Web Editing WG TPAC 2024 Meeting - 26 Sep 2024

Participants

-1 Lower Level - Catalina 5

(Please add your name!)

- Etienne Noël (Google)
- Johannes Wilm (IE, Chair)
- Sanket Joshi (Microsoft)
- Tomasz Jakut (IE)
- Siye Liu (Microsoft)
- Xiaoqian Wu (W3C, Team Contact)
- Kagami Rosylight (Mozilla, observing)
- Dan Clark (Microsoft)
- Gary Kacmarcik (Google)
- Olli Pettay (Mozilla)
- Anne van Kesteren (Apple)
- Christine Hollingsworth (Google)
- Ryosuke Niwa (Apple)
- Simon Pieters (Mozilla)
- Ayu Ishii (Google)
- Marijn Kruisselbrink (Google)
- Di Zhang (Google)
- Gardner (Apple)
- Sean Feng (Mozilla)

Remote Participation
<https://www.w3.org/events/meetings/952b3813-f07f-43fb-b3ff-f9ff8d59955f/>

Agenda

<https://github.com/w3c/editing/issues/469>

Minutes

Topic: EditContext

w3c/edit-context#103

Dan: which elements do we want to allow for editContext. We made previous selection - and that was not enough (<figcaption> didn't allow it). Question is which ones. We started out with all HTML elements, and then removed specific groups where it wouldn't make sense. I think we need to remove a few individual ones. We can't rely on HTML categories to select these tags. What does group think?
I removed a number of different ones. I think Ruby should have them. If it's not rendered, then it just won't receive.

Anne: why do we have a restriction at all?

Dan: To avoid weirdness with strange elements. MIght also be compatibility issues. Thinking of a safe list. Try to start from list of all HTML elements, eliminating those where it doesn't make sense.

Ryosuke: May be an idea to refer to a list of elements with shadow dom.

Dan: These are included, but then we have some extra. Unfortunately, I think there will be a list of exceptions, unfortunately.

Anne: … generally we should rely on these list of tags for implementation requirements. I think we should make a list of elements and then write that this matches these categories.

SImon: Excluding elements may exclude usecases we haven't thought of. Such as contenteditable on style element.

Johannes: We had actually thought of that. And were a bit afraid of the consequences.

Anne: Ideally, we figure out what the actual thing is, then we can just allow everything like contenteditable. We need to figure out what the behavior for contenteditable anyway.

Simon: Would be good to be in line with contenteditable. As we will need to figure the behavior out for contenteditable anyway.

Dan: summary is out. Every table-related element is out. Is it ok to exclude these?

Anne: Sounds reasonable but I will defer to Ryosuke and Megan

Ryosuke: I think td and th should be allowed. The editontext does not directly insert dom elements. So it should be possible to do something reasonable.

Anne: If contenteditable on tables isn't interoperable, then probably a good idea to exclude here. Do we really need a list?

Dan: eventually interoperable. Until then it's good to hold back.

Ryosuke: …

Anne: If it primarily depends on layout and not element type, we do we have a list?

Ryosuke: How about flex/grid?

Dan: We should test it. I think it's difficult to exclude based on layout. What do we do if the layout of an element is changed? Do we kick the editcontext off?

Olli: What do we actually test?

Dan: caret positions, getTargetRanges()

Ryosuke: We check the positions and see if they match.

Anne: I don't mind a list if we write the tests?

Johannes: we are not sure editcontext whether the DOM changes or the changes on the page that happen in the element that is associated with the editcontext to be. You could have some random element
that is associated with editcontext and then render everything in totally different elements. So list of allowed ones might not effectively stop users from using context on other elements.

Dan: Next category figure/fieldset/menu/ol/ul . Are they interoperable?

Sanket: Ol/Ul seem similar to table, probably should leave them out. Not sure how to think about figure.

Johannes: for figure, you put contenteditable false then on the fig caption you put the contenteditable true. To avoid the figure is a mathematical formula instead of an image

Simon: what's the problem with figure?

Johannes: typically to avoid people to write outside of the caption

Simon: we just need to revisit the list over and over because people are trying to do tings that we didn't think of

Anne: not point spending a lot of time on this.

Johannes: No point in having browser devs spend a lot of time developing things that are not generally used.

Dan: Ok, so conclusion: yes, to figure. No to the remaining ones.

Anne: Test needs to see what bounding boxes are like in duiffernet layout types.

Ryosuke: + selection extension. Really good to have a test for selection API as well.

RESOLUTION: we go for safe list (see github comment)

w3c/edit-context#96

Dan: [explains issue and discussion at previous meeting]

We will ask for each code unit. In case of a grapheme cluster, the JS will need to give back four times the same values if it's the same grapheme cluster

Anne: "character was unfortunate choice

Dan: problem is: User may use their own font and complicated characters, and they may be rendered apart or together.

