Q1 Face-to-face

Feb 14, 2025 | Part 2

Attendees:

- Westbrook Johnson
- Olli Pettay
- Sanket Joshi (Microsoft)
- Dan Clark (Microsoft)
- Jesse Jurman
- Maxime Bétrisey
- Ryosuke Niwa (Apple)
- Pascal Vos
- Owen Buckley
- …

Agenda:

1. [25 minutes] Reference target
    - final questions
    - Level 2 discussion
        - referenceTargetMap
        - referencing out
            - Button in shadow form outside
            - Label in shadow targeting input outside of that root
        - Other attributes
2. [20 minutes] Scoped Registries ongoing questions
    - Implementor appetite for Apple proposal?
3. [20 minutes] :has-slotted(...) & /slotted/
    - /slotted/ continue discussion from CSSWG
    - :has-slotted(...) the functional version
        - How do we support flatten vs not in this sort of syntax
4. [30 minutes] HTML Module Scripts
5. Overflow? Possibly, depends on what holds over from session no. 1

Time: [95 minutes] + [5 minute break]

Notes:

- Reference Target
    - Dan Clark: blocking issues for Phase 1 in https://github.com/WICG/webcomponents/issues/1086 - some of these are already close to consensus, but would be good to get agreement here
    - Dan Clark: Phase 2, release independently, there is a prototype in Chromium and a code review for WebKit.
    - Dan Clark opens Explainer for quick review
    - Dan Clark: when we want to apply a label from the light DOM, there is no way to plumb that through to the shadow root. This is true for any IDREF, like aria

attributes, and popover. This `referenceTarget` is an attribute on the shadowRoot (passed into the attachShadow constructor). Also works with the declarative case (shadowrootreferencetarget)

- Olli P: should phase 1 and phase 2 happen at the same time? I wish there was only one attribute, having two complicates the platform and makes it harder to explain
    - Dan Clark: good question, at TPAC we talked about it. There was a question if Phase 1 is good enough on its own. The sentiment is that Phase 2 is important, but Phase 1 should not be blocked and gets us 90% of the way there. That is the basis on where we are moving now. It's still valuable to think of both phases (so we don't back ourselves to a corner), but phase 1 has a lot of value. Working in parallel feels valuable.
- Keith: the list of IDREF is well defined and bounded - can you talk about the implementation here, since the other side of phase 2 - I imagine the implementation for both is relatively similar mapping, and phase 1 is just an ergonomic improvement.
    - Dan Clark: it feels like Phase 1 is the more simple version and the thing that most developers would want. From the implementation side, both are not super hard - in blink these ID references are not cached, and we're only looking in the light DOM, and we just recursively check for reference target. Phase 2 just checks which attribute, but yeah Implementation wise it's not much harder either way.
- Dan Clark: interesting question brought up by Ben, about reflecting attributes, that reflect IDREF, like `form`, and this is interesting in the reference target case, because we have a form association, where the form is in the shadow DOM, and now this is possible because of reference target. There is a question about what the `form` attribute should be though. We don't want to expose the `form` (that should be hidden in the shadow DOM). Alternatively we could expose the `host`, but that feels unexpected. Potentially it would be `null`? I feel like this is something we should try the host option, and re-evaluate after getting feedback.
    - Westbrook: a good number of people from the community group have discussed in the issue here. It does feel like `null` is a dead-end, where returning the host has the ability to work with the component author.
    - Olli: what happens when you change the id of the form, so it is not there anymore, what will happen? Will it start to return null.
    - Dan: yeah, it will return null, which is why returning host feels valuable, so you can tell when that has been disconnected.
    - Dan: it sounds like there aren't issues here, and there is a rough consensus on using host, so are we good to close this?
    - Westbrook: I think closing this is fine
- Dan: we should bikeshed the name - there isn't a great consensus on the name. One complaint is that it is too long, especially in phase 2, when the attribute gets really long. There is an idea to shorten it - refs, or delegaterefs. Not sure there is anything people have fallen in love with. Status quo would be to run with what we

have. If people want to advocate for a different thing, this might be a good time to do that.

