching LMDB: How We Made Meilisearch's Vector Store 333% Faster Creating a DAW in Rust - Playing Audio How to Check Code Coverage in Rust [video] RustCurious lesson 4: Structs and Resources – Copy vs Clone vs Move Miscellaneous Free TokioConf tickets for contributors and open source maintainers Crate of the Week This week's crate is grab, a command-line tool to quickly convert CSV to JSON. Thanks to Gábor Maksa for the 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, Cargo, Rustup or Rust language RFCs. 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 Bluesky 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. Oxidize Conference | CFP open until 2026-03-23 | Berlin, Germany | 2026-09-14 - 2026-09-16 EuroRust | CFP open until 2026-04-27 | Barcelona, Spain | 2026-10-14 - 2026-10-17 NDC Techtown 2026 | CFP open until 2026-05-03 | Kongsberg, Norway | 2026-09-21 - 2026-09-24 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 Bluesky or Mastodon! Updates from the Rust Project 427 pull requests were merged in the last week Compiler provide better suggestions for inference errors on .collect()? Library add From impls for wrapper types in Option::get_or_insert_with(), forget the None instead of dropping it fixed VecDeque::splice() not filling the buffer correctly when resizing the buffer on start = end range Cargo CARGO_TARGET_DIR doesn't have to be relative shell: Support OSC 9;4 progress on ptyxis compile: Stop on denying warnings without --keep-going avoid panic for package specs with an empty fragment util: exclude from iCloud Drive sync on macOS Rustdoc rustdoc-json: Add optional support for rkyv (de)serialization Clippy fix match_same_arms false positive with associated consts fix: question_mark suggestion caused error refactor implementation of unnecessary_{option,result}_map_or_else Rust-Analyzer don't trigger GC on slow tests SCIP generation should prime caches in parallel add naming convention validation for union types handle multi-byte UTF-8 identifiers in NameGenerator::suggest_name infer generic args for trait ref and its assoc type remove angle brackets if all lifetime args removed in inline type alias code assist replace make usage with SyntaxFactory in few ide-assists utils methods Rust Compiler Performance Triage Another fairly quiet week, with few changes and overall neutral performance. Triage done by @simulacrum. Revision range: 3945997a..5b61449e 1 Regression, 1 Improvement, 2 Mixed; 3 of them in rollups 35 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

Improvements to match formatting Fix SGX delayed host lookup via ToSocketAddr Rust RFCs Add homogeneous_try_blocks RFC Compiler Team (MCPs only) allow incomplete_features in UI tests Add -Zsanitizer=kernel-hwaddress Language Reference [type layout] usize and isize have the same size and alignment Leadership Council Discuss travel grants 2026 projections No Items entered Final Comment Period this week for Cargo, Language Team 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 Crate deletion allowances Avoid linting unreachable_code on todo!() Propose the Rust Foundation Maintainer fund Upcoming Events Rusty Events between 2026-03-18 - 2026-04-15 □ Virtual 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-18 | Virtual (Cardiff, UK) | Rust and C++ Cardiff Hybrid event with Rust Dortmund! 2026-03-18 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-20 | Virtual | Packt Publishing Limited Rust Adoption, Safety, and Cloud with Francesco Ciulla 2026-03-24 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2026-03-24 | Virtual (London, UK) | Women in Rust Lunch & Learn: Crates, Tips & Tricks Lightning Talks - Bring your ideas! 2026-03-25 | Virtual (Girona, ES) | Rust Girona Rust Girona Hack & Learn 03 2026 2026-03-26 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-04-01 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-04-01 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2026-04-02 | Virtual (Nürnberg, DE) | Rust Nuremberg Rust Nürnberg online 2026-04-04 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup 2026-04-09 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-04-14 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2026-04-14 | Virtual (London, GB) | Women in Rust □ Community Catch Up 2026-04-15 | Virtual (Vancouver, BC, CA) | Vancouver Rust Nushell Asia 2026-03-19 | Seoul, KR | Seoul Rust (Programming Language) Meetup Seoul Rust Meetup 2026-03-22 | Tel Aviv-yafo, IL | Rust □ TLV In person Rust March 2026 at AWS in Tel Aviv 2026-03-28 | Delhi, IN | Rust Delhi Rust Delhi Meetup #13 Europe 2026-03-18 | Dortmund, DE | Rust Dortmund Rust Dortmund Meetup - Intro to Embedded Rust - March 2026-03-19 - 2026-03-20 | Warsaw, PL | Rustikon Rustikon Conference 2026-03-23 | Augsburg, DE | Rust Meetup Augsburg Rust Meetup #18: Ludwig Weinzierl - Bevy: Spielentwicklung mit Rust 2026-03-23 | Amsterdam, NL | Open Source SecurityCon Open Source SecurityCon EU 2026 2026-03-24 | Aarhus, DK | Rust Aarhus Hack Night - Advent of Code 2026-03-24 | Manchester, UK | Rust Manchester Rust Manchester March Code Night 2026-03-24 | Trondheim, NO | Rust Trondheim Rust projects - show and tell in March 2026-03-25 | Dresden, DE | Rust Dresden First Meetup 2026-03-26 | Paris, FR | Rust Paris Rust meetup #84 2026-03-27 | Paris, FR | Rust in Paris Rust in Paris 2026-03-28 | Stockholm, SE | Stockholm Rust Ferris' Fika Forum #25 2026-04-01 | Berlin, DE | Rust Berlin Rust Berlin Talks: The next generation 2026-04-01 | Oxford, UK | Oxford ACCU/Rust Meetup. Rust/ACCU meetup. 2026-04-02 | London, GB | Rust London User Group LDN Talks Spring Community Showcase 2026-04-07 | Basel, CH | Rust Basel Rust Meetup #15 @ letsboot 2026-04-09 | Geneva, CH | Rust Meetup Geneva Rust Meetup Geneva 2026-04-09 | Oslo, NO | Rust Oslo Rust talks @ AutoStore – "Patterns for Event Driven Systems" and "Rust + WASM" North America 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-19 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2026-03-19 | Nashville, TN, US | Music City Rust Developers Applied Rust - Building Rust Applictions 2026-03-19 | New York, NY, US | Rust NYC Rust NYC: Social Interoperability - Rust, C++ and The Greater Good 2026-03-21 | Boston, MA, US | Boston Rust Meetup Porter Square Rust Lunch, Mar 21 2026-03-25 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2026-03-25 | New York, NY, US | Rust NYC Rust NYC's Digital Asset Adoption Special 2026-03-26 | Atlanta, GA, US | Rust Atlanta Rust-Atl 2026-04-02 | Saint Louis, MO, US | STL Rust SIUE Cruft Crawler with LLM 2026-04-09 | San Diego, CA, US | San Diego Rust San Diego Rust April Meetup - Back in person! 2026-04-14 | Charlottesville, VA, US |

Charlottesville Rust Meetup Sharpening Your Rust Skills for Job Interviews Oceania 2026-03-26 | Melbourne, AU | Rust Melbourne TBD March Meetup South America 2026-03-21 | São Paulo, BR | Rust São Paulo Meetup Encontro do Rust-SP (migrado pro Lumma) 2026-04-11 | Argentina, AR | Oxidar Org Oxidar.org Hackaton - Snakear - ¡Veni a hackear con Rust! 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 What we collectively build, beyond the code artifacts that the compiler+tools are, is a group of people who come back, who learn, who share their understanding, who align their tastes, who take input from the community, etc etc. Merging an LLM-generated PR feeds only the “we have code that works” part of the Project; it’s not participating in all the other feedback cycles that make the project alive. – Nadrieril on the Rust Project Perspectives on AI Despite another week without a suggestion, llogiq is pleased with his choice. Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell llogiq ericseppanen extrawurst U007D mariannegoldin bdillo opeolluwa bnchi KannanPalani57 tzilist Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- Thunderbird Blog: VIDEO: Meet the New Support Team! (2026/03/17 22:05)
  Welcome to the first Community Office Hours of 2026! We’re all recovered from conferences in Brussels and California, and we’re talking to members of one of our newest teams: Support. Wait, hasn’t Thunderbird always had a support team? Surprisingly, no! We’ve had individual staff members responsible for different parts of our support efforts, but never combined into one fantastic team. This includes new hires Lisa (Jill) Wess, Manager of Customer Support Operations, and MadHatter McGinnis, Support Knowledge and Enablement Lead, whose work is not only vital for Thundermail, but a better experience for donors, users, and contributors alike. We’re also joined by Roland Tanglao and Wayne Mery, two of our longstanding support and community staff who you might remember from our office hours on writing support articles and helping users on the support forums. Community Office Hours are just getting started this year. Thank you so much for joining us for these sneak peeks into how we make, improve, and expand Thunderbird! As always, if you have any ideas for future office hours topics, let us know in the comments! A New Model for Support While Lisa and MadHatter have only been at Thunderbird for less than a year, the impact they’ve made since then is notable. When Thundermail launches later this year, users will experience their work when they read a user guide, ask for support, or make a suggestion on ideas.tb.pro. This renewed focus on support won’t only be for Thundermail users, however! In the video, the members of the support team discuss how the team wants to make it easier to get help AND easier to provide it, whether the person answering questions is a staff member or a volunteer contributor. No matter what technical improvements the team has planned, human beings will always be at the heart of the Thunderbird (and Thundermail) support experience. This includes the existing Thunderbird support on the Mozilla Support forums. As always, there is so much our community can do to help out fellow Thunderbird users with questions or issues they might have. Roland and Wayne offer useful tips on where to get started on the support contributor journey. One path that might be new to readers is suggesting improvements or new knowledge base (KB) articles! If you have ideas on replacing outdated advice, updating screenshots, or ways to fill knowledge gaps with a new KB article, we’d love to hear from you, whether your advice is about the desktop or Android client. VIDEO (Also on Peertube): Resources: Resources for Suggesting Features:Thunderbird on Desktop and Mobile – https://connect.mozilla.orgThundermail and Thunderbird Pro – https://ideas.tb.pro Resources for Becoming a Community Support Contributor:Join the Support Crew Matrix Channel – https://matrix.to/#/#tb-support-crew:mozilla.orgLearn How to Write Support Articles – https://blog.thunderbird.net/2024/07/video-learn-about-thunderbird-support-articles-and-how-to-contribute/Learn How to Help Support Forum Users – https://blog.thunderbird.net/2024/08/video-how-to-answer-thunderbird-questions-on-mozilla-support/Provide Feedback on Desktop

Support Articles – https://github.com/thunderbird/knowledgebase-issues/issuesProvide Feedback on Android Support Articles – https://github.com/thunderbird/android-knowledgebase-issues/issuesAlready a SUMO Contributor? Join the Contributor Discussion Forum – https://support.mozilla.org/forums/contributors/ Resources for Getting Help with Thunderbird:Thunderbird for Android Support Channel (Matrix) – https://matrix.to/#/#tb-android:mozilla.orgThunderbird Desktop Support Channel (Matrix) – https://matrix.to/#/#thunderbird:mozilla.org The post VIDEO: Meet the New Support Team! appeared first on The Thunderbird Blog.

- Firefox Nightly: Firefox Profiler Dark Mode and Updated Smart Window Prompts – These Weeks in Firefox: Issue 197 (2026/03/17 19:57) Highlights Smart Window added contextual prompts for follow-ups bug 2012208 and conversation starters bug 2014946 SplitView is approaching completion in Nightly and hopes to ride the trains soon. If you see something, say something (File a bug in Firefox: Tabbed Browser: Split View) Session History (Jake Diagrams) is now available in DevTools Set behind a pref devtools.application.sessionHistory.enabled. Feel free to try it out! The Firefox Profiler now supports dark mode thanks to :arai! Friends of the Firefox team Resolved bugs (excluding employees) Volunteers that fixed more than one bug Beatriz Rizental Chris Vander Linden Khalid AlHaddad New contributors (= first patch) Alex Leykin [:alexff]: Firefox's new 'pinned web apps / taskbar web apps' windows (Firefox 143 on Windows 11) don't remember their size or position after being moved/resized and reopened Jacob: downloaded files do not respect referrer-policy when setting kMDItemWhereFroms Leonardo Paffi: Add capability for loadTestPage to take inline HTML Phil [:phigl]: Request size is wrong when request is served from cache Project Updates Add-ons / Web Extensions Addon Manager & about:addons Fixed issue hit by concurrent installations due to error handling potentially removing the entire staging directory used by the other concurrent XPI files pending installation – Bug 2006512 Fixed issue with "Available Updated" badge counter getting out of sync – Bug 1548836 WebExtensions Framework Disabled dom.keyboardevent.init_key_event.enabled_in_addons preference on all channels – Bug 2015079 / Bug 2013772 NOTE: 1Password 4.x relied on a deprecated, non-standard API (initKeyEvent) in Firefox. The deprecated feature has been disabled for the release channel in Firefox versions >= 147. Users still using the discontinued 1password classic extension should migrate to the maintained 1password extension or migrate to another maintained password manager. Investigated and fixed excessive process creation on rapidly clicking extension icons (e.g., uBlock) – Bug 1987679 / Bug 2007742 WebExtension APIs Exposed splitViewId property in tabs extension API to support split view feature – Bug 1993037 Improved cross-browser compatibility for the browserAction and pageAction openPopup API method, in particular allowing it to be called without user interaction (which is an ability that password managers leverages) – Bug 1799344 / Bug 1799347 / Bug 2011516 Fixed issue with browser.storage.onChanged API event not emitted for object writes after opening Storage tab in DevTools – Bug 1994355 Deprecating executeScript API methods support for dynamically executing code into moz-extension documents – Bug 2011234 NOTE: This deprecation is in the process of being documented and communicated to extensions developers. The restriction is active by default on Nightly, whereas it will be logging a deprecation warning on Beta and Release channels. DevTools Chris Vander Linden refactored AccessibilityRow.js and accessibility/panel.js to use proper ES private fields (#2015237, #2015238) Phil [:phigl] fixed the request size of cached request in Netmonitor (#1973003) Nicolas Chevobbe [:nchevobbe] added tooltip to show value of attribute in attr() function (#2011646) Julian Descottes [:jdescottes] made it possible to use Network Throttling in the Browser Toolbox (#2015030) Nicolas Chevobbe [:nchevobbe] fixed an issue where the content dialog (e.g. alert()) were covering the DevTools toolbox (#2013120) Nicolas Chevobbe [:nchevobbe] added inactive CSS indicator for position-area when the element is not absolutely positioned, and/or doesn't have a default anchor (#1895185) Julian Descottes [:jdescottes] similarly fixed an issue with the Browser Screenshot popup being displayed oddly when RDM is enabled (#2016109) Information Management/Sidebar We've started making xul:splitter keyboard accessible. Bug 2012633 – Splitter component is not

accessible landed for splitview splitter. There are others across the product that can make it focusable. New Tab Page With the startup crash fixed, train-hopping has been unblocked! 149.1.20260121.51415 went out to the release and beta channels recently. Nathan Barrett fixed reordered Top Sites showing mismatched names/icons after restart when placed after Add Shortcut, correcting persisted tile indexing in Activity Stream so Top Sites render with the right metadata across sessions. Maxx Crawford added thumbnail attachment to uploaded wallpapers, using the generated thumbnail for previews in the Customize panel to speed image rendering and reduce decode cost while keeping full-res when applied. Nathan Barrett enabled the image proxy backend for newtab, giving us the ability to route newtab image requests through a third-party proxy for better privacy characteristics. This is disabled by default while we investigate proxying strategies. Mike Conley added activation window experiment messaging variants on newtab. This is something we expect to experiment with for new installs starting later this month. Maxx Crawford replaced the Weather widget context menu with the shared panel-list, unifying menu styling and improving keyboard/focus behavior on the New Tab Page. Irene Ni removed auto-placement logic for the Weather widget, preventing unexpected moves and reducing layout shifts so the widget consistently stays where users place it. Performance Tools (aka Firefox Profiler) You can now copy the marker table as a markdown or plain text. Added a “Focus on self only” transform. Stack Chart now includes traced values if you have “JS execution tracing” feature enabled. Search and Navigation Moritz corrected various misbehaviors of the new Search Bar widget: The search bar should properly overflow, and work in the overflow panel – Bug 2006023 Go button was missing when dropping text into the search bar – Bug 2013536 Go button used the previous search engine after changing the default engine – Bug 2013883 The suggestions panel was appearing behind the urlbar when in menubar Bug 2007147 Couldn’t remove search bar history entries – Bug 2014258 Suggest: Drew enabled online Suggest (ohttp) for eligible users in Nightly: Bug 1998009, Bug 1996952, Bug 2014318, Bug 2014296 Drew fixed Show Less Frequently option for Add-on suggestions Bug 2011853 Other notable fixes: Dharma improved name matching for the contextual search feature Bug 2003798 Dale fixed a bug where the number of blocked trackers was wrongly reset in the trust panel when reloading the page Bug 2011494 Smart Window More Smart Window updates: Smartbar with autofill with appropriate cta (go vs search vs ask) bug 2008926 and suggestions list bug 2008977 Conversations stay with a tab when shifting from fullpage to sidebar assistant bug 2012536 Session restore smart window status for all windows bug 2010901 Hooking up footer actions, e.g., copy, retry without memories bug 2010189 Preferences show get started bug 2014852 vs personalization with browser.smartwindow.enabled bug 2014537 Handling error cases including 401 token expiration bug 2015116 and showing them in conversation bug 2010417

- Jonathan Almeida: Test sites for browser developers (2026/03/17 18:58)
  Working on the mobile Firefox team gives you the opportunity to touch on many different parts of the browser space. You often need to test the interaction between web content and the application integration's to another component, say for example, a site registering for a WebPush subscription and Firefox using Firebase Cloud Messaging to deliver the encrypted message to the end-user. Hunting around for an example to validate everything fine and dandy takes time. Sometimes a simple test site for your use case is helpful for initial validation or comparison against other browsers. Below is a list of tests that I've used in the past (in no particular order): WebAuthn / Passkeys & PRF Demo For WebAuthn login/registration, but also includes PRF extension support. Web Push: Data Encryption Test Page Push notifications requires a server to send a notification to the client (not the same as a WebNotification), so you can use this WebPush test site for validating just that. Simple Push Demo Similar to the above, but prettier. Test of HTML5 localStorage and sessionStorage persistence - sharonminsuk.com When you have to verify that localStorage is truly gone. This is especially helpful for Firefox Focus where private browsing is the primary feature. <input>: The HTML Input

element - MDN There are Too Many™ different prompt and input element types. The MDN docs have the best collection of all of them. Browsing privacy and Tracking Protection badssl.com Great for testing out the various error pages that a browser can show. (Safe) Safe Browsing Testing Links For making sure we are still using the Safe Browsing list correctly. senglehardt.com/test/trackingprotection Tracking Protection test page. Somewhat ancient so cookie blocking might not work. Forms and Autocomplete There are various form types and various heuristics to trigger completion options, so they deserve their own section. The more (test sites) the merrier! Registration, Login and Change Forms - MattN Web forms come in all shapes and sizes. Some simple forms to see if we can detect a login/registration form and fill a login entry into them. fill.dev More forms, but also includes credit card and address form filling. daleharvey.github.io/testapp Good for sanity testing simple forms, links that have same/different origins, or (location) permission prompts. Sign-Up & Login Forms - Dimi Sign-up and login forms behave differently, so they are handy to test separately. For example, autofilling a generated password is useful on a registration form but not on a login one. Make your own If you need to make your own, try to write out the code yourself so you can understand the reduced test case. If it's not straight-forward, try using the Testcase Reducer by Thomas Wisniewski. Comments With an account on the Fediverse or Mastodon, you can respond to this post. Since Mastodon is decentralized, you can use your existing account hosted by another Mastodon server or compatible platform if you don't have an account on this one. Known non-private replies are displayed below. Learn how this was implemented from the original source here. Load comments <noscript> Loading comments relies on JavaScript. Try enabling JavaScript and reloading, or visit the original post on Mastodon. </noscript> <noscript>You need JavaScript to view the comments.</noscript> &>"'

- The Mozilla Blog: More reasons to love Firefox: What’s new now, and what’s coming soon (2026/03/17 16:06)
  Firefox is for people who make their own choices online, from what stays private to the tools that help get things done. That commitment to choice shows up throughout the Firefox experience, and AI controls is just the latest example — making it possible to turn generative AI features off, on, or customize them feature by feature. Over the coming weeks, we’ll be rolling out a series of updates that build on that. Expect more control where it matters, better protections in the background, and a few new tools that make everyday browsing better. You may even spot a fresh face of Firefox along the way. Here’s a quick look at what to watch for: An easier way to personalize Firefox The Settings section has been improved with clear navigation and search, so it’s simpler to customize Firefox. Available soon in Firefox Nightly. Privacy upgrades, built in A free built-in VPN is coming to Firefox. Free VPNs can sometimes mean sketchy arrangements that end up compromising your privacy, but ours is built from our data principles and commitment to be the world’s most trusted browser. It routes your browser traffic through a proxy to hide your IP address and location while you browse, giving you stronger privacy and protection online with no extra downloads. Users will have 50 gigabytes of data monthly in the U.S., France, Germany and U.K. to start. Available in Firefox 149 starting March 24. We also recently shared that Firefox is the first browser to ship Sanitizer API, a new web security standard that blocks attacks before they reach you. New tools to get more done Smart Window, previously called AI Window, uses AI to give quick help while you browse — like quick definitions, article summaries, product comparisons and more — without leaving the page. It’s completely optional to use and will be opt in. Waitlist now open for updates and early access. Split view puts two webpages side by side in one window, making it easy to compare, copy and multitask without bouncing between tabs. Rolling out in Firefox 149 on March 24. Tab Notes let you add notes to any tab, another tool to help with multitasking and picking up where you left off. Available in Firefox Labs 149 starting March 24. “The roadmap for Firefox this year is the most exciting one we’ve developed in quite a while. We’re solely focused on building the best browser, and our features over the next few months and beyond are driven by the feedback

from our community," said Ajit Varma, head of Firefox. "We're improving the fundamentals like speed and performance. We're also launching innovative new open standards in Gecko to ensure the future of the web is open, diverse, and not controlled by a single engine. At the same time we're prioritizing features that give users real power, choice and strong privacy protections, built in a way that only Firefox can. And as always, we'll keep listening, inviting users to help shape what comes next and giving them more reasons to love Firefox." Welcome new and improved Firefox… and Kit! Firefox is also getting a fresh new look across our website, product and beyond. This includes updated themes, icons, and visual refinements across Firefox, including our toolbars, menus, and the homepage. It also brings usability improvements that make key features easier to access. The changes reflect user feedback and aim to modernize the browser while reinforcing a more distinctly Firefox look and feel. We appreciate the community feedback on the designs. We'll be sharing more soon. You also may have noticed something else showing up in Firefox recently: our new mascot, Kit. Kit is your companion in this new internet era, our way of making Firefox's support visible and bringing a little warmth and familiarity as you browse. More to come Choosing Firefox means using the internet on your own terms: options when you want them, safety when it matters most, and tools that make the web easier to navigate. Download the latest version of Firefox to try what's new, and let us know what you think on Mozilla Connect. Get Firefox on desktop and mobile Download now The post More reasons to love Firefox: What's new now, and what's coming soon appeared first on The Mozilla Blog.

- The Mozilla Blog: Meet Kit, your companion for a new internet era (2026/03/17 15:00)
  The web shouldn't feel like it's working against you. Yet so much of it now is designed to pull you off course: endless feeds, pop-up windows and content that looks credible until it isn't. Staying focused and trusting your next click takes more effort than it should. Firefox is here to help you navigate the web on your own terms. That means a browser built for people: one that doesn't sell your data, that's open and optional with AI and that isn't owned by billionaires. And as we enter a new internet era – one shaped by AI and a web that's harder to trust – Kit is our way of making our support visible, a companion that brings some warmth and familiarity when you're browsing with Firefox. Where you'll see Kit In Firefox, Kit may appear in moments that are meant to feel welcoming or encouraging: when you're getting started, discovering a new feature or hitting a small win (like when you've successfully made a setting change). You'll also see Kit outside the browser, like on our product website, the blog, across social media and in campaigns. If you want, you can set Kit as your new tab wallpaper (bottom right corner: Customize > Firefox). You may even spot Kit in the real world at our community events. Kit is a companion, not a commentator. They're not here to deliver punchlines. Kit shows up as a small signal that Firefox is working for you, then steps back so you can keep moving. How we made Kit To bring Kit to life, we partnered with creative agency JKR and illustrator Marco Palmieri. We chose JKR because they've helped iconic character brands evolve for modern audiences, and we needed collaborators who could build a companion with range – not just a mascot. Marco helped shape Kit's personality through the kind of craft that only comes from drawing characters for a living. He started the way he always does: with a pencil. "I tend to stay away from the computer at the beginning," he said. "I want as few obstacles as possible between me and what I'm trying to see." From there, the work moved into Illustrator, where Kit could be refined through many rounds of iteration. Two decisions shaped Kit along the way. First: the tail. It isn't just a signature detail. It's part of Kit's personality, helping carry motion and emotion even in still moments. Second: no mouth. We wanted Kit to feel expressive without tipping into "talking cartoon" territory. So expression lives in the eyes, posture and body language instead. Our teams explored how literal Kit should be to the Firefox logo. Fox proportions mattered, but so did making something new. At one point, we asked ourselves whether or not Kit should look more like a red panda (leading to a brief detour into red panda references). In the end, we agreed that Kit isn't a fox nor a red panda. Kit is a Firefox, its own creature — with attributes from both a fox and a red panda, and

perhaps a little fire magic.Marco wants Firefox fans to meet Kit with curiosity. “I hope that they would want to look,” he said. “Not just glance and move on, but feel like there’s a little character here you might actually want to get to know.” Kit was created by a team of people, not generated by AI. And Kit isn’t an AI assistant or a chatbot. Kit came from hundreds of small choices — tail flicks, textures, gradients, proportions — made deliberately, then refined until the character felt right to what Firefox stands for. More fire. More fox. The web can feel like it’s trying to wear you down: pulling at your attention, keeping you stuck with defaults and asking for more than it should. That’s why Firefox exists, and why the internet needs it. Firefox has your back. And we hope Kit helps make that promise visible, with conviction where it matters and lightness when it helps. You deserve a browser that respects your humanity, agency and privacy — one that’s built for people, not profit. The internet as it should be. Get Firefox on desktop and mobile Download now The post Meet Kit, your companion for a new internet era appeared first on The Mozilla Blog.

- Firefox Tooling Announcements: New Deploy of PerfCompare (March 16th) (2026/03/16 21:12)
  The latest version of PerfCompare is now live! Check out the change-log below to see the updates: Highlights: [kala-moz] Test version refactor: column config and rendering (#1004) BUG 2019125: Remove replicates toggle for mwu test version (#1006) BUG 2017544: update MWU tooltips with more info and links (#1008) [moijes12] Bug-1999558: Display complete commit message even when newline is absent (#1000) Bug-1915777: Make test tags theme aware (#993) Bug-1948879: Add links to docs and source in the Banner (#995) Thank you for the contributions! Bugs or feature request can be filed on Bugzilla. The team can also be found on the #perfcompare channel on Element. Come and chat! 1 post - 1 participant Read full topic
- Firefox Tooling Announcements: Mach Performance Improvements (2026/03/13 20:22)
  Hi all, I wanted to share some of the performance optimizations for mach that have landed recently or are landing soon: bug 1775197 reduced the overhead of mach’s initialization process by ~30%-50% (highly platform and command specific), which means that commands you invoke could begin executing up to a second sooner than before, which should make things feel much more responsive. bug 2018327 sped up a pre-test step that verifies all test files are in the right place before tests start running. On Windows, this shaved ~13 seconds off the time between invoking ./mach test/./mach xpcshell-test and tests actually starting to run (~75% speedup). Minor (if any) improvement on Linux/macOS. bug 2017746 (landing today) speeds up ./mach configure by adding caching to a taskcluster step that verifies you have up-to-date toolchains, which saves up to 10 seconds (~50% speedup) if none of the toolchain input files changed since the previous run. The most common scenario that benefits from this is editing your mozconfig then building. 1 post - 1 participant Read full topic
- Firefox Application Security Team: Bug Bounty Program Updates 2026 (2026/03/13 02:13)
  The Firefox bug bounty program is the longest-running security bug bounty program. Born out of Netscape’s bug bounty program, we’ve been awarding ingenious security research for over two decades, helping keep our hundreds of millions of users safe. With the threat landscape changing, we’re updating our security program along with it. As browser security architecture continues to improve, our bug bounty program is evolving to focus incentives on the highest-impact work and the most critical threats. Policy changes We are keeping updates to the reporting process to a minimum so that experienced and successful researchers do not need to change how their security bugs are reported right now. Raising the bar for report quality and novelty Quality: As part of our policy update, we are putting an increasing emphasis on actionable reports: Reducing the time spent on validating theoretical concerns and focusing on confirmed vulnerabilities is ensuring faster resolution, faster reward payments and a better Firefox. Our existing policy already differentiates between high-quality reports and reports that only provide sufficient

information to inform us of a new security vulnerability. Going forward, we will now require security researchers to provide simple and reproducible test cases for reports to be eligible for bounty rewards (e.g., demonstrable with an Address Sanitizer build). Reports without any credible evidence of a security issue will be assigned a much lower priority. Novelty: In light of a recent increase in duplicate security bug reports and timing issues where researchers were able to race some of our internal tools, we are reserving an increased time window of seven days for our internal automated methods to find security issues. While this might affect more bug reports than before, we also believe this encourages researchers to focus on novel research that genuinely benefits advancements in the security posture, rather than research that replicates existing open source tools or results already captured by internal processes. This puts original, high-quality research back in the center of our bug bounty program Aligning Security Architecture and Reward Structure Security Architecture: Firefox is already using multiple utility processes to reduce the level of privilege for security-critical code. We are therefore focusing our rewards for “Highest Impact” / “Sandbox Escape” solely on attacks that compromise the parent process. We are also going to release a separate, sandboxed GPU process on all Operating Systems in 2026, which means that from now on, we will assess vulnerabilities in our graphics stack according to their sandboxed context, rather than treating them as full sandbox escapes. These bugs will still be considered as “High Impact” memory corruptions. We are also working on better definitions for compromising other processes and will update the bounty program again with more eligible process pairs in the future. Changes to our Reward Structure: We limit our definition of sandbox escapes to true escapes that will allow the attacker to gain control over a higher privileged, unsandboxed process. This definition also excludes memory-reads as well as cross-process exploits targeting other sandboxed processes. As with above, we will continue to rate memory corruptions and attacks on other processes as “High Impact“ findings. Consequently, the highest reward will be limited to breaches of the most hardened security boundaries. Join our bug bounty program These updates are a natural step forward in maintaining a highly effective, quality-focused program that addresses the real and evolving security risks facing modern browsers. We encourage researchers to engage with these clarified guidelines and focus their efforts on high-impact vulnerabilities and continue working alongside us to strengthen Firefox by focusing on the vulnerabilities that matter most. We will provide additional guides and tips for improved bug finding and reporting based on previous bug reports in upcoming blog posts. We are also reminding our readers and security community that we welcome guest articles on our blog where successful researchers can share their discoveries.

- The Rust Programming Language Blog: Call for Testing: Build Dir Layout v2 (2026/03/13 00:00)
  We would welcome people to try and report issues with the nightly-only cargo -Zbuild-dir-new-layout. While the layout of the build dir is internal-only, many projects need to rely on the unspecified details due to missing features within Cargo. While we've performed a crater run, that won't cover everything and we need help identifying tools and process that rely on the details, reporting issues to these projects so they can update to the new layout or support them both. How to test this? With at least nightly 2026-03-10, run your tests, release processes, and anything else that may touch build-dir/target-dir with the -Zbuild-dir-new-layout flag. For example: $ cargo test -Zbuild-dir-new-layout Note: if you see failures, the problem may not be isolated to just -Zbuild-dir-new-layout. With Cargo 1.91, users can separate where to store intermediate build artifacts (build-dir) and final artifacts (still in target-dir). You can verify this by running with only CARGO_BUILD_BUILD_DIR=build set. We are evaluating changing the default for build-dir in #16147. Outcomes may include: Fixing local problems Reporting problems in upstream tools with a note on the the tracking issue for others Providing feedback on the the tracking issue Known failure modes: Inferring a [[bin]]s path from a [[test]]s path: Use std::env::var_os("CARGO_BIN_EXE_*") for Cargo 1.94+, maybe keeping the inference as a fallback for older Cargo versions Use env!("CARGO_BIN_EXE_*") Build scripts looking up target-dir from their binary or OUT_DIR: see Issue #13663 Update current workarounds to

support the new layout Looking up user-requested artifacts from rustc, see Issue #13672 Update current workarounds to support the new layout Library support status as of publish time: assert_cmd: fixed cli_test_dir: Issue #65 compiletest_rs: Issue #309 executable-path: fixed snapbox: fixed term-transcript: Issue #269 test_bin: Issue #13 trycmd: fixed What is not changing? The layout of final artifacts within target dir. Nesting of build artifacts under the profile and the target tuple, if specified. What is changing? We are switching from organizing by content type to scoping the content by the package name and a hash of the build unit and its inputs. Here is an example of the current layout, assuming you have a package named lib and a package named bin, and both have a build script: build-dir/ ├── CACHEDIR.TAG └── debug/ ├── .cargo-lock # file lock protecting access to this location ├── .fingerprint/ # build cache tracking │ ├── bin-[BUILD_SCRIPT_RUN_HASH]/* │ ├── bin-[BUILD_SCRIPT_BIN_HASH]/* │ ├── bin-[HASH]/* │ ├── lib-[BUILD_SCRIPT_RUN_HASH]/* │ ├── lib-[BUILD_SCRIPT_BIN_HASH]/* │ └── lib-[HASH]/* ├── build/ │ ├── bin-[BIN_HASH]/* # build script binary │ ├── bin-[RUN_HASH]/out/ # build script run OUT_DIR │ ├── bin-[RUN_HASH]/* # build script run cache │ ├── lib-[BIN_HASH]/* # build script binary │ ├── lib-[RUN_HASH]/out/ # build script run OUT_DIR │ └── lib-[RUN_HASH]/* # build script run cache ├── deps/ │ ├── bin-[HASH]* # binary and debug information │ ├── lib-[HASH]* # library and debug information │ └── liblib-[HASH]* # library and debug information ├── examples/ # unused in this case └── incremental/... # managed by rustc The proposed layout: build-dir/ ├── CACHEDIR.TAG └── debug/ ├── .cargo-lock # file lock protecting access to this location ├── build/ │ ├── bin/ # package name │ │ ├── [BUILD_SCRIPT_BIN_HASH]/ │ │ │ ├── fingerprint/* # build cache tracking │ │ │ └── out/* # build script binary │ │ ├── [BUILD_SCRIPT_RUN_HASH]/ │ │ │ ├── fingerprint/* # build cache tracking │ │ │ ├── out/* # build script run OUT_DIR │ │ │ └── run/* # build script run cache │ │ └── [HASH]/ │ │ ├── fingerprint/* # build cache tracking │ │ └── out/* # binary and debug information │ └── lib/ # package name │ ├── [BUILD_SCRIPT_BIN_HASH]/ │ │ ├── fingerprint/* # build cache tracking │ │ └── out/* # build script binary │ ├── [BUILD_SCRIPT_RUN_HASH]/ │ │ ├── fingerprint/* # build cache tracking │ │ ├── out/* # build script run OUT_DIR │ │ └── run/* # build script run cache │ └── [HASH]/ │ ├── fingerprint/* # build cache tracking │ └── out/* # library and debug information └── incremental/... # managed by rustc For more information on these Cargo internals, see the mod layout documentation. Why is this being done? ranger-ross has worked tirelessly on this as a stepping stone to cross-workspace caching which will be easier when we can track each cacheable unit in a self-contained directory. This also unblocks work on: Automatic cleanup of stale build units to keep disks space use constant over time More granular locking so cargo test and rust-analyzer don't block on each other Along the way, we found this helps with: Build performance as the intermediate artifacts accumulate in deps/ Content of deps/ polluting PATH during builds on Windows Avoiding file collisions among intermediate artifacts While the Cargo team does not officially endorse sharing a build-dir across workspaces, that last item should reduce the chance of encountering problems for those who choose to. Future work We will use the experience of this layout change to help guide how and when to perform any future layout changes, including: Efforts to reduce path lengths to reduce risks for errors for developers on Windows Experimenting with moving artifacts out of the --profile and --target directories, allowing sharing of more artifacts where possible In addition to narrowing scope, we did not do all of the layout changes now because some are blocked on the lock change which is blocked on this layout change. We would also like to work to decouple projects from the unspecified details of build-dir.