Anne: Across unicode revisions it changes what is a grapaheme cluster. I think code point would be nicer, but code units is more consistent with what we have otherwise. We don't have a way good way of measuring code points, so we should go with code units. As long as you put in some links to infra standard (that defines code units, etc.).

Dan: **Resolution**: clarify that unit is code unit. And link to infra spec.

execCommand/ContentEditable
#468

Demo: https://software.hixie.ch/utilities/js/live-dom-viewer/saved/13129 , https://software.hixie.ch/utilities/js/live-dom-viewer/saved/13130
Johannes: [explain the issue]

Anne: will it be a big UI change?

Olli: not for Firefox

Johannes: it’s not following the spec because it’s using an input so insert where it’s supposed to only insert text. But it’s inserting an entire element and the element comes out of nowhere

Anne: we can maybe change the event because it matches the platform behaviour. You should focus what event it should be instead

Johannes: ok. Maybe we should then have a special kind of

Anne: what happen when you do execCommand and then you start typing

Johannes: the thing is execCommand isn’t used, I mean, so I dont really know what’s happening, so we would need a different one

Anne: sorry I don't have enough context about this event. Perhaps we could have a format event? That might be a separate API that like typing start… reset the typing start to match the parent or something

Ryosuke: firing those events and adding a new type of event. A new API to get that typing information. Then editors have to get updated. I think this could defer to like an Edit or Maintainers. Case 1. You are using contentEditable, you want to retain the behavior that matches the platform; but if the App is, implementing their own code, they have their own text editing, editcontext,

Simon: in Firefox, the behaviour is different, depending on whether there is remaining text before the cursor

Megan: basically a style is a different tag, When you don’t have any text that element does exist

Anne: when the users starts typing, they migh expect it to be italic, but then the editor may prevent it. Once the element disappears from the tree, they also want to be able to reset the cursor, to be upright, when the user starts typing, they know it won’t be italic, and that requires a different solution, that’s why we might need both, or something else?

Olli: let’s say in Chrome, which adds this bold back, can you query that? Next time I’m typing here, will I get this extra bolder?

Ryosuke: query command should be done, state would return

@@

Simon: consider the idea of sending a format event when you start typing again, wouldn't you also need to send a format event when you delete the last character? So that's to send the states that the format is now gone

Johannes: when you delete the text, the delete, the last character which deletes the entire bold element, that's something you have on that event, a target range

Anne: but that's no format change

Johannes: no, so in deletion, if you did, it's basically the event you get, the whole bold thing is being deleted, and insert and deletion work slightly differently with the input type, so if you delete an entire element, that includes formatting

Anne: when you get like the deleted content, it does remove the bold, but you dont get a format change event, so the format is still the same

Johannes: I understand the underlying model works differently

RN: one thing we could do in the browser is to leave the element in a contentable, one thing we could consider doing is just not delete in the target range, it will not include the bold element, next is to figure out is this a thing browsers can implement

Johannes: sounds good

Megan: +1 to the idea of keeping the bold and italic element

Simon: the cursor style depending on the actual DOM, not the internal editing state
… we will need to send a delete event when you move the cursor

Johannes: the idea of with the contentEditable was that you can cancel all the before input event and do your DOM changes and you can rely on the browser not making its own dom changes that are not going through beforeinput events.
…

RESOLUTION: Chromium/Webkit will check viability of leaving <b>-element in place and sending delete event when caret moves.

#470

Johannes: What happens if contenteditable=true/plaintext are nested?

Dan: The outer one takes precedence. You can reset with contenteditable=false

Ryosuke: Same true for Webkit

Sanket: We should write this down in a spec. Current definition looks incomplete: https://html.spec.whatwg.org/multipage/interaction.html#editing-host

Resolution: follow model of outermost wins. File issue with HTML spec (Olli takes responsibility).

Dan: In both examples below, they should be richtext.

Resolution: Bug will need to be filed with Chromium (Dan)/Webkit (Ryosuke).

Topic: Input Events

w3c/input-events#159

Sanket: I think problem is larger in that dom changes differ.

Johannes: I think this is just about being consistent in how target ranges are collected.

Ryosuke: Difficult in Webkit as we have not yet implemented non-canonicalized selection …

Sanket: Yeah, also in Chromium

Ryosuke: Could imagine fancy editor doing different thing in bidi or so

Johannes: I wonder if it is about wording.

RESOLUTION: Olli follows up with Masayuki

w3c/input-events#158
RESOLUTION: We wait for #468 before going further with this.
w3c/input-events#156

RESOLUTION: for relevant input types and textarea, the default action should update the value and it's cancelable. Browser makers find out which input types this should apply to (Olli will test).

w3c/input-events#161

Tomasz: When doing beforeinput cancelltion, it also cancels other related things such as autocorrect, two spaces give a perido, etc.