- Anne van Kesteren: I don't mind delegate, but I don't feel particularly strongly.
- Olli: I don't feel strongly, but I also don't mind delegate.
- Mayank: I don't mind delegate - ref can be an overloaded term, especially with frameworks today. Between delegate and delegatesTo, the latter is more clear.
- Dan: I slightly prefer delegates as a term, the length is slightly punishing. On templates it would be `shadowrootdelegatesreferenceariadescribedby`
- Anne: can we just use delegates?
- Dan: is that confusing with delegatesfocus
- Keith Cirkel: could we do then shadowrootdelegatesariadescribedby (that is, remove the reference)
- Westbrook: (re delegates vs delegate) delegate is both a verb and a noun
- Mayank: should it be delegateId?
- Anne: it's not always an id - it's the delegate element
- Keith: the other id references don't use id in the name, they use `target`.
- <Some consensus on delegatesTo>
- Dan: would it be valuable to have a survey? If there's no champion for an alternative that may be worthwhile
- Keith: If no one is in objective to delegatesTo, then maybe we don't need a survey
- Dan: I'm okay with having that be the new default pick.
- Mayank: is that for both phase 1 and phase 2? It feels confusing for phase 2?
- Dan: yeah, potentially we could reword it, to have `to` at the end
- Anne: potentially you would remove the `to`
- Mayank: `delegatesPopover` does read like a boolean.
- Dan: maybe we could sort that out in a phase 2 discussion. It sounds like we can use `delegatesTo`.
- Ryosuke: there is a confusion around the implementation around delegatesFocus. Maybe we could have delegates take in a boolean? Or rethink about how delegatesFocus behaves.
    - Dan: maybe there is a thing we could think around delegates
    - Keith: in the presence of a delegatesFocus attribute, maybe that takes precedence, but if you were to provide a string, then it could change that behavior? We could tie it together.
    - Dan: maybe if you give it an ID, we can change that behavior. There is a question of how this should behave if you also have delegateFocus, but I don't know that we want to (in a default way) tie these together.

- Keith: theoretically, if we do think about the map case (phase 2), do you have list 19 of 20 if you don't want a singular one?

        - Dan: in a phase two approach, we should try to avoid that
    - Dan: I'll open an issue on the focus, because that is valuable
- Scoped Registries APIs
    - Anne: Since this was presented, there was feedback on the API shape. We have since updated the proposal. Not have create elements functions, and keep those on document. Additionally, Ryosuke has implemented this in WebKit, but not published yet
    - Ryosuke: will be 2 technical previews from now - so around March this should be available, second week of March - will let the group know when it is available
    - Justin Fagnani: can we see the new API shape, and look at where frameworks would have to update to use that, and get a sense of how easy or difficult it would be
        - Anne: in the dictionary, you pass the registry, as a new dictionary member.
        - Justin: thinking about how we monkey patch React's behavior.
        - Anne: I think if you were able to do that with the original proposal, you should be able to do that with the new proposal
    - Ryosuke: another change, we have an options dictionary to import.
        - Justin: does that take a deep options?
        - Anne: it might be called subtree, but yeah
        - Justin: is it default true?
        - Anne: no, we didn't want to flip the default.
        - Ryosuke: an empty dictionary, we want to treat all values as falsy
        - Anne: undefined we definitely want to be false
        - Justin: we would have to invert the name, to shallow
        - Sanket: is adding the dictionary a breaking change?
        - Anne: no, we can overload the type, we've done this before with eventListener
        - Sanket: Like the new pattern - if you pass null, do you still get the HTML Unknown element?
        - Anne: no, you get a null registry, but yeah, if you pass a null registry, and build a dashed element that isn't defined, it would be HTML Unknown
        - Justin: the existing behavior does already do something if you pass null.
        - Anne: that is worth bikeshedding - I'll make an issue for that
    - Anne: does the custom registry matter for equality?
        - Ryosuke: if you call the same constructor in multiple registries, the expectation is probably that it is true.
        - Justin: even if there is a small behavior difference, their children could be different types, even if the registry is similar.
        - Anne: if you compare elements across documents, they would be considered equal. Maybe that is a good reason to pursue that separately.
    - Anne: Do people want to be able to target these differently, with CSS? If they are created from different registries