- The Mozilla Blog: Under the hood: The AI powering Firefox’s Shake to Summarize (2026/03/12 17:57)
  We recently released a feature in the Firefox iOS mobile app called “Shake to Summarize”. The reception was remarkably positive, earning an honorable mention on Time Magazine’s best inventions of 2025. For anyone unfamiliar with Shake to Summarize, it’s just what the name implies: when you’re browsing a webpage, you can shake your phone to generate a short summary of the page’s content. The gesture is fun, the feature

is useful, and the whole thing feels simple and natural. From a technical standpoint, the application works just how you'd imagine: when a shake (or lightning bolt-icon press) is detected, we grab the web page content, pass it to an LLM for summarization, and then return the result to the user. But with the LLM landscape being as vast as it is, there is a lot to consider when bringing even a relatively straightforward application to the market. In this post, we'll discuss the ins and outs of our approach to model selection. We will leave prompt development and quality testing for a future article. Which model? These days, there are many LLMs available, with a steady stream of new releases arriving almost every week.. Each release is paired with a slate of benchmark scores, showing the new model's superiority along one dimension or another. The pace of development has been fast and furious and billions of dollars have been spent inching the numbers higher and higher.But what do these metrics mean in practice? At the end of the day, we are building a product for users. The most important metric for us is, "how useful are the summaries the model produces?" – something that isn't neatly captured by the benchmark scores. To select the best model for our applications, we need to run our own tests. For us, the best model had to excel along several dimensions: First, summary quality. That's the whole point, after all. Second, speed. The model needs to return summaries relatively quickly. If it takes the same amount of time to produce the summary as it does to read the article – we've lost. Third, cost. Since we do not charge for use of the Shake to Summarize feature, inference costs are entirely on us (you're welcome). Finally, open source. Supporting open source projects is a core value here at Mozilla. As such, we prefer to use open source models in our applications, when possible (in this case, we had to settle for open weights). Keeping the above in mind, we selected the following models for our initial evaluation: Mistral Nemo, Mistral Small, Jamba 1.5 mini, Gemini flash 2.0 flash and Llama 4 Maverick – all of which were hosted on Vertex AI. Note: this project began in early 2025 Quality Standard summary evaluation metrics such as BLEU and ROUGE rely on token overlap and do not correlate well with human judgement. Thus, we decided to use an LLM judge (GPT-4o) to evaluate our model candidates. We had each model generate summaries of the same set of webpages, and then asked the LLM judge to evaluate each summary on the following metrics: Coherence: Does the summary read logically and clearly as a standalone text? Consistency: Is the information in the summary accurate and faithful to the source? Are there any hallucinations? Relevance: Does the summary focus on the most important content from the document? Fluency: Is the summary grammatically correct, fluent, and well-written? To get a single, comparable metric, we then averaged these scores. From this analysis, we see that Google's Gemini 2.0 Flash, Meta's Llama 4 Maverick, and Mistral small are top performers – with Gemini consistently leading the pack. We see that the top three models are roughly equivalent on short passages up to around 2000 tokens (which is roughly the length of the average webpage), but performance separates more as passages get longer – particularly those containing over 5000 tokens*. *We note that, due in part to this performance degradation, we summarize only pages that are shorter than this 5000 token threshold. Speed For speed, the two metrics we looked at were time to first token (i.e. how long do you have to wait before the model starts generating its response) and tokens-per-second (total tokens generated / total generation time, including encoding time). In both of these tests, Mistral-Small and Gemini-2.0-flash are the clear winners. Both models are faster to begin generating output and produce tokens at a much faster clip than the other models we tested. Cost On Vertex AI serverless instances, as of November 2025, the cost for input tokens for our top 3 models are as follows. (See all Vertex AI pricing here): ModelPrice / M input tokensPrice / M output tokensGemini 2.5 Flash (2.0 no longer available)$0.30$2.50Llama4-Maverick$0.35$1.15Mistral Small$0.10$0.30 It's clear that Mistral small over-performs from a quality and performance / dollar standpoint, costing one-third of the price or less per input token (which is where the bulk of our token usage is) compared to the other two models. Open source Our top priority is building a great user experience. We also believe that open source software is an integral part of building a healthy internet. When we can support open source while still delivering the highest quality experience, we will. In this

category, Llama4 Maverick and Mistral Small come out ahead. While neither is fully open source (no training code or data has been released), both models have open weights paired with liberal usage policies. Gemini 2.5 Flash, on the other hand, is a proprietary model. Model selection When we considered all of the above, we decided to go with Mistral-Small to power our feature: it's fast, it's inexpensive, it has open weights, and it produces high quality summaries. What's not to like? Release and future directions After selecting the model, we iterated on the prompt to ensure that we were delivering the best experience (see the upcoming blog post: Shake to Summarize: Prompt Engineering), and we released the solution in September of 2025. This project was an early foray in building LLM-powered features into the browser. As such, the model selection process we developed here helped us chart the course for model selection in our later AI integrations. Notably, the soon-to-be released Smart Window required choosing not just one, but multiple models to power the application—giving users increased control over their experience.Throughout this process, we learned that the "best" model isn't the one with highest benchmark scores. It's the one that fits the context in which it's used—aligning with the task, the budget, and Mozilla's commitment to open source. The post Under the hood: The AI powering Firefox's Shake to Summarize appeared first on The Mozilla Blog.

- Firefox Tooling Announcements: Happy BMO Push Day! (20260312.1) (2026/03/12 17:43)
  Github Link The following changes have been pushed to bugzilla.mozilla.org: Bug 1995468 - Support URL pasting in markdown editor Bug 2018260 - "Fields You Can Search On" is blocking people from making it through quicksearch.html doc Bug 2022581 - Support Git trailer convention when parsing bug numbers out of commit messages Bug 2022370 - Support updated uplift request form field names from Lando Discuss these changes in the BMO Matrix Room 1 post - 1 participant Read full topic
- The Mozilla Blog: The web should remain anonymous by default (2026/03/12 12:00)
  The unique architecture of the web enables a much higher degree of user privacy than exists on other platforms. Many factors contribute to this, but an essential one is that you don't need to log in to start browsing. Sharing details about yourself with a website is an optional step you can take when you have reason to do so, rather than the price of admission. These norms mirror those of a free society. You can walk down the street without wearing a name tag or prove who you are to passersby. You can enter a store without introducing yourself, and only open your wallet if you decide to buy something. You aren't hiding anything, but society shows restraint in what it asks and observes, which allows you to be casually anonymous. When this is the default, everyone can freely enjoy the benefits of privacy without having to go to great lengths to hide their identity – something that isn't practical for most people. It's easy to take casual anonymity for granted, but it depends on a fragile equilibrium that is under constant threat. One way to erode casual anonymity is with covert surveillance, like a snoop following you around town or listening to your phone calls. For more than a decade, Mozilla has worked hard to close technical loopholes — like third-party cookies and unencrypted protocols — used by third parties to learn much more about you than you intended to share with them. The work is far from done, but we're immensely proud of how much less effective this kind of surveillance has become. But there's also a different kind of threat, which is that sites begin to explicitly reject the norm of casual anonymity and move to a model of "papers, please". This isn't a new phenomenon: walled gardens like Facebook and Netflix have long operated this way. However, several recent pressures threaten to tip the balance towards this model becoming much more pervasive. First, increasing volume and sophistication of bot traffic — often powering and powered by AI — is overwhelming sites. Classic approaches to abuse protection are becoming less effective, leading sites to look for alternatives like invasive fingerprinting or requiring all visitors to log in. Second, jurisdictions around the world are beginning to mandate age restrictions for certain categories of content, with many implementations requiring users to present detailed identity information in order to access often-sensitive

websites. Third, new standardized mechanisms for digital government identity make it much more practical for sites to demand hard identification and thus use it for all sorts of new purposes, which may be expedient for them but not necessarily in the interest of everyone’s privacy. All of these pressures stem from real problems that people are trying to solve, and ignoring them will not make them go away. Left unchecked, the natural trajectory here would be the end of casual anonymity. However, Mozilla exists to steer emerging technology and technical policy towards better outcomes. In that vein, we’ve identified promising technical approaches to address each of these three pressures while maintaining or even strengthening the privacy we enjoy online today. A common theme across these approaches is the use of cryptography: some new, some old. For example, most people have at least one online relationship with an entity who knows them well (think banks, major platforms, etc). Zero-knowledge proof protocols can let other sites use that knowledge to identify visitors as real humans, not bots. Careful design of the protocols maintains privacy by preventing sites from learning any additional information beyond personhood. We’ll be sharing more about these approaches over the coming months. Some details are still evolving in collaboration with our partners in the ecosystem, but we are confident it is possible to address abuse, age assurance, and civic authentication without requiring the web to abandon casual anonymity. The web is special and irreplaceable — let’s work together to preserve what makes it great. The post The web should remain anonymous by default appeared first on The Mozilla Blog.

- The Rust Programming Language Blog: Announcing rustup 1.29.0 (2026/03/12 00:00)
  The rustup team is happy to announce the release of rustup version 1.29.0. Rustup is the recommended tool to install Rust, a programming language that empowers everyone to build reliable and efficient software. What's new in rustup 1.29.0 Following the footsteps of many package managers in the pursuit of better toolchain installation performance, the headline of this release is that rustup has been enabled to download components concurrently and unpack during downloads in operations such as rustup update or rustup toolchain and to concurrently check for updates in rustup check, thanks to a GSoC 2025 project. This is by no means a trivial change so a long tail of issues might occur, please report them if you have found any! Furthermore, rustup now officially supports the following host platforms: sparcv9-sun-solaris x86_64-pc-solaris Also, rustup will start automatically inserting the right $PATH entries during rustup-init for the following shells, in addition to those already supported: tcsh xonsh This release also comes with other quality-of-life improvements, to name a few: When running rust-analyzer via a proxy, rustup will consider the rust-analyzer binary from PATH when the rustup-managed one is not found. This should be particularly useful if you would like to bring your own rust-analyzer binary, e.g. if you use Neovim, Helix, etc. or are developing rust-analyzer itself. Empty environment variables are now treated as unset. This should help with resetting configuration values to default when an override is present. rustup check will use different exit codes based on whether new updates have been found: it will exit with 100 on any updates or 0 for no updates. Furthermore, @FranciscoTGouveia has joined the team. He has shown his talent, enthusiasm and commitment to the project since the first interactions with rustup and has played a significant role in bring more concurrency to it, so we are thrilled to have him on board and are actively looking forward to what we can achieve together. Further details are available in the changelog! How to update If you have a previous version of rustup installed, getting the new one 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. In particular, anti-malware scanners might block rustup or stop it from creating or copying files, especially when installing rust-docs which contains many small files. Issues like this should be automatically

resolved in a few weeks when the anti-malware scanners are updated to be aware of the new rustup release. Thanks Thanks again to all the contributors who made this rustup release possible!

- This Week In Rust: This Week in Rust 642 (2026/03/11 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.bsky.social on Bluesky 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 1.94.0 | Rust Blog Newsletters State of Rust Survey Findings Project/Tooling Updates Release 0.7.0 · uutils/coreutils mdterm v1.0.0 - A terminal-based Markdown browser The Anatomy of a 500ns Parser: Porting libphonenumber to Rust mini-agent: A Rust AI Agent Framework ClickHouse meets SeaORM: Arrow-powered data pipeline Rustaceans.AI Leptodon 1.0.0: UI toolkit for the Leptos WASM framework Signing Rust Binaries Shouldn't Require Shell Scripts Observations/Thoughts symbolic derivatives and the rust rewrite of RE# | ian erik varatalu The State of Allocators in 2026 [series] FORTRAN to Rust: part 1 The Cost of Indirection in Rust Why SeaORM over JavaScript client database options? Rust is slowly but surely eating PostgreSQL: Deep dive into Neon, ParadeDB, PgDog and more What Happens When You Constrain an Event-Driven System to Three Primitives My Rust dev setup in 2026 [audio] Netstack.FM episode 30 — uReq with Martin Algesten Weighing up Zngur and CXX for Rust/C++ Interop Rust Walkthroughs ZK snarks for rust developer part 1/8 Get in Line (Part 2) - Vyukov's Queue and its specializations How to stop fighting with coherence and start writing context-generic trait impls Rewriting Our Database in Rust OpenTelemetry for Rust Developers - The Complete Implementation Guide Miscellaneous Rust Shined Over Python for My CLI Tool - Smiling Dev Blog Write small Rust scripts Crate of the Week This week's crate is sentencex, a fast sentence segmentation library. Thanks to Santhosh Thottingal for the 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, Cargo, Rustup or Rust language RFCs. 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. diesel-guard - REFRESH MATERIALIZED VIEW without CONCURRENTLY diesel-guard - ADD CHECK CONSTRAINT without NOT VALID diesel-guard - ADD FOREIGN KEY without NOT VALID diesel-guard - no lock_timeout/statement_timeout before DDL 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 Bluesky 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. Rust India Conference 2026 | CFP open until 2026-03-14 | Bangalore, IN | 2026-04-18 Oxidize Conference | CFP open until 2026-03-23 | Berlin, Germany | 2026-09-14 - 2026-09-16 EuroRust | CFP open until 2026-04-27 | Barcelona, Spain | 2026-10-14 - 2026-10-17 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 Bluesky or Mastodon! Updates from the Rust Project 483 pull requests were merged in the last week Compiler enable PassMode::Indirect { on_stack: true, .. } tail call

arguments Library constify Vec::{into, from}_raw_parts{_in|_alloc} implement MaybeDangling compiler support stabilize control_flow_ok Cargo compile: Turn warning summaries into errors also fix: Switch from ad-hoc to structured warnings script: surpress unused_features lint for embedded tests: allow for 'could not' as well as couldn't in test output add missing truncate when writing .crate files ignore implicit std dependencies in unused-crate-dependencies lint let git decide when to run gc split build-dir lock into dedicated lock Clippy add manual_pop_if lint doc_paragraphs_missing_punctuation: Trim picture symbols do not materialize snippets when it is not needed to fix ICE in match_same_arms fix ICE in swap_binop() fix ICE when using the min_generic_const_args incomplete feature fix infinite_loop wrong suggestion inside conditional branches fix redundant_closure suggests wrongly when local is derefed to callable fix unnecessary_safety_comment false positive on code blocks inside inner docs fix semicolon-inside-block inside try_blocks optimize allow_unwrap_types evaluation to eliminate performance regression Rust-Analyzer do not re-query source roots per crate in analysis-stats offer destructure_struct_binding on self param when going to def on ? on Result that goes through From, go to the From impl add has_pending methods to Incoming/Outgoing/ReqQueue in lsp_server cfg_select supports non token-tree tokens align is_rust() with rustc by correcting constructor ABI in next solver do not use PostAnalysis TypingMode for IDE method resolution file watcher should watch directories recursively fix wrong descend range for add_missing_match_arms offer block .let in ref-expr in match arm Rust Compiler Performance Triage Almost no regressions this week, while there was a handful of performance improvements caused by the ongoing refactoring of the compiler query system. The largest one was from #153521. Triage done by @kobzol. Revision range: ddd36bd5..3945997a Summary: (instructions:u) mean range count Regressions (primary) 0.4% [0.4%, 0.5%] 3 Regressions (secondary) 0.6% [0.1%, 1.2%] 8 Improvements (primary) -0.9% [-2.5%, -0.1%] 110 Improvements (secondary) -0.8% [-2.7%, -0.1%] 77 All (primary) -0.9% [-2.5%, 0.5%] 113 0 Regressions, 6 Improvements, 3 Mixed; 5 of them in rollups 31 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 Remove ATTRIBUTE_ORDER No Items entered Final Comment Period this week for Rust RFCs, Cargo, Compiler Team (MCPs only), Language Team, Language Reference, Leadership Council 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: Custom lint profiles Upcoming Events Rusty Events between 2026-03-11 - 2026-04-08 Virtual 2026-03-11 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-12 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-03-17 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-18 | Virtual (Cardiff, UK) | Rust and C++ Cardiff Hybrid event with Rust Dortmund! 2026-03-18 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-20 | Virtual | Packt Publishing Limited Rust Adoption, Safety, and Cloud with Francesco Ciulla 2026-03-24 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2026-03-24 | Virtual (London, UK) | Women in Rust Lunch & Learn: Crates, Tips & Tricks Lightning Talks - Bring your ideas! 2026-03-25 | Virtual (Girona, ES) | Rust Girona Rust Girona Hack & Learn 03 2026 2026-03-26 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-04-01 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-04-01 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2026-04-02 | Virtual (Nürnberg, DE) | Rust Nuremberg Rust Nürnberg online 2026-04-04 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup Asia 2026-03-22 | Tel Aviv-yafo, IL | Rust TLV In person Rust March 2026 at AWS in Tel Aviv Europe 2026-03-11 | Amsterdam, NL | Rust Developers Amsterdam Group Meetup @ Instruqt 2026-03-11 |

Frankfurt, DE | Rust Rhein-Main Writing a Python compiler in Rust 2026-03-12 | Bern, CH | Rust Bern 2026 Rust Talks Bern #1 @bespinian 2026-03-12 | Geneva, CH | Post Tenebras Lab Rust Meetup Geneva 2026-03-18 | Dortmund, DE | Rust Dortmund Rust Dortmund Meetup - Intro to Embedded Rust - March 2026-03-19 - 2026-03-20 | Warsaw, PL | Rustikon Rustikon Conference 2026-03-23 | Augsburg, DE | Rust Meetup Augsburg Rust Meetup #18: Ludwig Weinzierl - Bevy: Spielentwicklung mit Rust 2026-03-24 | Aarhus, DK | Rust Aarhus Hack Night - Advent of Code 2026-03-24 | Manchester, UK | Rust Manchester Rust Manchester March Code Night 2026-03-24 | Trondheim, NO | Rust Trondheim Rust projects - show and tell in March 2026-03-26 | Paris, FR | Rust Paris Rust meetup #84 2026-03-27 | Paris, FR | Rust in Paris Rust in Paris 2026-04-01 | Oxford, UK | Oxford ACCU/Rust Meetup. Rust/ACCU meetup. North America 2026-03-12 | Lehi, UT, US | Utah Rust An Interpreter for Computability theory, Written the Hard Way 2026-03-12 | San Diego, CA, US | San Diego Rust San Diego Rust March Meetup - Back in person! 2026-03-14 | Boston, MA, US | Boston Rust Meetup North End Rust Lunch, Mar 14 2026-03-17 | San Francisco, CA, US | San Francisco Rust Study Group Rust Hacking in Person 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-19 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2026-03-19 | Nashville, TN, US | Music City Rust Developers Applied Rust - Building Rust Applictions 2026-03-19 | New York, NY, US | Rust NYC Rust NYC: Social Interoperability - Rust, C++ and The Greater Good 2026-03-21 | Boston, MA, US | Boston Rust Meetup Porter Square Rust Lunch, Mar 21 2026-03-25 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2026-03-25 | New York, NY, US | Rust NYC Rust NYC's Digital Asset Adoption Special 2026-03-26 | Atlanta, GA, US | Rust Atlanta Rust-Atl 2026-04-02 | Saint Louis, MO, US | STL Rust SIUE Cruft Crawler with LLM Oceania 2026-03-12 | Brisbane City, AU | Rust Brisbane Rust Brisbane Mar 2026 2026-03-26 | Melbourne, AU | Rust Melbourne TBD March Meetup South America 2026-03-21 | São Paulo, BR | Rust São Paulo Meetup Encontro do Rust-SP (migrado pro Lumma) 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 Happy "Clippy, you are very helpful" day for those who celebrates! – Manpacket on functional.cafe Despite a lamentable lack of suggestions, llogiq is exceedingly pleased with his choice. Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell llogiq ericseppanen extrawurst U007D mariannegoldin bdillo opeolluwa bnchi KannanPalani57 tzilist Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- Firefox Nightly: AI Controls – These Weeks in Firefox: Issue 196 (2026/03/10 00:59)
  Highlights Shout out to mstriemer for getting the AI Controls work landed and uplifted! You choose. We've also enabled New Tab with Sections by default in the UK! Friends of the Firefox team Resolved bugs (excluding employees) Volunteers that fixed more than one bug Henry Yeary Khalid AlHaddad Sam Johnson Thomas Sileo New contributors (= first patch) aaryamannsingh3 made it so that we don't show the "Move Tab to New Window" tab context menu option if there's only a single tab in the window Henry Yeary fixed a glitch with one of our reorderable stories from using the wrapped argType! Henry also cleaned up some dead code in our PDF reader. kian fixed a bug in our DevTools Inspector tool where shadowroots were sometimes not always visible simon cleaned up a redundant argument in part of our layout code! Thomas Sileo made it so that we use the auto_bookmark transition type via the WebNavigation WebExtension API for navigations triggered by the bookmarks toolbar, and also fixed an intermittent test failure! Project Updates Add-ons / Web Extensions WebExtension APIs As part of the work for the AI toggle switch, we have applied some small tweaks to improve the behavior of the the trialML WebExtensions API when it is disable/enabled globally at runtime through AI toggle switch – Bug 2012543 As part of improving cross-browser compatibility of the browserAction / action WebExtensions API, the isEnabled API method has been tweaked to optimally accept a tabId (as currently expected on Chrome) and to return the global state instead of

rejecting when called without any parameter (aligning it to both Chrome and Safari) – Bug 2012727 / Bug 2013477 Fixed webNavigation transitionType for webpages loaded from the Firefox bookmarks panels by making sure it is set to auto_bookmark instead of the fallback link transitionType – Bug 1623654 Thanks to Thomas Sileo for contributing this fix and improving the cross-browser alignment of the webNavigation WebExtensions API! Smart Window generate memories based on browsing and chat history to use in conversations schedule memories generation 2010143 when activating smart window 2011478 toggle memories per conversation 2011323 and show when they're used 2011774 manage memories from about:preferences 2001453 refer to tabs with @mentions rich multiline inputbox to @ tabs 2003064 rendering mentions in the conversation 2001526 chat conversation history in firefox view 2001496 start new chat from sidebar 2003657 require account when opening 2012806 and switching to smart window 2012014 callout to remind how to switch back to classic 2010897 improve reopening conversations by id 2009866 hookup new urlbar provider for smartbar chat 2003067 renamed prefs from aiwindow to smartwindow 2010128 DevTools Kian fixed a bug in the Inspector where Shadow Root children couldn't be inspected (#1643915) Matt M improved the layout of Netmonitor's "Edit and Resend" UI (#2012236) Chris Vander Linden refactored devtools/client/accessibility/accessibility-proxy.js to use proper private fields (#2013442) Leo McArdle [:leo] made the markup view to omit the attribute value when it is empty. For example we used to show <input checked="">, now it will only be <input checked>) (#2009915) Nicolas Chevobbe [:nchevobbe] made attr() strike-through the attribute name or the fallback, depending if the attribute is set on the element (#1854104) Nicolas Chevobbe [:nchevobbe] fixed a couple issues in the computed panel. We were missing nested media query declarations (#2012412), and we would incorrectly order selectors when an inherited rule declaration had a higher priority than non-inherited "direct" rule declaration (#2013327) Alexandre Poirot [:ochameau] fixed an issue that could prevent to re-open devtools (#2003810) Nicolas Chevobbe [:nchevobbe] made the stacktrace from log points (when checking "show stacktrace") to also show async frames (e.g. setTimeout callback, Promise then, …) (#2006092) Julian Descottes [:jdescottes] addressed a bug that would show an empty Timings tab for some network requests(#2007846) WebDriver Khalid AlHaddad updated asserts for the WebDriver BiDi network wdspec tests to take a single object for the expected events. Khalid AlHaddad updated wdspec tests to use the "iframe" fixture when inlining iframes. Henrik Skupin updated geckodriver to improve browser session cleanup, particularly to always locate minidump files produced by content process crashes. Henrik Skupin updated wptrunner to better distinguish between required and default preferences for Firefox on desktop and mobile. Default preferences can now always be overridden via the –setpref mach argument or through manifest entries. Julian Descottes updated the bidi-har-export package (npm) to collect request and response bodies when recording HAR files. This brings it to feature parity with DevTools HAR recording but with lower performance overhead, making it ideal for performance-focused tools like Browsertime. Lint, Docs and Workflow ahal has fixed the source-docs-generate and source-docs-upload tasks to run when changes occur to the JavaScript files which are used in generating docs. The list of files for generating has moved to docs/config.yml When posting patches for review, if you get a message such as The analysis task source-test-doc-upload failed, but we could not detect any defect , please do check the task output manually by visiting the link. There's a bug on file to improve the output there. New Tab Page We paused all train-hops last week (including a planned one) while we dealt with a surprise start-up crasher caused by the mechanism. We have fixed the crasher, and uplifted the fix to Beta and aim to have it go out tomorrow in the upcoming 147.0.3 dot release. Train-hops will continue after that. Dré made refined cards and favicons the default for all New Tab cards, improving visual clarity and consistency across Discovery Stream. mconley added a capability to set distinct Top Sites/Top Stories defaults during the first 48 hours of a new profile, boosting first-run relevance. Scott fixed an issue where the first sponsored shortcut couldn't be dismissed after installing the New Tab add-on, restoring expected control of sponsored tiles. mconley fixed a crash in

firefox_on_glean::factory::create_and_register_metric during NTP telemetry init, reducing startup crashes on New Tab when performing a train-hop. Reem added context menu actions and switching logic to the Weather widget, enabling hide/show and unit toggles. Reem also added infrastructure for a compact Weather widget variant, reducing space usage for smaller layouts. Irene shipped the new “In the know” component in Activity Stream (behind a pref/rollout), adding a curated updates surface to New Tab. Irene also added topic labels to Daily Briefing cards, improving scannability and content grouping. Irene added “Updated x ago” timestamps to In the know, making content freshness visible at a glance. Irene added a context menu button to In the know cards, enabling quick dismiss and customize actions. Dré fixed an overlap where the MREC ad z-index covered the Customize panel, restoring access to panel controls. Irene fixed a decorative image not being programmatically identified in Settings > Manage topics, improving screen reader output. Picture-in-Picture Special thanks to jsudiaman who fixed a long-standing bug where an open Picture-in-Picture player window would close if the mirrored <video> element was moved around in the DOM! Screenshots :sfoster (readonly) landed Split view related bug: Bug 1997929 – Save full page and Save visible buttons are displayed on the right view when the screenshot option is used on the left view. We needed the buttons panel to show up on the right of the *browser*, which is not the same as the window anymore. Search and Navigation Drew and Daisuke are consolidating urlbar results UI, according to new design specs. Dao and Moritz are continuing polishing the new search bar Fixed recent search terms when changing search engines – Bug 2008429 Allow dragging current loaded url on the search widget – Bug 2011252 Search bar corrupted after customization – Bug 2002275 X button position in RTL – Bug 2010289 Dale continue polishing the new identity box Fixed identity popup not opening on trusted about pages – Bug 2010361 Connection secure shown for self-signed certificates – Bug 2010045 Clicking on privacy settings doesn’t close the trust panel – Bug 2006633 James improved the autofill ranking, to address recent feedback. The patch will be uplifted to Beta, we’re evaluating Release – Bug 2010941 Moritz addressed a performance issue when dragging large text over the address bar – Bug 2002920 Mark finished cleaning up the Search Service after deCOMtaminizing it – Bug 2003302, Bug 2003300, Bug 2003298, Bug 1643008 Storybook/Reusable Components/Acorn Design System Thanks to Henry Yeary for fixing Bug 2010907 – In Storybook, don’t set `wrapped` for reorderable lists and don’t show the `wrapped` control Thanks to :standard8 for fixing Bug 2011040 – Add a medium-icon option for moz-box-item Thanks to :standard8 for fixing a flickering scrollbar issue when dragging items in a reorderable moz-box-group, Bug 2008841 – Dragging an element to the bottom of a reorderable moz-box-group causes a scroll bar to appear and flicker Thanks to :scunnane for fixing an issue with moz-message-bar not reappearing when expected after dismissing, Bug 2013419 – Dismissed moz-message-bar does not reappear if applicable in the same session Thanks to :yjamora for cleaning up tokens related to moz-badge and adding a new story for moz-badge, Bug 1987750 – Add “New” variant to moz-badge and move toolkit/themes/shared/menu.css badge related tokens to tokens files :akulyk has improved our hover styles of interactive moz-box components in high contrast mode (HCM) – Bug 1973552 – Improve hover styles of interactive moz-box components in HCM :tgiles fixed a DOM reordering issue in moz-box-group, Bug 2010132 – Prevent DOM reordering in moz-box-group :mkennedy added support for dividers in moz-select, Bug 1969475 – Support dividers in moz-select :mkennedy fixed a phantom focus element that would appear in moz-box-group, Bug 2005570 – moz-box-group has a phantom focus element Settings Redesign Thanks to :jaws for converting the sidebar to config-based prefs, Work is continuing on converting sections to the new config-based framework, with only a couple remaining (meta bug for these conversions) :tgiles converted the website languages, Bug 1972082 – Convert Website languages to config-based prefs box group :hjones is working on the Backup settings, Bug 1995062 – Convert Backup settings to config-based prefs :akulyk is working on the Firefox Updates section, Bug 1990961 – Convert Firefox Updates section to config-based prefs Finn is working on the spell checking section, Bug 1972083 – Convert Spell checking to config-based prefs :jhirsch is working on the Firefox language

section,

- Frederik Braun: Composing Sanitizer configurations (2026/03/07 23:00)
  The HTML Sanitizer API allows multiple ways to customize the default allow list and this blog post aims to describe a few variations and tricks we came up with while writing the specification. Safe and unsafe Configurations Examples in this post will use configuration dictionaries. These dictionaries might be used …
- Frederik Braun: Perfect types with `setHTML()` (2026/03/06 23:00)
  TLDR: Use require-trusted-types-for 'script'; trusted-types 'none'; in your CSP and nothing besides setHTML() works, essentially removing all DOM-XSS risks. Background: Sanitizer API I was guest at the ShopTalkShow Podcast to talk about setHTML() and the HTML Sanitizer API. Feel free to listen to the whole episode, if you want to …
- The Mozilla Blog: Hardening Firefox with Anthropic’s Red Team (2026/03/06 10:30)
  For more than two decades, Firefox has been one of the most scrutinized and security-hardened codebases on the web. Open source means our code is visible, reviewable, and continuously stress-tested by a global community. A few weeks ago, Anthropic’s Frontier Red Team approached us with results from a new AI-assisted vulnerability-detection method that surfaced more than a dozen verifiable security bugs, with reproducible tests. Our engineers validated the findings and landed fixes ahead of the recently shipped Firefox 148. For users, that means better security and stability in Firefox. Adding new techniques to our security toolkit helps us identify and fix vulnerabilities before they can be exploited in the wild. An emerging technique, pressure-tested by Firefox engineers AI-assisted bug reports have a mixed track record, and skepticism is earned. Too many submissions have meant false positives and an extra burden for open source projects. What we received from the Frontier Red Team at Anthropic was different. Anthropic’s team got in touch with Firefox engineers after using Claude to identify security bugs in our JavaScript engine. Critically, their bug reports included minimal test cases that allowed our security team to quickly verify and reproduce each issue. Within hours, our platform engineers began landing fixes, and we kicked off a tight collaboration with Anthropic to apply the same technique across the rest of the browser codebase. In total, we discovered 14 high-severity bugs and issued 22 CVEs as a result of this work. All of these bugs are now fixed in the latest version of the browser. In addition to the 22 security-sensitive bugs, Anthropic discovered 90 other bugs, most of which are now fixed. A number of the lower-severity findings were assertion failures, which overlapped with issues traditionally found through fuzzing, an automated testing technique that feeds software huge numbers of unexpected inputs to trigger crashes and bugs. However, the model also identified distinct classes of logic errors that fuzzers had not previously uncovered. Anthropic has also published a technical write-up of their research process and findings, which we invite you to read here. The scale of findings reflects the power of combining rigorous engineering with new analysis tools for continuous improvement. We view this as clear evidence that large-scale, AI-assisted analysis is a powerful new addition in security engineers’ toolbox. Firefox has undergone some of the most extensive fuzzing, static analysis, and regular security review over decades. Despite this, the model was able to reveal many previously unknown bugs. This is analogous to the early days of fuzzing; there is likely a substantial backlog of now-discoverable bugs across widely deployed software. Firefox was not selected at random. It was chosen because it is a widely deployed and deeply scrutinized open source project — an ideal proving ground for a new class of defensive tools. Mozilla has historically led in deploying advanced security techniques to protect Firefox users. In that same spirit, our team has already started integrating AI-assisted analysis into our internal security workflows to find and fix vulnerabilities before attackers do. Building in the open for users Firefox has always championed building publicly and working with our community to build a browser that puts users first. This work reflects Mozilla’s long-standing

commitment to applying emerging technologies thoughtfully and in service of user security. The Frontier Red Team at Anthropic showed what collaboration in this space looks like in practice: responsibly disclosing bugs to maintainers, and working together to make them as actionable as possible. As AI accelerates both attacks and defenses, Mozilla will continue investing in the tools, processes, and collaborations that ensure Firefox keeps getting stronger and that users stay protected. The post Hardening Firefox with Anthropic’s Red Team appeared first on The Mozilla Blog.