Sanket: We should keep to existing model. Maybe add a second APi to do auto correction. Would that fix it?

Tomasz: Only in this case.There are other things like auto-capitalization, etc.

Dan: What if you had three events?

Johannes: What if the editor doesn't update the dom on one of the first two events? Could this not be solved with EditContext instead?

Sanket: It still makes sense to me. It only happens on specific keys. … Maybe something to put on EditContext.

Sanket: I think we need a way to trigger the autocorrect.

Dan: Same privacy issues (on Mac)

Ryosuke: At the same time we need to support this.

Johannes: Ryosuke, do you agree that it should no t be an issue in EditContext [explain]?

Ryosuke: Yes

Olli: But we also need to support this in other places - like textinput, etc.

Siye: The event should still be triggered, even if prevent default was called on an enter-event.

Dan: That is risky. We need more time. Difficult to fix without privacy issues.

RESOLUTION: We try to think about a solution. Table until next meeting.

Dan difference to spell checking: less privacy issue as the JS cannot see wha words are underlined and the user has to decide

w3c/input-events#162

Sanket: Sounds fine to give access to HTMl when pasting even in plaintext.

Dan: How about input?

Johannes: We prefixed all the types of inputTypes to make it possible for an old editor to handle new input types with the same prefix.

Sanket/Ryosuke: We could add another attribute that would not cause the same web compat issues.
https://w3c.github.io/uievents/#idl-inputeventinit

Dan: Another option is to only deliver plaintext.

Dan: If we can do whatever, we can add new input type. But webcompat is an issue.

Johannes: second best is attribute. Where could that live.

Sanket: we could put it into the init where isComposing lives.

Olli: Maybe put it in the data transfer?

(Olli: what is the order in DataTransferItemList? Could ctrl+shift+v put text/plain first? https://mozilla.pettay.fi/moztests/paste_contenteditable.html Firefox gives only text/plain when doing ctrl+shift+v. text/html is there without 'shift'. Chrome doesn't have .dataTransfer with insertFromPaste ? Tested on linux. The test is about normal contenteditable, not plaintext-only)

Topic: Clipboard API

https://github.com/MicrosoftEdge/MSEdgeExplainers/blob/main/ClipboardAPI/clipboard-change-event-explainer.md

w3c/clipboard-apis#227

Rohan: Discuss some proposed changes to the clipboardchange event
What is this event? Fired whenever the clipboard is changed, inside or outside the browser
This allows the app to respond to these changes
We already have a basic spec.
We'd like to talk about the proposed changes.

Usage scenario:
(1) In remote access, synchronizing the clipboard between the local and remote machine.
To make a seamless experience
(2) rich web editors. in a WP you might have multiple case format options and want to show those options to the user. Only allow the available options. DIsable those options which are not available.

Olli: Wondering if it could ever be implemented in Firefox. What about other browsers?
Megan: Not sure if Webkit would do it either.

Olli: Only implemented in Chrome, and they have a Permission.

Etienne: This could be done by polling, right?

Olli: Not on FF because it requires a user interaction to use.

Sanket: What mitigations could be in place to make you feel better about it.

Etienne: Your concerns are mostly privacy-related?

Olli: Also, with the current Permissions model in FF (and Safari?) this would not be safe to implement.

w3c/clipboard-apis#225

Rohan: In the spec the clipboardchange event, the event is only fired when the user has docs on the webpage.

Proposing that the events are still allowed as long as the window had focus within the last 5 seconds.

Firefox already supports has a timeout associated with user gestures.

To restate: For the clipboardchange event, you need to have transient user activation.

Olli: What is the proposed model

Rohan: Event won't fire if you don't have focus. But they want it to be able to fire for 5 seconds after the user leaves focus.

Olli: There should not be transient user activation if the page doesn't have focus anymore. As soon as focus is lost user activation should expire, if it doesn't that's a bug in HTML.

Sanket: If I understood Olli's question correctly, we check for user activation and don't fire if it's not present.

Maybe it's 2 separate things in this issue, we can focus on the user activation.

Olli: would you require transient user activation, would the event fire?

No

Sanket: Are we OK with the event firing if we have transient user activation.

Olli: ...maybe.. but it's a very different model than what Firefox and Safari do

Megan: Safari has a pop-up, so this wouldn't really work. Expect users to have clicked on a "Paste" button.

Sanket: The clipboardchange event does not contain the clipboard contents, it only indicates that the clipboard contents have changed.
Relevant info for potential mitigations.

Since this is mostly useful for identifying clipboard changes that occur outside the user agent, when is the event fired? Does it wait until focus is returned to the page?

Current spec - clipboardchange event must fire when the user agent regains focus.

Mostly useful in the case where you can access the clipboard (with permission) without a user gesture. So this is less useful for FF and Safari.