        - Justin: doesn't feel like a real use case. For your own component, you wouldn't get in that scenario. For micro-frontends, people will target those in the container.
        - Anne: if identity doesn't matter too much there, maybe it won't matter for equality either.
        - Justin: do we have usage counters on these APIs (isEqualNode?)
    - Anne: this is good - I guess the one thing would be for people to look at the issue, and review what's there. The PRs will be updated.
        - Westbrook: lets make sure all the links are visible and possible to get to
- Return to Reference Target
    - Dan: reference target IDL should be nullable? It gets into weirdness, if you set an id to null, it becomes the string "null". Feels like a pitfall, especially if you want to check if it is null (falsy). The question really is how do we unset this value. Leaning towards making it nullable. Other takes?
        - Keith: is this specified as reflection of the attribute? It has to be limited to known values, right?
        - Dan: it won't reflect elements, just what you set it. The next issues get into more interesting cases.
        - Dan: it sounds like no disagreement, so we can close that out.
    - Dan: reflection behavior for IDL properties like `el.ariaLabelledByElements`. These can be set and return back actual elements. Some examples in the issue that show some complex behavior with a combo control. With a broken reference target. If you set one of these IDL properties, should we return null? You can think of interesting cases where the shadowRoot changes. Should the platform keep a host reference around? You can think of analogous cases where if you move elements between shadow and light dom, those references will work. Is it okay that the platform works this way, or do we want to do something different here?
        - Ryosuke: here you have a reference target, and you are setting the value with IDL?
        - Dan: instead of establishing the reference with an id, I can set it dynamically with the reference to the shadow host element. It's combining reference target with IDL properties. I can have a general reference target, but the reflection isn't obvious.
        - Dan: potentially when we have the spec written, it will be more obvious, but are there any red flags on the behavior here?
        - Dan: we're taking how they work today, and we're just resolving similar to event retargeting, in a way that doesn't leak shadow internals.
        - Westbrook: if one of the elements passed into the IDL was invalid, and one was valid, what is the behavior?
        - Dan: we should follow the same behavior, if you had two valid elements, and moved one of them out of the document.
        - Keith: if you fixed up the reference, would the order change?

        - Dan: that's a good question, I think the order stays the same, because the platform retains the order, and filters based on the state of the DOM when returned. We shouldn't change it - whatever it is doing now should be what we do.
        - Dan: more to be reviewed in the spec
    - Westbrook: how current is the version in Blink, before the spec was written?
        - Dan: you should see the behavior as suggested here. That is the way tests are written. If its not, let me know! It's probably a bug.
    - Pascal: if you don't have delegatesFocus, could there be a default here?
        - Dan: I'd be careful about tying this behavior to focus. I would be hesitant to combine them in that way.
    - Pascal: when we get to Phase 2, it feels similar as the parts case - maybe there is a shared solution here.
        - Dan: yeah, using prior art here would be valuable. TAG was insistent on not doing that if we don't have to. But yeah, if something already exists, that would be great to work off of.
    - Dan: Folks should definitely look at the issues in the Github!
- :has-slotted
    - Keith: I would direct everyone to https://github.com/w3c/csswg-drafts/issues/7922
    - Keith: we're at a fork in the road, in terms of what the desires are of the community, and what's possible in the engine. Two options on the table, a :has-slotted function, providing an argument could be a compound selector - the problem there is that it would be unprecedented to have tree traversal / complex combinators there. The alternative would be a slotted combinator, but I don't know if that is a blocker for WebKit - I believe there were concerns about a deep combinator
    - Justin: confused about this, the functional form of has-slotted, doesn't create more power, because slotted already has matching. That's why we need the functional form.
    - Keith: the top-level elements are easy, the problem becomes a design-space and scope. If we want to match more than the top level element, do we want the functional form, or a combinator.
    - Justin: I thought we always wanted has-slotted to match when slotted would match. We want them to have equal behavior, just that one targets the element in the slot, and one targets the slot.
    - Keith: if we had a slotted combinator, a lot of the questions get answered by that. The ::slotted would be effectively deprecated, because you have the slotted combinator.
    - Justin: the whole purpose of the functional form, is when you have nodes that match in the assigned nodes. It doesn't need to go further down the tree. If we want more expressive power here, that should be in ::slotted
    - Keith: some feedback has been around ::slotted having more power
    - Justin: that should be a separate conversation, directed at ::slotted. There is an association of ::slotted and :has-slotted, and we shouldn't break that.