- Jonathan Almeida: My Firefox for Android local build environment (2026/03/06 00:32)
  The Firefox for Android app has always had a complicated build process - we're cramping a complex cross-platform browser engine and all the related components that make it work on Android into one package. In its current form, it lives in the Firefox mono-repo at mozilla-central (now mozilla-firefox using the git repository). I wanted to document my "artifact-mode" environment here since it's worked quite successfully for me for many years with minor changes. NOTE: After a fresh clone of the mono-repo, don't forget to first run and follow the prompts of ./mach bootstrap . mozconfig My mozconfig below is enabled for artifact mode, but occasionally I switch between various configurations. You can see those commented out, with these few extra notes: I like to separate out my objdirs to avoid cache pollution between the different build types. I think you can get away without needing to specify this and an objdir for your build type and arch will be generated. sccache speeds up the native portion of full builds after the first slow one, but it's a hit or miss if you fetch from the remote repository but don't need to rebuild as often. I don't care to manually run the clobber step, and I don't truly appreciate why that isn't always automatically done. Emilio's mozconfig manager looks like a better solution, however my needs are very simple. # Build GeckoView/Firefox for Android: ac_add_options --enable-application=mobile/android # Targeting the following architecture. # For regular phones, no --target is needed. # For x86 emulators (and x86 devices, which are uncommon): # ac_add_options --target=i686 # For newer phones or Apple silicon ac_add_options --target=aarch64 # For x86_64 emulators (and x86_64 devices, which are even less common): # ac_add_options --target=x86_64 # sccache will significantly speed up your builds by caching # compilation results. The Firefox build system will download # sccache automatically. # This only works for non-artifact builds. #ac_add_options --with-ccache=sccache # Enable artifact builds; manager-mode. ac_add_options --enable-artifact-builds # Write build artifacts to.. ## Full build dir #mk_add_options MOZ_OBJDIR=./objdir-droid #mk_add_options MOZ_OBJDIR=./objdir-desktop ## Artifact builds mk_add_options MOZ_OBJDIR=./objdir-frontend # Automatic clobbering; don't ask me. mk_add_options AUTOCLOBBER=1 JAVA_HOME Sometimes you might find yourself needing to run a (non-mach) command in the terminal. Those typically will need to invoke some parts of gradle for an Android build, so it's best to make sure those are using the same JDK as the bootstrapped one in the mono-repo. This avoids weird build errors where something that compiles in one place isn't working in another (like Android Studio). The location for the JDKs are typically in ~/.mozbuild/jdk/, and if you've between around for ~6 months you end up with multiple versions after every JDK bump: $ ls -l ~/.mozbuild/jdk/ drwxr-xr-x@ - jalmeida 15 Apr 2025 jdk-17.0.15+6 drwxr-xr-x@ - jalmeida 15 Jul 2025 jdk-17.0.16+8 drwxr-xr-x@ - jalmeida 21 Oct 2025 jdk-17.0.17+10 drwxr-xr-x@ - jalmeida 20 Jan 09:00 jdk-17.0.18+8 drwxr-xr-x@ - jalmeida 26 Feb 15:04 mozboot You can find some way to point your latest JDK to one location or you can be lazy like me and pick the latest version to assign as your JAVA_HOME property by adding this to your shell's RC file: export JAVA_HOME="$(ls -1dr -- $HOME/.mozbuild/jdk/jdk-* | head -n 1)/Contents/Home" Android Studio Similarly for Android Studio, let's do the same so that environment is identical. Head to, Settings | Build, Execution, Deployment | Build Tools | Gradle, and ensure that "Gradle JDK" path is set to JAVA_HOME. Lately, the default seems to be for it to follow GRADLE_LOCAL_JAVA_HOME which is a property we can't easily override, so we have to manually set this ourselves. Debugging This section is for miscellaneous build error situations that come-up, but

assuming mach build work and there are no known Android build changes, my solution has typically always been the same. For example, the other day I fetched another engineers patch to test out locally1 as part of reviewing it where I faced the error message below: Execution failed for task ':components:feature-pwa:compileDebugKotlin'. FAILURE: Build failed with an exception. * What went wrong: Execution failed for task ':components:feature-pwa:compileDebugKotlin'. > A failure occurred while executing org.jetbrains.kotlin.compilerRunner.GradleCompilerRunnerWithWorkers$GradleKotlinCompilerWorkAction > Internal compiler error. See log for more details * Try: > Run with --info or --debug option to get more log output. > Run with --scan to generate a Build Scan (powered by Develocity). > Get more help at https://help.gradle.org. * Exception is: org.gradle.api.tasks.TaskExecutionException: Execution failed for task ':components:feature-pwa:compileDebugKotlin'. at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.lambda$executeIfValid$1(ExecuteActionsTaskExecuter.java:135) at org.gradle.internal.Try$Failure.ifSuccessfulOrElse(Try.java:288) at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.executeIfValid(ExecuteActionsTaskExecuter.java:133) at org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.execute(ExecuteActionsTaskExecuter.java:121) at org.gradle.api.internal.tasks.execution.ProblemsTaskPathTrackingTaskExecuter.execute(ProblemsTaskPathTrackingTaskExecuter.java:41) at org.gradle.api.internal.tasks.execution.FinalizePropertiesTaskExecuter.execute(FinalizePropertiesTaskExecuter.java:46) at org.gradle.api.internal.tasks.execution.ResolveTaskExecutionModeExecuter.execute(ResolveTaskExecutionModeExecuter.java:51) at org.gradle.api.internal.tasks.execution.SkipTaskWithNoActionsExecuter.execute(SkipTaskWithNoActionsExecuter.java:57) at org.gradle.api.internal.tasks.execution.SkipOnlyIfTaskExecuter.execute(SkipOnlyIfTaskExecuter.java:74) at org.gradle.api.internal.tasks.execution.CatchExceptionTaskExecuter.execute(CatchExceptionTaskExecuter.java:36) at org.gradle.api.internal.tasks.execution.EventFiringTaskExecuter$1.executeTask(EventFiringTaskExecuter.java:77) at org.gradle.api.internal.tasks.execution.EventFiringTaskExecuter$1.call(EventFiringTaskExecuter.java:55) at org.gradle.api.internal.tasks.execution.EventFiringTaskExecuter$1.call(EventFiringTaskExecuter.java:52) at org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:209) at org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:204) at org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:66) at org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:59) at org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:166) at org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:59) at org.gradle.internal.operations.DefaultBuildOperationRunner.call(DefaultBuildOperationRunner.java:53) at org.gradle.api.internal.tasks.execution.EventFiringTaskExecuter.execute(EventFiringTaskExecuter.java:52) at org.gradle.execution.plan.DefaultNodeExecutor.executeLocalTaskNode(DefaultNodeExecutor.java:55) at org.gradle.execution.plan.DefaultNodeExecutor.execute(DefaultNodeExecutor.java:34) at org.gradle.execution.taskgraph.DefaultTaskExecutionGraph$InvokeNodeExecutorsAction.execute(DefaultTaskExecutionGraph.java:355) at org.gradle.execution.taskgraph.DefaultTaskExecutionGraph$InvokeNodeExecutorsAction.execute(DefaultTaskExecutionGraph.java:343) at org.gradle.execution.taskgraph.DefaultTaskExecutionGraph$BuildOperationAwareExecutionAction.lambda$execute$0(DefaultTaskExecutionGrap

h.java:339) at org.gradle.internal.operations.CurrentBuildOperationRef.with(CurrentBuildOperationRef.java:84) at org.gradle.execution.taskgraph.DefaultTaskExecutionGraph$BuildOperationAwareExecutionAction.execute(DefaultTaskExecutionGraph.java:339) at org.gradle.execution.taskgraph.DefaultTaskExecutionGraph$BuildOperationAwareExecutionAction.execute(DefaultTaskExecutionGraph.java:328) at org.gradle.execution.plan.DefaultPlanExecutor$ExecutorWorker.execute(DefaultPlanExecutor.java:459) at org.gradle.execution.plan.DefaultPlanExecutor$ExecutorWorker.run(DefaultPlanExecutor.java:376) at org.gradle.internal.concurrent.ExecutorPolicy$CatchAndRecordFailures.onExecute(ExecutorPolicy.java:64) at org.gradle.internal.concurrent.AbstractManagedExecutor$1.run(AbstractManagedExecutor.java:47) Caused by: org.gradle.workers.internal.DefaultWorkerExecutor$WorkExecutionException: A failure occurred while executing org.jetbrains.kotlin.compilerRunner.GradleCompilerRunnerWithWorkers$GradleKotlinCompilerWorkAction at org.gradle.workers.internal.DefaultWorkerExecutor$WorkItemExecution.waitForCompletion(DefaultWorkerExecutor.java:289) at org.gradle.internal.work.DefaultAsyncWorkTracker.lambda$waitForItemsAndGatherFailures$2(DefaultAsyncWorkTracker.java:130) at org.gradle.internal.Factories$1.create(Factories.java:33) at org.gradle.internal.work.DefaultWorkerLeaseService.lambda$withoutLocks$2(DefaultWorkerLeaseService.java:344) at org.gradle.internal.work.ResourceLockStatistics$1.measure(ResourceLockStatistics.java:42) at org.gradle.internal.work.DefaultWorkerLeaseService.withoutLocks(DefaultWorkerLeaseService.java:342) at org.gradle.internal.work.DefaultWorkerLeaseService.withoutLocks(DefaultWorkerLeaseService.java:326) at org.gradle.internal.work.DefaultWorkerLeaseService.withoutLock(DefaultWorkerLeaseService.java:331) at org.gradle.internal.work.DefaultAsyncWorkTracker.waitForItemsAndGatherFailures(DefaultAsyncWorkTracker.java:126) at org.gradle.internal.work.DefaultAsyncWorkTracker.waitForItemsAndGatherFailures(DefaultAsyncWorkTracker.java:92) at org.gradle.internal.work.DefaultAsyncWorkTracker.waitForAll(DefaultAsyncWorkTracker.java:78) at org.gradle.internal.work.DefaultAsyncWorkTracker.waitForCompletion(DefaultAsyncWorkTracker.java:66) at org.gradle.api.internal.tasks.execution.TaskExecution$3.run(TaskExecution.java:260) at org.gradle.internal.operations.DefaultBuildOperationRunner$1.execute(DefaultBuildOperationRunner.java:29) at org.gradle.internal.operations.DefaultBuildOperationRunner$1.execute(DefaultBuildOperationRunner.java:26) at org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:66) at org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:59) at org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:166) at org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:59) at org.gradle.internal.operations.DefaultBuildOperationRunner.run(DefaultBuildOperationRunner.java:47) at org.gradle.api.internal.tasks.execution.TaskExecution.executeAction(TaskExecution.java:237) at org.gradle.api.internal.tasks.execution.TaskExecution.executeActions(TaskExecution.java:220) at org.gradle.api.internal.tasks.execution.TaskExecution.executeWithPreviousOutputFiles(TaskExecution.java:203) at org.gradle.api.internal.tasks.execution.TaskExecution.execute(TaskExecution.java:170) at

org.gradle.internal.execution.steps.ExecuteStep.executeInternal(ExecuteStep.java:105) at
org.gradle.internal.execution.steps.ExecuteStep.access$000(ExecuteStep.java:44) at
org.gradle.internal.execution.steps.ExecuteStep$1.call(ExecuteStep.java:59) at
org.gradle.internal.execution.steps.ExecuteStep$1.call(ExecuteStep.java:56) at
org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:209) at
org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:204) at
org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:66) at
org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:59) at
org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:166) at
org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:59) at
org.gradle.internal.operations.DefaultBuildOperationRunner.call(DefaultBuildOperationRunner.java:53) at
org.gradle.internal.execution.steps.ExecuteStep.execute(ExecuteStep.java:56) at
org.gradle.internal.execution.steps.ExecuteStep.execute(ExecuteStep.java:44) at
org.gradle.internal.execution.steps.CancelExecutionStep.execute(CancelExecutionStep.java:42) at
org.gradle.internal.execution.steps.TimeoutStep.executeWithoutTimeout(TimeoutStep.java:75) at
org.gradle.internal.execution.steps.TimeoutStep.execute(TimeoutStep.java:55) at
org.gradle.internal.execution.steps.PreCreateOutputParentsStep.execute(PreCreateOutputParentsStep.java:50) at
org.gradle.internal.execution.steps.PreCreateOutputParentsStep.execute(PreCreateOutputParentsStep.java:28) at
org.gradle.internal.execution.steps.RemovePreviousOutputsStep.execute(RemovePreviousOutputsStep.java:68) at
org.gradle.internal.execution.steps.RemovePreviousOutputsStep.execute(RemovePreviousOutputsStep.java:38) at
org.gradle.internal.execution.steps.BroadcastChangingOutputsStep.execute(BroadcastChangingOutputsStep.java:61) at
org.gradle.internal.execution.steps.BroadcastChangingOutputsStep.execute(BroadcastChangingOutputsStep.java:26) at
org.gradle.internal.execution.steps.CaptureOutputsAfterExecutionStep.execute(CaptureOutputsAfterExecutionStep.java:69) at
org.gradle.internal.execution.steps.CaptureOutputsAfterExecutionStep.execute(CaptureOutputsAfterExecutionStep.java:46) at
org.gradle.internal.execution.steps.ResolveInputChangesStep.execute(ResolveInputChangesStep.java:39) at
org.gradle.internal.execution.steps.ResolveInputChangesStep.execute(ResolveInputChangesStep.java:28) at
org.gradle.internal.execution.steps.BuildCacheStep.executeWithoutCache(BuildCacheStep.java:189) at
org.gradle.internal.execution.steps.BuildCacheStep.lambda$execute$1(BuildCacheStep.java:75) at
org.gradle.internal.Either$Right.fold(Either.java:176) at org.gradle.internal.execution.caching.CachingState.fold(CachingState.java:62) at
org.gradle.internal.execution.steps.BuildCacheStep.execute(BuildCacheStep.java:73) at
org.gradle.internal.execution.steps.BuildCacheStep.execute(BuildCacheStep.java:48) at
org.gradle.internal.execution.steps.StoreExecutionStateStep.execute(StoreExecutionStateStep.java:46) at
org.gradle.internal.execution.steps.StoreExecutionStateStep.execute(StoreExecutionStateStep.java:35) at
org.gradle.internal.execution.steps.SkipUpToDateStep.executeBecause(SkipUpToDateStep.java:75) at

org.gradle.internal.execution.steps.SkipUpToDateStep.lambda$execute$2(SkipUpToDateStep.java:53) at
org.gradle.internal.execution.steps.SkipUpToDateStep.execute(SkipUpToDateStep.java:53) at
org.gradle.internal.execution.steps.SkipUpToDateStep.execute(SkipUpToDateStep.java:35) at
org.gradle.internal.execution.steps.legacy.MarkSnapshottingInputsFinishedStep.execute(MarkSnapshottingInputsFinishedStep.java:37) at
org.gradle.internal.execution.steps.legacy.MarkSnapshottingInputsFinishedStep.execute(MarkSnapshottingInputsFinishedStep.java:27) at
org.gradle.internal.execution.steps.ResolveIncrementalCachingStateStep.executeDelegate(ResolveIncrementalCachingStateStep.java:49) at
org.gradle.internal.execution.steps.ResolveIncrementalCachingStateStep.executeDelegate(ResolveIncrementalCachingStateStep.java:27) at
org.gradle.internal.execution.steps.AbstractResolveCachingStateStep.execute(AbstractResolveCachingStateStep.java:71) at
org.gradle.internal.execution.steps.AbstractResolveCachingStateStep.execute(AbstractResolveCachingStateStep.java:39) at
org.gradle.internal.execution.steps.ResolveChangesStep.execute(ResolveChangesStep.java:64) at
org.gradle.internal.execution.steps.ResolveChangesStep.execute(ResolveChangesStep.java:35) at
org.gradle.internal.execution.steps.ValidateStep.execute(ValidateStep.java:62) at
org.gradle.internal.execution.steps.ValidateStep.execute(ValidateStep.java:40) at
org.gradle.internal.execution.steps.AbstractCaptureStateBeforeExecutionStep.execute(AbstractCaptureStateBeforeExecutionStep.java:76) at
org.gradle.internal.execution.steps.AbstractCaptureStateBeforeExecutionStep.execute(AbstractCaptureStateBeforeExecutionStep.java:45) at
org.gradle.internal.execution.steps.AbstractSkipEmptyWorkStep.executeWithNonEmptySources(AbstractSkipEmptyWorkStep.java:136) at
org.gradle.internal.execution.steps.AbstractSkipEmptyWorkStep.execute(AbstractSkipEmptyWorkStep.java:66) at
org.gradle.internal.execution.steps.AbstractSkipEmptyWorkStep.execute(AbstractSkipEmptyWorkStep.java:38) at
org.gradle.internal.execution.steps.legacy.MarkSnapshottingInputsStartedStep.execute(MarkSnapshottingInputsStartedStep.java:38) at
org.gradle.internal.execution.steps.LoadPreviousExecutionStateStep.execute(LoadPreviousExecutionStateStep.java:36) at
org.gradle.internal.execution.steps.LoadPreviousExecutionStateStep.execute(LoadPreviousExecutionStateStep.java:23) at
org.gradle.internal.execution.steps.HandleStaleOutputsStep.execute(HandleStaleOutputsStep.java:75) at
org.gradle.internal.execution.steps.HandleStaleOutputsStep.execute(HandleStaleOutputsStep.java:41) at
org.gradle.internal.execution.steps.AssignMutableWorkspaceStep.lambda$execute$0(AssignMutableWorkspaceStep.java:35) at
org.gradle.api.internal.tasks.execution.TaskExecution$4.withWorkspace(TaskExecution.java:297) at
org.gradle.internal.execution.steps.AssignMutableWorkspaceStep.execute(AssignMutableWorkspaceStep.java:31) at
org.gradle.internal.execution.steps.AssignMutableWorkspaceStep.execute(AssignMutableWorkspaceStep.java:22) at
org.gradle.internal.execution.steps.ChoosePipelineStep.execute(ChoosePipelineStep.java:40) at
org.gradle.internal.execution.steps.ChoosePipelineStep.execute(ChoosePipelineStep.java:23) at
org.gradle.internal.execution.steps.ExecuteWorkBuildOperationFiringStep.lambda$execute$2(ExecuteWorkBuildOperationFiringStep.java:67) at
org.gradle.internal.execution.steps.ExecuteWorkBuildOperationFiringStep.execute(ExecuteWorkBuildOperationFiringStep.java:67) at
org.gradle.internal.execution.steps.ExecuteWorkBuildOperationFiringStep.execute(ExecuteWorkBuildOperationFiringStep.java:39) at
org.gradle.internal.execution.steps.IdentityCacheStep.execute(IdentityCacheStep.java:46) at
org.gradle.internal.execution.steps.IdentityCacheStep.execute(IdentityCacheStep.java:34) at

org.gradle.internal.execution.steps.IdentifyStep.execute(IdentifyStep.java:44) at
org.gradle.internal.execution.steps.IdentifyStep.execute(IdentifyStep.java:31) at
org.gradle.internal.execution.impl.DefaultExecutionEngine$1.execute(DefaultExecutionEngine.java:64) at
org.gradle.api.internal.tasks.execution.ExecuteActionsTaskExecuter.executeIfValid(ExecuteActionsTaskExecuter.java:132) ... 30 more Caused by:
org.jetbrains.kotlin.gradle.tasks.FailedCompilationException: Internal compiler error. See log for more details at
org.jetbrains.kotlin.gradle.tasks.TasksUtilsKt.throwExceptionIfCompilationFailed(tasksUtils.kt:22) at
org.jetbrains.kotlin.compilerRunner.GradleKotlinCompilerWork.run(GradleKotlinCompilerWork.kt:112) at
org.jetbrains.kotlin.compilerRunner.GradleCompilerRunnerWithWorkers$GradleKotlinCompilerWorkAction.execute(GradleCompilerRunnerWithWorkers.kt:75) at org.gradle.workers.internal.DefaultWorkerServer.execute(DefaultWorkerServer.java:68) at
org.gradle.workers.internal.NoIsolationWorkerFactory$1$1.create(NoIsolationWorkerFactory.java:64) at
org.gradle.workers.internal.NoIsolationWorkerFactory$1$1.create(NoIsolationWorkerFactory.java:61) at
org.gradle.internal.classloader.ClassLoaderUtils.executeInClassloader(ClassLoaderUtils.java:100) at
org.gradle.workers.internal.NoIsolationWorkerFactory$1.lambda$execute$0(NoIsolationWorkerFactory.java:61) at
org.gradle.workers.internal.AbstractWorker$1.call(AbstractWorker.java:44) at
org.gradle.workers.internal.AbstractWorker$1.call(AbstractWorker.java:41) at
org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:209) at
org.gradle.internal.operations.DefaultBuildOperationRunner$CallableBuildOperationWorker.execute(DefaultBuildOperationRunner.java:204) at
org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:66) at
org.gradle.internal.operations.DefaultBuildOperationRunner$2.execute(DefaultBuildOperationRunner.java:59) at
org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:166) at
org.gradle.internal.operations.DefaultBuildOperationRunner.execute(DefaultBuildOperationRunner.java:59) at
org.gradle.internal.operations.DefaultBuildOperationRunner.call(DefaultBuildOperationRunner.java:53) at
org.gradle.workers.internal.AbstractWorker.executeWrappedInBuildOperation(AbstractWorker.java:41) at
org.gradle.workers.internal.NoIsolationWorkerFactory$1.execute(NoIsolationWorkerFactory.java:58) at
org.gradle.workers.internal.DefaultWorkerExecutor.lambda$submitWork$0(DefaultWorkerExecutor.java:176) at
org.gradle.internal.work.DefaultConditionalExecutionQueue$ExecutionRunner.runExecution(DefaultConditionalExecutionQueue.java:194) at
org.gradle.internal.work.DefaultConditionalExecutionQueue$ExecutionRunner.access$700(DefaultConditionalExecutionQueue.java:127) at
org.gradle.internal.work.DefaultConditionalExecutionQueue$ExecutionRunner$1.run(DefaultConditionalExecutionQueue.java:169) at
org.gradle.internal.Factories$1.create(Factories.java:33) at
org.gradle.internal.work.DefaultWorkerLeaseService.lambda$withLocksAcquired$0(DefaultWorkerLeaseService.java:269) at
org.gradle.internal.work.ResourceLockStatistics$1.measure(ResourceLockStatistics.java:42) at
org.gradle.internal.work.DefaultWorkerLeaseService.withLocksAcquired(DefaultWorkerLeaseService.java:267) at
org.gradle.internal.work.DefaultWorkerLeaseService.withLocks(DefaultWorkerLeaseService.java:259) at
org.gradle.internal.work.DefaultWorkerLeaseService.runAsWorkerThread(DefaultWorkerLeaseService.java:127) at

org.gradle.internal.work.DefaultWorkerLeaseService.runAsWorkerThread(DefaultWorkerLeaseService.java:132) at org.gradle.internal.work.DefaultConditionalExecutionQueue$ExecutionRunner.runBatch(DefaultConditionalExecutionQueue.java:164) at org.gradle.internal.work.DefaultConditionalExecutionQueue$ExecutionRunner.run(DefaultConditionalExecutionQueue.java:133) ... 2 more The full trace was long and didn't seem related to a code failure in the module itself. So I employed the solution, which is always the same: ./mach build In Android Studio, File > Sync Project with Gradle Files. Yup, that's all. Very simple and boring. 1 With Jujutsu, this is the moz-phab command I use which has made it easier to manage review patches: moz-phab patch <patch-id> --no-branch --apply-to main@origin Comments With an account on the Fediverse or Mastodon, you can respond to this post. Since Mastodon is decentralized, you can use your existing account hosted by another Mastodon server or compatible platform if you don't have an account on this one. Known non-private replies are displayed below. Learn how this was implemented from the original source here. Load comments <noscript> Loading comments relies on JavaScript. Try enabling JavaScript and reloading, or visit the original post on Mastodon. </noscript> <noscript>You need JavaScript to view the comments.</noscript> &>"'

- Firefox Tooling Announcements: MozPhab 2.9.0 Released (2026/03/05 21:47)
  Issues resolved in Moz-Phab 2.9.0: use Lando’s Mercurial patch application method Discuss these changes in #engineering-workflow on Slack or #Conduit Matrix. 1 post - 1 participant Read full topic
- The Mozilla Blog: Ajit Varma on Firefox’s new AI controls: ‘We believe in user choice’ (2026/03/05 17:06)
  This is an edited transcript of an episode of Outside the Fox, Firefox’s flagship podcast, where we explore what’s happening online and why it matters. Stay up to date by subscribing on YouTube, Apple Podcasts, Spotify, or your favorite podcast app. On Outside the Fox, my co-host Kim Horcher and I spend a lot of time talking about big shifts on the web, but also the quieter product decisions that shape everyday internet life. In this episode, we sat down with Ajit Varma, Head of Firefox, to talk about AI controls, our philosophy behind product decisions, and building a browser around user choice. Steve Flavin:Ajit, welcome to the podcast. We’ve had you on before — but for those who might be new to the show, could you give us a little intro? Ajit Varma:Yeah, thanks for having me. It’s always great to talk to you. As Steve mentioned, I’m the head of Firefox. I’ve been at Firefox for about a year now. And it’s been a great year as we’ve really launched new features, and I feel like we’ve really gotten back to basics of creating the best browser. Kim Horcher:So let’s get to it. Can you give us a quick overview of the new AI Controls in Firefox 148? Ajit:AI Controls is a simple one-stop destination where users who have preferences in how they use AI can easily make their choices. This could be to completely turn off AI, turn off all notifications about AI features, any future features, or have more fine-grained control if there’s specific AI features that you might want. And then it’s really trying to make it easy for users to create the browser experience that they prefer. Kim:Awesome, thank you for breaking that down. What I want to know is: why is Firefox launching this now? What problem is Firefox trying to solve? Ajit:Over the last year, we’ve started to introduce a few AI features into Firefox. And when we build features, we really listen to our community on how we can build a better product. And as we launched a few of these features, it was clear that for some users they did not want to use AI now or in the future. And the reasons vary a lot between people, but some are societal concerns, some just don’t feel like the feature is the right feature for them. And so as we heard this feedback, it became clear to us that we could do a better job of helping these users turn things off if that’s what they wanted. And so that was kind of the reason behind us prioritizing this work. Steve:We love that. I know that we’ve been working on these features for some time. And I’ve got to say, the implementation is super simple and straightforward. I also really appreciate how customizable it is. Can you elaborate a little bit on the specific features you can control in this new hub, and how you developed

the UI? Ajit:This was a question that we spent a lot of time on, because there isn't one definition for AI. It means different things to different people. There are features that have existed in Firefox for a long time that people didn't really consider AI—the world evolves. And so we spent time talking to our users. We spent time looking at the technology behind the features that we launched, and we came up with a set of features that we think best aligned with people's concerns around AI. So these are features like translations, which allow you to go to a website and translate the content into a native language of your choosing. We had a feature in our PDF editor that allowed creation of alternative text to help people with accessibility needs understand what the image was about. When you hovered over a link, we would provide summarizations of the content on that page as an optional feature. And then there are features that we launched in the last year like tab groups that we wanted to make more intelligent by helping users automatically organize tab groups with fewer clicks and suggesting titles by looking at the content of tabs. With all these features, there is now the ability to turn all of these off, but it'll also apply to future features as well. So in November, we talked about Smart Window, which is a new mode that allows even more AI innovation. But if you decide that you want to use this AI Control feature, then future features we build will also be turned off by this toggle, and we wouldn't notify a user about any of those upcoming or existing features as well. This was very informed by feedback that we heard. We encourage feedback from our user base and anyone who uses Firefox. If you feel like there are other features that you expected or other ways we can make this easier, please send us that feedback and we'll continue to improve the feature based on it. Kim:Out of curiosity, what happens if a user chooses to block all features inside AI Controls? And is it reversible if I happen to change my mind? Ajit:Yeah. So, if you go to the page, there's a toggle at the top that you can flip on or off. If you choose to flip it off, then all the features that would be in this AI bucket would then be toggled off. So if you tried something in the past and you decide you don't want it, this is a single spot that would remove all those features. But if you decide to use this feature and there's one feature that you want to turn back on — say translations because you're traveling — you can come back and just turn on a specific feature. Or if you decide that AI is something that you want, you can flip everything back on and you'll get notifications of upcoming features. We've tried to make it a choose-your-own-adventure. It's not paywalling things or putting users through hoops. It is very user-friendly and gives people the ability to choose and control how they use AI, if at all. Steve:Something I'd love to touch on is this moment in the browser space. People are talking about browsers again. That's cool. They've always been relevant—but in recent years, they've come to be viewed almost as a kind of utility. Whereas now, there's a lot of experimentation happening, particularly with AI browsers and AI functionality. Across the industry, it's generating a lot of conversations, some might even say a lot of noise. What would you say is the key differentiator of Firefox's approach to AI? Ajit:First off, I'm so happy about the competition and the conversation. That goes back to Firefox's roots. We were one of the first browsers that provided competition to an entrenched competitor, and this is what ultimately moved the internet forward. But it's important for us to talk about what makes Firefox differentiated. For us, that is about choice, control, and privacy. These values go to the core of the mission for Firefox, which is to help create a healthy and open internet. When you look at other browsers, whether newly emerging ones or existing browsers changing into AI-first browsers, it's becoming apparent that some are not being created because there's a desire to create the best browser. Many companies are looking at how to take users' data, how to get more adoption for AI, how to create more entry points into that company's AI. At Firefox, we have a singular mission: to create the best browser. We don't have billions of dollars spent on building an LLM that we need to force upon users. We believe in user choice. That can mean using on-device models. It can mean choosing the AI you want and not just the AI of the company who built the browser. Steve:That's exciting to hear. As all of us are navigating this new landscape together—as AI reshapes the web as we know it—how do you view Firefox's role? How can Firefox lead by example? Ajit:With any new technology, there are going to be pros and cons. Some

we can anticipate, and some we'll adjust to as we see what users want. We think there are AI features that can improve the browsing experience. Translations can create better connection and empathy. Accessibility features can make the internet more accessible to more people. We're looking very thoughtfully at how we launch AI features to make sure they create better experiences. But we're also focused on features outside of AI. Over the last year, we've launched tab groups, vertical tabs, sidebar. We have customizable hotkeys and split view coming. And over the next few months, we have many privacy features launching. I'd say this is probably the most exciting roadmap I've seen in years as we get back to the basics of creating the best browser. We're excited to hear from users, and build something that serves everyone's needs. You can watch or listen to Outside the Fox on YouTube, Apple Podcasts, Spotify and other major podcast platforms. The post Ajit Varma on Firefox's new AI controls: 'We believe in user choice' appeared first on The Mozilla Blog.

- Tom Ritter: telemetry helps. you still get to turn it off (2026/03/05 15:12)
  Phew, it's been a minute since I last wrote anything, hasn't it. And this blog design is pretty dated... Let me start with this: it is your right to disable telemetry. I fully support that right, and in many cases I disable telemetry myself. If your threat model says "nope", or you simply don't like it, flip the switch. Your relationship with the software and the author of it is a great guide for whether you want to enable telemetry. What I don't buy is the claim I keep seeing that telemetry is useless and doesn't actually help. I can only speak to Firefox telemetry, but I presume the lesson generalizes. Telemetry has paid for itself many times over on the technical side - stability, security, performance, and rollout safety. If you trust the publisher and want to help them improve the thing you use every day, turning on telemetry is the lowest-effort way to do it. If you don't trust them, or just don't want to... cool. But be forewarned - if you're one of a very few people doing a very weird thing, we won't even know we need to support that thing. (More on that later.) What I mean (and don't mean) by "telemetry" Telemetry is a catch-all for measurements and signals a program sends home. In browsers that includes "technical and interaction data" (performance, feature usage, hardware basics), plus things like crash reports that are often controlled by a separate checkbox. To me, telemetry is things you send to the publisher and you don't directly receive anything in return. In contrast, there are lots of other phone-home things I wouldn't call telemetry. Software update pings, for example. The publisher can derive data about this - in fact it's one of the only things Tor Browser 'collects' - but the purpose isn't to tell the publisher something, it's to get you the latest version and that's a direct benefit you gain. Firefox obviously has update pings, but it also has something called Remote Settings which is a tool to sync data to your browser for lots of other useful things. You phone home to get this data. Here's the list of collections, and here's a random one (it's overrides for the password autofill to fix certain websites). Overall it's stuff like graphics driver blocklists, addon blocklists, certificate blocklists, data for CRLite, exemptions to tracking protection to unbreak sites, and so on. And then finally there are things that seem like gratuitous phoning home that I also don't consider telemetry. I don't know the status of all these features and if they still exist, or under what circumstances they happen, but these are things like pinging a known-good website to determine if you're under a captive portal, or roughtime to figure out if all your cert validation is going to break. Now even for Telemetry - I'm not going to talk about product decisions like "is anyone clicking this button?" Those exist, sure, but they're not my world most days. I don't have any personal success stories from that world - I deal with technical telemetry - the kind that finds crashes and hangs, proves that risky security changes won't brick Nightly, and helps us pick the fastest safe implementation. And I'm also not going to argue that you should trust Firefox's telemetry. I think you should make an informed decision - but if you're informed about what we collect (and all the mish-mash of data review approvals); how we collect it including 'regular telemetry' (discards your IP immediately), OHTTP (we never see your IP), Prio (privacy preserving calculations); and how we store it (automatic deletion of old data, segmented and unlinked datasets, etc) - and you still think we aren't doing enough to preserve

your privacy... Well I can't argue with that. We aren't the absolute best in the world; we're far from the worst. And if we don't meet your threshold, turn it off. But my point is: it's not pointless. It's not useless. It helps. It's shipped features you rely on. As a super simple example you can easily poke at yourself - Mozilla's Background Hang Reporter (BHR) exists specifically to collect stacks during hangs on pre-release channels so engineers can find and fix the slow paths. That's telemetry. Concrete wins from Firefox Telemetry (just from me) This is a tiny slice from one developer. There are hundreds more across the project. Killing eval in the parent process (1473549) Eval is bad, right? It can lead to XSS attacks, and when your browser process is (partially) written with JavaScript - that can be a sandbox escape. We tried to eliminate eval in the parent (UI) process, shipped it to Nightly, and immediately broke Nightly. The entire test suite was green and Mozillians had dogfooded the feature for weeks... and it still blew up on real users with real customizations. We had to revert fast and spin a new build. It was a pretty big incident, and not a good day. So we re-did our entire approach here and put in several rounds of extensive telemetry. That told us where eval was still happening in the wild, including Mozilla code paths we didn't have tests for and, crucially, a thriving community of Firefox tinkerers using userChromeJS and friends. Because telemetry surfaced those scripts, I could go talk to that community, explain the upcoming change, and work around the breakages. See the public thread on the firefox-scripts repo for a flavor of that conversation. There's no way we could have safely shipped this without telemetry, and certainly no way we could have preserved your ability to hack Firefox to do what you want. Background Hang Reporter saved me from myself (1721840) BHR data showed specific interactions where my code hung - no apparent reason, never would have guessed. I refactored, and the hang graphs dropped. That feedback loop doesn't exist without telemetry being on in pre-release. Fission (site isolation) and data minimization (1708798) Chrome has focused a lot on removing cross origin data from content processes, as well as the IPC security boundary for cross origin data retrieval. Coming from Tor Browser (where I am also a developer, although not too active) - I was also pretty concerned with personal user data unrelated to origin data. Stuff like your printer or device name. As part of Fission, I worked to eliminate both cross-origin data and personally identifiable things from the content process so a web process running a Spectre attack couldn't get those details. Telemetry helped us confirm we weren't breaking user workflows as we pulled those identifiers out. Ending internet-facing jar: usage Years ago Firefox allowed jar: URIs from web content, and the security model was... not great. Telemetry let us show that real-web usage was basically nonexistent, which made closing that attack surface from the web a no-brainer. Same story brewing for XSLT Chrome has been pushing to deprecate/remove XSLT in the browser due to security/maintenance risk and very low usage; I'm supportive. Usage telemetry is the only way we're able to justify removing a feature from the web. Picking the fastest safe canvas noise (1972586) For anti-fingerprinting canvas noise generation, I used telemetry to measure which implementation was actually fastest across CPUs: it's SHA-256 if you have SHA extensions; SipHash if you don't - or if the input is under ~2.5KB. That choice matters when you multiply it by billions of calls. Font allowlist for anti-fingerprinting (Lists, 1795460) Fonts are a huge fingerprinting vector. We built a font allowlist and font-visibility controls; by design, Firefox's fingerprinting protection avoids using your locally installed one-off fonts on the web. This dramatically shrinks the entropy of "which fonts do you have?" without breaking normal sites. While many browsers do this now, telemetry has helped us continue to improve these defenses and I'm pretty sure we're still the only one that has a font allowlist for Android. Reality check on Resist Fingerprinting users Folks who manually enable our "Resist Fingerprinting" preference (which we don't officially support, and I don't generally recommend - but hey, you do you) are very loud on Bugzilla. VERY loud. To the point where I've had a lot of managers and executives come telling me "Everyone is complaining about this breaking stuff, we really need to disable this so people can't accidentally turn it on." Telemetry let me show that despite being SO LOUD they're still a minute portion of the population. Management's question "Should we block it?" became "No." You're welcome. That's just my lane. People I work

closely with used telemetry to: Ship CRLite (privacy-preserving certificate revocation that's finally practical). Telemetry was instrumental in making this happen. Roll out TLS features like Certificate Transparency support and HTTPS-First behavior, watching real-world fallout and compatibility. Tighten OS sandboxes. I've been working at Mozilla close to 10 years, and I vividly remember the days we lagged behind Chrome in how tight we had our sandbox. (We're on par now, if you didn't realize.) The only way we could do this was by continually running experiments and monitoring telemetry and crash reports as we identified more and more things we broke and needed to fix before we could ship it. Gabriele Svelto works in the stability and crash reporting team and has written extensively about the unexpected things he finds and diagnoses using crash reports. I could give more examples, but I think you get the idea. "I use Foo browser because it disables telemetry." Every major browser either implements telemetry or outsources the job to the upstream engine, and benefits from their having it. Period. Even Brave does telemetry, and they're quite public about their design (P3A): collected into buckets/histograms with privacy techniques like shuffling/thresholding. That's a perfectly respectable approach. We can debate the efficacy or privacy properties of different telemetry designs. We can both stand aghast at overcollection of things that shouldn't be collected. We can debate whether it should be opt-out or opt-in. But only if we both start from the position that telemetry isn't philosophically bad, it can just be implemented badly. Every Foo browser that brags about disabling telemetry is relying on their upstream source - whether it's Firefox or Chrome - to improve the Foo browser using someone else's telemetry - all while trying to take this moral high ground. If you want to use Foo because it adds features you like, or you trust its publisher to choose defaults more than upstream - those are completely valid reasons to use it. But if the reason is "Telemetry is just a way for Firefox to spy on me", hopefully I've dented that perception.