How did this even get into the spec in the first place!

Gary: It was added to the spec at the same time as the Async Clipboard APIs were added, however it was not implemented at the time.

The spec currently states that clipboard read permission is based on the result of a user interaction with a "Paste" element activated.

Async clipboard demo:
https://glitch.com/edit/#!/async-clipboard-text?path=index.html%3A1%3A0

Example: On iOS, if JavaScript initiates a paste, the OS pops up a small paste button to get confirmation before allowing the paste.
Keyboard combos are considered to be the same as the user opting into a paste.

Etienne:Is that the same on iOS with an external keyboard?
Yes, because there is user interaction there.

Sanket: We need to think more about how this could be done in a way that could be supported by FF and Safari. Perhaps we can brainstorm for a bit.

Sanket: Safari already has a permission pop-up, could that be leveraged for this?
Megan: Maybe

Sanket: Olli do you have thoughts on the permission
Olli: Since we don't have permissions for paste, it would be challenging. We think our current model is the best for clipboard, so it would be hard to come up with an alternative.

Olli: Since there is only 1 implementation, it should be removed from the spec

Sanket: Actually 0 implementation, so that makes sense to remove.

Sanket: What about parallels with Camera access? You ask for permission for that?
Olli: Yes
Sanket: You don't do it for clipboard, because instead of permission, you prefer to rely on a user action
Olli: Yes and it's short lived. Because the info is soooooo privacy sensitive.

Sanket: What about a permission for the clipboard change?
Olli: Not useful if you can act on that event and access the clipboard.

Johannes: Could you show a notification to the user?
Olli: that's not what the event is for?
Johannes: But could you use it for that?
Olli: The user isn't trying to paste anything at that time, so it doesn't really make sense.

Sanket: Re: comparison with Camera permission model
If you block the camera API, then the camera does not work
In this case, if you block the permission, then the event doesn't happen.

Ryousuke: What is the use case for it, if you can't access the clipboard.

What about having a new permission for something like: "Monitoring the Clipboard". Would that be reasonable?

Ryosuke: I don't think so.

Megan: On iOS, each JS-initiated clipboard action results in the pop-up. There is no 5 second window. It applies to each action.

Megan: iOS does have some site-specific permissions (like Geolocation) that will last for some time (24-hours), but there is no option to grant a site permission in perpetuity.

Megan: Speaking hypothetically. In theory, if there was a site-level permission (like Geolocation) would that make sense for this? Yes. But we don't think such a permission is something we're going to add.

Sanket: Is that something we could reconsider? It would allow privacy to be controlled by the user and allow this feature to be added.

Sanket: Is there room for discussion about this.

Olli: Basically, we'd need to change our model for Paste.

Sanket: I don't think you'd have to change your model, but we'd all have to agree on a permission for this.

Scenario:
Permission to allow clipboardchange
Website tries to read clipboard, an (OS) "Paste" bubble pops up, which grants access

Olli: You know nothing about the clipboard, you can request types.

Gary: Are the types needed?

Sanket: Is it worth continuing with the other issues?

Sanket: Probably not. This is important. We need to digest this info and revisit.

Olli: We should make sure to remove the clipboardchange event from the spec.

Megan: Wenson says you can get clipboard type info without a permission, but this doesn't work for custom types.

Later clarification: This requires a simple user action. For read access, a click allows you to prompt for access (with native UI)

So you can only get clipboard types as part of a paste

Sanket: So something like this could work (in theory) for native types.

*** Welcoming Wenson and bringing him up to speed. ***

Relevant scenario:
Clipboard change event
Get clipboard types
Update the UI elements

Megan: Is there any appetite in Webkit for having a paste permission model (for the websites) like what we have for native apps?

Wenson: The reason why we have this per-session model is that we don;t know if you've copied malicious text from a website, and paste into a doc. It might be fine today, but maybe tomorrow you have a password on your clipboard, so you might not want to share it

Megan: Compare to Geolocaiton where it's only available for 24 hours. How about that? Or is this something we ever want to do.

Wenson: We certainly don't want the permission to stick around. Probably allow it as long as the clipboard hasn't changed

Megan: Not useful for this particular case.

Wenson: What about reading custom data

Sanket: Let's say yes for the sake of argument

Wenson: That has a bunch of risk associated with it.

Sanket: OK. How about restricted to native types? Would that be acceptable?

Megan: My suggestion was for a time-based permission. Would we be OK doing this for a period of time. I'm not necessarily advocating for this, just exploring whether there would be an appetite for this. If not, we don't need to discuss further.

Wenson: Lots of privacy concerns. Hard to make end-users understand the implications.

Wenson: It's worth exploring, but we need to be cautious. We would try to expose the most restricted set of information as possible.

Wenson: We're not talking about exposing any info. Just that the clipboard has changed

Wenson: That seems fairly safe.