- Westbrook: the CSS working group is interested in if a combinator can replace BOTH syntaxes, whether it would be functional or not. So that, someone could use `:has`, rather than having a dedicated :has-slotted. One of the larger questions is, how do you maintain, the limitations of ::slotted, if you were to change it to a slotted combinator, without breaking the dam of selectability. If we were willing to have a limited combinator, would that be enough, and would that remove that duplicative need here. The immediate worry, from the CSS working group is what we open up here.
- Justin: so could you do :has(::slotted(foo))
  - Westbrook: no, only :has(/sllotted/ foo)
- Keith: a lot of this isn't because of invalid syntax, the engine just doesn't do this today.
- Justin: so there is a question of selecting descendants of slotted elements.
- Justin: one issue is that the reason for these decisions just isn't written down. We don't have a canonical place to point people to here.
- Keith: if we can't, if the slotted combinator can not select deeply nested elements, then we already have the answer here. Which maybe means that the :has-slotted functional version doesn't give us anything.
- Westbrook: slotted as a combinator, there is no syntax (regardless of combinator) lets you target the presence of text. Not shipping combinator slotted, at the preference of :has-slotted, does prevent a number of issues in this issue.
- Keith: would :has-slotted, first child, or nth-child ::slotted, fix that? Or allow for that?
  - :has-slotted(:nth-child(1))::slotted(*)
- Justin: there are relevant questions in the most recent CSSWG minutes
- Westbrook: the problem space, is if I have multiple elements (a div next to a p), how do I target the second element.
- Justin: there is a capability issue here. The question is, can implementers find a way to select descendants of slotted children.
- Westbrook: it's not clear if the performance issues that have been challenging before can be solved here
- Justin: should we have a direct issue that asks this question?
- Keith: I think we need concrete use-cases
- Westbrook: Lea has these examples listed in the issue
- Keith: summarizing these would be good - and targeting assigned siblings(?) vs deep, would be a good conversation to have. Would people expect the slotted elements, how we disambiguate "div next to a p" case?
- Keith: one limited version of this, is that this works on assigned nodes, and not the tree. The take away I'm getting, is that we shouldn't pursue has-slotted functional use-case.
- Sanket: is this covering nested slots use-case?
  - Keith: yeah, both ::slotted and :has-slotted, are flattened, so they allow nested slots.
  - Sanket: but for target one node, but not all of them.

    - Westbrook: it seems like the next steps is to get the concrete use-cases.
        - Justin: do we have a clear understanding that this wouldn't be a performance problem? Do we know if this can even be done?
        - Keith: more or solid use-cases allow us to more appropriately prioritize work. Knowing that there are issues that can not be resolved otherwise, would be good to prioritize this work. E.g. :has
        - Ryosuke: concrete use-cases to motivate this feature, would be useful. Especially as we try to decide options here.
        - Keith: it doesn't feel like it is worthwhile to resolve :has-slotted functional until we have an answer here.
- Ryosuke: there is a breakout day in March - so we should discuss in an issue there what breakouts we want to do there.
    - https://www.w3.org/2025/03/breakouts-day-2025/
    -

Feb 7, 2025 | Part 1

Attendees:

- Westbrook Johnson
- Jesse Jurman
- Olli Pettay
- Keith Cirkel
- Kurt Catti-Schmidt
- Steve Orvell
- Jochen Kühner
- Alison Maher
- Justin Fagnani
- Dan Clark
- Sanket Joshi
- Owen Buckley
- Rob Eisenberg
- Alex Keng
- Pascal Vos
- …

Agenda:

1. [5 minutes] Celebrating the 2023 report
2. [20 minutes] @sheet
    - resolved by the CSSWG, proposed by Microsoft
3. [20 minutes] Declarative CSS Module Scripts
    - proposed by Microsoft
4. [30 minutes] styling
    - adopting non-constructed style sheet in Firefox

    - expanding the export parts API
    - :host and :has
    - decomposed features, is there appetite to pursue some version of this?
5. [30 minutes] theming
    - exposing shadow tree
    - ::theme()
6. Overflow? Likely not. Most likely things fall into the next week.

Time: [105 minutes] + [5 minute break]

Notes:

- Celebrating the 2023 Report
    - DSD, shipped
    - CSS slot content detection, resolved and starting to land
    - Reference Target (nee: Cross Root ARIA) resolved and starting to land
    - Scoped Registries, great new proposal from Apple is getting close to resolution
- @sheet
    - Explainer - https://github.com/MicrosoftEdge/MSEdgeExplainers/blob/efd1742b8dc082fc29b51d34adf0f099cdc2f58d/AtSheet/explainer.md
    - Kurt Catti-Schmidt - slides
        - https://docs.google.com/presentation/d/1OUr5IYQ-t4jlqPCalkTG55e8SHbkXTGCeIqv1uR1dyA/edit#slide=id.p
        - Kurt: discussions with the CSS working group recently (last week), quick moving proposal, links in the first slides
        - Kurt: took a real-world site (CNN), and found gains 0.4% (small, but that is something!); also helps with logical organization. Namely, it helps for sharing inline styles with Declarative Shadow DOM - a nice extra bonus
        - Kurt: in 2023, CSSWG accepted sheets, but with URL fragment referencing rule. There is some JS syntax around this, specifically around import
        - Kurt: when we brought up the URL fragments recently, there were concerns about local styles. “MIME types need to match for fragments”. Proposed solution is to have a “sheet” attribute on the link tag. Needs to be proposed to WHATWG next.
        - Justin: can you describe the MIME type issue
            - Kurt: if you have a fragment (can drop a link), any fragment identifier, the MIME type needs to match the rest of the URL. What had been proposed, if the href directly reference the defined at-sheet (earlier in the document), there would be an issue.
        - Steve: When you say there is two different proposals, is that for declarative CSS module scripts? (Kurt: no). Having the sheet = something, looks like the specifier.

        - Kurt: consider @sheet as completely different from CSS modules. By href you say reference some file. By using the attribute, we're pointing to "@sheet sheet1". At the time of embedding, you can determine what styles to apply.
        - Kurt: there is a potential duplication, where the href could be set to "sheet.css#sheet1" and support "sheet=sheet1".
        - Steve: how do I use this in the same document?
        - Kurt: you can have style-id, and you can `@import sheet1 from "#style_tag"`
        - Justin: that doesn't work today right?
        - Kurt: yes, that would require support in WHATWG and CSSWG
        - Justin: in HTML modules, we started digging into how things can reference within their own document and others
        - Kurt: yeah, there is some overlap here - more of that in the modules topic
- Kurt: we have two proposals - "@sheet" and "Declarative CSS Modules", they can accomplish some of the same goals, but there is overlap, and we should discuss
- Jochen: does the sheet allow multiple sheets, or global sheet?
    - Kurt: I think multiple sheets is something that _should_ work, and makes sense to support, but we have edge cases we would need to figure out.
    - Kurt: I haven't thought about the global. The global sheet is already applied, but that is an interesting idea, and a good suggestion to discuss.
- Steve: I think that, in constructed stylesheets, you can't use import. This is something we should consider.
    - Justin: the way link-rel stylesheet, they are supposed to make multiple network requests - this is one of the reasons we kicked the can on CSS imports
    - Kurt: another good non-obvious thing to look into and worth spec-ing out
- Keith: gonna be a CSS identifier - maybe it doesn't matter as much, but it would also limit you, you couldn't use "from" as a sheet name. It also gets very confusing with other syntaxes - there is the layer syntax. Importing a single sheet is fine, but there are some complexities if you define a layer in a sheet. You couldn't assign multiple sheets in a single layer (?). It would be the first token of its kind, traditionally it has always been a URL. It is worth getting alignment with the existing stuff? CSSWG will probably have more thoughts here.
    - Kurt: link-rel could support more sheets, but other parts would need to be thought through
- Justin: This is the first instance of exporting a named export on the JS side. We should be careful about what we consider to be in the named