- The Rust Programming Language Blog: Announcing Rust 1.94.0 (2026/03/05 00:00)
  The Rust team is happy to announce a new version of Rust, 1.94.0. 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 1.94.0 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 1.94.0. 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 1.94.0 stable Array windows Rust 1.94 adds array_windows, an iterating method for slices. It works just like windows but with a constant length, so the iterator items are &[T; N] rather than dynamically-sized &[T]. In many cases, the window length may even be inferred by how the iterator is used! For example, part of one 2016 Advent of Code puzzle is looking for ABBA patterns: "two different characters followed by the reverse of that pair, such as xyyx or abba." If we assume only ASCII characters, that could be written by sweeping windows of the byte slice like this: fn has_abba(s: &str) -> bool { s.as_bytes() .array_windows() .any(|[a1, b1, b2, a2]| (a1 != b1) && (a1 == a2) && (b1 == b2)) } The destructuring argument pattern in that closure lets the compiler infer that we want windows of 4 here. If we had used the older .windows(4) iterator, then that argument would be a slice which we would have to index manually, hoping that runtime bounds-checking will be optimized away. Cargo config inclusion Cargo now supports the include key in configuration files (.cargo/config.toml), enabling better organization, sharing, and management of Cargo configurations across projects and environments. These include paths may also be marked optional if they might not be present in some circumstances, e.g. depending on local developer choices. # array of paths include = ["frodo.toml", "samwise.toml",] # inline tables for more control include = [{ path = "required.toml" }, { path = "optional.toml", optional = true },] See the full include documentation for more details. TOML 1.1 support in Cargo Cargo now parses TOML v1.1 for manifests and configuration files. See the TOML release notes for detailed changes, including: Inline tables

across multiple lines and with trailing commas \xHH and \e string escape characters Optional seconds in times (sets to 0) For example, a dependency like this: serde = { version = "1.0", features = ["derive"] } ... can now be written like this: serde = { version = "1.0", features = ["derive"], } Note that using these features in Cargo.toml will raise your development MSRV (minimum supported Rust version) to require this new Cargo parser, and third-party tools that read the manifest may also need to update their parsers. However, Cargo automatically rewrites manifests on publish to remain compatible with older parsers, so it is still possible to support an earlier MSRV for your crate's users. Stabilized APIs <[T]>::array_windows <[T]>::element_offset LazyCell::get LazyCell::get_mut LazyCell::force_mut LazyLock::get LazyLock::get_mut LazyLock::force_mut impl TryFrom<char> for usize std::iter::Peekable::next_if_map std::iter::Peekable::next_if_map_mut x86 avx512fp16 intrinsics (excluding those that depend directly on the unstable f16 type) AArch64 NEON fp16 intrinsics (excluding those that depend directly on the unstable f16 type) f32::consts::EULER_GAMMA f64::consts::EULER_GAMMA f32::consts::GOLDEN_RATIO f64::consts::GOLDEN_RATIO These previously stable APIs are now stable in const contexts: f32::mul_add f64::mul_add Other changes Check out everything that changed in Rust, Cargo, and Clippy. Contributors to 1.94.0 Many people came together to create Rust 1.94.0. We couldn't have done it without all of you. Thanks!

- Jonathan Almeida: Update jj bookmarks to the latest revision (2026/03/04 23:52)
  Update: As of v0.39.0, tug is now built-in to jj as bookmark advance! :tada: Got this one from another colleague as well but it seems like most folks use some version of this daily that it might be good to have this built-in. Before I can jj git push my current bookmark to my remote, I need to update where my (tracked) bookmark is, to the latest change: @ ptuqwsty git@jonalmeida.com 2026-01-05 16:00:22 451384bf <-- move 'main' here. │ TIL: Update remote bookmark to the latest revision ◆ xoqwkuvu git@jonalmeida.com 2025-12-30 19:50:51 main git_head() 9ad7ce11 │ TIL: Preserve image scale with ImageMagick ~ A quick one-liner jj tug does that for me: @ ptuqwsty git@jonalmeida.com 2026-01-05 16:03:54 main* 6e7173b4 │ TIL: Update remote bookmark to the latest revision ◆ xoqwkuvu git@jonalmeida.com 2025-12-30 19:50:51 main@origin git_head() 9ad7ce11 │ TIL: Preserve image scale with ImageMagick ~ The alias is quite straight-forward: [aliases] # Update your bookmarks to your latest rev. tug = ["bookmark", "move", "--from", "heads(::@ & bookmarks())", "--to", "@"]
- This Week In Rust: This Week in Rust 641 (2026/03/04 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.bsky.social on Bluesky 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 2025 State of Rust Survey Results Newsletters The Embedded Rustacean Issue #66 Project/Tooling Updates Compendium: Adding eBPF for Kernel-Level Visibility Danube Messaging migration from ETCD Feedr v0.4.0 - Terminal-based RSS feed reader dag_exec: DAG executor for CPU-heavy pipelines Supercharge Rust functions with implicit arguments using CGP v0.7.0 vscreen: AI agents browser Ply 1.0: Building apps in Rust shouldn't be this hard Observations/Thoughts Using Rust and Postgres for everything: patterns learned over the years Kovan: From Production MVCC Systems to Wait-Free Memory Reclamation Never snooze a future Rust zero-cost abstractions vs. SIMD Nobody ever got fired for using a struct Debugging Reproducibility Issues in Rust Software Designing Backpressure in a Parallel DAG Executor Testing Concurrency Invariants in a Parallel Executor [audio] Netstack.FM episode 29 — Hyper With Sean McArthur (Ep 2 Remastered) Rust Walkthroughs Tutorial: let's make a resumable Pi Spigot with SQLite Apache Iggy's migration journey to thread-per-core architecture powered by io_uring Formal methods for the unsafe side of the Force Quantifying the Swiss marriage tax Fast Python with

Rust: a data-oriented approach [video] Rust: compiling to WASM to make a browser-based game using canvas [video] Daniel Almeida Interview, Writing a Linux GPU Kernel Driver in Rust Miscellaneous TokioConf Update: What to Expect Crate of the Week This week's crate is office2pdf, a standalone library or binary to generate PDF from OOXML (docx, xlsx, etc.) files. Thanks to One 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, Cargo, Rustup or Rust language RFCs. 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 Bluesky 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. Rust India Conference 2026 | CFP open until 2026-03-14 | Bangalore, IN | 2026-04-18 Oxidize Conference | CFP open until 2026-03-23 | Berlin, Germany | 2026-09-14 - 2026-09-16 EuroRust | CFP open until 2026-04-27 | Barcelona, Spain | 2026-10-14 - 2026-10-17 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 Bluesky or Mastodon! Updates from the Rust Project 414 pull requests were merged in the last week Compiler improve the forcing/promotion functions in DepKindVTable codegen: Restore noundef On PassMode::Cast Args In Rust ABI Library BTreeMap::merge optimized make atomic primitives type aliases of Atomic<T> neon fast path for str::contains prepare NonNull for pattern types re-add #[inline] to Eq::assert_fields_are_eq stabilize new RangeToInclusive type Cargo fix: Inject an edition into scripts help: display manpage for nested commands host-config: fix panic when cross-compiling with host-config toml: show required rust-version in unstable edition error improve parent workspace search error msg Clippy fix cmp_owned suggests wrongly on PathBuf fix explicit_counter_loop false positive when the initializer is not integral fix suboptimal_flops false negative on add and sub assign handle core panics in all format lints Rust-Analyzer detect E0804 when casting raw ptr-to-dyn adds auto traits don't panic on invalid LSP notifications fix scrutinee expr indent for replace_if_let_with_match no complete enum variant qualifier in pat use ExprIsRead::Yes for rhs of binary operators implement Span::SpanParent for proc-macro-srv Rust Compiler Performance Triage A positive week with a few nice improvements coming from query system cleanups. Triage done by @panstromek. Revision range: eeb94be7..ddd36bd5 Summary: (instructions:u) mean range count Regressions □ (primary) 0.3% [0.3%, 0.3%] 1 Regressions □ (secondary) 0.2% [0.0%, 0.3%] 3 Improvements □ (primary) -0.8% [-2.1%, -0.1%] 141 Improvements □ (secondary) -1.1% [-6.6%, -0.1%] 90 All □□ (primary) -0.8% [-2.1%, 0.3%] 142 2 Regressions, 5 Improvements, 5 Mixed; 4 of them in rollups 30 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 Always check ConstArgHasType even when otherwise ignoring Always make tuple elements a coercion site deny-by-default & report in deps uninhabited_static Never break between empty parens Compiler Team (MCPs only) Remove soft_unstable Parse unstable keywords for experimental syntax Language Reference Mitigation enforcement No Items entered Final Comment Period this week for Rust RFCs, Cargo, Language Team, Leadership

Council 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 2026-03-04 - 2026-04-01 🦀 Virtual 2026-03-04 | Virtual (Cardiff, UK) | Rust and C++ Cardiff Getting Started with Rust Part 4: Module Handling in a Project 2026-03-04 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2026-03-05 | Virtual (Charlottesville, VA, US) | Charlottesville Rust Meetup Presentation: Tock OS Part #3 - Capsules and lower-level hardware drivers 2026-03-05 | Virtual (Nürnberg, DE) | Rust Nuremberg Rust Nürnberg online 2026-03-07 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup 2026-03-10 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2026-03-10 | Virtual (London, UK) | Women in Rust 🦀 Community Catch Up 2026-03-11 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-12 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-03-17 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-18 | Virtual (Cardiff, UK) | Rust and C++ Cardiff Hybrid event with Rust Dortmund! 2026-03-18 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-20 | Virtual | Packt Publishing Limited Rust Adoption, Safety, and Cloud with Francesco Ciulla 2026-03-24 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2026-03-24 | Virtual (London, UK) | Women in Rust Lunch & Learn: Crates, Tips & Tricks Lightning Talks - Bring your ideas! 2026-03-25 | Virtual (Girona, ES) | Rust Girona Rust Girona Hack & Learn 03 2026 2026-03-26 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-04-01 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-04-01 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing Asia 2026-03-22 | Tel Aviv-yafo, IL | Rust 🦀 TLV In person Rust March 2026 at AWS in Tel Aviv Europe 2026-03-04 | Barcelona, ES | BcnRust Rust at MWC Talent Arena — Workshops + Community Meetup 2026-03-04 | Hamburg, DE | Rust Meetup Hamburg Rust Hack & Learn March 2026 2026-03-04 | Köln, DE | Rust Cologne Rust in March: Abstractions, but at what cost? 2026-03-04 | Oxford, UK | Oxford ACCU/Rust Meetup. Records, Shredded on Ice: A Primer on Parquet and Iceberg 2026-03-04 | Paris, FR | Rust Paris Rust meetup #83 2026-03-05 | Oslo, NO | Rust Oslo Rust Hack'n'Learn at Kampen Bistro 2026-03-11 | Amsterdam, NL | Rust Developers Amsterdam Group Meetup @ Instruqt 2026-03-11 | Frankfurt, DE | Rust Rhein-Main Writing a Python compiler in Rust 2026-03-12 | Bern, CH | Rust Bern 2026 Rust Talks Bern #1 @bespinian 2026-03-12 | Geneva, CH | Post Tenebras Lab Rust Meetup Geneva 2026-03-18 | Dortmund, DE | Rust Dortmund Rust Dortmund Meetup - Intro to Embedded Rust - March 2026-03-19 - 2026-03-20 | Warsaw, PL | Rustikon Rustikon Conference 2026-03-24 | Aarhus, DK | Rust Aarhus Hack Night - Advent of Code 2026-03-24 | Manchester, UK | Rust Manchester Rust Manchester March Code Night 2026-03-27 | Paris, FR | Rust in Paris Rust in Paris 2026-04-01 | Oxford, UK | Oxford ACCU/Rust Meetup. Rust/ACCU meetup. North America 2026-03-04 | New York, NY, US | Rust NYC Rust NYC: Custom Metrics Collector & Embedded Rust! 2026-03-05 | Chicago, IL, US | Chicago Rust Meetup Rust Happy Hour 2026-03-05 | Mountain View, CA, US | Hacker Dojo RUST MEETUP at HACKER DOJO 2026-03-05 | Saint Louis, MO, US | STL Rust Rust Project Night 2026-03-07 | Boston, MA, US | Boston Rust Meetup MIT Rust Lunch, Mar 7 2026-03-12 | Lehi, UT, US | Utah Rust An Interpreter for Computability theory, Written the Hard Way 2026-03-14 | Boston, MA, US | Boston Rust Meetup North End Rust Lunch, Mar 14 2026-03-17 | San Francisco, CA, US | San Francisco Rust Study Group Rust Hacking in Person 2026-03-18 | Hybrid (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-19 | Nashville, TN, US | Music City Rust Developers Applied Rust - Building Rust Applictions 2026-03-21 | Boston, MA, US | Boston Rust Meetup Porter Square Rust Lunch, Mar 21 2026-03-25 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2026-03-26 | Atlanta, GA, US | Rust Atlanta Rust-Atl Oceania 2026-03-12 | Brisbane City, AU | Rust Brisbane Rust Brisbane Mar 2026 2026-03-26 | Melbourne, AU | Rust

Melbourne TBD March Meetup South America 2026-03-21 | São Paulo, BR | Rust São Paulo Meetup Encontro do Rust-SP (migrado pro Lumma) 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 After all, Rust only became as good as it is by going through a rather drastic transformation. At one point it had a GC and Green Threads, famously. There's no substitute for making it exist and seeing how it does on a real problem. – scottmcm on rust-users Thanks to Jonas Fassbender for the suggestion! Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell llogiq ericseppanen extrawurst U007D mariannegoldin bdillo opeolluwa bnchi KannanPalani57 tzilist Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- Firefox Tooling Announcements: Firefox Profiler Deployment (March 3, 2026) (2026/03/03 18:55)
  The latest version of the Firefox Profiler is now live! Check out the full changelog below to see what’s changed. Highlights: [fatadel] Persist selected marker in URL and show sticky tooltip on load (#5847) [Markus Stange] Implement the “collapse resource” transform with the help of the “collapse direct recursion” transform. (#5824) [Markus Stange] Share stackTable, frameTable, funcTable, resourceTable and nativeSymbols between threads (#5482) [fatadel] Add support for ternaries in marker labels (#5857) Other Changes: [fatadel] Fix crash when nativeSymbol index is out of bounds in assembly view (#5850) [Nazım Can Altınova] Fix the color of dark mode back arrow svg (#5863) [fatadel] Force canvas redraw when system theme changes (#5861) [Nazım Can Altınova] Fix unhandled promise rejection in setupInitialUrlState (#5864) [Markus Stange] Remove async attribute from module script tag. (#5870) [Nazım Can Altınova] Update the docsify package that’s used in the user documentation (#5872) [Nazım Can Altınova] Escape CSS URLs that are coming from profiles (#5874) [fatadel] Update home page message for the other browser case (#5866) [Markus Stange] Reduce allocations for getStackLineInfo + getStackAddressInfo (#5761) Big thanks to our amazing localizers for making this release possible: de: Ger fy-NL: Fjoerfoks it: Francesco Lodolo [:flod] nl: Fjoerfoks ru: berry ru: Valery Ledovskoy zh-TW: Pin-guang Chen Find out more about the Firefox Profiler on profiler.firefox.com! If you have any questions, join the discussion on our Matrix channel! 1 post - 1 participant Read full topic
- Thunderbird Blog: Illustrating Roc’s World: A Spotlight on Michaela Martin (2026/03/02 20:38)
  Thunderbird’s mascot, Roc, is a bit of an unsung hero. If you’ve ever donated to the project, he’s the happy blue bird who thanks you for supporting our work. For the 2024 End of Year appeal, the Thunderbird team commissioned design artist Michaela Martin to broaden Roc’s world. Her whimsical illustration, which is also available as a wallpaper download, shows Roc soaring through a sunlit forest as he delivers the mail to its denizens. Likewise, we’d like to shine the spotlight on Michaela Martin, who has brought our mascot to life in what is soon becoming the Roc Illustrated Universe! She not only answered our questions about her artistic background and creative process, but also provided us some visual peeks into how she turned Roc’s flight from a first draft to a finished product. What motivated you to become an illustrator? What was your training/education as an artist? What have been some of your favorite illustrations? I have always had a passion for creating art; I find the process of bringing stories to life particularly fascinating and inspiring. Art can allow others to take a glimpse at what otherwise might exist only in someone’s mind. As far as training and education, I attended the College for Creative Studies in Detroit where I studied Animation, with a focus on Design (character/prop design, visual development, and color keys, for example). I learned a lot during my years in school, but I do not want to understate the value of learning from my fellow peers and colleagues, as well as resources available online (classes, software tutorials, etc). Most of my favorite illustration works happen to be from the field of animation. The background illustrations, concept art, color keys, and character design are all elements that I find quite inspiring. The Prince of Egypt, The Iron Giant, The Road to El Dorado, and Spirited Away are a

handful of films that I would say are particularly inspirational to me. Having worked on several animated projects myself now, I have a much greater sense of appreciation for these works of art than I did when I was much younger. Tell us about your work process! How do you go from idea to first drafts? About how long, on average, does an illustration take? I don't have a particularly exciting answer for this – I visualize and imagine various designs in my head, and I translate them onto canvas with my hand as best I can. I tend to start off with rough sketches that I can get out quickly, which allows me to explore a lot of concepts in a short amount of time. The length of time it takes me to finish an illustration tends to vary, depending on various factors such as image size/detail, as well as client needs (revisions, adjustments, etc). Usually it takes around 1-3 months to fully complete an illustration, give or take a couple weeks. What's in your toolkit? Do you work mostly digitally, or in more traditional mediums? I primarily use Photoshop. I do enjoy sketching traditionally in sketchbooks (or printer paper, napkins, notepads, anything that is available). I tend to do more rough works on paper, and transfer them to Photoshop once I am ready to work on them with more detail. I enjoy the feel of working traditionally, but I also enjoy how forgiving digital mediums can be. Tell us about how you created the first illustration, aka Roc flying through the sunlit forest? Did you have a direction from the Thunderbird team on what to illustrate, or more free reign? Can you tell us how you imagined and then created Roc's world? I worked closely with Laurel (the team's Design Manager) on both illustrations. She provided me with some details and the main idea; Roc flying over the forest, delivering the mail to the various inhabitants (hinted at through the appearance of houses amongst the foliage). I had a decent amount of free reign for exploration, though I wanted to add some elements that I thought would work with our previously established Roc design, since we had just finished establishing that. I wanted to keep the design elements soft and friendly, similar to Roc's finalized design. There are not many sharp angles in the design of the houses or foliage, for example. I have included some very early concept art for the final illustration that shows the exploration phase and a couple of unused ideas. In your illustration for this year's appeal, you gave us an entire group of Rocs, working together! Can you tell us more about this design and how you brought it to life? These were initially just different pose options for Roc that I did very early in the sketch phase. I was exploring character placement, and only expected one to be picked! It was later discussed that all three poses could potentially be used together in the finished illustration – the different poses would have a very simple animation; turning visible and invisible. It would look as though Roc were jumping between different screens at his workstation. What part of Roc's world would you like to explore in your next piece? We'd love to have a sneak peak into the greater Roc's illustrated universe! I think that more characters would be fun to explore. Possibly Roc's family, friends, neighbors, etc. Maybe some of them can have jobs alongside Roc, or Roc delivers their mail. More exploration on Roc's little forest village would be quite fun to do as well (different styles of buildings for different species, different biomes of the forest, etc). Perhaps these explorations could reflect different features of the Thunderbird service! Were you a Thunderbird user before you began illustrating for us? If not, have you tried us out since? I poked around Thunderbird a bit, but I have not actually utilized it to the full capacity as of yet! I am not the best at consistently using new technology, even if it is quite helpful. I do use Firefox quite frequently, though! If you'd like to see more of Michaela's work, be sure to visit her website at MichaelaM Art and/or follow her on Instagram. We'd like to thank her again for answering our questions, sharing her sketches and explorations, and designing Roc's world for us and our community! The post Illustrating Roc's World: A Spotlight on Michaela Martin appeared first on The Thunderbird Blog.

- Firefox Tooling Announcements: MozPhab 2.8.3 Released (2026/03/02 16:18)
  Bugs resolved in Moz-Phab 2.8.3: bug 2016153 moz-phab uplift should strip DONTBUILD from commit message Discuss these changes in #engineering-workflow on Slack or #Conduit Matrix. 1 post - 1 participant Read full topic

- Jan-Erik Rediger: Eight-year Moziversary (2026/03/02 14:00)
  In March 2018 I started a job as a Telemetry engineer at Mozilla. Eight years later I'm still here on my Moziversary, as I was in 2019, 2020, 2021, 2022, 2023, 2024 and 2025. In the past year we had 1 CEO change, 14 reorgs, 12 Firefox releases1 and 31 Glean releases2. My team is still part of the Infrastructure Org, now under a bigger organization called Core Services. Other Data Engineers however have been moved over to the neighboring Data org. I was told nothing else changes and priorities stay the same. My goals individual expectations will be adjusted accordingly. I haven't seen my team in person since Dublin, due to policy changes around work travel in and out of the US. Despite all the company changes, the work I'm directly involved in did remain largely the same. I'm still focused on Glean, as expected. We (and by that I mean mostly chutten) moved legacy telemetry to Glean fully (we are Glean now). In the last half of 2025 I finally found dedicated time for performance work. We now have the first benchmarks on Glean and I was able to close out some performance gaps previously identified. More work remains. Early this year then I was able to revive my prototype for moving Glean to a new storage backend based on SQLite and early benchmarks are very promising. This year I will focus on finalizing that work, deploying a new Glean version with a more reliable (and faster!) client-side storage, accompanied by benchmarks we can run and rely on. Along with that I can do some refactoring and cleanups that will make our codebase easier to work with. Other work includes a better integration with other Rust components that want to use Glean directly. The rest of Mozilla is still working on many big and small things. Leadership is still chasing the AI hype, with questionable outcomes3. I don't know where that will take the company and I disagree with some of the approaches. So far I've been spared from the worst and no one is forcing AI tools on me just yet. Thank you Eight years in the same job is a long time. I get to make decisions. I have to live with them. I have to live with decisions of others and maintain them. None of that however would be possible without a team, which makes it a joy to work alongside. So a big thank you to my team members Chris, Travis, Charlie and Abhishek for being the Data Collection Tools team. And also a thank you to Alessio, our team manager. There's many others across Mozilla I get to work with and chat with. Thank you! Footnotes: Contrary to popular belief there is not a fixed reorg schedule (that I know of), there is a Firefox release schedule though. ↩ Now with a release song playlist ↩ You can disable the things you don't want in Firefox. ↩
- The Rust Programming Language Blog: 2025 State of Rust Survey Results (2026/03/02 00:00)
  Hello, Rust community! Once again, the survey team is happy to share the results of the State of Rust survey, this year celebrating a round number - the 10th edition! The survey ran for 30 days (from November 17th to December, 17th 2025) and collected 7156 responses, a slight decrease in responses compared to last year. In this blog post we will shine a light on some specific key findings. As usual, the full report is available for download. SurveyStartedCompletedCompletion rateViews 20249 4507 31077.4%13 564 20259 3897 15676.2%20 397 Overall, the answers we received this year pretty closely match the results of last year, differences are often under a single percentage point. The number of respondents decreases slightly year over year. In 2025, we published multiple surveys (such as the Compiler Performance or Variadic Generics survey), which might have also contributed to less people answering this (longer) survey. We plan to discuss how (and whether) to combine the State of Rust survey with the ongoing work on the Rust Vision Doc. Also to be noted that these numbers should be taken in context: we cannot extrapolate too much from a mere 7 000 answers and some optional questions have even less replies. Let's point out some interesting pieces of data: Screenshotting Rust use Challenges and wishes about Rust Learning about Rust Industry and community Screenshotting Rust use Confirmed that people develop using the stable compiler and keep up with releases, trusting our stability and compatibility guarantees. On the other hand, people use nightly out of "necessity" (for example, something not yet stabilized). Compared to last year (link) we seem to have way less nightly users. This may not be a significant data point because we are looking at a sliding window of releases and differences could depend

on many factors (for example, at a specific point in time we might have more downloads of the nightly compiler because of a highly anticipated feature). One example might be the very popular let chains and async closures features, which were stabilized last year. <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] <noscript> </noscript> [PNG] [SVG] We are also interested to hear from (and grateful to) people not using Rust (or not anymore) when they tell us why they dropped the language. In most cases it seems to be a "see you again in the future" rather than a "goodbye". <noscript> </noscript> [PNG] [SVG] <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] Some specific topic we were interested in: how often people download crates using a git repository pinned in the Cargo.toml (something like foo = { git = "https://github.com/foo/bar" }). <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] and if people actually find the output of --explain useful. Internal discussions hinted that we were not too sure about that but this graph contradicts our prior assumption. Seems like many Rust users actually do find compiler error code explanations useful. <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] Challenges and wishes about Rust We landed long-awaited features in 2025 (let chains and async closures) and the survey results show that they are indeed very popular and often used. That's something to celebrate! Now generic const expressions and improved trait methods are bubbling up in the charts as the most-wanted features. Most of the other desired features didn't change significantly. <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] When asked about which non-trivial problems people encounter, little changes overall compared to 2024: resource usage (slow compile times and storage usage) is still up there. The debugging story slipped from 2nd to 4th place (~2pp). We just started a survey to learn more about it! <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] Learning about Rust Noticeable (within a ~3pp) flection in attendance for online and offline communities to learn about Rust (like meetups, discussion forums and other learning material). This hints at some people moving their questions to LLM tooling (as the word cloud for open answers suggests). Still, our online documentation is the preferred canonical reference, followed by studying the code itself. <noscript> </noscript> [PNG] [SVG] [Wordcloud of open answers] <noscript> </noscript> [PNG] [SVG] Industry and community Confirmed the hiring trend from organisations looking for more Rust developers. The steady

growth may indicate a structural market presence of Rust in companies, codebases consolidate and the quantity of Rust code overall keeps increasing. <noscript>

</noscript> [PNG] [SVG] As always we try to get a picture of the concerns about the future of Rust. Given the target group we are surveying, unsurprisingly the majority of respondents would like even more Rust! But at the same time concerns persist about the language becoming more and more complex. Slight uptick for "developer and maintainers support". We know and we are working on it. There are ongoing efforts from RustNL (https://rustnl.org/fund) and on the Foundation side. Funding efforts should focus on retaining talents that otherwise would leave after some time of unpaid labor. This graph is also a message to companies using Rust: please consider supporting Rust project contributors and authors of Rust crates that you use in your projects. Either by joining the Rust Foundation, by allowing some paid time of your employees to be spent on Rust projects you benefit from or by funding through other collect funds (like https://opencollective.com, https://www.thanks.dev and similar) or personal sponsorships (GitHub, Liberapay or similar personal donation boxes). Trust in the Rust Foundation is improving, which is definitively good to hear. <noscript>

</noscript> [PNG] [SVG] [Wordcloud of open answers] As a piece of trivia we ask people which tools they use when programming in Rust. The Zed editor did a remarkable jump upward in the preferences of our respondents (with Helix as a good second). Editors with agentic support are also on the rise (as the word cloud shows) and seems they are eroding the userbase of VSCode and IntelliJ, if we were to judge by the histogram. We're happy to meet again those 11 developers still using Atom (hey !) and we salute those attached to their classic editors choice like Emacs and Vim (or derivatives). <noscript>

</noscript> [PNG] [SVG] [Wordcloud of open answers] And finally, here are some data about marginalized groups, out of all participants who completed our survey: Marginalized groupCountPercentage Lesbian, gay, bisexual, queer, or otherwise non-heterosexual75210.59% Neurodivergent7069.94% Trans5487.72% Woman or perceived as a woman4576.43% Non-binary gender2924.11% Disabled (physically, mentally, or otherwise)2183.07% Racial or ethnic minority2173.06% Political beliefs2112.97% Educational background1702.39% Cultural beliefs1391.96% Language1341.89% Religious beliefs1001.41% Other610.86% Older or younger than the average developers I know220.31% While some of these numbers have slightly improved, this still shows that only a very small percentage of the people who are part of marginalized groups make it to our project. While we still do better than many other tech communities, it is a reminder that we need to keep working hard on being a diverse and welcoming FOSS community for everyone, which has always been and always will be one of our core values. Conclusions Overall, no big surprises and a few trends confirmed. If you want to dig more into details, feel free to download the PDF report. We want once again to thank all the volunteers that helped shaping and translating this survey and to all the participants, who took the time to provide us a picture of the Rust community. A look back Since this year we publish a round number, if you fancy a trip down the memory lane here the blog posts with the past years' survey results: 2024 State of Rust Survey results 2023 Rust Annual Survey results 2022 Rust Annual Survey results 2021 Rust Survey results 2020 Rust Survey results 2019 Rust Survey results 2018 Rust Survey results 2017 Rust Survey results 2016 State of Rust survey

- The Servo Blog: January in Servo: preloads, better forms, details styling, and more! (2026/02/28 00:00)
  Servo 0.0.5 is here, bringing with it lots of improvements in web platform features. Some highlights: <link rel=preload> (@TimvdLippe, @jdm,