Olli: Concerns about how useful this is.

Sanket: Maybe we should explore options further for this.

Sanket: Any further questions

Kagami: If we can get the type when we get user activation, would it still require a clipboardchange event? (or the scenario of showing the available menu options)

Sanket: If the clipboard changed after the menu was shown.

Marijn: But the clipboard change would have been initiated by a user action.

Johannes: You could show this menu after the paste, right?

Sanket: Yes

Rohan: Can we look at it from the perspective of a native app?

What protections do we have there? Can we duplicate that? Have a similar model for websites?

Johannes: And with that, we're done with clipboard.

Johannes: No need to discuss the remaining issues because they are dependent on resolving the above discussion.

Topic: Selection API

w3c/selection-api#331

w3c/selection-api#176

Di: solving the problem with selection across shadow trees
… the current spec status is that it was resolved to add a new function, getComposedRanges
… it will give selection as a StaticRange of the two endpoints, or a rescoped version of the selection
… 176 is to request a change in the parameter from a rest parameter to a dictionary
… the first reason is that this is more in line with other APIs, like highlightFromPoint, which interact with shadow roots
… the second reason is this keeps the API more flexible if more options are wanted in the future
… the other issue seems to have support from WebKit and Gecko (and Blink)
… looking for feedback to resolve this

Sean (mozilla): breaking change?

Di Zhang: I understand that this was shipped, but getComposedRange is not fully specced at this point, marked as experimental on MDN
… it also has not passed full review yet

Ryosuke: we're open, but there is the possibility of web compat issues

Mason: could be specced as rest param or options bag, for compat?

Anne: could try it and see

Ryosuke: probably okay; this is a very recent addition to the selection API

Sanket: Resolve to make the update in the spec. Webkit can try to change their implementation and come back to the WG if there are issues?

Daniel Clark (MS): PR changes getComposedRanges signature, but further down in the spec, the actual definition references shadow roots variable arguments

Anne: leave a review comment

RESOLVED: Update the API in the spec to take the options parameter. Webkit can try to change their implementation and come back to the WG if there are issues.

w3c/selection-api#180

Di: building on top of previous issue; we'd like to recommend a new option called selectionRoot to the options for getComposedRanges
… was in the original proposal a few years back
… so that the returned StaticRange has endpoints which are always descendants of the shadow root
… because rich editors want to get selection information only within the component they care about
… receiving endpoints from unknown shadow roots was counterproductive
… so that selection roots are always available, even when a custom element is contained within an ancestor shadow root

Ryosuke: when you specify the shadow root, it automatically [?]

Mason: deepest/innermost shadow root is enough
… if you only provide selectionRoot, it will automatically populate shadowRoot

Ryosuke: if you do specify shadowRoots, it doesn't modify that?

Mason: I think it adds to what you specify; selectionRoot will add one more
… this came from the discussion in 2021; I think it was yours, Ryosuke
… in an editor, you know your own shadow root – it should be easy to provide just that one root and the other behavior falls out from that

Anne: how does this work? does it scope the selection in some way?

Di: Yes. Given you have an endpoint, if the node is inside the selectionRoot, no rescoping needs to be done. If it's before the selection root, it's rescoped to be at the start. If it's after the selection root, it's rescoped to the end.

Ryosuke: before start and end could be anywhere in the DOM tree; with this, we automatically adjust the start and end so that it's within the root

Mason: motivating use case is GitHub comment editor; if you start a selection inside the editor, the editor doesn't care if you drag outside the editor to the end of the page

Daniel: selection already seems to be limited to the editor

Ryosuke: only limits editable region

Anne: might need this if you have contenteditable?

Ryosuke: no, selection doesn't span beyond contenteditable region
… selection is segregated between different contenteditable

Mason: can start a selection outside and drag it inside

Anne: that doesn't work in Safari but works in other browsers
… in Firefox, if you have a bit outside the contenteditable area, you can start a selection there and drag it inside the contenteditable area, and end up selecting both

Daniel: In Safari, I was able to select

???: you can't have a partial selection through a contenteditable

Simon: GitHub comment area in Firefox for me seems to be a textarea

Ryosuke: contenteditable does the same thing

Anne: not in Firefox; you can do partial

smaug: shadow root which is an ancestor of selectionRoot (just a node), would the method give only endpoints which are inside the selection root

???: Correct

smaug: okay, so selection root always scopes
… let's say you just pass a selection root, you can scope the selection

Mason: Correct

Simon: is the selection behavior useful outside of editing?