export. In the userland, people will export class names, people are going to want to do that, so we might want to make an early proposal for exporting named variables from JS modules, and how we handle collisions (if there would be).
        - Keith: Presumably, the userland CSS modules is a bit of plunge, since you can only export certain things. Sheet is the answer to that no?
        - Justin: the use case is important, what people want to do is export a className, and have that be shared everywhere. That use-case is important, and we should try to make that spec-able.
    - Steve: When I look at the “What about inline styles”, it looks like a general “Sharable styles” solution. For example, “@scope” rules. When you squint, it looks like a more general “sharing styles” solution, and maybe that’s beyond scope, but that could be really compelling.
        - Justin: this is a really common thing people will want to do with other things, like “mixins”
        - Keith: looking at a stylesheet with a JS API, you get stylesheet objects back, and if we closed the scope down, maybe this is the interface for getting those keyframes, etc with the OM
        - Justin: why is sheets special, if we can use OM
        - Kurt: sheet is a little special, in that it is a sub-file
        - Justin: I don’t view sheets as special, because if you look at user-land solutions, people are importing these other attributes, like classNames, mixins, functions, and they aren’t importing sheets (which don’t exist).
        - Justin: do we need some sort of export rules? Maybe there is a broad topic on “what is exported from a stylesheet”
        - Steve: We had importing modules in weird ways, before we had a spec, and if we could get to some parallel here, we could get to a good place
    - Westbrook: what are next steps?
    - Kurt: I have a lot of action items, we should document a lot of the things that were brought up here, that all need further discussions. I can open threads on these different topics. Imports, exports, multiple stylesheets, etc need more discussion offline.
- Declarative CSS Module Scripts
    - Kurt - another slide deck https://docs.google.com/presentation/d/1zSIF71F59PBfI6blYH26r98RtsAiUvbdDfhkvyNl4XU/edit?usp=sharing
    - Kurt: original proposal had script tag with CSS syntax - TAG said that @sheet felt like a better solution. Another piece of feedback, for CSS modules, “we don’t want to spec this just for CSS, we want this to work for HTML modules and WASM modules”. It became a much more broad proposal - which makes sense, we don’t want to dig a hole for just CSS, we want to think of a holistic picture.