#40059) <style blocking> and <link blocking> (@TimvdLippe, #42096) <img align> (@mrobinson, #42220) <select disabled> (@simonwuelker, #42036) OGG files can now be played in <audio> (@jdm, #41789) ‘cursor-color’ (@mrobinson, #41976) ‘content: <image>’ works on all elements (@andreubotella, #41480) ‘::details-content’ on <details> (@lukewarlow, #42107) ‘:open’ on <details> (@lukewarlow, #42195) ‘:active’ on <input type=button> (@mrobinson, #42095) Origin API (@WaterWhisperer, #41712) MouseEvent.detail (@mrobinson, #41833) Request.keepalive (@TimvdLippe, @WaterWhisperer, #41457, #41811) Cyclic imports, import attributes, and JSON modules (@Gae24, #41779) navigator.sendBeacon() is enabled by default (@TimvdLippe, #41694) https_proxy, HTTPS_PROXY, and NO_PROXY (@Narfinger, #41689) ML-KEM, ML-DSA, and AES-OCB in Crypto (@kkoyung, #41604, #41617, #41615, #41627, #41628, #41647, #41659, #41676, #41791, #41822, #41813, #41829) Web APIs Servo now plays OGG media inside <audio> elements (@jdm, #41789)! We disabled this feature many years ago due to bugs in GStreamer, our media playback engine, but those bugs have since been fixed. We now support non-px sizes for width and height attributes in <svg> elements (@rodio, #40761). Inactive documents will now correctly reject fullscreen mode changes (@stevennovaryo, #42068). We’ve enabled support for the navigator.sendBeacon() by default (@TimvdLippe, #41694); the dom_navigator_sendbeacon_enabled preference has been removed. As part of this work, we implemented the keepalive feature of the Request API (@TimvdLippe, @WaterWhisperer, #41457, #41811). That’s not all for network-related improvements! Quota errors from the fetchLater() API provide more details (@TimvdLippe, #41665), and fetch response body promises now reject when invalid gzip content is encountered (@arayaryoma, #39438). Meanwhile, EventSource connections will no longer endlessly reconnect for permanent failures (@WaterWhisperer, #41651, #42137), and now use the correct ‘Last-Event-Id’ header when reconnecting (@WaterWhisperer, #42103). Finally, Servo will create PerformanceResourceTiming entries for requests that returned unsuccessful responses (@bellau, #41804). There has been lots of work related to navigating pages and loading iframes. We process URL fragments more consistently when navigating via window.location (@TimvdLippe, #41805, #41834), and allow evaluating javascript: URLs when a document’s domain has been modified (@jdm, #41969). XML documents loaded in an <iframe> no longer inherit their encoding from the parent document (@simonwuelker, #41637). We’re also made it possible to use blob: URLs from inside ‘about:blank’ and ‘about:srcdoc’ documents (@jdm, #41966, #42104). Finally, constructed documents (e.g. new Document()) now inherit the origin and domain of the document that created them (@TimvdLippe, #41780), and we implemented the new Origin API (@WaterWhisperer, #41712). Servo’s mixed content protections are steadily increasing. Insecure requests (e.g. HTTP) originating from <iframe> elements can now be upgraded to secure protocols (@WaterWhisperer, #41661), and redirected requests now check the most recent URL when determining if the protocol is secure (@WaterWhisperer, #41832). <style blocking> and <link blocking> can now be used to block rendering while loading stylesheets that are added dynamically (@TimvdLippe, #42096), and stylesheets loaded when parsing the document will block the document ‘load’ event more consistently (@TimvdLippe, @mrobinson, #41986, #41987, #41988, #41973). We also fire the ‘error’ event if a fetched stylesheet response is invalid (@TimvdLippe, @mrobinson, #42037). Servo now leads other browsers in support for new Web Cryptography algorithms! This includes full support for ML-KEM (@kkoyung, #41604, #41617, #41615, #41627), ML-DSA (@kkoyung, #41628, #41647, #41659, #41676), and AES-OCB (@kkoyung, #41791, #41822, #41813, #41829), plus improvements to AES-GCM (@kkoyung, #41950). Additionally, the error messages returned by many Crypto APIs are now more detailed (@PaulTreitel, @danilopedraza, #41964, #41468, #41902). JS module loading received a lot of attention – we’ve improved support for cyclic imports (@Gae24, #41779), import attributes (@Gae24, #42185), and JSON modules (@Gae24, @jdm, #42138). Additionally, the <link rel=preload> attribute now triggers preload fetch operations that can improve page load speeds (@TimvdLippe, @jdm, #40059). IndexedDB support continues to make progress, though for now

the feature is disabled by default (--pref dom_indexeddb_enabled). This month we gained improvements to connection queues (@gterzian, #41500, #42053) and request granularity (@gterzian, #41933). We were accidentally persisting SessionStorage data beyond the current session, but this has been corrected (@arihant2math, #41326). Text input fields have received a lot of love this month. Clicking in an input field will position the cursor accordingly (@mrobinson, @jdm, @Loirooriol, #41906, #41974, #41931), as will clicking past the end of a multiline input (@mrobinson, @Loirooriol, #41909). Selecting text with the mouse in input fields works (@mrobinson, #42049), and double and triple clicks now toggle selections (@mrobinson, #41926). Finally, we fixed a bug causing the input caret to be hidden in <input> elements inside of Shadow DOM content (@stevennovaryo, #42233). ‘cursor-color’ is respected when rendering the input cursor (@mrobinson, #41976), and newlines can no longer be pasted into single line inputs (@mrobinson, #41934). Finally, we fixed a panic when focusing a text field that is disabled (@mrobinson, #42078), as well as panics in APIs like HTMLInputElement.setRangeText() that confused bytes and UTF-8 character indices (@mrobinson, #41588). We also made time to improve form controls! The default styling of many controls received some care (@mrobinson, #42085), while <input type=button> can now be styled with the ‘:active’ pseudo-class (@mrobinson, #42095). Conversely, disabled <select> elements can no longer be activated (@simonwuelker, #42036). Mouse events triggered by the embedder are more complete; MouseEvent.detail correctly reports the click count for ‘mouseup’ and ‘mousedown’ events (@mrobinson, #41833), and many other members are now consistent with other mouse events (@mrobinson, #42013). Performing a pinch zoom on mobile is now reflected in the VisualViewport API (@stevennovaryo, #41754), though for now the feature is disabled by default (--pref dom_visual_viewport_enabled). We’ve changed the behaviour of Web APIs that use the [Clamp] annotation (such as Blob.slice()). The previous implementation would cast floating point values to their integer equivalents, but the standard requires more specific rounding logic (@Taym95, #41640). The RGBA8 constant is now available in WebGL 1 rendering contexts; it was previously only available in WebGL 2 contexts (@simonwuelker, #42048). Fonts were another area of focus this month. Loading web fonts from file: URLs works as expected (@TimvdLippe, #41714), as does using web fonts within Shadow DOM content (@minghuaw, #42151). Each web font request now creates a PerformanceResourceTiming entry (@lumi-me-not, #41784). Servo supports font variations as of November 2025, so as of this month, the FontFace constructor no longer ignores the ‘font-variation-settings’ property (@muse254, #41968). Cursive scripts now ignore the ‘letter-spacing’ CSS property (@mrobinson, #42165), and we significantly reduced the time and memory required when rendering non-ASCII text (@mrobinson, @Loirooriol, #42105, #42162) and when text nodes share the same font (@mrobinson, #41876). CSS There were lots of improvements to block layout algorithms (@Loirooriol, #41492, #41624, #41632, #41655, #41652, #41683). These often affect pages where a block element (such as a) exists within some other layout mode (such as an inline , or a flexbox context), and fixes like these ensure Servo matches the output of other browsers. Elements with scrollable overflow can be scrolled more consistently, even with CSS transforms applied to them (@stevennovaryo, #41707, #42005). You can now use ‘content: <image>’ on any element (@andreubotella, #41480). Generated image content used to only work with pseudo-elements, but that restriction no longer applies. <details> elements can now be styled with the ‘::details-content’ pseudo-element (@lukewarlow, #42107), as well as the ‘:open’ pseudo-class (@lukewarlow, #42195). CSS styles now inherit correctly through ‘display: contents’ as well as <slot> elements in Shadow DOM content (@longvatrong111, @Loirooriol, @mrobinson, #41855). ‘overflow-clip-margin’ now works correctly when ‘border-radius’ is present (@Loirooriol, #41967). We fixed bugs involving text inside flexbox elements: they now use consistent baselines for alignment (@lukewarlow, @mrobinson, #42038), and style updates are propagated to the text correctly (@mrobinson, #41951). <img align> now aligns the image as expected (@mrobinson, #42220). ‘word-break: keep-all’ now prevents line breaks in CJK text (@RichardTjokroutomo, #42088). We also fixed some bugs involving floats, collapsing margins, and

phantom line boxes (@Loiroriol, #41812), which sound much cooler than they actually are. Finally, we upgraded our Stylo dependency to the latest changes as of January 1 2026 (@Loiroriol, #41916, #41696). Stylo powers our CSS parsing and style resolution engine, and this upgrade improves support for parsing color functions like ‘color-mix()’, and improves our CSS animations and transitions for borders and overflow clipping. Automation and introspection Last month Servo gained support for HTTP proxies. We now support HTTPS proxies as well (@Narfinger, #41689), which can be configured with the https_proxy or HTTPS_PROXY environment variables, or the network_https_proxy_uri preference. In addition, the NO_PROXY environment variable or the network_http_no_proxy preference can disable any proxy for particular domains. Our developer tools integration continues to improve. Worker globals are now categorized correctly in the UI (@atbrakhi, #41929), and the Sources panel is populated for very short documents (@atbrakhi, #41983). Servo will report console messages that were logged before the developer tools are opened (@eerii, @mrobinson, #41895). Finally, we fixed a panic when selecting nodes in the layout inspector that have no style information (@eerii, #41800). We’re working towards supporting pausing in the JS debugger (@eerii, @atbrakhi, @jdm, #42007), and breakpoints can be toggled through the UI (@eerii, @atbrakhi, #41925, #42154). While the debugger is paused, hovering over JS objects will report the object’s properties for builtin JS classes (@eerii, @atbrakhi, #42186). Stay tuned for more JS debugging updates in next month’s blog post! Servo’s WebDriver server is also maturing. Evaluating a synchronous script that returns a Promise will wait until that promise settles (@yezhizhen, #41823). ‘touchmove’ events are fired for pointer actions when a button is pressed (@yezhizhen, #41801), and ‘touchcancel’ events are fired for canceled pointer action items (@yezhizhen, #41937). Finally, any pointer actions that would trigger duplicate ‘mousemove’ events are silently discarded (@mrobinson, #42034). Element Clear commands now test whether the element is interactable (@yezhizhen, #42124). Now a null script execution timeout value will never trigger a timeout (@yezhizhen, #42184), and synthesized ‘pointermove’ events have a consistent pointerId value (@yezhizhen, #41726). Embedding You can now cross-compile Servo using Windows as the host (@yezhizhen, #41748). We’ve pinned all git dependencies to specific revisions, to reduce the risk of build failures (@Narfinger, #42029). We intend to eventually forbid git dependencies in Servo libraries, which will help unblock releasing Servo on crates.io. SiteDataManager now has a new clear_site_data() method to clear all stored data for a particular host (@janvarga, #41618, #41709, #41852). Our nightly testing UI, servoshell, now respects any customized installation path on Windows (@yezhizhen, #41653). We fixed a crash in the Android app when pausing the application (@NiklasMerz, #41827). Additionally, clicking inside a webview in the desktop app will remove focus from any browser UI (@mrobinson, #42080). We’ve laid more groundwork towards exposing accessibility tree information from webviews (@delan, @lukewarlow, @alice, #41924). There’s nothing to test yet, but keep an eye on our tracking issue if you want to be notified when nightly builds are ready for testing! Stability & performance We’ve converted many uses of IPC channels in the engine to channels that are more efficient when multiprocess mode is disabled (@Narfinger, @jdm, @sagudev, @mrobinson, #41178, #41071, #41733, #41806, #41380, #41809, #41774, #42032, #42033, #41412). Since multiprocess mode is not yet enabled by default (--multiprocess), this is a significant boost to Servo’s everyday performance. Servo now sets a socket timeout for HTTP connections (@Narfinger, @mrobinson, #41710). This is controlled by the network_connection_timeout preference, and defaults to 15 seconds. Each instance of Servo now starts four fewer threads (@Narfinger, #41740). Any network operations that trigger a synchronous UI operation (such as an HTTP authentication prompt) no longer blocks other network tasks from completing (@Narfinger, @jdm, #41965, #41857). It’s said that one of the hardest problems in computer science is cache invalidation. We improved the memory usage of dynamic inline SVG content by evicting stale SVG tree data from a cache (@TomRCummings, #41675). Meanwhile, we added a new cache to reduce memory usage and improve rendering performance for pages with animating images (@Narfinger, #41956). Servo’s JS engine now

accounts for 2D and 3D canvas-related memory usage when deciding how often to perform garbage collection (@sagudev, #42180). This can reduce the risk of out-of-memory (OOM) errors on pages that create large numbers of short-lived WebGL or WebGPU objects. To reduce the risk of panics involving the JS engine integration, we're continuing to use the Rust type system to make certain kinds of dynamic borrow failures impossible (@sagudev, #41692, #41782, #41756, #41808, #41879, #41878, #41955, #41971, #42123). We also continue to identify and forbid code patterns that can trigger rare crashes when garbage collection happens while destroying webviews (@willypuzzle, #41717, #41783, #41911, #41911, #41977, #41984, #42243). This month also brought fixes for panics in parallel layout (@mrobinson, #42026), WebGPU (@WaterWhisperer, #42050), <link> fetching (@jdm, #42208), Element.attachShadow() (@mrobinson, #42237), text input methods (@mrobinson, #42240), Web Workers when the developer tools are active (@mrobinson, #42159), IndexedDB (@gterzian, #41960), and asynchronous session history updates (@mrobinson, #42238). Node.compareDocumentPosition() is now more efficient (@webbeef, #42260), and selections in text inputs no longer require a full page layout (@mrobinson, @Loirooriol, #41963). Donations Thanks again for your generous support! We are now receiving 7007 USD/month (−1.4% over December) in recurring donations. This helps us cover the cost of our speedy CI and benchmarking servers, one of our latest Outreachy interns, and funding maintainer work that helps more people contribute to Servo. Servo is also on thanks.dev, and already 33 GitHub users (+3 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. We now have sponsorship tiers that allow you or your organisation to donate to the Servo project with public acknowlegement of your support. A big thanks from Servo to our newest Bronze Sponsor: str4d! If you're interested in this kind of sponsorship, please contact us at join@servo.org. 7007 USD/month 10000 Use of donations is decided transparently via the Technical Steering Committee's public funding request process, and active proposals are tracked in servo/project#187. For more details, head to our Sponsorship page. Conference talks and blogs There were two talks about Servo at FOSDEM 2026 (videos and slides here): Implementing Streams Spec in Servo – Taym Haddadi (@taym95) described the challenges of implementing the Streams Standard. The Servo project and its impact on the web platform – Manuel Rego (@rego) highlighted the ways that Servo has shaped the web platform and contributed to web standards since it started in 2012.

- Mozilla Localization (L10N): Localizer Spotlight: Marcelo (2026/02/27 16:31)
  About You My name is Marcelo Poli. I live in Argentina, and I speak Spanish and English. I started contributing to Mozilla localization with Phoenix 0.3 — 24 years ago. Mozilla Localization Journey Q: How did you first get involved in localizing Mozilla products? A: There was a time when alternative browsers were incompatible with many websites. "Best with IE" appeared everywhere. Then Mozilla was reborn with Phoenix. It was just the browser — unlike Mozilla Suite (the old name for SeaMonkey) — and it was the best option. At first, it was only available in English, so I searched and found an opportunity to localize my favorite browser. There were already some Spanish localization for the Suite, and that became the base for my work. It took me two releases to complete it, and Phoenix 0.3 shipped with a full language pack — the first Spanish localization in Phoenix history. The most amazing part was that Mozilla let me do it. Q: Do you have a favorite product? Do you use the ones you localize regularly? A: Firefox is always my favorite. Thunderbird comes second — it's the simplest and most powerful email software. Firefox has been my default browser since the Phoenix era, and since many Mozilla products are connected, working on one often makes you want to contribute to others as well. Q: What moments stand out from your localization journey? A: Being part of the Firefox 1.0 release was incredible. The whole world was talking about the new browser, and my localization was part of it. Another unforgettable moment was seeing my name — along with hundreds of others — on the Mozilla Monument in San Francisco. Q: Have you shared your work with family and friends? A: Yes. I usually say,

"Try this, it's better," and many times they agree. Sometimes I have to explain the concept of free software. When they say, "But I didn't pay for the other browsers," I use the classic explanation: "Free as in freedom and free as in free beer." I wear Mozilla T-shirts, but I don't brag about managing the Argentinian localization. Still, some tech-savvy friends have found my name in the credits. Community & Collaboration Q: How does the Argentinian localization community work together today? Marcelo (right) with fellow Argentinian Mozillians Gabriela and Guillermo A: In the beginning, the Suite localization, Firefox localization, and the Argentinian community were separate. Mozilla encouraged us to join forces, and I eventually became the l10n manager.The community has grown and shrunk over time. Right now it's smaller, but localization remains the most active part, keeping products up to date. We stay in touch through an old mailing list, Matrix, and direct messages. I've also participated in many community events, although living far from Buenos Aires limits how often I can attend. Q: How do you coordinate translation, review, and testing? A: We're a small group, which actually makes coordination easier. Since we contribute in our free time, even small contributions matter, and three people can approve strings at any time. We test using Nightly as our main browser. Priorities are set in Pontoon — once the five-star products are complete, we move on to others. Usually, the number of untranslated strings is small, so it's manageable. Q: How has your role evolved over time? A: The old Mozilla folks — the "original cast," you could say — were essential in the early days. Before collaborative tools existed, I explained DTD and properties file structures to others. Some contributors had strong language skills but less technical background. Since the Phoenix years, I've been responsible for es-AR localization. At first, I worked alone; later others joined. Today, I hold the manager title in Pontoon. As Uncle Ben once said, "With great power comes great responsibility," so I check Pontoon daily. Q: What best practices would you share with other localizers? A: Pontoon is easy to use. The key is respecting terminology and staying consistent across the localization. If you find a typo or a better phrasing, suggest it directly in Pontoon. You don't need to contact a manager, and it doesn't matter how small the change is. Every contribution matters — even if it isn't approved. Professional Background & Skills Q: What is your professional background, and how has it helped your localization work? A: I studied programming, so I understand software structure and how it works. That helped a lot in the early days when localization required editing files directly — especially dealing with encoding and file structure. Knowledge of web development also helped with Developer Tools strings, and as a heavy user, I'm familiar with the terminology for almost everything you can do in software. Q: What have you gained beyond translation? A: Mozilla allows you to be part of something global — meeting people from different countries and learning how similar or different we are. Through community events and hackathons, I learned how to collaborate internationally. As a side effect, I became more fluent speaking English face to face than I expected. Q: After so many years, what keeps you motivated? A: My main motivation is being able to use Mozilla products in my own language. Mozilla is unique in having four Spanish localization. Most projects offer only one for all Spanish-speaking countries — or at best, one for Spain and one for Latin America. I'm not the most social person in the community, so recruiting isn't really my role. The best way I motivate others is simply by continuing to work on the projects. Many years ago, I contributed a few strings to Ubuntu localization — maybe they're still there. Fun Facts Marcelo as a DJ I was a radio DJ for many years — sometimes just playing music, sometimes talking about it. Paraphrasing Sting, I was born in the '60s and witnessed the first home computers like Texas Instruments and Commodore. My first personal computer was pre-Windows, with text-based screens, and I used Netscape Navigator on dial-up. I still prefer a big screen over a cellphone and mechanical keyboards over on-screen ones. These days, I'm learning how to build mobile apps.

- Niko Matsakis: How Dada enables internal references (2026/02/27 10:20)
  In my previous Dada blog post, I talked about how Dada enables composable sharing. Today I'm going to start diving into Dada's permission system; permissions are Dada's equivalent to Rust's borrow checker. Goal: richer, place-based permissions Dada aims to exceed Rust's

capabilities by using place-based permissions. Dada lets you write functions and types that capture both a value and things borrowed from that value. As a fun example, imagine you are writing some Rust code to process a comma-separated list, just looking for entries of length 5 or more: let list: String = format!("...something big, with commas..."); let items: Vec<&str> = list .split(",") .map(|s| s.trim()) // strip whitespace .filter(|s| s.len() > 5) .collect(); One of the cool things about Rust is how this code looks a lot like some high-level language like Python or JavaScript, but in those languages the split call is going to be doing a lot of work, since it will have to allocate tons of small strings, copying out the data. But in Rust the &str values are just pointers into the original string and so split is very cheap. I love this. On the other hand, suppose you want to package up some of those values, along with the backing string, and send them to another thread to be processed. You might think you can just make a struct like so… struct Message { list: String, items: Vec<&str>, // ---- // goal is to hold a reference // to strings from list } …and then create the list and items and store them into it: let list: String = format!("...something big, with commas..."); let items: Vec<&str> = /* as before */; let message = Message { list, items }; // ---- // | // This *moves* `list` into the struct. // That in turn invalidates `items`, which // is borrowed from `list`, so there is no // way to construct `Message`. But as experienced Rustaceans know, this will not work. When you have borrowed data like an &str, that data cannot be moved. If you want to handle a case like this, you need to convert from &str into sending indices, owned strings, or some other solution. Argh! Dada's permissions use places, not lifetimes Dada does things a bit differently. The first thing is that, when you create a reference, the resulting type names the place that the data was borrowed from, not the lifetime of the reference. So the type annotation for items would say ref[list] String1 (at least, if you wanted to write out the full details rather than leaving it to the type inferencer): let list: given String = "...something big, with commas..." let items: given Vec[ref[list] String] = list .split(",") .map(_.trim()) // strip whitespace .filter(_.len() > 5) // ------- I *think* this is the syntax I want for closures? // I forget what I had in mind, it's not implemented. .collect() I've blogged before about how I would like to redefine lifetimes in Rust to be places as I feel that a type like ref[list] String is much easier to teach and explain: instead of having to explain that a lifetime references some part of the code, or what have you, you can say that "this is a String that references the variable list". But what's also cool is that named places open the door to more flexible borrows. In Dada, if you wanted to package up the list and the items, you could build a Message type like so: class Message(list: String items: Vec[ref[self.list] String] // --------- // Borrowed from another field!) // As before: let list: String = "...something big, with commas..." let items: Vec[ref[list] String] = list .split(",") .map(_.strip()) // strip whitespace .filter(_.len() > 5) .collect() // Create the message, this is the fun part! let message = Message(list.give, items.give) Note that last line – Message(list.give, items.give). We can create a new class and move list into it along with items, which borrows from list. Neat, right? OK, so let's back up and talk about how this all works. References in Dada are the default Let's start with syntax. Before we tackle the Message example, I want to go back to the Character example from previous posts, because it's a bit easier for explanatory purposes. Here is some Rust code that declares a struct Character, creates an owned copy of it, and then gets a few references into it. struct Character { name: String, class: String, hp: u32, } let ch: Character = Character { name: format!("Ferris"), class: format!("Rustacean"), hp: 22 }; let p: &Character = &ch; let q: &String = &p.name; The Dada equivalent to this code is as follows: class Character(name: String, klass: String, hp: u32,) let ch: Character = Character("Tzara", "Dadaist", 22) let p: ref[ch] Character = ch let q: ref[p] String = p.name The first thing to note is that, in Dada, the default when you name a variable or a place is to create a reference. So let p = ch doesn't move ch, as it would in Rust, it creates a reference to the Character stored in ch. You could also explicitly write let p = ch.ref, but that is not preferred. Similarly, let q = p.name creates a reference to the value in the field name. (If you wanted to move the character, you would write let ch2 = ch.give, not let ch2 = ch as in Rust.) Notice that I said let p = ch "creates a reference to the Character stored in ch". In particular, I did not say "creates a reference to ch". That's a subtle choice of

wording, but it has big implications. References in Dada are not pointers The reason I wrote that let p = ch “creates a reference to the Character stored in ch” and not “creates a reference to ch” is because, in Dada, references are not pointers. Rather, they are shallow copies of the value, very much like how we saw in the previous post that a shared Character acts like an Arc<Character> but is represented as a shallow copy. So where in Rust the following code… let ch = Character { … }; let p = &ch; let q = &ch.name; …looks like this in memory… # Rust memory representation Stack Heap ——— ——— ┌——► ch: Character { │ ┌——► name: String { │ │ buffer: ————————► "Ferris" │ │ length: 6 │ │ capacity: 12 │ │ }, │ │ … │ │ } │ │ └——— p │ └— q in Dada, code like this let ch = Character(…) let p = ch let q = ch.name would look like so # Dada memory representation Stack Heap ——— ——— ch: Character { name: String { buffer: ————————┐► "Ferris" length: 6 │ capacity: 12 │ }, │ .. │ } │ │ p: Character { │ name: String { │ buffer: ————————┤ length: 6 │ capacity: 12 │ … │ } │ } │ │ q: String { │ buffer: ————————————┘ length: 6 capacity: 12 } Clearly, the Dada representation takes up more memory on the stack. But note that it doesn’t duplicate the memory in the heap, which tends to be where the vast majority of the data is found. Dada talks about values not references This gets at something important. Rust, like C, makes pointers first-class. So given x: &String, x refers to the pointer and *x refers to its referent, the String. Dada, like Java, goes another way. x: ref String is a String value – including in memory representation! The difference between a given String, shared String, and ref String is not in their memory layout, all of them are the same, but they differ in whether they own their contents.2 So in Dada, there is no *x operation to go from “pointer” to “referent”. That doesn’t make sense. Your variable always contains a string, but the permissions you have to use that string will change. In fact, the goal is that people don’t have to learn the memory representation as they learn Dada, you are supposed to be able to think of Dada variables as if they were all objects on the heap, just like in Java or Python, even though in fact they are stored on the stack.3 Rust does not permit moves of borrowed data In Rust, you cannot move values while they are borrowed. So if you have code like this that moves ch into ch1… let ch = Character { … }; let name = &ch.name; // create reference let ch1 = ch; // moves `ch` …then this code only compiles if name is not used again: let ch = Character { … }; let name = &ch.name; // create reference let ch1 = ch; // ERROR: cannot move while borrowed let name1 = name; // use reference again …but Dada can There are two reasons that Rust forbids moves of borrowed data: References are pointers, so those pointers may become invalidated. In the example above, name points to the stack slot for ch, so if ch were to be moved into ch1, that makes the reference invalid. The type system would lose track of things. Internally, the Rust borrow checker has a kind of “indirection”. It knows that ch is borrowed for some span of the code (a “lifetime”), and it knows that the lifetime in the type of name is related to that lifetime, but it doesn’t really know that name is borrowed from ch in particular.4 Neither of these apply to Dada: Because references are not pointers into the stack, but rather shallow copies, moving the borrowed value doesn’t invalidate their contents. They remain valid. Because Dada’s types reference actual variable names, we can modify them to reflect moves. Dada tracks moves in its types OK, let’s revisit that Rust example that was giving us an error. When we convert it to Dada, we find that it type checks just fine: class Character(…) // as before let ch: given Character = Character(…) let name: ref[ch.name] String = ch.name // -- originally it was borrowed from `ch` let ch1 = ch.give // ------- but `ch` was moved to `ch1` let name1: ref[ch1.name] = name // --- now it is borrowed from `ch1` Woah, neat! We can see that when we move from ch into ch1, the compiler updates the types of the variables around it. So actually the type of name changes to ref[ch1.name] String. And then when we move from name to name1, that’s totally valid. In PL land, updating the type of a variable from one thing to another is called a “strong update”. Obviously things can get a bit complicated when control-flow is involved, e.g., in a situation like this: let ch = Character(…) let ch1 = Character(…) let name = ch.name if some_condition_is_true() { // On this path, the type of `name` changes // to `ref[ch1.name] String`, and so `ch` // is no longer considered borrowed. ch1 = ch.give ch = Character(…) // not borrowed, we can mutate } else { // On this path, the type of

`name` // remains unchanged, and `ch` is borrowed. } // Here, the types are merged, so the // type of `name` is `ref[ch.name, ch1.name] String`. // Therefore, `ch` is considered borrowed here. Renaming lets us call functions with borrowed values OK, let's take the next step. Let's define a Dada function that takes an owned value and another value borrowed from it, like the name, and then call it: fn character_and_name(ch1: given Character, name1: ref[ch1] String,) { // ... does something ... } We could call this function like so, as you might expect: let ch = Character(...) let name = ch.name character_and_name(ch.give, name) So...how does this work? Internally, the type checker type-checks a function call by creating a simpler snippet of code, essentially, and then type-checking that. It's like desugaring but only at type-check time. In this simpler snippet, there are a series of let statements to create temporary variables for each argument. These temporaries always have an explicit type taken from the method signature, and they are initialized with the values of each argument: // type checker "desugars" `character_and_name(ch.give, name)` // into more primitive operations: let tmp1: given Character = ch.give // --------------- ------- // | taken from the call // taken from fn sig let tmp2: ref[tmp1.name] String = name // --------------------- ---- // | taken from the call // taken from fn sig, // but rewritten to use the new // temporaries If this type checks, then the type checker knows you have supplied values of the required types, and so this is a valid call. Of course there are a few more steps, but that's the basic idea. Notice what happens if you supply data borrowed from the wrong place: let ch = Character(...) let ch1 = Character(...) character_and_name(ch, ch1.name) // --- wrong place! This will fail to type check because you get: let tmp1: given Character = ch.give let tmp2: ref[tmp1.name] String = ch1.name // -------- // has type `ref[ch1.name] String`, // not `ref[tmp1.name] String` Class constructors are "just" special functions So now, if we go all the way back to our original example, we can see how the Message example worked: class Message(list: String items: Vec[ref[self.list] String]) Basically, when you construct a Message(list, items), that's "just another function call" from the type system's perspective, except that self in the signature is handled carefully. This is modeled, not implemented I should be clear, this system is modeled in the dada-model repository, which implements a kind of "mini Dada" that captures what I believe to be the most interesting bits. I'm working on fleshing out that model a bit more, but it's got most of what I showed you here.5 For example, here is a test that you get an error when you give a reference to the wrong value. The "real implementation" is lagging quite a bit, and doesn't really handle the interesting bits yet. Scaling it up from model to real implementation involves solving type inference and some other thorny challenges, and I haven't gotten there yet – though I have some pretty interesting experiments going on there too, in terms of the compiler architecture.6 This could apply to Rust I believe we could apply most of this system to Rust. Obviously we'd have to rework the borrow checker to be based on places, but that's the straight-forward part. The harder bit is the fact that &T is a pointer in Rust, and that we cannot readily change. However, for many use cases of self-references, this isn't as important as it sounds. Often, the data you wish to reference is living in the heap, and so the pointer isn't actually invalidated when the original value is moved. Consider our opening example. You might imagine Rust allowing something like this in Rust: struct Message { list: String, items: Vec<&{self.list} str>, } In this case, the str data is heap-allocated, so moving the string doesn't actually invalidate the &str value (it would invalidate an &String value, interestingly). In Rust today, the compiler doesn't know all the details of what's going on. String has a Deref impl and so it's quite opaque whether str is heap-allocated or not. But we are working on various changes to this system in the Beyond the & goal, most notably the Field Projections work. There is likely some opportunity to address this in that context, though to be honest I'm behind in catching up on the details. I'll note in passing that Dada unifies str and String into one type as well. I'll talk in detail about how that works in a future blog post. ↩ This is kind of like C++ references (e.g., String&), which also act "as if" they were a value (i.e., you write s.foo(), not s->foo()), but a C++ reference is truly a pointer, unlike a Dada ref. ↩ This goal was in part inspired by a conversation I had early on within Amazon, where a (quite experienced) developer told me, "It took me months to understand what

variables are in Rust". ↩ I explained this some years back in a talk on Polonius at Rust Belt Rust, if you'd like more detail. ↩ No closures or iterator chains! ↩ As a teaser, I'm building it in async Rust, where each inference variable is a "future" and use "await" to find out when other parts of the code might have added constraints. ↩

- Hacks.Mozilla.Org: Why is WebAssembly a second-class language on the web? (2026/02/26 16:02)
  This post is an expanded version of a presentation I gave at the 2025 WebAssembly CG meeting in Munich. WebAssembly has come a long way since its first release in 2017. The first version of WebAssembly was already a great fit for low-level languages like C and C++, and immediately enabled many new kinds of applications to efficiently target the web. Since then, the WebAssembly CG has dramatically expanded the core capabilities of the language, adding shared memories, SIMD, exception handling, tail calls, 64-bit memories, and GC support, alongside many smaller improvements such as bulk memory instructions, multiple returns, and reference values. These additions have allowed many more languages to efficiently target WebAssembly. There's still more important work to do, like stack switching and improved threading, but WebAssembly has narrowed the gap with native in many ways. Yet, it still feels like something is missing that's holding WebAssembly back from wider adoption on the Web. There are multiple reasons for this, but the core issue is that WebAssembly is a second-class language on the web. For all of the new language features, WebAssembly is still not integrated with the web platform as tightly as it should be. This leads to a poor developer experience, which pushes developers to only use WebAssembly when they absolutely need it. Oftentimes JavaScript is simpler and "good enough". This means its users tend to be large companies with enough resources to justify the investment, which then limits the benefits of WebAssembly to only a small subset of the larger Web community. Solving this issue is hard, and the CG has been focused on extending the WebAssembly language. Now that the language has matured significantly, it's time to take a closer look at this. We'll go deep into the problem, before talking about how WebAssembly Components could improve things. What makes WebAssembly second-class? At a very high level, the scripting part of the web platform is layered like this: WebAssembly can directly interact with JavaScript, which can directly interact with the web platform. WebAssembly can access the web platform, but only by using the special capabilities of JavaScript. JavaScript is a first-class language on the web, and WebAssembly is not. This wasn't an intentional or malicious design decision; JavaScript is the original scripting language of the Web and co-evolved with the platform. Nonetheless, this design significantly impacts users of WebAssembly. What are these special capabilities of JavaScript? For today's discussion, there are two major ones: Loading of code Using Web APIs Loading of code WebAssembly code is unnecessarily cumbersome to load. Loading JavaScript code is as simple as just putting it in a script tag: <script src="script.js"></script> WebAssembly is not supported in script tags today, so developers need to use the WebAssembly JS API to manually load and instantiate code. let bytecode = fetch(import.meta.resolve('./module.wasm')); let imports = { ... }; let { exports } = await WebAssembly.instantiateStreaming(bytecode, imports); The exact sequence of API calls to use is arcane, and there are multiple ways to perform this process, each of which has different tradeoffs that are not clear to most developers. This process generally just needs to be memorized or generated by a tool for you. Thankfully, there is the esm-integration proposal, which is already implemented in bundlers today and which we are actively implementing in Firefox. This proposal lets developers import WebAssembly modules from JS code using the familiar JS module system. import { run } from "/module.wasm"; run(); In addition, it allows a WebAssembly module to be loaded directly from a script tag using type="module": <script type="module" src="/module.wasm"></script> This streamlines the most common patterns for loading and instantiating WebAssembly modules. However, while this mitigates the initial difficulty, we quickly run into the real problem. Using Web APIs Using a Web API from JavaScript is as simple as this: console.log("hello, world"); For WebAssembly, the situation is much more complicated.