Mason: that was the use case I heard

Simon: if Safari and Chrome agree on scoping selection for contenteditable=true, and Firefox doesn't, we could change that

Mason: in all of them you can start the selection outside and select across, so selection root would still be useful
… this makes writing an editor a little easier

… regardless of whether the user selected the entire page, the selection is still inside the DOM that they understand

Ryosuke: that's a strong behavior, for the site to only act on a portion of the selection because it's inside a shadow root

Mason: if the user selected the entire page and click Bold, the editor should only bold the editor contents

Ryosuke: the fact that you can select the whole thing seems like a bug, in that case

Mason: as a user I'd like to be able to select and copy the whole page, even if part of it is contenteditable

smaug: you can; it selects everything

Daniel: if I click Ctrl+B then, the editor shouldn't explode because it's trying to process the whole document

Ryosuke: I think Bold shouldn't do anything, because you selected the whole region

Simon: like if you select something that includes a textarea and backspace, it shouldn't delete its contentes

Anne: did Chrome get developer feedback asking for this feature?

Mason: it was a suggestion from Ryosuke, which seems reasonable to me

Anne: we should leave it for now, and see if we get more questions later
… it doesn't seem like we have a very good story

Ryosuke: I don't recall suggesting this, but if I did, maybe I should retract it
… there doesn't seem to be a valid use case where this would be useful

Mason: from Chrome's POV it's fine to wait

Anne: people can write a library for this, if they want to
… we can evaluate if there's a lot of libraries and we'd like to lighten their load by adding it to the browser
… it's not a new primitive

RESOLVED: wait on this until we have developer feedback w3c/selection-api#169

Di: this was opened by Sean and has had multiple conversations, there's a comment at the end that summarizes
… three big questions
… we are currently prototyping for getComposedRanges, but there are some changes we need to make to allow selections to be across trees
… getComposedRanges gets the current selection StaticRange
… first: output of getComposedRanges
… currently it returns a list of one StaticRange
… in spec, a StaticRange is valid if the following are true: start and end are in the same node tree (and other things)
… that is no longer true – potentially in different shadow trees
… first option: change StaticRange to allow start/end in different trees
… second option: new StaticComposedRange in different trees in same doc
… third option: Add new “shadow-including valid” definition.
… fourth option: change getComposedRanges to return multiple ranges, one per tree

Ryosuke: returning multiple ranges is my least favorite solution
… have to wrestle with dozens of different ranges, each on a different tree
… have to have discontinuous ranges, too, if a selection starts in one shadow root and ends in another shadow root in slotted content, for example, you can have part of shadow host children selected but not the whole thing

Anne: you don't get multiple ranges, but you do need to do some of that with the masking, when you have a selection that goes inside the shadow tree and call window.getSelection()
… getSelectionRange can only return portion that is outside of the shadow tree

Ryosuke: range which represents the selection which crosses the shadow tree
… the point of getComposedRange is to get the information inside the shadow tree

Anne: you do, though, still need to do some of the partial computation

Mason: agree that #4 is bad
… re. #1, this sounds best and what it should have been in the first place
… is there compat risk?

Sanket: the intent was about disconnected trees
… highlight API doesn't make sense for disconnected trees

Anne: should highlights work across shadow boundary?

Sanket: should

Anne: valid is only used for highlights
… option 3 seems interesting but we should consider that once there is a caller
… "option 5" is don't do anything, leave defn of valid as-is because the only caller (highlight) doesn't span the boundary
… whatever we return from getComposedRanges isn't "invalid" aside from that

Ryosuke: we could rename "valid"

Anne: could call it something opposite of "shadow-including"

Simon: what is the compat risk with option 1?

Anne: would have to change highlight

Simon: valid wasn't exposed?

various: no, just a spec concept

Anne: it is observable via highlight

Daniel: can change highlight so it continues to do the same thing, file a separate issue to evaluate changing its behavior

Anne: I don't see the need for a change, except maybe valid is a confusing word
… keep it as-is until they decide they want to change it

smaug: in our implementation, we return invalid StaticRanges

Sean: it can have nodes in different trees

Ryosuke: keep spec dfn as-is, and just let getComposedRanges return "invalid" ranges
… might be misleading to authors, if we don't rename it something clearer

Anne: we probably want to look into changing highlight (or not)

Daniel: maybe the dfn of valid should move into highlight

Anne: should ranges be able to cross the boundary?

Simon: should file a spec issue against highlight to consider this

smaug: this is complicated and highlight API doesn't have the issue we have here
… you can reorder the visual representation using slots

… Ranges, otoh, look at the DOM trees
… the start/endpoint might not correspond to the visual

Ryosuke: wouldn't that problem already exist today?

smaug: it exists in getComposedRanges
… when to use flat tree traversal?