- Kurt: There is a lot of demand to solve declarative shadow dom. People don't want to have separate files, or to use script tags.
- Steve: I think that, the use-case for this, was not stated correctly. Right now, in imperative shadowDOM you can use adoptive stylesheets to share stylesheets. There is no way to do this in declarative shadowDOM. Embedding a link element does not share that stylesheet. I think it is very clear, this is just a missing feature, that there is no way to share a stylesheet declaratively.
    - Kurt: yeah, all the solutions today do not truly "Share" stylesheets.
    - Steve: a good test would be, if you add a rule to a stylesheet, does it update all those existing elements?
- Justin: in support of streaming, you could imagine bundling, a requirement would be a stylesheet that is deeply nested in an SSR served bundle. Any solution here will require a global namespace, so that you can reference across shadow boundaries. Maybe that is why "@sheet" is appealing, because that lives in the global namespace. I think fundamentally we are seralizing javascript references. I think generalizing it is too big of a problem.
    - Kurt: yeah, the key difference here is the scoping. If that is the big difference, that might justify the difference here. The module map is already global, and is a good argument
- Dan: the namespace for module map and identifier involves scoping questions. Any kind of DOM level of scoping could be a non-starter. Important consideration for the module-map here. Declarative ShadowDOM true way to share styles - I'm assuming (thinking about the @sheet case), if I pull that @sheet, is that pulling a copy? Is that the case?
- Ollie: having external files is very important for performance, browsers are set up for this
- Steve: Defining styles in a script tag feels odd… I like the idea of using the module graph. Potentially if the scoping thing can be "you can use the module graph to export whatever you want". A single solution would be really desirable.
    - Kurt: a lot of the other modules feel scripty, so that's the preference on script tag
    - Steve: as-is, people don't like inline style or script tags because the syntax difference in the same document
    - Kurt: script tag does mean that the styles won't be applied
    - Kurt: there is a tooling problem too, we need to have those IDEs know the syntax difference
    - Justin: are users writing this by hand? Is this only for tooling?
    - Steve: we should not be specifically build something just for tools
- Jesse: template tag would be a good inert element that can have style / script tags. Lots of comments in the chat also express that this feels like a worthwhile idea
- Justin: +1 to Ollie's comment, external files and cache can give us deduplication, and in the SSR use case. One big problem with SSR is how not to double send, and if we can get that, that would be a good selling point.

    - Steve: to me, if we put all this together, and populate the module graph with @sheet, and expose the OM there. That gives use a way way to access directly rules and attributes from CSS. That could be the type of thing that could go in a tree-shaking tool.
        - Justin: I have an issue open for that, for tree-shaking based on reference
    - Kurt: I'll update the explainer. We're still fleshing out the shape of this.
- Styling - Adopting non-constructed style sheets in Firefox
    - Keith: one of the restrictions on adopting style sheets, if it is not constructed, it can throw an error. Cross-document stylesheets need to throw an exception, but I don't know if that needs to exist on constructed sheets. I think we can loosen the restriction.
    - Steve: why was this important to not allow in the first place? Why is cross-document an obvious no-go.
        - Keith: I have asked around, and haven't been able to get a good answer, there's no historically good answer, it was probably a defensive answer.
        - Keith: in the firefox code-base, there are a lot of places that assume a cross-document stylesheet is non-constructed. There is a lot of code to change here, and probably has a high cost of solving vs simply solving for same-document constructed sheets initially.
        - Steve: we have a lit user, who wants to share elements across documents, because you try to adopt a stylesheet and it throws
        - Justin: comes up with the picture-in-picture API, which breaks
        - Steve: some of the initial restriction, was around getting strange URLs. base-url was added later, and maybe we can lean on that.
        - Keith: those are great ideas for tests - will help people who are motivated to write tests here.
        - Westbrook: one point made in the chat was the support for @import, not sure off-hand why. Is that a blocker?
        - Dan Clark: not for any technical reason, other than it wasn't obvious how they should work. The @import creates an entry in the module graph, and it could lead to chains of imports in the graph. In order to not block releasing CSS module graphs, we decided to block the ability to do this.
        - Justin: there might be some desire to fix this - see https://github.com/w3c/csswg-drafts/issues/6130
        - Justin: maybe we should change the behavior of import, maybe there is a middle ground?
        - Jochen: was it an issue that constructed stylesheets could be modified.
            - Steve: they should support it
            - Dan: constructed style sheets that have been adopted would not be updated
    - Keith: the implementation in Firefox, removes the assertion that throws, when constructed is set. The thing that I'm looking at, which of the existing mechanics break that would cause this assertion to be important. Could we surgically add those restrictions.