WebAssembly has no direct access to Web APIs and must use JavaScript to access them. The same single-line console.log program requires the following JavaScript file: // We need access to the raw memory of the Wasm code, so // create it here and provide it as an import. let memory = new WebAssembly.Memory(...); function consoleLog(messageStartIndex, messageLength) { // The string is stored in Wasm memory, but we need to // decode it into a JS string, which is what DOM APIs // require. let messageMemoryView = new UInt8Array(memory.buffer, messageStartIndex, messageLength); let messageString = new TextDecoder().decode(messageMemoryView); // Wasm can't get the `console` global, or do // property lookup, so we do that here. return console.log(messageString); } // Pass the wrapped Web API to the Wasm code through an // import. let imports = { "env": { "memory": memory, "consoleLog": consoleLog, }, }; let { instance } = await WebAssembly.instantiateStreaming(bytecode, imports); instance.exports.run(); And the following WebAssembly file: (module ;; import the memory from JS code (import "env" "memory" (memory 0)) ;; import the JS consoleLog wrapper function (import "env" "consoleLog" (func $consoleLog (param i32 i32))) ;; export a run function (func (export "run") (local i32 $messageStartIndex) (local i32 $messageLength) ;; create a string in Wasm memory, store in locals ... ;; call the consoleLog method local.get $messageStartIndex local.get $messageLength call $consoleLog)) Code like this is called “bindings” or “glue code” and acts as the bridge between your source language (C++, Rust, etc.) and Web APIs. This glue code is responsible for re-encoding WebAssembly data into JavaScript data and vice versa. For example, when returning a string from JavaScript to WebAssembly, the glue code may need to call a malloc function in the WebAssembly module and re-encode the string at the resulting address, after which the module is responsible for eventually calling free. This is all very tedious, formulaic, and difficult to write, so it is typical to generate this glue automatically using tools like embind or wasm-bindgen. This streamlines the authoring process, but adds complexity to the build process that native platforms typically do not require. Furthermore, this build complexity is language-specific; Rust code will require different bindings from C++ code, and so on. Of course, the glue code also has runtime costs. JavaScript objects must be allocated and garbage collected, strings must be re-encoded, structs must be deserialized. Some of this cost is inherent to any bindings system, but much of it is not. This is a pervasive cost that you pay at the boundary between JavaScript and WebAssembly, even when the calls themselves are fast. This is what most people mean when they ask “When is Wasm going to get DOM support?” It’s already possible to access any Web API with WebAssembly, but it requires JavaScript glue code. Why does this matter? From a technical perspective, the status quo works. WebAssembly runs on the web and many people have successfully shipped software with it. From the average web developer’s perspective, though, the status quo is subpar. WebAssembly is too complicated to use on the web, and you can never escape the feeling that you’re getting a second class experience. In our experience, WebAssembly is a power user feature that average developers don’t use, even if it would be a better technical choice for their project. The average developer experience for someone getting started with JavaScript is something like this: There’s a nice gradual curve where you use progressively more complicated features as the scope of your project increases. By comparison, the average developer experience for someone getting started with WebAssembly is something like this: You immediately must scale “the wall” of wrangling the many different pieces to work together. The end result is often only worth it for large projects. Why is this the case? There are several reasons, and they all directly stem from WebAssembly being a second class language on the web. 1. It’s difficult for compilers to provide first-class support for the web Any language targeting the web can’t just generate a Wasm file, but also must generate a companion JS file to load the Wasm code, implement Web API access, and handle a long tail of other issues. This work must be redone for every language that wants to support the web, and it can’t be reused for non-web platforms. Upstream compilers like Clang/LLVM don’t want to know anything about JS or the web platform, and not just for lack of effort. Generating and maintaining JS and web glue code is a specialty skill that is difficult for already

stretched-thin maintainers to justify. They just want to generate a single binary, ideally in a standardized format that can also be used on platforms besides the web. 2. Standard compilers don't produce WebAssembly that works on the web The result is that support for WebAssembly on the web is often handled by third-party unofficial toolchain distributions that users need to find and learn. A true first-class experience would start with the tool that users already know and have installed. This is, unfortunately, many developers' first roadblock when getting started with WebAssembly. They assume that if they just have rustc installed and pass a –target=wasm flag that they'll get something they could load in a browser. You may be able to get a WebAssembly file doing that, but it will not have any of the required platform integration. If you figure out how to load the file using the JS API, it will fail for mysterious and hard-to-debug reasons. What you really need is the unofficial toolchain distribution which implements the platform integration for you. 3. Web documentation is written for JavaScript developers The web platform has incredible documentation compared to most tech platforms. However, most of it is written for JavaScript. If you don't know JavaScript, you'll have a much harder time understanding how to use most Web APIs. A developer wanting to use a new Web API must first understand it from a JavaScript perspective, then translate it into the types and APIs that are available in their source language. Toolchain developers can try to manually translate the existing web documentation for their language, but that is a tedious and error prone process that doesn't scale. 4. Calling Web APIs can still be slow If you look at all of the JS glue code for the single call to console.log above, you'll see that there is a lot of overhead. Engines have spent a lot of time optimizing this, and more work is underway. Yet this problem still exists. It doesn't affect every workload, but it's something every WebAssembly user needs to be careful about. Benchmarking this is tricky, but we ran an experiment in 2020 to precisely measure the overhead that JS glue code has in a real world DOM application. We built the classic TodoMVC benchmark in the experimental Dodrio Rust framework and measured different ways of calling DOM APIs. Dodrio was perfect for this because it computed all the required DOM modifications separately from actually applying them. This allowed us to precisely measure the impact of JS glue code by swapping out the "apply DOM change list" function while keeping the rest of the benchmark exactly the same. We tested two different implementations: "Wasm + JS glue": A WebAssembly function which reads the change list in a loop, and then asks JS glue code to apply each change individually. This is the performance of WebAssembly today. "Wasm only": A WebAssembly function which reads the change list in a loop, and then uses an experimental direct binding to the DOM which skips JS glue code. This is the performance of WebAssembly if we could skip JS glue code. The duration to apply the DOM changes dropped by 45% when we were able to remove JS glue code. DOM operations can already be expensive; WebAssembly users can't afford to pay a 2x performance tax on top of that. And as this experiment shows, it is possible to remove the overhead. 5. You always need to understand the JavaScript layer There's a saying that "abstractions are always leaky". The state of the art for WebAssembly on the web is that every language builds their own abstraction of the web platform using JavaScript. But these abstractions are leaky. If you use WebAssembly on the web in any serious capacity, you'll eventually hit a point where you need to read or write your own JavaScript to make something work. This adds a conceptual layer which is a burden for developers. It feels like it should just be enough to know your source language, and the web platform. Yet for WebAssembly, we require users to also know JavaScript in order to be a proficient developer. How can we fix this? This is a complicated technical and social problem, with no single solution. We also have competing priorities for what is the most important problem with WebAssembly to fix first. Let's ask ourselves: In an ideal world, what could help us here? What if we had something that was: A standardized self-contained executable artifact Supported by multiple languages and toolchains Which handles loading and linking of WebAssembly code Which supports Web API usage If such a thing existed, languages could generate these artifacts and browsers could run them, without any JavaScript involved. This format would be easier for languages to support and could potentially exist in standard upstream

compilers, runtimes, toolchains, and popular packages without the need for third-party distributions. In effect, we could go from a world where every language re-implements the web platform integration using JavaScript, to sharing a common one that is built directly into the browser. It would obviously be a lot of work to design and validate a solution! Thankfully, we already have a proposal with these goals that has been in development for years: the WebAssembly Component Model. What is a WebAssembly Component? For our purposes, a WebAssembly Component defines a high-level API that is implemented with a bundle of low-level WebAssembly code. It's a standards-track proposal in the WebAssembly CG that's been in development since 2021. Already today, WebAssembly Components… Can be created from many different programming languages. Can be executed in many different runtimes (including in browsers today, with a polyfill). Can be linked together to allow code re-use between different languages. Allow WebAssembly code to directly call Web APIs. If you're interested in more details, check out the Component Book or watch "What is a Component?". We feel that WebAssembly Components have the potential to give WebAssembly a first-class experience on the web platform, and to be the missing link described above. How could they work? Let's try to re-create the earlier console.log example using only WebAssembly Components and no JavaScript. NOTE: The interactions between WebAssembly Components and the web platform have not been fully designed, and the tooling is under active development. Take this as an aspiration for how things could be, not a tutorial or promise. The first step is to specify which APIs our application needs. This is done using an IDL called WIT. For our example, we need the Console API. We can import it by specifying the name of the interface. component { import std:web/console; } The std:web/console interface does not exist today, but would hypothetically come from the official WebIDL that browsers use for describing Web APIs. This particular interface might look like this: package std:web; interface console { log: func(msg: string); … } Now that we have the above interfaces, we can use them when writing a Rust program that compiles to a WebAssembly Component: use std::web::console; fn main() { console::log("hello, world"); } Once we have a component, we can load it into the browser using a script tag. <script type="module" src="component.wasm"></script> And that's it! The browser would automatically load the component, bind the native web APIs directly (without any JS glue code), and run the component. This is great if your whole application is written in WebAssembly. However, most WebAssembly usage is part of a "hybrid application" which also contains JavaScript. We also want to simplify this use case. The web platform shouldn't be split into "silos" that can't interact with each other. Thankfully, WebAssembly Components also address this by supporting cross-language interoperability. Let's create a component that exports an image decoder for use from JavaScript code. First we need to write the interface that describes the image decoder: interface image-lib { record pixel { r: u8; g: u8; b: u8; a: u8; } resource image { from-stream: static async func(bytes: stream<u8>) -> result<image>; get: func(x: u32, y: u32) -> pixel; } } component { export image-lib; } Once we have that, we can write the component in any language that supports components. The right language will depend on what you're building or what libraries you need to use. For this example, I'll leave the implementation of the image decoder as an exercise for the reader. The component can then be loaded in JavaScript as a module. The image decoder interface we defined is accessible to JavaScript, and can be used as if you were importing a JavaScript library to do the task. import { Image } from "image-lib.wasm"; let byteStream = (await fetch("/image.file")).body; let image = await Image.fromStream(byteStream); let pixel = image.get(0, 0); console.log(pixel); // { r: 255, g: 255, b: 0, a: 255 } Next Steps As it stands today, we think that WebAssembly Components would be a step in the right direction for the web. Mozilla is working with the WebAssembly CG to design the WebAssembly Component Model. Google is also evaluating it at this time. If you're interested to try this out, learn to build your first component and try it out in the browser using Jco or from the command-line using Wasmtime. The tooling is under heavy development, and contributions and feedback are welcome. If you're interested in the in-development specification itself, check out the component-model proposal repository. WebAssembly has come very far from when it was first

released in 2017. I think the best is still yet to come if we're able to turn it from being a "power user" feature, to something that average developers can benefit from. The post Why is WebAssembly a second-class language on the web? appeared first on Mozilla Hacks - the Web developer blog.

- This Week In Rust: This Week in Rust 640 (2026/02/25 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.bsky.social on Bluesky 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 2026 Rust debugging survey 2026 Foundation Guest Blog: FOSDEM 2026 — Rust Devroom in Review Project/Tooling Updates Zed: Split Diffs are Here CHERIoT Rust: Status update #0 SeaORM now supports Arrow & Parquet Releasing bincode-next v3.0.0-rc.1 Introducing Almonds SafePilot v0.1: self-hosted AI assistant Hitbox 0.2.0: declarative cache orchestration Observations/Thoughts What it means that Ubuntu is using Rust Read Locks Are Not Your Friends Achieving Zero Bugs: Rust, Specs, and AI Coding [video] device-envoy: Making Embedded Fun with Rust—by Carl Kadie Rust Walkthroughs About memory pressure, lock contention, and Data-oriented Design Breaking SHA-2: length extension attacks in practice with Rust device-envoy: Making Embedded Fun with Rust, Embassy, and Composable Device Abstractions Research Auditing Rust Crates Effectively Miscellaneous Hieratic Prompt Compression: From Prototype to Production Crate of the Week This week's crate is docstr, a macro crate providing a macro to create multiline strings out of doc comments. Thanks to Nik Revenco for the 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, Cargo, Rustup or Rust language RFCs. 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 Bluesky 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. Rust India Conference 2026 | CFP open until 2026-03-14 | Bangalore, IN | 2026-04-18 Oxidize Conference | CFP open until 2026-03-23 | Berlin, Germany | 2026-09-14 - 2026-09-16 EuroRust | CFP open until 2026-04-27 | Barcelona, Spain | 2026-10-14 - 2026-10-17 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 Bluesky or Mastodon! Updates from the Rust Project 450 pull requests were merged in the last week Compiler bring back enum DepKind simplify the canonical enum clone branches to a copy statement stabilize if let guards (feature(if_let_guard)) Library add try_shrink_to and try_shrink_to_fit to Vec fixed ByteStr not padding within its Display trait when no specific alignment is mentioned reflection TypeId::trait_info_of reflection TypeKind::FnPtr just pass Layout directly to box_new_uninit stabilize cfg_select! Cargo cli: Remove --lockfile-path job_queue: Handle Clippy CLI arguments in fix message fix parallel locking when -Zfine-grain-locking is enabled Clippy add

unnecessary_trailing_comma lint add new disallowed_fields lint clone_on_ref_ptr: don't add a & to the receiver if it's a reference needless_maybe_sized: don't lint in proc-macro-generated code str_to_string: false positive non-str types useless_conversion: also fire inside compiler desugarings add allow-unwrap-types configuration for unwrap_used and expect_used add brackets around unsafe or labeled block used in else allow deprecated(since = "CURRENT_RUSTC_VERSION") do not suggest removing reborrow of a captured upvar enhance collapsible_match to cover if-elses enhance manual_is_variant_and to cover filter chaining is_some fix explicit_counter_loop false negative when loop counter starts at non-zero fix join_absolute_paths to work correctly depending on the platform fix redundant_iter_cloned false positive with move closures and coroutines fix unnecessary_min_or_max for usize fix panic/assert message detection in edition 2015/2018 handle Result<T, !> and ControlFlow<!, T> as T wrt #[must_use] make unchecked_time_subtraction to better handle Duration literals make unnecessary_fold commutative the path from a type to itself is Self Rust-Analyzer add partial selection for generate_getter_or_setter offer block let fallback postfix complete offer on is_some_and for replace_is_method_with_if_let_method fix some TryEnum reference assists add handling for cycles in sizedness_constraint_for_ty() better import placement + merging complete .let on block tail prefix expression complete derive helpers on empty nameref correctly parenthesize inverted condition in convert_if_to_bool_... exclude macro refs in tests when excludeTests is enabled fix another case where we forgot to put the type param for PartialOrd and PartialEq in builtin derives fix predicates of builtin derive traits with two parameters defaulting to Self generate method assist uses enclosing impl block instead of first found no complete suggest param in complex pattern offer toggle_macro_delimiter in nested macro prevent qualifying parameter names in add_missing_impl_members implement Span::SpanSouce for proc-macro-srv Rust Compiler Performance Triage Overall, a bit more noise than usual this week, but mostly a slight improvement with several low-level optimizations at MIR and LLVM IR building landing. Also less commits landing than usual, mostly due to GitHub CI issues during the week. Triage done by @simulacrum. Revision range: 3c9faa0d..eeb94be7 3 Regressions, 4 Improvements, 4 Mixed; 3 of them in rollups 24 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 Gate #![reexport_test_harness_main] properly Observe close(2) errors for std::fs::{copy, write} warn on empty precision refactor 'valid for read/write' definition: exclude null Compiler Team (MCPs only) Remove -Csoft-float Place-less cg_ssa intrinsics Optimize repr(Rust) enums by omitting tags in more cases involving uninhabited variants. Proposal for a dedicated test suite for the parallel frontend Promote tier 3 riscv32 ESP-IDF targets to tier 2 Proposal for Adapt Stack Protector for Rust Cargo feat(help): display manpage for nested commands No Items entered Final Comment Period this week for Rust RFCs, Language Reference, Language Team, Leadership Council 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 Cargo: hints.min-opt-level Cargo RFC for min publish age Place traits RFC: Extend manifest dependencies with used Upcoming Events Rusty Events between 2026-02-25 - 2026-03-25 □ Virtual 2026-02-25 | Virtual (Cardiff, UK) | Rust and C++ Cardiff Getting Started with Rust Part 3: Patterns and Matching 2026-02-25 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-02-26 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-03-04 | Virtual (Indianapolis, IN, US) | Indy Rust Indy.rs - with Social Distancing 2026-03-05 | Virtual (Charlottesville, VA, US) | Charlottesville Rust Meetup Presentation: Tock OS Part #3 - Capsules and lower-level hardware drivers 2026-03-05 | Virtual (Nürnberg, DE) | Rust Nuremberg Rust Nürnberg online 2026-03-07 | Virtual (Kampala, UG) | Rust Circle Meetup Rust Circle Meetup 2026-03-10 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Second Tuesday 2026-03-10 | Virtual (London, UK)| Women in Rust □ Community

Catch Up 2026-03-11 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-12 | Virtual (Berlin, DE) | Rust Berlin Rust Hack and Learn 2026-03-17 | Virtual (Washington, DC, US) | Rust DC Mid-month Rustful 2026-03-18 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session 2026-03-18 | Virtual (Vancouver, BC, CA) | Vancouver Rust Embedded Rust 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-20 | Virtual | Packt Publishing Limited Rust Adoption, Safety, and Cloud with Francesco Ciulla 2026-03-24 | Virtual (Dallas, TX, US) | Dallas Rust User Meetup Fourth Tuesday 2026-03-24 | Virtual (London, UK) | Women in Rust Lunch & Learn: Crates, Tips & Tricks Lightning Talks - Bring your ideas! 2026-03-25 | Virtual (Girona, ES) | Rust Girona Sessió setmanal de codificació / Weekly coding session Asia 2026-03-22 | Tel Aviv-yafo, IL | Rust □ TLV In person Rust March 2026 at AWS in Tel Aviv Europe 2026-02-25 | Copenhagen, DK | Copenhagen Rust Community Rust meetup #65 Sponsored by Factbird 2026-02-26 | Prague, CZ | Rust Czech Republic Informační teorie vs. filtry: Proč filtrování bitcoinového mempoolu NEFUNGUJE 2026-02-28 | Stockholm, SE | Stockholm Rust Ferris' Fika Forum #24 - crablings edition 2026-03-04 | Barcelona, ES | BcnRust Rust at MWC Talent Arena — Workshops + Community Meetup 2026-03-04 | Hamburg, DE | Rust Meetup Hamburg Rust Hack & Learn March 2026 2026-03-04 | Oxford, UK | Oxford ACCU/Rust Meetup. Records, Shredded on Ice: A Primer on Parquet and Iceberg 2026-03-05 | Oslo, NO | Rust Oslo Rust Hack'n'Learn at Kampen Bistro 2026-03-11 | Amsterdam, NL | Rust Developers Amsterdam Group Meetup @ Instruqt 2026-03-12 | Geneva, CH | Post Tenebras Lab Rust Meetup Geneva 2026-03-18 | Dortmund, DE | Rust Dortmund Rust Dortmund Meetup - Intro to Embedded Rust - March 2026-03-19 - 2026-03-20 | | Rustikon Rustikon Conference 2026-03-24 | Aarhus, DK | Rust Aarhus Hack Night - Advent of Code North America 2026-02-25 | Austin, TX, US | Rust ATX Rust Lunch - Fareground 2026-02-25 | Los Angeles, CA, US | Rust Los Angeles Rust LA: Rust as a Glue Layer- Infrastructure for AI-Native Applications 2026-02-26 | Atlanta, GA, US | Rust Atlanta Rust-Atl 2026-02-26 | New York, NY, US | Rust NYC Rust NYC: Compile-Time Solutions 2026-02-28 | Boston, MA, US | Boston Rust Meetup Boston University Rust Lunch, Feb 28 2026-03-05 | Saint Louis, MO, US | STL Rust TBD 2026-03-07 | Boston, MA, US | Boston Rust Meetup MIT Rust Lunch, Mar 7 2026-03-14 | Boston, MA, US | Boston Rust Meetup North End Rust Lunch, Mar 14 2026-03-17 | San Francisco, CA, US | San Francisco Rust Study Group Rust Hacking in Person 2026-03-19 | Hybrid (Seattle, WA, US) | Seattle Rust User Group March, 2026 SRUG (Seattle Rust User Group) Meetup 2026-03-21 | Boston, MA, US | Boston Rust Meetup Porter Square Rust Lunch, Mar 21 2026-03-25 | Austin, TX, US | Rust ATX Rust Lunch - Fareground Oceania 2026-03-26 | Melbourne, VIC, AU | Rust Melbourne TBD March Meetup 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 This is actually just Rust adding support for C++-style duck-typed templates, and the long and mostly-irrelevant information contained in the ICE message is part of the experience. – robofinch on rust-users Thanks to Kyllingene for the suggestion! Please submit quotes and vote for next week! This Week in Rust is edited by: nellshamrell llogiq ericseppanen extrawurst U007D mariannegoldin bdillo opeolluwa bnchi KannanPalani57 tzilist Email list hosting is sponsored by The Rust Foundation Discuss on r/rust

- The Mozilla Blog: How to turn off AI features in Firefox, or choose the ones you want (2026/02/24 17:56)
  Other browsers force AI features on users. Firefox gives you a choice. In the latest desktop version of Firefox, you'll find an AI controls section where you can turn off AI features entirely — or decide which ones stay on. Here's how to set things up the way you want. But first, what AI features can you manage in Firefox? AI is a broad term. Traditional machine learning — like systems that classify, rank or personalize an experience — have been part of browsers for a while. AI controls focus on newer AI-enhanced features on Firefox, including website translations, image alt text in PDFs, AI-enhanced tab groups, link previews and an AI chatbot in the sidebar. Block AI features with a single switch To block

current and future AI-enhanced features: Menu bar > Firefox > Settings (or Preferences) > AI Controls Turn on Block AI enhancements Or block specific features You can also manage AI features individually. Block link previews? Up to you. Change your mind on translations? You can turn them on any time, while keeping all other AI features blocked. To block specific features: Menu bar > Firefox > Settings (or Preferences) > AI Controls Find the AI feature > dropdown menu > Blocked In the dropdown menu, Available means you can see and use the feature; Enabled means you've opted in to use it; and Blocked means it's hidden and can't be used. Choose the AI features you want to use Enable image alt text in PDFs to improve accessibility. Keep AI-enhanced tab group suggestions if they're useful. Block anything that isn't — the decision is yours. You can update your choices in Setting anytime. For more details on AI controls, head to our Mozilla Support page. Have feedback? We'd love to hear it on Mozilla Connect – or come chat with us during our Reddit AMA on Feb. 26. The post How to turn off AI features in Firefox, or choose the ones you want appeared first on The Mozilla Blog.

- Firefox Developer Experience: Firefox WebDriver Newsletter 148 (2026/02/24 14:26)
  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 148 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 148, a WebDriver bug was fixed by a contributor: Sameem added a helper to assert and transform browsing and user contexts in emulation commands. 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 Fixed a race condition during initialization of required browser features when opening a new window, preventing issues when navigating immediately to another URL. Fixed an interoperability issue between Marionette and WebDriver BiDi where the BiDi clientWindow ID was incorrectly used as a window handle in Marionette. WebDriver BiDi Added initial support for interacting with the browser's chrome scope (the Firefox window itself). The browsingContext.getTree command now accepts the vendor specific moz:scope parameter and returns chrome contexts when set to chrome and Firefox was started with the --remote-allow-system-access argument. These contexts can be used with script.evaluate and script.callFunction to execute privileged JavaScript with access to Gecko APIs. Other commands do not yet support chrome contexts, but support will be added incrementally as needed. Updated the emulation.setGeolocationOverride and emulation.setScreenOrientationOverride commands to implement the new reset behavior: contexts are reset only when the contexts parameter is provided, and user contexts only when the userContexts parameter is specified. Fixed a race condition in browsingContext.create where opening a new tab in the foreground could return before the document became visible. Fixed an issue that occurred when a navigation redirected to an error page. Fixed an issue in network.getData that caused a RangeError when decoding chunked response bodies due to a size mismatch. Fixed an issue where the browsingContext.userPromptOpened and browsingContext.userPromptClosed events incorrectly reported the top-level context ID instead of the iframe's context ID. Improved the performance of WebDriver BiDi commands by approximately 100 ms when the selected context is no longer available during the command execution. Marionette Added the Reporting:GenerateTestReport command to generate a test report via the Reporting API.
- Hacks.Mozilla.Org: Goodbye innerHTML, Hello setHTML: Stronger XSS Protection in Firefox 148 (2026/02/24 13:00)

Cross-site scripting (XSS) remains one of the most prevalent vulnerabilities on the web. The new standardized Sanitizer API provides a straightforward way for web developers to sanitize untrusted HTML before inserting it into the DOM. Firefox 148 is the first browser to ship this standardized security enhancing API, advancing a safer web for everyone. We expect other browsers to follow soon. An XSS vulnerability arises when a website inadvertently lets attackers inject arbitrary HTML or JavaScript through user-generated content. With this attack, an attacker could monitor and manipulate user interactions and continually steal user data for as long as the vulnerability remains exploitable. XSS has a long history of being notoriously difficult to prevent and has ranked among the top three web vulnerabilities (CWE-79) for nearly a decade. Firefox has been deeply involved in solutions for XSS from the beginning, starting with spearheading the Content-Security-Policy (CSP) standard in 2009. CSP allows websites to restrict which resources (scripts, styles, images, etc.) the browser can load and execute, providing a strong line of defense against XSS. Despite a steady stream of improvements and ongoing maintenance, CSP did not gain sufficient adoption to protect the long tail of the web as it requires significant architectural changes for existing web sites and continuous review by security experts. The Sanitizer API is designed to help fill that gap by providing a standardized way to turn malicious HTML into harmless HTML — in other words, to sanitize it. The setHTML( ) method integrates sanitization directly into HTML insertion, providing safety by default. Here is an example of sanitizing a simple unsafe HTML: document.body.setHTML(`<h1>Hello my name is <img src="x" onclick="alert('XSS')">`); This sanitization will allow the HTML <h1> element while removing the embedded <img> element and its onclick attribute, thereby eliminating the XSS attack resulting in the following safe HTML: <h1>Hello my name is</h1> Developers can opt into stronger XSS protections with minimal code changes by replacing error-prone innerHTML assignments with setHTML(). If the default configuration of setHTML( ) is too strict (or not strict enough) for a given use case, developers can provide a custom configuration that defines which HTML elements and attributes should be kept or removed. To experiment with the Sanitizer API before introducing it on a web page, we recommend exploring the Sanitizer API playground. For even stronger protections, the Sanitizer API can be combined with Trusted Types, which centralize control over HTML parsing and injection. Once setHTML( ) is adopted, sites can enable Trusted Types enforcement more easily, often without requiring complex custom policies. A strict policy can allow setHTML( ) while blocking other unsafe HTML insertion methods, helping prevent future XSS regressions. The Sanitizer API enables an easy replacement of innerHTML assignments with setHTML( ) in existing code, introducing a new safer default to protect users from XSS attacks on the web. Firefox 148 supports the Sanitizer API as well as Trusted Types, which creates a safer web experience. Adopting these standards will allow all developers to prevent XSS without the need for a dedicated security team or significant implementation changes. Image credits for the illustration above: Website, by Desi Ratna; Person, by Made by Made; Hacker by Andy Horvath. The post Goodbye innerHTML, Hello setHTML: Stronger XSS Protection in Firefox 148 appeared first on Mozilla Hacks - the Web developer blog.

- Firefox Tooling Announcements: Firefox Profiler Deployment (February 24, 2026) (2026/02/24 12:15)
  The latest version of the Firefox Profiler is now live! Check out the full changelog below to see what’s changed. Highlights: [fatadel] Provide correct instructions on the home page for Firefox for Android (Fenix) (#5816) [Markus Stange] Migrate to esbuild (#5589) [Nazım Can Altınova] Make the source table non-optional in the gecko profile format (#5842) Other Changes: [Markus Stange] Move transparent fill check to a less expensive place. (#5826) [fatadel] Refine docs for ctrl key (#5831) [Nazım Can Altınova] Update the team members inside CONTRIBUTING.md (#5829) [Markus Stange] Make markers darker (#5839) [Markus Stange] Directly use photon-colors (#5821) [Markus Stange] Use data URLs for SVG files. (#5845) [Markus Stange] Tweak dark mode colors some more. (#5846) [fatadel] fix foreground color of the button on error page (#5849) [Nazım Can Altınova] Sync: l10n → main (February 24, 2026) (#5855) Big thanks to our amazing localizers for making this release

possible: de: Michael Köhler el: Jim Spentzos en-GB: Ian Neal es-CL: ravmn fr: Théo Chevalier fy-NL: Fjoerfoks a: Melo46 it: Francesco Lodolo [:flod] nl: Mark Heijl ru: Valery Ledovskoy sv-SE: Andreas Pettersson zh-TW: Pin-guang Chen Find out more about the Firefox Profiler on profiler.firefox.com! If you have any questions, join the discussion on our Matrix channel! 1 post - 1 participant Read full topic

- Niko Matsakis: What it means that Ubuntu is using Rust (2026/02/23 13:14)
  Righty-ho, I'm back from Rust Nation, and busily horrifying my teenage daughter with my (admittedly atrocious) attempts at doing an English accent1. It was a great trip with a lot of good conversations and some interesting observations. I am going to try to blog about some of them, starting with some thoughts spurred by Jon Seager's closing keynote, "Rust Adoption At Scale with Ubuntu". There are many chasms out there For some time now I've been debating with myself, has Rust "crossed the chasm"? If you're not familiar with that term, it comes from a book that gives a kind of "pop-sci" introduction to the Technology Adoption Life Cycle. The answer, of course, is it depends on who you ask. Within Amazon, where I have the closest view, the answer is that we are "most of the way across": Rust is squarely established as the right way to build at-scale data planes or resource-aware agents and it is increasingly seen as the right choice for low-level code in devices and robotics as well – but there remains a lingering perception that Rust is useful for "those fancy pants developers at S3" (or wherever) but a bit overkill for more average development3. On the other hand, within the realm of Safety Critical Software, as Pete LeVasseur wrote in a recent rust-lang blog post, Rust is still scrabbling for a foothold. There are a number of successful products but most of the industry is in a "wait and see" mode, letting the early adopters pave the path. "Crossing the chasm" means finding "reference customers" The big idea that I at least took away from reading Crossing the Chasm and other references on the technology adoption life cycle is the need for "reference customers". When you first start out with something new, you are looking for pioneers and early adopters that are drawn to new things: What an early adopter is buying [..] is some kind of change agent. By being the first to implement this change in the industry, the early adopters expect to get a jump on the competition. – from Crossing the Chasm But as your technology matures, you have to convince people with a lower and lower tolerance for risk: The early majority want to buy a productivity improvement for existing operations. They are looking to minimize discontinuity with the old ways. They want evolution, not revolution. – from Crossing the Chasm So what is most convincing to people to try something new? The answer is seeing that others like them have succeeded. You can see this at play in both the Amazon example and the Safety Critical Software example. Clearly seeing Rust used for network services doesn't mean it's ready to be used in your car's steering column4. And even within network services, seeing a group like S3 succeed with Rust may convince other groups building at-scale services to try Rust, but doesn't necessarily persuade a team to use Rust for their next CRUD service. And frankly, it shouldn't! They are likely to hit obstacles. Ubuntu is helping Rust "cross the (user-land linux) chasm" All of this was on my mind as I watched the keynote by Jon Seager, the VP of Engineering at Canonical, which is the company behind Ubuntu. Similar to Lars Bergstrom's epic keynote from year's past on Rust adoption within Google, Jon laid out a pitch for why Canonical is adopting Rust that was at once visionary and yet deeply practical. "Visionary and yet deeply practical" is pretty much the textbook description of what we need to cross from early adopters to early majority. We need folks who care first and foremost about delivering the right results, but are open to new ideas that might help them do that better; folks who can stand on both sides of the chasm at once. Jon described how Canonical focuses their own development on a small set of languages: Python, C/C++, and Go, and how they had recently brought in Rust and were using it as the language of choice for new foundational efforts, replacing C, C++, and (some uses of) Python. Ubuntu is building the bridge across the chasm Jon talked about how he sees it as part of Ubuntu's job to "pay it forward" by supporting the construction of memory-safe foundational utilities. Jon meant support both in terms of finances – Canonical is sponsoring the Trifecta Tech Foundation's to develop sudo-rs and ntpd-rs and

sponsoring the uutils org's work on coreutils – and in terms of reputation. Ubuntu can take on the risk of doing something new, prove that it works, and then let others benefit. Remember how the Crossing the Chasm book described early majority people? They are "looking to minimize discontinuity with the old ways". And what better way to do that than to have drop-in utilities that fit within their existing workflows. The challenge for Rust: listening to these new adopters With new adoption comes new perspectives. On Thursday night I was at dinner5 organized by Ernest Kissiedu6. Jon Seager was there along with some other Rust adopters from various industries, as were a few others from the Rust Foundation and the open-source project. Ernest asked them to give us their unvarnished takes on Rust. Jon made the provocative comment that we needed to revisit our policy around having a small standard library. He's not the first to say something like that, it's something we've been hearing for years and years – and I think he's right! Though I don't think the answer is just to ship a big standard library. In fact, it's kind of a perfect lead-in to (what I hope will be) my next blog post, which is about a project I call "battery packs"7. To grow, you have to change The broader point though is that shifting from targeting "pioneers" and "early adopters" to targeting "early majority" sometimes involves some uncomfortable changes: Transition between any two adoption segments is normally excruciatingly awkward because you must adopt new strategies just at the time you have become most comfortable with the old ones. [..] The situation can be further complicated if the high-tech company, fresh from its marketing success with visionaries, neglects to change its sales pitch. [..] The company may be saying "state-of-the-art" when the pragmatist wants to hear "industry standard". – Crossing the Chasm (emphasis mine) Not everybody will remember it, but in 2016 there was a proposal called the Rust Platform. The idea was to bring in some crates and bless them as a kind of "extended standard library". People hated it. After all, they said, why not just add dependencies to your Cargo.toml? It's easy enough. And to be honest, they were right – at least at the time. I think the Rust Platform is a good example of something that was a poor fit for early adopters, who want the newest thing and don't mind finding the best crates, but which could be a great fit for the Early Majority.8 Anyway, I'm not here to argue for one thing or another in this post, but more for the concept that we have to be open to adapting our learned wisdom to new circumstances. In the past, we were trying to bootstrap Rust into the industry's consciousness – and we have succeeded. The task before us now is different: we need to make Rust the best option not just in terms of "what it could be" but in terms of "what it actually is" – and sometimes those are in tension. Another challenge for Rust: turning adoption into investment Later in the dinner, the talk turned, as it often does, to money. Growing Rust adoption also comes with growing needs placed on the Rust project and its ecosystem. How can we connect the dots? This has been a big item on my mind, and I realize in writing this paragraph how many blog posts I have yet to write on the topic, but let me lay out a few interesting points that came up over this dinner and at other recent points. Investment can mean contribution, particularly for open-source orgs First, there are more ways to offer support than $$. For Canonical specifically, as they are an open-source organization through-and-through, what I would most want is to build stronger relationships between our organizations. With the Rust for Linux developers, early on Rust maintainers were prioritizing and fixing bugs on behalf of RfL devs, but more and more, RfL devs are fixing things themselves, with Rust maintainers serving as mentors. This is awesome! Money often comes before a company has adopted Rust, not after Second, there's an interesting trend about $$ that I've seen crop up in a few places. We often think of companies investing in the open-source dependencies that they rely upon. But there's an entirely different source of funding, and one that might be even easier to tap, which is to look at companies that are considering Rust but haven't adopted it yet. For those "would be" adopters, there are often individuals in the org who are trying to make the case for Rust adoption – these individuals are early adopters, people with a vision for how things could be, but they are trying to sell to their early majority company. And to do that, they often have a list of "table stakes" features that need to be supported; what's more, they often have access to some budget to make these things happen. This came up