Mason: fundamental problem: any new API should use flat tree, because that's what the user sees and how the selection is painted

Ryosuke: I think the problem of StaticRange not covering whole tree, and only covering partial tree, already exists because you can start a selection from a sibling of a shadow host, and end inside slotted content
… depending where the slotted content appears in the shadow tree, two discontiguous ranges can be highlighted

smaug: browsers are inconsistent about how they behave in that case
… toString gives DOM stringification, not visual stringification
… this is an existing issue, which we should fix

Mason: toString should return the thing that is painted
… there is no such thing as a discontinuous endpoint

Ryosuke: do we need to update the definition of before/after to support the shadow tree?

Di: yes; that's point #2
… existing spec uses before/after for start/end

Ryosuke: those dfns need to be updated to shadow-including before etc
… do we need to change that in the DOM spec?

Di: in both Selection API spec and DOM spec

Anne: we also wanted to make the relationship between selection and the range it owns more explicit
… when you have a selection and it owns a range, mutating the range mutates the selection

Ryosuke: that is wonky right now because selection needs to internally have a live range

Anne: the other algorithm needs to be updated

Ryosuke: we don't want to change the behavior of Range objects, seems like a web compat issue anne: maybe there should be an opt-in way to have a shadow-including range, or composed range

Ryosuke: what are the use cases for web devs?

Mason: I thought that was StaticRange, expand it to handle across shadow roots

Ryosuke: now talking about extending regular range behavior to optionally cross shadow boundary, e.g. with ctor argument

Anne: keep live composed ranges as an internal concept, only spread StaticRange to new APIs
… not too happy with live ranges, even though we have to continue using them for this purpose
… that's a lot of typing

smaug: what do we do if you mutate a cross-shadow-DOM selection?

Sean: clear it, pretty much
… there is a use case to expose live composed range, so that pointInRange method can use flat tree order

Anne: I could imagine if you have use cases for ranges today, you also have use cases for composed ranges
… the problem with live composed range is we have to do these updates, but we have to figure them out anyway
… it seems weird if once you mutate a shadow tree, it works differently to mutating the normal tree
… we don't necessarily have to expose that API to web developers; we could just give them static ranges and then give bounding rect APIs for those, including across the boundary

Ryosuke: what use cases require live ranges which cross the boundary, beyond static ranges?

Anne: we have one – selection

Ryosuke: is there any point in exposing that as a DOM API?

Mason: no use case should require a live range; you can do anything with getComposedRanges and set base/extent
… which updates a live composed range, but one that's not exposed

Ryosuke: it seems tricky to implement the exact same behavior
… even if you use a MutationObserver, your selection API using static ranges might potentially see state before the ranges are updated, between when the endpoints are updated due to the mutation and when the observer fires

… I don't think there is anything you should be able to do with live composed ranges that you cannot do with static composed range

Sanket: that's the direction we've been trying to go, nudge away from live range

Anne: cost is high, but there is an ergonomic issue, and potentially the issue Ryosuke mentioned
… though editors usually tightly control their own node trees
… if that's not the case, we'll hear about it

Sanket: if we expose it to Range, we can't take it away; if we start with StaticRange then expanding is always an option

Sean: flattened tree order: what if end boundary is an unslotted node?

Mason: should behave as though it's out of the document

Sean: but it doesn't have an order

Mason: so it's as if the two endpoints are in different documents; it's invalid

Ryosuke: with shadow trees, you can have endpoints that are not in the flat tree

Anne: you're saying a child of the shadow host that doesn't end up anywhere, and when stuff does get slotted, the actual contents of the <slot> element

Ryosuke: should behave the same as if display:none were applied
… today if you apply display:none we don't treat such selection as invalid
… it just happens to start after that element
… we should do the same thing here

Anne: selection uses the layout tree, or layout info, to compute what actual contents are

Ryosuke: for painting, you have to have some logic to determine where the selection starts

smaug: contents is coming from the range

Anne: if you select around a display:none thing which contains the word "test" and toString, does it contain "test"?

Sanket: selection.getRangeAt(0).toString() would

(https://mozilla.pettay.fi/moztests/reorder.html something to try. Select something and check ^ toString() in web console)

Ryosuke: selection does skip display:none

Mason: when the endpoint is not in the flat tree (e.g. unslotted light DOM content), proposal is to act as if the selection contains the entire shadow host?

Ryosuke: walk the tree in the direction you're intending to walk, find the first node that is in the shadow tree

Anne: we only have to care about Selection toString, because the live range isn't crossing the boundary
… static ranges don't have toString
… apparently Selection toString is already somewhat magic

smaug: in live DOM, all the nodes you select are in the live DOM and you can reorder them with slots

Anne: then it just does the normal tree walk? Range toString just does a tree-order traversal

smaug: Range toString returns DOM representation even though visual representation is different

Anne: already possible

Ryosuke: just regular tree walk, concatenate whatever is visible

Anne: we wouldn't have to touch toString

Sanket: first thing was: whether we update the dfn of valid
… agreed on not changing dfn for now, I will file CSS highlight issue
… second issue: Di, do you want to offer a resolution?