    - Keith: Ryosuke - do you know any off-hand
        - Ryosuke - the things discussed before, and specifically cross-document makes sense.
    - Westbrook: we should build on the web platform tests, to validate the behavior in cross-platform documents
- Styling - Expanding export-parts API
    - Steve sharing https://github.com/WICG/webcomponents/issues/1051
    - Steve: The context is, part goes one level. If you do this, in a way where you want to make everything themable, you end up with a ridiculous large attribute value. There was a proposal to export with "*" to export all named parts with "*" or prefixed with some namespace. It would make this much easier to recommend to developers. It was part of the original spec, and just hasn't been done. Is there developer pushback on a technical level?
        - Dan: you may get some pushback on the micro-syntax here. It might not be avoidable. How far does exportparts="*" alone.
        - Ryosuke: could the wildcard appear only at the end, or could that appear anywhere?
        - Steve: I would imagine you could do that anywhere
        - Ryosuke: we'd have to handle if a regex matches a part multiple times, e.g. a-* and a-b, it matches it twice - do we export both names?
        - Steve: yeah, you can export different parts with the same name or vice-versa. That's something that exists today. It's almost certainly worth documenting how this is expected to behave, and spec-ing that out.
    - Sanket: could we consider other keywords, like "all" or "parent" or "ancestor"? That might reduce the complexity here.
        - Keith: we probably couldn't do keywords, because it would be ambiguous if it was a part name, or one of these keywords.
        - Sanket: the comma separation should allow us to define unique syntax here
    - Keith: there is a potential performance cliff, as you get larger and large exporting parts. We need to be judicious about how we use star. They need to be computed, at runtime, and could be an issue.
        - Steve: the list of parts could also be dynamic, which could require a deep search.
        - Steve: today, export parts is a live feature today.
        - Keith: it is somewhat bounded
        - Ryosuke: if you update a node 1000 nodes deep, you need to update all the ancestors
        - Keith: with star, this becomes a live map, just to compute and re-validate those attributes, causes an explosion of re-computation.
        - Ryosuke: you can quickly get an N to the N situation
        - Keith: star existing introduces a pathological performance issues.

        - Steve: the use case, I'm rendering a list, and the list changes, and it has a bunch of parts on it, that's probably a bad case. Invalidating the tree every single time it updates would be unideal.
        - Keith: there isn't something happening in the CSS engine, so we can't work with those optimizations.
    - Steve: is it the case that the return on investment is "some", and the concern is strong enough here, that we should try think about the options here more.
        - Olli: this might be too powerful a feature, and while super useful, probably worth thinking through the performance issue.
- Styling - :host and :has
    - Westbrook opens web platform tests, looking at the wpt/css/css-scoping
        - https://wpt.fyi/results/css/css-scoping?label=master&label=experimental&aligned&q=host%20has
    - Westbrook opens a codepen with `:host(:has(a))`
    - Westbrook: the host has internal, is supported in chrome, but not firefox and safari, and chrome does not support the opposite use-case.
    - Westbrook: it's not spec'd, so we need feedback on this
    - Olli & Ryosuke: need to talk to colleagues
    - Westbrook: it would be good to hear from implementors in a month about what the expected behaviors should be, and if there is something that needs to be spec'd here.
    - Sanket: what's the exact use-case here?
        - Westbrook: if you've been slotted content from a user, in the case you have a direct child, and you want to act a specific way because it is there. E.g. receiving data from child elements (rather than visual elements). I can also give more expansive demos here.
        - Sanket: it sounds like the slotted content?
        - Sanket: `:host(:has(a))` the colon has is matching in the light dom, where the `:host:has(a)` is matching in the shadow DOM.
        - Steve: has supports a sub-tree, it would be independent of slotting. My use-case is if a decedent has a checked checkbox.
        - Steve: the second use-case is 10x more justifiable
        - Justin: I have justification for both. Light dom has is valuable for slot assignment. Maybe has-slotted would take care of this, but there are things you can express, that you can't currently express with slot. The second one is important for dynamic DOM and you do want to style the host differently, and this would be immediately serialiable for SSR. Right now, if you want to react to that, you have to style a wrapper or react to the change in the host attributes. It's been something I've needed a lot.
        - Sanket: what's the canonical use-case for host-has without brackets.
        - Westbrook: it's about styling the host when it has an element in it.
        - Sanket: is there a distinction between author / consumer?
        - Justin: just needed this use-case, having a component with a checkbox and styling the wrapper.

- Ryosuke: we should also plan for discussions / meeting for W3C session in March

Q2 Face-to-face