when I was talking to Alexandru Radovici, the Foundation's Silver Member Directory, who said that many safety critical companies have money they'd like to spend to close various gaps in Rust, but they don't know how to spend it. Jon's investments in Trifecta Tech and the uutils org have the same character: he is looking to close the gaps that block Ubuntu from using Rust more. Conclusions…? Well, first of all, you should watch Jon's talk. "Brilliant", as the Brits have it. But my other big thought is that this is a crucial time for Rust. We are clearly transitioning in a number of areas from visionaries and early adopters towards that pragmatic majority, and we need to be mindful that doing so may require us to change some of the way that we've always done things. I liked this paragraph from Crossing the Chasm: To market successfully to pragmatists, one does not have to be one – just understand their values and work to serve them. To look more closely into these values, if the goal of visionaries is to take a quantum leap forward, the goal of pragmatists is to make a percentage improvement–incremental, measurable, predictable progress. [..] To market to pragmatists, you must be patient. You need to be conversant with the issues that dominate their particular business. You need to show up at the industry-specific conferences and trade shows they attend. Re-reading Crossing the Chasm as part of writing this blog post has really helped me square where Rust is – for the most part, I think we are still crossing the chasm, but we are well on our way. I think what we see is a consistent trend now where we have Rust champions who fit the "visionary" profile of early adopters successfully advocating for Rust within companies that fit the pragmatist, early majority profile. Open source can be a great enabler to cross the chasm… It strikes me that open-source is just an amazing platform for doing this kind of marketing. Unlike a company, we don't have to do everything ourselves. We have to leverage the fact that open source helps those who help themselves – find those visionary folks in industries that could really benefit from Rust, bring them into the Rust orbit, and then (most important!) support and empower them to adapt Rust to their needs. …but only if we don't get too "middle school" about it This last part may sound obvious, but it's harder than it sounds. When you're embedded in open source, it seems like a friendly place where everyone is welcome. But the reality is that it can be a place full of cliques and "oral traditions" that "everybody knows"9. People coming with an idea can get shutdown for using the wrong word. They can readily mistake the, um, "impassioned" comments from a random contributor (or perhaps just a troll…) for the official word from project leadership. It only takes one rude response to turn somebody away. What Rust needs most is empathy So what will ultimately help Rust the most to succeed? Empathy in Open Source. Let's get out there, find out where Rust can help people, and make it happen. Exciting times! I am famously bad at accents. My best attempt at posh British sounds more like Apu from the Simpsons. I really wish I could pull off a convincing Greek accent, but sadly no. ↩ Another of my pearls of wisdom is "there is nothing more permanent than temporary code". I used to say that back at the startup I worked at after college, but years of experience have only proven it more and more true. ↩ Russel Cohen and Jess Izen gave a great talk at last year's RustConf about what our team is doing to help teams decide if Rust is viable for them. But since then another thing having a big impact is AI, which is bringing previously unthinkable projects, like rewriting older systems, within reach. ↩ I have no idea if there is code in a car's steering column, for the record. I assume so by now? For power steering or some shit? ↩ Or am I supposed to call it "tea"? Or maybe "supper"? I can't get a handle on British mealtimes. ↩ Ernest is such a joy to be around. He's quiet, but he's got a lot of insights if you can convince him to share them. If you get the chance to meet him, take it! If you live in London, go to the London Rust meetup! Find Ernest and introduce yourself. Tell him Niko sent you and that you are supposed to say how great he is and how you want to learn from the wisdom he's accrued over the years. Then watch him blush. What a doll. ↩ If you can't wait, you can read some Zulip discussion here. ↩ The Battery Packs proposal I want to talk about is similar in some ways to the Rust Platform, but decentralized and generally better in my opinion– but I get ahead of myself! ↩ Betteridge's Law of Headlines has it that "Any headline that ends in a question mark can be answered by the word no". Well, Niko's law of open-source2 is that "nobody actually knows

anything that ’everybody’ knows”. ↩

- Ludovic Hirlimann: Are mozilla's fork any good? (2026/02/23 11:39)
  To answer that question, we first need to understand how complex, writing or maintaining a web browser is. A "modern" web browser is :a network stack,and html+[1] parser, and image+[2] decoder,a javascript[3] interpreter compiler,a User's interface,integration with the underlying OS[4],And all the other things I'm currently forgetting. Of course, all the above point are interacting with one another in different ways. In order for "the web" to work, standards are developed and then implemented in the different browsers, rendering engines.In order to "make" the browser, you need engineers to write and maintain the code, which is probably around 30 Million lines of code[5] for Firefox. Once the code is written, it needs to be compiled [6] and tested [6]. This requires machines that run the operating system the browser ships to (As of this day, mozilla officially ships on Linux, Microslop Windows and MacOS X - community builds for *BSD do exists and are maintained). You need engineers to maintain the compile (build) infrastructure. Once the engineers that are responsible for the releases [7] have decided what codes and features were mature enough, they start assembling the bits of code and like the engineers, build, test and send the results to the people using said web browser. When I was employed at Mozilla (the company that makes Firefox) around 900+ engineers were tasked with the above and a few more were working on research and development. These engineers are working 5 days a week, 8 hours per day, that's 1872000 hours of engineering brain power spent every year (It's actually less because I have not taken vacations into account) on making Firefox versions. On top of that, you need to add the cost of building and running the test before a new version reaches the end user. The current browsing landscape looks dark, there are currently 3 choices for rendering engines, KHTML based browsers, blink based ones and gecko based ones. 90+% of the market is dominated by KHTML/blink based browsers. Blink is a fork of KHTML. This leads to less standard work, if the major engine implements a feature and others need to play catchup to stay relevant, this has happened in the 2000s with IE dominating the browser landscape[8], making it difficult to use macOS 9 or X (I'm not even mentioning Linux here :)). This also leads to most web developers using Chrome and once in a while testing with Firefox or even Safari. But if there's a little glitch, they can still ship because of market shares.Firefox was started back in 1998, when embedding software was not really a thing with all the platform that were to be supported. Firefox is very hard to embed (eg use as a softwrae library and add stuff on top). I know that for a fact because both Camino and Thunderbird are embeding gecko. In the last few years, Mozilla has been itching the people I connect to, who are very privacy focus and do not see with a good eye what Mozilla does with Firefox. I believe that Mozilla does this in order to stay relevant to normal users. It needs to stay relevant for at least two things :Keep the web standards open, so anyone can implement a web browser / web services.to have enough traffic to be able to pay all the engineers working on gecko. Now that, I've explained a few important things, let's answer the question "Are mozilla's fork any good?"I am biased as I've worked for the company before. But how can a few people, even if they are good and have plenty of free time, be able to cope with what maintaining a fork requires :following security patches and porting said patches.following development and maintain their branch with changes coming all over the placehow do they test?Are they able to pull something like https://www.youtube.com/watch?app=desktop&v=YQEq5s4SRxY If you are comfortable with that, then using a fork because Mozilla is pushing stuff you don't want is probably doable. If not, you can always kill those features you don't like using some `about:config` magic. Now, I've set a tone above that foresees a dark future for open web technologies. What Can you do to keep the web open and with some privacy focus?Keep using Mozilla NightlyGive servo a try [1] HTML is interpreted code, that's why it needs to be parsed and then rendered.[2] In order to draw and image or a photo on a screen, you need to be able to encode it or decode it. Many file formats are available.[3] Is a computer language that transforms HTML into something that can interact with the person using the web browser. See

https://developer.mozilla.org/en-US/docs/Glossary/JavaScript[4] Operating systems need to the very least know which program to open files with. The OS landscape has changed a lot over the last 25 years. These days you need to support 3 major OS, while in the 2000s you had more systems, IRIX for example. You still have some portions of the Mozilla code base that support these long dead systems.[5]https://math.answers.com/math-and-arithmetic/How_many_lines_of_code_in_mozillafirefox [6] Testing implies, testing the code and also having engineers or users using the unfinished product to see that it doesn't regress. Testing Mozilla, is explained at https://ehsanakhgari.org/wp-content/uploads/talks/test-mozilla/[7] Read a release equals a version. Version 1.5 is a release, as is version 3.0.1.[8] https://en.wikipedia.org/wiki/Browser_wars

- The Rust Programming Language Blog: Rust debugging survey 2026 (2026/02/23 00:00)
  We're launching a Rust Debugging Survey. Various issues with debugging Rust code are often mentioned as one of the biggest challenges that annoy Rust developers. While it is definitely possible to debug Rust code today, there are situations where it does not work well enough, and the quality of debugging support also varies a lot across different debuggers and operating systems. In order for Rust to have truly stellar debugging support, it should ideally: Support (several versions!) of different debuggers (such as GDB, LLDB or CDB) across multiple operating systems. Implement debugger visualizers that are able to produce quality presentation of most Rust types. Provide first-class support for debugging async code. Allow evaluating Rust expressions in the debugger. Rust is not quite there yet, and it will take a lot of work to reach that level of debugger support. Furthermore, it is also challenging to ensure that debugging Rust code keeps working well, across newly released debugger versions, changes to internal representation of Rust data structures in the standard library and other things that can break the debugging experience. We already have some plans to start improving debugging support in Rust, but it would also be useful to understand the current debugging struggles of Rust developers. That is why we have prepared the Rust Debugging Survey, which should help us find specific challenges with debugging Rust code. You can fill out the survey here. Filling the survey should take you approximately 5 minutes, and the survey is fully anonymous. We will accept submissions until Friday, March 13th, 2026. After the survey ends, we will evaluate the results and post key insights on this blog. We would like to thank Sam Kellam (@hashcatHitman) who did a lot of great work to prepare this survey. We invite you to fill the survey, as your responses will help us improve the Rust debugging experience. Thank you!
- Mozilla Privacy Blog: Behind the Velvet Rope: The AI Divide on Display at the India AI Impact Summit 2026 (2026/02/20 20:47)
  TLDR: No one could agree what ‘sovereignty’ means, but (almost) everyone agreed that AI cannot be controlled by a few dominant companies. This past week, droves of AI experts and enthusiasts descended on New Delhi, bringing their own agendas, priorities, and roles in the debate to the table. I scored high for my ability to move between cars, rickshaws and foot for transport (mostly thanks to colleagues), but low for being prepared with snacks. So, based on my tightly packed agenda combined with high hunger levels, here’s my read out: The same script, different reactions As with any global summit, the host government made the most of having the eyes of the world and deep pockets of AI investors in town. While some press were critical of India seeking deals and investments, it wasn’t notable – or outside of the norm. What should be notable, and indeed were reflected in the voluntary agreements, were the key themes that drove conversations, including democratisation of AI, access to resources, and the vital role of open source to drive the benefits of AI. These topics were prominent in the Summit sessions and side events throughout the week. In the name of innovation, regulation has become a faux pas The EU has become a magnet for criticism given its recent attempts to regulate AI. I’m not going to debate this here, but it’s clear that the EU AI Act (AIA) is being deployed and PRed quite expertly as a cautionary tale. While healthy debate around regulation is absolutely critical, much of the public commentary surrounding the AIA (and not just at

this Summit) has been factually incorrect. Interrogate this reality by all means — we live in complex times — but it's hard not to see invalid criticisms as a strategic PR effort by those who philosophically (and financially) opposed governance. There is certainly plenty to question in the AIA, but the reality is much more nuanced than critics suggest. What's more likely to kill a start up: the cost of compliance, or the concentration of market power in the hands of a few dominant players? It's true that regulation can absolutely create challenges. However, it is also worth looking at whether the greater obstacle is the control a small number of tech companies hold. A buy-out as an exit is great for many start-ups, but if that is now the most hopeful option, it raises important questions about the long-term health and competitiveness of the larger tech ecosystem. A note of optimism: developing global frameworks on AI may still seem like a pipe dream in today's macro political climate, but ideas around like-minded powers working together and building trust makes me think that alignment beyond pure voluntary principles may be something we see grow. Frequent references to the Hiroshima Process as a middle ground were notable. AI eats the world There were pervasive assumptions that bigger — and then bigger still — is the inevitable direction of AI deployment, with hyperscale seen as the only viable path forward, in terms of inputs needed. However, the magnitude of what's required to fuel the current LLM-focused market structure met a global majority-focused reality: hyperscaling isn't sustainable. There were two primary camps at the Summit — the haves and the rest of us — and while the Summit brought them together, the gulf between them continues to grow. Open source has to win At the first AI Safety Summit in the UK, the concept of open source AI was vilified as a security risk. At the France AI Action Summit, the consensus began to shift meaningfully. At the India AI Impact Summit, we saw undeniable recognition of the vital role that open source plays in our collective AI future. With proprietary systems, winning means owning. With open source approaches, winning means we're not just renting AI from a few companies and countries: we're working collectively to build, share, secure and inspect AI systems in the name of economic growth and the public interest. Before the Paris Summit, this was a difficult vision to push for, but after New Delhi, it's clear that open source is on the right side of history. Now, it's time for governments to build out their own strategies to promote and procure this approach. Consolidation ≠ Competition Global South discussions made one message clear: dependency orientated partnerships are not true partnerships and they're not a long term bet. Many countries are becoming more vocal that they want autonomy of their data and choice in their suppliers to lessen harmful impacts on citizens, and increase their impact to responsibly govern. That is not today's reality. I was, however, encouraged to find that attendees were far less starry-eyed over big tech than at previous Summits. The consensus agreed that it met no one's definition of sovereignty for a select few companies to own and control AI. Despite agreement amongst the majority, addressing market concentration remained an elephant in the room. The narrative deployed against regulation became a blanket mantra, applied to anything from AI governance to competition action. However, it fails to address the fact that the AI market is already skewed toward a small number of powerful companies and traditional competition rules that act only after problems arise (and often through long legal processes) are not enough to keep up with fast-paced digital industries. Some participants were downbeat and questioned if it was too late. The challenge is in demonstrating that it isn't. There is no single approach. But we know that concentration can be countered with a mix of technical and legal interventions. Options can be sweeping, or lighter touch and surgical in their focus. We are currently seeing is a wave of countries pass, draft, debate and consider new ex ante rules, providing learnings, data and inspiration. It's important that we watch this space. Whose safety are we talking about exactly? The India AI Impact Summit has been criticised for letting safety discussions fall off the radar. That's not necessarily true. Instead of focusing on the view that AI is a cause for human annihilation, discussions focused on impacts that we can evidence now: on language, culture, bias, online safety, inclusion, and jobs. Less headline-grabbing, less killer robots, far more human. The path forward It's difficult to know if these Summits will continue in the long term. There is a lot of fuss, expense,

marketing, diplomacy, traffic and word salads involved. However, the opportunity to convene world leaders, businesses, builders, engineers, civil society and academics in one place, for what we are constantly reminded is a transformational technology, feels needed. Tracking progress on voluntary commitments over time might be illustrative. And while many of the top sessions are reserved for the few, witnessing the diverse debates this past week gives me hope that there is an opportunity for the greater voice to shape AI to be open, competitive and built for more than just the bottom line. The post Behind the Velvet Rope: The AI Divide on Display at the India AI Impact Summit 2026 appeared first on Open Policy & Advocacy.

- The Rust Programming Language Blog: Rust participates in Google Summer of Code 2026 (2026/02/19 00:00)
  We are happy to announce that the Rust Project will again be participating in Google Summer of Code (GSoC) 2026, same as in the previous two years. 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 months long 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. We would also like to bring your attention to our GSoC AI policy. 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 16, 2026. From that date you can submit project proposals into the GSoC dashboard. The project proposal application period ends on March 31, 2026 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. Our GSoC contributors were quite successful in the past two years (2024, 2025), so we are excited what this year's GSoC will bring! 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.
- Tarek Ziadé: Running SmolLM-135M in rustnn with flexible inputs (2026/02/18 00:00)
  WebNN is emerging as a portable, browser-friendly inference API. But LLMs hit a hard wall: dynamic inputs. Autoregressive transformers fundamentally mutate state at runtime. KV cache tensors evolve at every step, sequence lengths vary with prompts, and shape expressions flow through operators like Shape, Gather, Concat, Reshape, and Expand. Today, this does not map cleanly to WebNN’s static-graph constraints. At step 1, KV cache length is 1. At step 512, KV cache length is 512. That is not a static graph. Why this matters If this is not solved, WebNN stays limited to fixed-shape demos for many LLM use cases. For real product workloads, people need variable prompt lengths, efficient token-by-token

decode, and stable latency as context grows. Without that, local browser LLM UX degrades quickly and teams default back to heavier alternatives. Dynamic inputs in practice This problem is now well documented in the WebNN WG in Issue #883: Support flexible input sizes. The issue captures exactly what we see in practice: Vision models with runtime-determined resolution Speech/decoder models where KV cache grows by one token per step LLMs with arbitrary prompt lengths and dynamic cache shapes In other words: modern inference workloads. The ONNX Runtime workaround (and why it hurts) ONNX Runtime WebNN has had to work around this limitation by routing dynamic-shape parts away from WebNN and into WASM execution paths. It works, but performance is terrible for autoregressive generation because you bounce between fast backend code and slow fallback code in the hottest part of the loop. This architecture can make demos pass, but it creates significant performance penalties in real autoregressive workloads. In preliminary runs, keeping decode on one backend avoids the repeated fallback round-trips that dominate token latency. So instead of accepting that, we decided to push WebNN support further. I started this in Malta I started prototyping this while on vacation in Malta. What began as a small experiment quickly turned into deep changes across three repositories: converter internals, runtime shape validation, and KV-cache plumbing. The work happened across: webnn-graph: ONNX lowering and dynamic-input metadata support rustnn (tarek-flexible-input): dynamic dimensions in graph/runtime + checked execution pywebnn: Python demos and loading-from-Hub workflows I also made sure to surface rustnn through Python (pywebnn) very early. Most ML engineers live in Python land, and I wanted this work to be reachable by that ecosystem immediately: easier model validation, easier parity checks against transformers, and faster feedback from people who already ship models every day. What changed in practice The key was to support bounded dynamic dimensions end to end. Why bounded and not fully dynamic? Because many backends still need strong compile-time guarantees for validation, allocation, and kernel planning. Fully unbounded shapes are hard to optimize and hard to validate safely. Bounded dynamic dimensions are the practical compromise: keep symbolic runtime flexibility, but define a maximum range so memory and execution planning remain deterministic. This allows the full autoregressive decode loop to stay inside the WebNN backend, without bouncing into slower fallback paths. This is also better than common alternatives: Padding everything to worst-case shapes wastes memory and compute Re-exporting one graph per shape explodes complexity Falling back dynamic parts to WASM in hot decode loops kills throughput For example, you can bound a sequence dimension to 2048 tokens: large enough for real prompts, still finite for backend planning and allocation. In rustnn, tensor dimensions can now be static values or dynamic descriptors with a name and a max size, then checked at runtime: { "inputs": { "x": { "dataType": "float32", "shape": [{ "name": "batch", "maxSize": 16 }, 128] } } } In webnn-graph, ONNX conversion can preserve unresolved input dynamics while still lowering shape-driving expressions needed by WebNN. That lets us keep flexibility where possible while still emitting valid WebNN graphs. SmolLM-135M converted and running With flexible inputs supported end to end, SmolLM-135M converts cleanly: no shape rewriting hacks, no per-length exports, no WASM fallback in the decode loop. The artifacts are published here: tarekziade/SmolLM-135M-webnn Then I built a Python demo in pywebnn/examples/smollm_from_hub.py that: downloads model.webnn, model.weights, and manifest from the Hub downloads tokenizer.json runs token-by-token generation with dynamic KV cache growth optionally compares output against transformers A few extracts from that demo: The demo defaults to the Hub-hosted WebNN artifacts: DEFAULT_MODEL_ID = "tarekziade/SmolLM-135M-webnn" model_files = resolve_model_files(args.model_id, force=args.force_download) graph = webnn.MLGraph.load(model_files["graph"], manifest_path=model_files["manifest"], weights_path=model_files["weights"],) past_key_values holds the growing KV-cache tensors returned by the previous step. The decode loop feeds them back on every token: def run_step(token_id: int, position: int) -> np.ndarray: inputs = { "input_ids": np.array([[token_id]], dtype=np.int64), "position_ids": np.array([[position]], dtype=np.int64), "attention_mask": np.ones((1, position

+ 1), dtype=np.int64), **past_key_values, } outputs = context.compute(graph, inputs) ... The demo can also run a transformers baseline and fail fast on divergence: if args.compare_transformers: hf_generated, hf_text, hf_prompt_ids = run_transformers_baseline(...) ... if generated_text != hf_text: print("[ERROR] WebNN and transformers generated different text output") sys.exit(1) Correctness checks against transformers are critical. Performance improvements mean nothing if generation diverges. Lessons learned Fully unbounded dynamic shapes are rarely necessary for practical decode loops Bounded flexibility captures most real workloads while keeping backends sane Python exposure (pywebnn) accelerates model validation and ecosystem feedback What is next Flexible inputs will likely be important if WebNN is to support real LLM workloads. Static graphs alone are not enough for modern inference. Bounded flexibility is the pragmatic bridge. And while this work pushes WebNN forward, we are also giving a lot of love to the TensorRT backend these days, because high-performance local inference matters just as much as API design. Links rustnn docs: https://rustnn.github.io/rustnn/ pywebnn docs: https://rustnn.github.io/pywebnn/ rustnn WPT conformance dashboard: https://rustnn.github.io/rustnnpt/

- Firefox Tooling Announcements: New Deploy of PerfCompare (Feb 17th) (2026/02/17 22:57)
  The latest version of PerfCompare is now live! Check out the changelog below to see the updates: [gmierz] - Add a link to Bugzilla in the MWU warning banner #981 [kala-moz] - Add coverage for test version dropdown #983 - Bug 2004156 - Add the mozharness framework as an available option #986. - Bug 2009257: trigger loader to rerun when setting the test version and framework in url #988 - Set mann-whitney-u as the default test version with replicates enabled by default #999 Our new contributor [moijes12] - Bug 1919317: Fix the tooltip for Total Runs column #992 - Bug-2003016: Constrain the clickable area of the Home button #996 Thank you for the contributions! Bugs or feature requests can be filed on Bugzilla. The team can also be found on the #perfcompare channel on Element. 1 post - 1 participant Read full topic
- Niko Matsakis: Sharing in Dada (2026/02/14 11:49)
  OK, let’s talk about sharing. This is the first of Dada blog posts where things start to diverge from Rust in a deep way and I think the first where we start to see some real advantages to the Dada way of doing things (and some of the tradeoffs I made to achieve those advantages). We are shooting for a GC-like experience without GC Let’s start with the goal: earlier, I said that Dada was like “Rust where you never have to type as_ref”. But what I really meant is that I want a GC-like experience–without the GC. We are shooting for a “composable” experience I also often use the word “composable” to describe the Dada experience I am shooting for. Composable means that you can take different things and put them together to achieve something new. Obviously Rust has many composable patterns – the Iterator APIs, for example. But what I have found is that Rust code is often very brittle: there are many choices when it comes to how you declare your data structures and the choices you make will inform how those data structures can be consumed. Running example: Character Defining the Character type Let’s create a type that we can use as a running example throughout the post: Character. In Rust, we might define a Character like so: #[derive(Default)] struct Character { name: String, class: String, hp: u32, } Creating and Arc’ing the Character Now, suppose that, for whatever reason, we are going to build up a character programmatically: let mut ch = Character::default(); ch.name.push_str("Ferris"); ch.class.push_str("Rustacean"); ch.hp = 44; So far, so good. Now suppose I want to share that same Character struct so it can be referenced from a lot of places without deep copying. To do that, I am going to put it in an Arc: let mut ch = Character::default(); ch.name.push_str("Ferris"); // ... let ch1 = Arc::new(ch); let ch2 = ch1.clone(); OK, cool! Now I have a Character that is readily sharable. That’s great. Rust is composable here, which is cool, we like that Side note but this is an example of where Rust is composable: we defined Character once in a fully-owned way and we were able to use it mutably (to build it up imperatively over time) and then able to “freeze” it and get a read-only, shared copy of Character. This gives us the advantages of an imperative

programming language (easy data construction and manipulation) and the advantages of a functional language (immutability prevents bugs when things are referenced from many disjoint places). Nice! Creating and Arc'ing the Character Now, suppose that I have some other code, written independently, that just needs to store the character's name. That code winds up copying the name into a lot of different places. So, just like we used Arc to let us cheaply reference a single character from multiple places, it uses Arc so it can cheaply reference the character's name from multiple places: struct CharacterSheetWidget { // Use `Arc<String>` and not `String` because // we wind up copying this into name different // places and we don't want to deep clone // the string each time. name: Arc<String>, // ... assume more fields here ... } OK. Now comes the rub. I want to create a character-sheet widget from our shared character: fn create_character_sheet_widget(ch: Arc<Character>) -> CharacterSheetWidget { CharacterSheetWidget { // FIXME: Huh, how do I bridge this gap? // I guess I have to do this. name: Arc::new(ch.name.clone()), // ... assume more fields here ... } } Shoot, that's frustrating! What I would like to do is to write name: ch.name.clone() or something similar (actually I'd probably like to just write ch.name, but anyhow) and get back an Arc<String>. But I can't do that. Instead, I have to deeply clone the string and allocate a new Arc. Of course any subsequent clones will be cheap. But it's not great. Rust often gives rise to these kind of "impedance mismatches" I often find patterns like this arise in Rust: there's a bit of an "impedance mismatch" between one piece of code and another. The solution varies, but it's generally something like clone some data – it's not so big anyway, screw it (that's what happened here). refactor one piece of code – e.g., modify the Character class to store an Arc<String>. Of course, that has ripple effects, e.g., we can no longer write ch.name.push_str(...) anymore, but have to use Arc::get_mut or something. invoke some annoying helper – e.g., write opt.as_ref() to convert from an &Option<String> to a Option<&String> or write a &**r to convert from a &Arc<String> to a &str. The goal with Dada is that we don't have that kind of thing. Sharing is how Dada copies So let's walk through how that same Character example would play out in Dada. We'll start by defining the Character class: class Character(name: String, klass: String, # Oh dang, the perils of a class keyword! hp: u32,) Just as in Rust, we can create the character and then modify it afterwards: class Character(name: String, klass: String, hp: u32) let ch: given Character = Character("", "", 22) # ----- remember, the "given" permission # means that `ch` is fully owned ch.name!.push("Tzara") ch.klass!.push("Dadaist") # - and the `!` signals mutation The .share operator creates a shared object Cool. Now, I want to share the character so it can be referenced from many places. In Rust, we created an Arc, but in Dada, sharing is "built-in". We use the .share operator, which will convert the given Character (i.e., fully owned character) into a shared Character: class Character(name: String, klass: String, hp: u32) let ch = Character("", "", 22) ch!.push("Tzara") ch!.push("Dadaist") let ch1: shared Character = ch.share # ------ ----- # The `share` operator consumes `ch` # and returns the same object, but now # with *shared* permissions. shared objects can be copied freely Now that we have a shared character, we can copy it around: class Character(name: String, klass: String, hp: u32) # Create a shared character to start let ch1 = Character("Tzara", "Dadaist", 22).share # ----- # Create another shared character let ch2 = ch1 Sharing propagates from owner to field When you have a shared object and you access its field, what you get back is a shared (shallow) copy of the field: class Character(...) # Create a `shared Character` let ch: shared Character = Character("Tristan Tzara", "Dadaist", 22).share # ------ ----- # Extracting the `name` field gives a `shared String` let name: shared String = ch1.name # ------ Propagation using a Vec To drill home how cool and convenient this is, imagine that I have a Vec[String] that I share with .share: let v: shared Vec[String] = ["Hello", "Dada"].share and then I share it with v.share. What I get back is a shared Vec[String]. And when I access the elements of that, I get back a shared String: let v = ["Hello", "Dada"].share let s: shared String = v[0] This is as if one could take a Arc<Vec<String>> in Rust and get out a Arc<String>. How sharing is implemented So how is sharing implemented? The answer lies in a not-entirely-obvious memory layout. To see how it works, let's walk how a Character would be laid out in

memory: # Character type we saw earlier. class Character(name: String, klass: String, hp: u32) # String type would be something like this. class String { buffer: Pointer[char] initialized: usize length: usize } Here Pointer is a built-in type that is the basis for Dada's unsafe code system.1 Layout of a given Character in memory Now imagine we have a Character like this: let ch = Character("Duchamp", "Dadaist", 22) The character ch would be laid out in memory something like this (focusing just on the name field): [Stack frame] [Heap] ch: Character { _flag: 1 name: String { _flag: 1 { _ref_count: 1 buffer: ⟶'D' initialized: 7 ... capacity: 8 'p' } } klass: ... hp: 22 } Let's talk this through. First, every object is laid out flat in memory, just like you would see in Rust. So the fields of ch are stored on the stack, and the name field is laid out flat within that. Each object that owns other objects begins with a hidden field, _flag. This field indicates whether the object is shared or not (in the future we'll add more values to account for other permissions). If the field is 1, the object is not shared. If it is 2, then it is shared. Heap-allocated objects (i.e., using Pointer[]) begin with a ref-count before the actual data (actually this is at the offset of -4). In this case we have a Pointer[char] so the actual data that follows are just simple characters. Layout of a shared Character in memory If I were to instead create a shared character: let ch1 = Character("Duchamp", "Dadaist", 22).share # ----- The memory layout would be the same, but the flag field on the character is now 2: [Stack frame] [Heap] ch: Character { _flag: 2 □ (This is 2 now!) name: String { _flag: 1 { _ref_count: 1 buffer: ⟶'D' initialized: 7 ... capacity: 8 'p' } } klass: ... hp: 22 } Copying a shared Character Now imagine that we created two copies of the same shared character: let ch1 = Character("Duchamp", "Dadaist", 22).share let ch2 = ch1 What happens is that we will copy all the fields of _ch1 and then, because _flag is 2, we will increment the ref-counts for the heap-allocated data within: [Stack frame] [Heap] ch1: Character { _flag: 2 name: String { _flag: 1 { _ref_count: 2 buffer: ⟶'D' □ initialized: 7 │ ... (This is capacity: 8 │ 'p' } 2 now!) } │ class: ... │ hp: 22 │ } │ │ ch2: Character { │ _flag: 2 │ name: String { │ _flag: 1 │ buffer: ⟶ initialized: 7 capacity: 8 } class: ... hp: 22 } Copying out the name field Now imagine we were to copy out the name field, instead of the entire character: let ch1 = Character("Duchamp", "Dadaist", 22).share let name = ch1.name ...what happens is that: traversing ch1, we observe that the _flag field is 2 and therefore ch1 is shared we copy out the String fields from name. Because the character is shared: we modify the _flag field on the new string to 2 we increment the ref-count for any heap values The result is that you get: [Stack frame] [Heap] ch1: Character { _flag: 2 name: String { _flag: 1 { _ref_count: 2 buffer: ⟶'D' initialized: 7 │ ... capacity: 8 │ 'p' } } │ class: ... │ hp: 22 │ } │ │ name: String { │ _flag: 2 │ buffer: ⟶ initialized: 7 capacity: 8 } "Sharing propagation" is one example of permission propagation This post showed how shared values in Dada work and showed how the shared permission propagates when you access a field. Permissions are how Dada manages object lifetimes. We've seen two so far the given permission indicates a uniquely owned value (T, in Rust-speak); the shared permission indicates a copyable value (Arc<T> is the closest Rust equivalent). In future posts we'll see the ref and mut permissions, which roughly correspond to & and &mut, and talk out how the whole thing fits together. Dada is more than a pretty face This is the first post where we started to see a bit more of Dada's character. Reading over the previous few posts, you could be forgiven for thinking Dada was just a cute syntax atop familiar Rust semantics. But as you can see from how shared works, Dada is quite a bit more than that. I like to think of Dada as "opinionated Rust" in some sense. Unlike Rust, it imposes some standards on how things are done. For example, every object (at least every object with a heap-allocated field) has a _flag field. And every heap allocation has a ref-count. These conventions come at some modest runtime cost. My rule is that basic operations are allowed to do "shallow" operations, e.g., toggling the _flag or adjusting the ref-counts on every field. But they cannot do "deep" operations that require traversing heap structures. In exchange for adopting conventions and paying that cost, you get "composability", by which I mean that permissions in Dada (like shared) flow much more naturally, and types that are semantically equivalent (i.e., you can do the same things with them) generally have the same layout in memory. Remember that I have not

implemented all this, I am drawing on my memory and notes from my notebooks. I reserve the right to change any and everything as I go about implementing. ↩

- Thunderbird Blog: Mobile Progress Report: February 2026 (2026/02/13 15:19)
  The first Mobile Progress Report of 2026 provides a high-level overview to our mobile plans and priorities for the coming year. Android Our primary focus this year revolves around a better user experience and includes a major push to improve quality. We want to make the app stable, reduce our bugs, and speed up our development process. To do this, we have to make some big changes and improvements to the app's basic structure and database. We're moving toward modern Android standards, which includes using technologies like Compose and creating a single, consistent design system for our User Interface. But we don't want a year of just code fixes; we know we need to add features, especially around messaging & notifications. We're making sure we deliver features and improve the user experience along the way. It's a tricky balance between making the app better for users and overhauling the inner workings. We think these changes are worth the investment because they'll lead to a better app, and ultimately, a better app for everyone. So the focus is better quality and simplifying the code to make us quicker. Thunderbird has a new product – Thunderbird Pro – and as it comes more online, we plan to connect the Android app to it. Here are our priorities for the year. P1 is the top focus: P1 Get the Android app into a state that's easier to maintain Improve the database structure Message List & View improvements Unified Account Unified Design System (for both iOS and Android) P2 HTML signatures Looking into JMAP support P3 Looking into Exchange support Calendar exploration iOS The main thing we're trying to do for iOS this year is successfully launch Version 1 of our app. That sounds simple, but it involves building a lot of complicated, low-level foundational things. This quarter, we're concentrating on finishing up the IMAP and SMTP pieces, getting our design system established, and building the basic UI so we can start using these pieces. After that, we'll shift to implementing OAuth. This will stop users from having to use confusing processes, like creating an app token, and let them sign in easily through the standard account import process with a simple User Interface. Once we have IMAP and OAuth ready, we'll have the absolute bare minimum for a mail app, allowing users to send and receive email. But there are other features you'd expect in a mail app, like mailboxes, signatures, rich text viewing, attachment handling, and the compose experience. We've already made great progress on the underlying functionality, and we have a clear vision of what needs to be implemented to make this successful. Our key priorities for iOS are: P1 Account creation flow IMAP support Full email writing and reading experience P2 JMAP support HTML signatures It's exciting to see the momentum that the iOS app is gaining and to get a clearer picture of what we need to do for the Android app to simplify things. We are getting farther on fewer, more targeted goals. I look forward to communicating with you over the next few months and share the progress that we are making. — Jon Bott (he/him) Manager, Mobile Apps The post Mobile Progress Report: February 2026 appeared first on The Thunderbird Blog.
- The Mozilla Blog: Heading to India AI Impact Summit, Mozilla leaders call for investment in open source AI as a path to sovereignty (2026/02/13 13:00)
  Mozilla is headed to New Delhi, India for the India AI Impact Summit 2026 next week with a message: Open Source is the path to both economic and digital sovereignty. Participating in dozens of events across the weeklong global forum, Mozilla leaders will make the case that a different kind of AI future is possible, and that global action is urgently needed to build a global AI ecosystem firmly grounded in the public interest. "We're at a crossroads for AI, where the world can continue to rent from a few big global corporations, or can take back control," said Mark Surman, president of Mozilla. "To build national resilience and lower costs for their domestic stakeholders, countries should leave India ready to meaningfully invest in open source AI as a transformational solution." As part of the India AI Impact Summit 2026, Mozilla is curating three

official events that showcase its work to help build a more decentralized AI ecosystem. These include panels on the state of competition in AI, how open source AI can operationalize digital sovereignty, and the launch of a new convenings program aimed at Bollywood artists and filmmakers to explore the future of creativity in the age of AI. Mozilla will also host a community party for open source AI developers and founders. At the Summit, Mozilla will speak to growing concerns about consolidation in the AI landscape, with just a few large global corporations essentially renting access to AI to the rest of the world. As more powerful AI systems are built and controlled behind closed platforms, users are left with little visibility into how these systems work and almost no ability to shape or govern them. This concentration of control means a small number of companies can decide who gets access to advanced AI, on what terms, and at what cost, with far-reaching consequences for innovation, public institutions, and digital sovereignty. Mozilla believes this growing concentration of AI control threatens innovation, fair competition and public accountability. To counter this, Mozilla is investing in people, products, and organizations working to build a more open and human-centered AI ecosystem. Open source AI is a key part of this effort, providing practical, real-world infrastructure that allows systems to be inspected, improved locally, and governed in the public interest. In partnership with G5A, Mozilla Foundation will also launch “The Origin of Thought” at the Summit — a new initiative to explore the intersections of culture and AI. The program will convene Bollywood artists, filmmakers, technologists, and cultural practitioners to help shape a nuanced, multi-faceted understanding of AI’s impact on our lives — not just as a tool but as a potential cultural force. The first taster session, held at The Oddbird Theatre in New Delhi, will feature filmmakers Nikkhil Advani and Shakun Batra. “Creativity has always been how people make sense of change, long before policy frameworks or product roadmaps catch up. As AI reshapes how culture is made, shared, and remembered, we need spaces that slow the conversation down enough to ask what we’re protecting and why,” said Nabiha Syed, executive director of Mozilla Foundation. “The Origin of Thought brings artists, technologists, and decision-makers together to look beyond efficiency and novelty, and toward the human stakes of this moment. This work reflects our belief that imagination is a critical safeguard, not a luxury. When we center creative voices, we make it possible to build technologies that expand opportunity, dignity, and livelihoods.” Through its participation at the India AI Impact Summit 2026, Mozilla will reaffirm its commitment to building an AI future that is open, accountable, and shaped by the societies it is meant to serve, and to ensuring that open source AI remains a central pillar of global AI governance and innovation. “Technology should adapt to humanity, not the other way around,” said Raffi Krikorian, CTO of Mozilla. “Across our entire portfolio, Mozilla is working to decentralize the future of AI — from building new tools for developers to supporting creators to building all levels of the open source AI ecosystem. India, with its incredible community of founders, startups, and developers, knows firsthand that open source AI is where innovation is going next.” The post Heading to India AI Impact Summit, Mozilla leaders call for investment in open source AI as a path to sovereignty appeared first on The Mozilla Blog.

- The Rust Programming Language Blog: crates.io: an update to the malicious crate notification policy (2026/02/13 00:00)
  The crates.io team will no longer publish a blog post each time a malicious crate is detected or reported. In the vast majority of cases to date, these notifications have involved crates that have no evidence of real world usage, and we feel that publishing these blog posts is generating noise, rather than signal. We will always publish a RustSec advisory when a crate is removed for containing malware. You can subscribe to the RustSec advisory RSS feed to receive updates. Crates that contain malware and are seeing real usage or exploitation will still get both a blog post and a RustSec advisory. We may also notify via additional communication channels (such as social media) if we feel it is warranted. Recent crates Since we are announcing this policy change now, here is a retrospective summary of the malicious crates removed since our last blog post and today: finch_cli_rust, finch-rst, and sha-rst: the Rust security response working group was notified on December 9th, 2025 by Matthias Zepper of

National Genomics Infrastructure Sweden that these crates were attempting to exfiltrate credentials by impersonating the finch and finch_cli crates. Advisories: RUSTSEC-2025-0150, RUSTSEC-2025-0151, RUSTSEC-2025-0152. polymarket-clients-sdk: we were notified on February 6th by Socket that this crate was attempting to exfiltrate credentials by impersonating the polymarket-client-sdk crate. Advisory: RUSTSEC-2026-0010. polymarket-client-sdks: we were notified on February 13th that this crate was attempting to exfiltrate credentials by impersonating the polymarket-client-sdk crate. Advisory: RUSTSEC-2026-0011. In all cases, the crates were deleted, the user accounts that published them were immediately disabled, and reports were made to upstream providers as appropriate. Thanks Once again, our thanks go to Matthias, Socket, and the reporter of polymarket-client-sdks for their reports. We also want to thank Dirkjan Ochtman from the secure code working group, Emily Albini from the security response working group, and Walter Pearce from the Rust Foundation for aiding in the response.

- Hacks.Mozilla.Org: Launching Interop 2026 (2026/02/12 17:07)
  The Interop Project is a cross-browser initiative to improve web compatibility in areas that offer the most benefit to both users and developers. The group, including Apple, Google, Igalia, Microsoft, and Mozilla, takes proposals of features that are well defined in a sufficiently stable web standard, and have good test suite coverage. Then, we come up with a subset of those proposals that balances web developer priorities (via surveys and bug reports) with our collective resources. We focus on features that are well-represented in Web Platform Tests as the pass-rate is how we measure progress, which you can track on the Interop dashboard. Once we have an agreed set of focus areas, we use those tests to track progress in each browser throughout the year. And after that, we do it all again! But, before we talk about 2026, let’s take a look back at Interop 2025… Interop 2025 Firefox started Interop 2025 with a score of 46, so we’re really proud to finish the cycle on 99. But the number that really matters is the overall Interop score, which is a combined score for all four browsers – and the higher this number is, the fewer developer hours are lost to frustrating browser differences. The overall Interop score started at 25, and it’s now 95. As a result, huge web platform features became available cross-browser, such as Same-Document View Transitions, CSS Anchor Positioning, the Navigation API, CSS @scope, and the URLPattern API. That’s the headline-grabbing part, but in my experience, it’s way more frustrating when a feature is claimed to be supported, but doesn’t work as expected. That’s why Interop 2025 also focused on improving the reliability of existing features like WebRTC, CSS Flexbox, CSS Grid, Pointer Events, CSS backdrop-filter, and more. But it’s not just about passing tests With some focus areas, in particular CSS Anchor Positioning and the Navigation API, we noticed that it was possible to achieve a good score on the tests while having inconsistent behavior compared to other browsers. In some cases this was due to missing tests, but in some cases the tests contradicted the spec. This usually happens when tests are written against a particular implementation, rather than the specified behavior. I experienced this personally before I joined Mozilla – I tried to use CSS Anchor Positioning back when it was only shipping in Chrome and Safari, and even with simple use-cases, the results were wildly inconsistent. Although it caused delays in these features landing in Firefox, we spent time highlighting these problems by filing issues against the relevant specs, and ensured they got priority in their working groups. As a result, specs became less ambiguous, tests were improved, and browser behavior became more reliable for developers. Okay, that’s enough looking at the past. Let’s move on to… Interop 2026 Over 150 proposals were submitted for Interop 2026. We looked through developer feedback, on the issues themselves, and developer surveys like The State of HTML and The State of CSS. As an experiment for 2026, we at Mozilla also invited developers to stack-rank the proposals, the results of which we used in combination with the other data to compare developer preferences between individual features – this is something we want to expand on in the future. After carefully examining all the proposals, the Interop group has agreed on 20 focus areas (formed of 33 proposals) and 4 investigation areas. See the Interop repository for the full list, but here are the highlights: New features As with

2025, part of the effort is about bringing new features to all browser engines. Cross-document View Transitions allow transitions to work across documents, without any JavaScript. The sub-features rel="expect" and blocking="render" are included in this focus area.. Scroll-driven animations allow you to drive animations based on the user's scroll position. This replaces heavy JavaScript solutions that run on the main thread. WebTransport provides a low-level API over HTTP/3, allowing for multiple unidirectional streams, and optional out-of-order delivery. This is a modern alternative to WebSockets. CSS container style queries allow you to apply a block of styles depending on the computed values of custom properties on the nearest container. This means, for example, you can have a simple --theme property that impacts a range of other properties. JavaScript Promise Integration for Wasm allows WebAssembly to asynchronously 'suspend', waiting on the result of an external promise. This simplifies the compilation of languages like C/C++ which expect APIs to run synchronously. CSS attr() has been supported across browsers for over 15 years, but only for pseudo-element content. For Interop 2026, we're focusing on more recent changes that allow attribute values to be used in most CSS values (with URLs being an exception). CSS custom highlights let you register a bunch of DOM ranges as a named highlight, which you can style via the ::highlight(name) pseudo-element. The styling is limited, but it means these ranges can span between elements, don't impact layout, and don't disrupt things like text selection. Scoped Custom Element Registries allow different parts of your DOM tree (such as a shadow root) to use a different set of custom elements definitions, meaning the same tag name can refer to different custom elements depending on where they are in the DOM. CSS shape() is a reimagining of path() that, rather than using SVG path syntax, uses a CSS syntax, allowing for mixed units and calc(). In practice, this makes it much easier to design responsive clip-paths and offset-paths. And more, including CSS contrast-color, accent-color, dialog closedby, popover="hint", fetch upload streams, IDB getAllRecords(), media pseudo-classes such as :playing, and the Navigation API's precommitHandler. Existing feature reliability improvements Like in previous years, the backbone of Interop is in improving the reliability of existing features, removing frustrations for web developers. In 2026, we'll be focusing these efforts on particular edge cases in: Range headers & form data in fetch The Navigation API CSS scroll snap CSS anchor positioning Same-document View Transitions JavaScript top-level await The event loop WebRTC CSS user-select CSS zoom Some of these are carried over from 2025 focus areas, as shortcomings in the tests and specs were fixed, but too late to be included in Interop 2025. Again, these are less headline-grabbing than the shiny new features, but it's these edge cases where us web developers lose hours of our time. Frustrating, frustrating, hours. Interop investigations Sometimes, we see a focus area proposal that's clearly important, but doesn't fit the requirements of Interop. This is usually because the tests for the feature aren't sufficient, are in the wrong format, or browsers are missing automation features that are needed to make the feature testable. In these cases, we identify what's missing, and set up an investigation area. For interop 2026, we're looking at… Accessibility. This is a continuation of work in 2025. Ultimately, we want browsers to produce consistent accessibility trees from the same DOM and CSS, but before we can write tests for this, we need to improve our testing infrastructure. Mobile testing. Another continuation from 2025. In particular, in 2026, we want to figure out an approach for testing viewport changes caused by dynamic UI, such as the location bar and virtual keyboard. JPEG XL. The current tests for this are sparse. Existing decoders have more comprehensive test suites, but we need to figure out how these relate to browsers. For example, progressive rendering is an important feature for developers, but how and when browsers should do this (to avoid performance issues) is currently being debated. WebVTT. This feature allows for text to be synchronised to video content. The investigation is to go through the test suite and ensure it's fit for purpose, and amend it where necessary. It begins… again The selected focus areas mean we've committed to more work compared to the other browsers, which is quite the challenge being the only engine that isn't owned by billionaires. But it's a challenge we're happy to take on! Together with other members of the Interop group, we're looking forward to delivering

features and fixes over the next year. You can follow along with the progress of all browsers on the Interop dashboard. If your favorite feature is missing from Interop 2026, that doesn't mean it won't be worked on. JPEG XL is a good example of this. The current test suite meant it wasn't a good fit for Interop 2026, but we've challenged the JPEG XL team at Google Research to build a memory-safe decoder in Rust, which we're currently experimenting with in Firefox, as is Chrome. Interop isn't the limit of what we're working on, but it is a cross-browser commitment. If you're interested in details of features as they land in Firefox, and discussions of future features from spec groups, you can follow us on: BlueSky Mastodon LinkedIn Threads YouTube TikTok Instagram Partner Announcements This is a team effort, and we've all made announcement posts like this one. Get other members' take on it: Apple Google Igalia Microsoft The post Launching Interop 2026 appeared first on Mozilla Hacks - the Web developer blog.

- Mozilla Localization (L10N): Pontoon Translation Search: Unifying Localization Across Mozilla (2026/02/12 08:14)
  Here at Mozilla, we are tirelessly working to bring our products to a global audience. Pontoon, our in-house translation management system plays a central role, where volunteer localizers work to bring translations for Firefox, Thunderbird, SUMO and other Mozilla products. Recently, we have been working to unify localization tools and give localizers and developers smoother, more streamlined workflows. This is why we are excited to introduce Pontoon's new Translation Search feature, where everyone can search for strings across all projects and locales at Mozilla. What is the Translation Search Feature? Translation Search is Pontoon's latest way to access our extensive collection of translations, built up through years of localization work by fellow Mozillians. This new feature allows localizers to search for strings across all projects and all locales at Mozilla. Inspired by the functionality of Transvision, it is intended to be a suitable replacement and includes many of the various features that localizers rely on. Let's go through some of its features and how they can apply to your localization workflows. Searching for Translations Pontoon's new Translation Search, where users can search for strings through all projects and locales. Searching for strings in Translation Search is simple. Similar to how Transvision operates, you can search within a specific project and locale, as well as filter by string identifiers, case sensitivity and whole word search. Unlike Transvision, Pontoon Translation Search covers all products localized at Mozilla. It is also completely integrated with other Pontoon elements, including Translate, such that you can seamlessly navigate to Translate after your search. Transvision's search functionality, which we intend to replace. Finding Entity Translations Pontoon's new Entity Translations page, which lists available translations for a source string. If you want to get a better picture of a source string with different translations, go to the Entity Translations page. By clicking the "All Locales" button in Translation Search for a string, you can see the source string translated into every available locale. This is useful for comparing similar locale translations and getting a broader picture of a string's context. This feature is intended to replace Transvision's translation list for a particular entity. Transvision's string translation list, which we intend to replace. Searching from Firefox Address Bar If you have the Pontoon Add-on installed to the latest version, you can now search for strings directly from the address bar in Firefox: you can select Pontoon from the list of search engines, or start typing pontoon and press TAB. How to use Translation Search with Pontoon Add-on. Pontoon Translation Search is Live! This feature is available now and ready for you to use. Head over to pontoon.mozilla.org/search to try out the new search experience and streamline your localization work! If you have any questions or concerns about Translation Search, do not hesitate to contact us on Matrix or file an issue. You can also consult the documentation here.
- The Rust Programming Language Blog: Announcing Rust 1.93.1 (2026/02/12 00:00)
  The Rust team has published a new point release of Rust, 1.93.1. 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 1.93.1 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 1.93.1 Rust 1.93.1 resolves three regressions that were introduced in the 1.93.0 release. Don't try to recover a keyword as a non-keyword identifier, fixing an internal compiler error (ICE) that especially affected rustfmt. Fix a clippy::panicking_unwrap false-positive on field access with an implicit dereference. Revert an update to wasm-related dependencies, fixing file descriptor leaks on the wasm32-wasip2 target. This only affects the rustup component for this target, so downstream toolchain builds should check their own dependencies too. Contributors to 1.93.1 Many people came together to create Rust 1.93.1. We couldn't have done it without all of you. Thanks!

- Thunderbird Blog: Thunderbird Monthly Development Digest: February 2026 (2026/02/11 17:50)
  Welcome back to the Thunderbird blog and the first post of 2026! We’re rested, recharged, and ready to keep our community updated on all of our progress on the desktop and mobile clients and with Thunderbird Pro! Hello again from the Thunderbird development team! After a restful and reflective break over the December holidays, the team returned recharged and ready to take on the mountain of priorities ahead. To everyone we met during the recent FOSDEM weekend, thank you! The conversations, encouragement, and thoughtful feedback you shared were genuinely energizing, and many of your insights are already helping us better understand the real-world challenges you’re facing. The timing couldn’t have been better, as FOSDEM provided a strong early-year boost of inspiration, collaboration, and perspective. FOSDEM – Collaboration, learning and real conversations This year, a larger contingent of the Thunderbird team joined our Mozilla colleagues in Brussels for an intense and rewarding FOSDEM weekend. Across talks, hallway chats, and long discussions at the Thunderbird booth, we dug into standards, shared hard-won lessons, debated solutions, and explored what’s next for open communication tools. The highlight, as always, was meeting users face-to-face. Hearing your stories about what’s working, what’s painful, and what you’d like to see next continues to be one of the most motivating parts of our work. Several recurring themes stood out in these discussions, and we’re keen to help move the needle on some of the bigger, knottier challenges, including: Unblocking Oauth issues for Microsoft exchange Enterprise management feature support and extension Add-ons and feature requests to enable the continued move to FOSS solutions for many European institutions and orgs These conversations don’t end when FOSDEM does but help shape our priorities for the months ahead, and we’re grateful to everyone who took the time to stop by, ask questions, or share their experiences. Exchange Email Support After releasing Exchange support for email to the Monthly release channel, we’ve had some great feedback and helpful diagnosis of edge case problems that we’ve been prioritizing for the past few weeks Work completed during this period includes: Concurrency, queuing and prioritization of requests Classless folder handling Subfolder copy/move operations Starring a message (which proved to be far more painful than imagined) Custom Oauth configuration support Work on supporting the Graph API protocol for email is moving steadily through the early stages, with these basic components already shipped to Daily: Initial scaffolding & rust crates Account Hub changes to support the addition of Graph protocol Login via Oauth Connectivity check Keep track of our Graph API implementation here. Account Hub The team met in person following FOSDEM and have planned out work to allow the new Account Hub UX to be used as the default experience in our next Extended Support Release this summer, which will ensure users benefit from changes we’ve made to enable custom Oauth settings and configuration specific to Microsoft Exchange. Follow progress in the meta bugs for the last few pieces of phase 3 and telemetry, as well as the work we’ve defined to enable an interim experience for users setting up Thunderbird for the first time. Calendar UI Rebuild The new Calendar UI work has advanced at a good pace in recent weeks and the team met in person to break the work apart into chunks which have been prioritized alongside some of the “First Time User Experience” milestones. The team has recently: Completed sprint planning for upcoming milestones Assigned tasks and estimated work for the next 2 milestones Continued preparation for adopting Redux-based state

management during the “Event Add/Edit” milestone Stay tuned to our milestones here: Event Read Dialog Event Read Enhancements Read Dialog inline editing Event Add/Edit Dialog Reminders UI Invitations Dialog Calendar Views Maintenance, Upstream adaptations, Recent Features and Fixes Over the past couple of months, a significant portion of the team’s time has gone into responding to upstream changes that have impacted build stability, test reliability, and CI. Sheriffing continues to be a challenge, with frequent breakages requiring careful investigation to separate upstream regressions from Thunderbird-specific changes. Alongside this ongoing maintenance work, we’ve also benefited greatly from contributions across the wider development community. Thanks to that collective effort, a steady stream of fixes and improvements has landed. More broadly and focusing on our roadmap, the last two months have seen solid progress on Fluent migrations, as well as planning and early groundwork for rolling out Redux and the Design System more widely across the codebase. Support from the community and team has resulted in some notable patches landing in recent weeks, with the following of particular help: Fixing unified toolbar sizing Hardening local search against crashes Migration of strings to modern Fluent – Filters & Plurals and many more which are listed in release notes for beta. If you would like to see new features as they land, and help us find some early bugs, you can try running daily and check the pushlog to see what has recently landed. This assistance is immensely helpful for catching problems early. Toby Pilling Senior Manager, Desktop Engineering The post Thunderbird Monthly Development Digest: February 2026 appeared first on The Thunderbird Blog.

- Support.Mozilla.Org: What’s up with SUMO – H2 2025 (2026/02/11 07:00)
  Hi everyone, We may already be a few weeks into 2026, but it’s never too late to say Happy New Year! As we dive into the year ahead, we also want to take a moment to reflect on everything we accomplished together in the second half of 2025. In this recap, you’ll find highlights from community campaigns, major platform updates and product launches, as well as forum and Knowledge Base data that offer a clearer picture of how the community performed. We hope this snapshot helps celebrate what went well, while also shining a light on areas where we can continue to grow together. Let’s jump in! Highlights The biggest highlight from H2 2025 was probably our first Ask a Fox event that brought contributors from both Firefox and Thunderbird together to respond to user questions in real time. Our total contributors aroused by 41.6 %, and we hit the highest weekly reply rate of the year during the event. The energy, collaboration, and collective impact of that week highlighted the power of coordinated community action, so much so that we’re now exploring ways to make it a recurring event. We launched the Semantic History Search campaign in September, which reaffirmed our community’s strong interest in shaping early-stage features. Contributor participation in this testing phase showed once again how eager our communities are to help shape the future of Firefox. The SUMO engineering team has also introduced Machine Translation (MT) for Knowledge Base articles that we first announced in August, with Italian and Spanish enabled later in September. This marked the start of a long-term effort to improve scalability and freshness of support content, particularly in core locales, where we’ve seen content freshness improve from 38% to 96%. At the same time, we recognize that this shift brought significant changes to long-established contributor workflows. We understand the rollout hasn’t been without its bumps. For many in the community, it raised concerns about quality, trust, and the role of contributors in shaping localized content. These are important conversations and we remain committed to listening carefully, learning from your feedback, and making the transition as transparent and collaborative as possible. We sincerely appreciate the patience, feedback, and ongoing dedication of everyone who has helped us navigate this complex change together. In October, we also helped host a Reddit AMA with Firefox leadership, and the response from the community was overwhelmingly positive. The thread generated 184 comments and 218 upvotes, making it our most engaging AMA on Reddit to date. This level of interaction reflects a strong appetite for direct and transparent conversations with users. The AMA not only created space for meaningful dialogue but also surfaced valuable insights that can

inform future product decisions. Building on this momentum, we’re committed to hosting more AMAs in the future to continue strengthening that connection. Earlier in the year, we made the tough call to sunset Social Support and Mobile Store Support, a decision made to focus our energy and resources more intentionally on the Community Forums. While we know this transition wasn’t easy for everyone, the impact became clear by year-end: reply rates rebounded from 49.9% in H2 2024 to 62% in average (May-December 2025), a nearly 12% increase. This shift signaled that our community didn’t just adapt, but they rallied, making the forum stronger, faster, and more effective than it had been in months. We wrapped the year with a celebration of identity and collaboration. Kit, our charming new mascot, made its debut in November. And shortly after, we announced that Mozilla Connect officially joined forces with the SUMO team. The alignment felt natural, uniting support and feedback under one community umbrella. Community stats Forum Support General stats The forum continues to show encouraging signs of community growth and maturity. Most notably, the solve rate more than doubled, jumping by 124% to reach 11.9%. This is a clear signal that contributors are not only engaging with users, but successfully resolving their issues at a much higher rate. We also saw a 9.8% increase in OP reply rate, suggesting stronger two-way engagement between users and contributors. Improvements in speed reinforce this trend: first response time dropped by nearly 30%, while time to resolution was cut in half, falling by 48.9%. Combined with a 12.7% rise in reply rate and 15.7% more threads being actively supported, these results point to a community that’s not just growing, but becoming more efficient, responsive, and impactful. With the automatic spam moderation introduced in the first half of year, we’ve seen fewer total questions coming in H2 2025 but higher quality interactions overall. This shift suggests a more focused and intentional support environment. Taken together, these trends suggest that it’s time to elevate solve rate from a “nice to have” to a core success metric, a meaningful reflection of contributor expertise, community trust, and the maturity of our Community Forums. Total valid questions 14803 (-20%) Reply rate 63.5 (+12.67%) Solve rate 11.9% (+124%) Total responses posted 14191 (+20.4%) Total threads interacted 9599 (+15.7%) Average first response time (in hour) 22.4 (-29.8%) Average time to resolution (in day) 2.55 (-48.9%) Total new registration 455k (+0.5%) Total contributors 963 (-1.3%) Helpful rate 60.8% (+3.92%) OP reply rate 27% (+9.8%) Top forum contributors All credit for this impressive performance goes to our incredible forum community, who continue to raise the bar with each quarter. Their dedication, consistency, and responsiveness are what make these results possible. We’re proud to highlight the top 3 contributors on the English forum, along with the leading contributors across other locales. This shows a true reflection of the global impact of our support network. Contributor name Total responses Total threads engaged Paul (en-US) 1900 1395 Denyshon (en-US) 1284 924 Jefferson Scher (en-US) 930 871 @next (it) 557 438 Gerardo (es) 523 457 Mark Heijl (nl) 186 155 Selim (tr) 126 87 Ansamb (cs) 87 65 Poljos-moz (cs) 81 59 Sandromonteiro (pt-BR) 15 15 Balázs Meskó (hu) 51 48 Vexi (sl) 5 5 Emerging forum contributors Contributor name Total responses Total threads engaged George Kitsoukakis 202 171 Mark 136 110 sjohnn 65 45 t.r.ernst 64 60 Jeff-g 61 54 Knowledge Base General stats We’ve seen an impressive uptick in article contributions in the second half of 2025, with 925 revisions submitted to the English Knowledge Base, a 21.9% increase compared to the previous period. This continued growth reflects not just dedication, but real momentum of the growing spirit to keep our Knowledge Base fresh and helpful for users worldwide. This level of participation directly supports our broader direction towards improving and streamlining the content request workflow. As we continue investing in clearer processes and better documentation, it’s clear that contributors are willing to step up when the pathway to impact is well defined. Total en-US revisions 925 (+21.9%) Total articles 248 (+2.9%) Total revisions reviewed 821 (+19.9%) Total revisions approved 778 (+17.7%) Total authors 97 Total reviewers 19 (+11.8%) Top KB contributors The numbers may show progress, but the real story is the people behind them. Behind these revisions, 97 unique contributors stepped in to create the updates and 19 reviewers helped guide their contributions. And here’s just a glimpse of the top 5 contributors: Contributor name Total

revisions Total articles Total reviews AliceWyman 467 246 334 Pierre Mozinet 125 100 – Mark Heijl 101 88 – Michele Rodaro 50 46 35 Paul 26 21 5 Article Localization The localization community delivered an outstanding performance in H2 2025, despite undergoing significant changes. We saw 4807 non-English revisions submitted, a 37.5% increase, covering 2664 articles across locales (+29.7%). In total, 4,259 revisions were approved and 4,296 reviewed, reflecting consistent contributor dedication to quality and accuracy. Most notably, 314 unique contributors stepped up to author content, representing a 54.7% increase from earlier this year. These results are especially meaningful given the rollout of Machine Translation in August, a major shift in localization workflow that understandably sparked concern and discussion across the community. Adjusting to MT required both flexibility and trust, and we're grateful that many contributors responded by showing up in full force. Your continued involvement ensured that translations remained thoughtful, context-aware, and aligned with Mozilla's values of openness and quality. This success is a testament to the strength, resilience, and care of our contributor base, and we're deeply grateful for your ongoing contribution. General stats (minus sumoBot) Total non en-US revisions 4807 (+37.5%) Total articles 2664 (+29.7%) Total revisions reviewed 4296 (+33.5%) Total revisions approved 4259 (+33.6%) Total authors 314 (+54.7%) Total reviewers 45 (-6%) Top localization contributors Contributor name Total revisions Total articles Total reviews Michele Rodaro (it) 819 393 801 Jim Spentzos (el) 727 566 555 Valery Ledovskoy (ru) 627 400 635 Mark Heijl (nl) 575 359 – Milupo (dsb & hsb) 460 224 330 Emerging localization contributors Contributor name Total revisions Total articles 普莱是袋熊 (zh-CN) 35 31 Zhengyang3552 (zh-CN) 22 21 xanhAD (vi) 13 11 Stay connected with the community Join the Conversation Participate in ongoing discussions on the Contributor Forum to catch up on the latest updates and share your input. Drop by our Matrix channel for more casual chats with fellow contributors. Attend Our Monthly Community Call Every month, we host a community call to share updates about Firefox and community activities. Watch past recordings from 2026! Don't hesitate to join the call in person if you can. We try our best to provide a safe space for everyone to contribute. Don't feel pressured to turn on your camera or speak if you're not comfortable. You can also: Submit your questions ahead of time via the Contributor Forum or Matrix Lurk silently and absorb the updates—your presence is still valued! Stay Informed Follow the SUMO Blog for the latest community news and updates. Subscribe to the Mozilla Community Newsletter to see what's happening across the wider Mozilla ecosystem. Sign up for the Firefox Daily Digest (Mon–Fri) for curated Firefox news and updates from around the web. Explore What We're Building Curious about what the platform team is working on? Visit the SUMO Engineering Board to see what the platform team is cooking in the engine room. You can also view our latest release notes to stay informed about recent changes and improvements.

- Niko Matsakis: Dada: moves and mutation (2026/02/11 00:29)
  Let's continue with working through Dada. In my previous post, I introduced some string manipulation. Let's start talking about permissions. This is where Dada will start to resemble Rust a bit more. Class struggle Classes in Dada are one of the basic ways that we declare new types (there are also enums, we'll get to that later). The most convenient way to declare a class is to put the fields in parentheses. This implicitly declares a constructor at the same time: class Point(x: u32, y: u32) {} This is in fact sugar for a more Rust like form: class Point { x: u32 y: u32 fn new() -> Point { Point { x, y } } } And you can create an instance of a class by calling the constructor: let p = Point(22, 44) // sugar for Point.new(22, 44) Mutating fields I can mutate the fields of p as you would expect: p.x += 1 p.x = p.y Read by default In Dada, the default when you declare a parameter is that you are getting read-only access: fn print_point(p: Point) { print("The point is {p.x}, {p.y}") } let p = Point(22, 44) print_point(p) If you attempt to mutate the fields of a parameter, that would get you an error: fn print_point(p: Point) { p.x += 1 # <-- ERROR! } Use ! to mutate If you declare a parameter with !, then it becomes a mutable reference to a class instance from your caller: fn translate_point(point!: Point, x: u32, y: u32) { point.x += x point.y += y } In Rust, this would be like point: &mut Point. When you call

translate_point, you also put a ! to indicate that you are passing a mutable reference: let p = Point(22, 44) # Create point print_point(p) # Prints 22, 44 translate_point(p!, 2, 2) # Mutate point print_point(p) # Prints 24, 46 As you can see, when translate_point modifies p.x, that changes p in place. Moves are explicit If you're familiar with Rust, that last example may be a bit surprising. In Rust, a call like print_point(p) would move p, giving ownership away. Trying to use it later would give an error. That's because the default in Dada is to give a read-only reference, like &x in Rust (this gives the right intuition but is also misleading; we'll see in a future post that references in Dada are different from Rust in one very important way). If you have a function that needs ownership of its parameter, you declare that with given: fn take_point(p: given Point) { // ... } And on the caller's side, you call such a function with .give: let p = Point(22, 44) take_point(p.give) take_point(p.give) # <-- Error! Can't give twice. Comparing with Rust It's interesting to compare some Rust and Dada code side-by-side: Rust Dada vec.len() vec.len() map.get(&key) map.get(key) vec.push(element) vec!.push(element.give) vec.append(&mut other) vec!.append(other!) message.send_to(&channel) message.give.send_to(channel) Design rationale and objectives Convenient is the default The most convenient things are the shortest and most common. So we make reads the default. Everything is explicit but unobtrusive The . operator in Rust can do a wide variety of things depending on the method being called. It might mutate, move, create a temporary, etc. In Dada, these things are all visible at the callsite– but they are unobtrusive. This actually dates from Dada's "gradual programming" days – after all, if you don't have type annotations on the method, then you can't decide foo.bar() should take a shared or mutable borrow of foo. So we needed a notation where everything is visible at the call-site and explicit. Postfix operators play more nicely with others Dada tries hard to avoid prefix operators like &mut, since they don't compose well with . notation.

- Dave Townsend: Can't you do this faster with AI? (2026/02/09 18:00)
  I'm hearing this question asked a lot lately. Both within Mozilla and from others in the industry. You come up with a plan for implementing some feature, put your best estimate on how long it will take to implement, and then you get push back from folks several levels removed from the project along the lines of "Wouldn't this be faster if you used AI?", or "Can't Claude Code do most of this?".
- Niko Matsakis: Hello, Dada! (2026/02/09 11:03)
  Following on my Fun with Dada post, this post is going to start teaching Dada. I'm going to keep each post short – basically just what I can write while having my morning coffee.1 You have the right to write code Here is a very first Dada program println("Hello, Dada!") I think all of you will be able to guess what it does. Still, there is something worth noting even in this simple program: "You have the right to write code. If you don't write a main function explicitly, one will be provided for you." Early on I made the change to let users omit the main function and I was surprised by what a difference it made in how light the language felt. Easy change, easy win. Convenient is the default Here is another Dada program let name = "Dada" println("Hello, {name}!") Unsurprisingly, this program does the same thing as the last one. "Convenient is the default." Strings support interpolation (i.e., {name}) by default. In fact, that's not all they support, you can also break them across lines very conveniently. This program does the same thing as the others we've seen: let name = "Dada" println(" Hello, {name}! ") When you have a " immediately followed by a newline, the leading and trailing newline are stripped, along with the "whitespace prefix" from the subsequent lines. Internal newlines are kept, so something like this: let name = "Dada" println(" Hello, {name}! How are you doing? ") would print Hello, Dada! How are you doing? Just one familiar String Of course you could also annotate the type of the name variable explicitly: let name: String = "Dada" println("Hello, {name}!") You will find that it is String. This in and of itself is not notable, unless you are accustomed to Rust, where the type would be &'static str. This is of course a perennial stumbling block for new Rust users, but more than that, I find it to be a big annoyance – I hate that I have to

write "Foo".to_string() or format!("Foo") everywhere that I mix constant strings with strings that are constructed. Similar to most modern languages, strings in Dada are immutable. So you can create them and copy them around: let name: String = "Dada" let greeting: String = "Hello, {name}" let name2: String = name Next up: mutation, permissions OK, we really just scratched the surface here! This is just the “friendly veneer” of Dada, which looks and feels like a million other languages. Next time I’ll start getting into the permission system and mutation, where things get a bit more interesting. My habit is to wake around 5am and spend the first hour of the day doing “fun side projects”. But for the last N months I’ve actually been doing Rust stuff, like symposium.dev and preparing the 2026 Rust Project Goals. Both of these are super engaging, but all Rust and no play makes Niko a dull boy. Also a grouchy boy. ↩