Di: proposed resolution: new concepts of before/after which are composed-aware, and use that internally but not expose it?

Anne: need to have live composed ranges, and adjust tree mutation algorithms to account for those
… for now they would be only internal
… why do we need to update before?

Di: before/after are used to compare boundary points, used by setStart, setEnd, …

Ryosuke: whether start comes before end, etc
… selection API needs to do before/after in composed trees

Anne: in order to mutate live composed range? okay
… live composed range concept, plus algorithms, and Selection API needs to be updated to use those while not messing up and exposing more info than it should through its public Range object
… traversal is the Selection toString case, not sure we have a resolution

Di: also relates to comparing positions

Anne: not sure; that one primarily looks at layout

Di: say start is in document as normal, end is unslotted, how can we compare those?

Ryosuke: just compare in the composed tree

Di: it doesn't exist in the flat tree; it's unslotted

Ryosuke: composed tree and flat tree are different – everything in the DOM exists in the composed tree
… you have to do a conversion from composed tree to flat tree, which doesn't exist today
… that's where display:none, display:contents, etc apply

Anne: how do we not have a conversion from composed to flat? isn't that what flat tree computation is?

Ryosuke: we can convert the composed tree to flat tree, but can't convert point in composed tree to point in flat tree

Anne: so you need some adjustment for the endpoints, okay

Ryosuke: someone will have to write a spec for that when we spec toString behavior

Sanket: want to introduce live composed range concept; update selection API to use live composed range concept

Ryosuke: define new before/after for composed range
… as for selections where endpoint are not in flat tree, do traversal to find first node in the flat tree and use that – must be after the start for the end, etc

Sanket: next: Selection::containsNode() mismatch

Di: right now, if you do a selection.getRangeAt(0), you can isPointInRange, StaticRange doesn't have that
… would we want to add isPointInRange?

Daniel: should it be a composed tree traversal?

Di: probably, yes

Ryosuke: in addition to regular range?
… why not also consider comparePoint and intersectsNode?
… if you are adding isPointInRange, those are equally useful

Sanket: proposal is to move them up to AbstractRange

smaug: behavior is different
… we want something that uses flat tree

various: composed tree

Mason: (1) can we change existing APIs, (2) should we have a new API which definitely uses the flat tree, if we can't

Ryosuke: doesn't seem web-compatible to change to using the flat tree
… for consistency reasons, isPointInRange etc should continue to use composed tree, because that's what these APIs currently do

Mason: should we add new APIs that use the flat tree, since that's more useful?

Anne: if we offer that synchronously, it requires layout
… we've tried not to expose flat tree for that reason

Ryosuke: definitely requires updating style for selection
… what are the use cases where this would be handy?
… the only difference is if an element is unslotted shadow host child, or fallback slot content

Mason: or when you rearrange things so the last DOM element gets pulled to the first via a slot
… a node could be outside selection in composed tree, inside in flat tree

Ryosuke: in all those cases, because the selection will follow the composed tree order, wouldn't it be more useful to use the composed tree to get this answer?
… if nodes are swapped due to slotting, what's selected is what's in the middle in the composed tree, not the flat tree

smaug: no, what's selected is what's in the flat tree – what the user sees

Mason: agreed with smaug; if a user sees a selection which contains "a", then it is in the selection smaug: yes, browsers are broken today

Ryosuke: it would mean that we have to support discontiguous ranges, because that's the only way you can select a visually contiguous region

Anne: not necessarily – if you have this magic function that computes from points in the composed tree to the flat tree, then you have a range in the flat tree

Ryosuke: that's not what the selection reflects, as argued – only when you can select a visually contiguous region when the slots rearrange the order, … [didn't catch this entire statement]
… consider a concrete example: nodes A, B, C
… A and C are swapped in slots: C, B, A
… if you select from C to B, in the DOM sense that selects B and C

Sanket: still can be contiguous within the range

Ryosuke: A, B, C, swap to B, A, C
… select B to C, select will look discontiguous to the user
… in order to select from end of B to end of C, you have to select A and node C, but that requires that you have discontiguous selection regions
… we could entertain that route, but that would have massive implications for the rest of the API

smaug: same argument we've had at Mozilla; I've been arguing for the flat tree (better for the user, even though composed is easier)

Megan: multi is specced, but nobody correctly or fully implements it
… it would be very powerful or useful, but huge rat's nest

Ryosuke: don't want to change the entire model of selection for this particular case

Sanket: for that particular case, let the API behave as it would today?

Ryosuke: but then we go back to the original question: isPointInRange, etc., I don't see when you want to have a function which checks the intersection in the flat tree because that's not what is selected
… WK and Blink don't have that capability today

smaug: Blink seems to select using flat tree, but uses composed tree for stringification
… don't know what WebKit does

Megan: not that, just DOM order only

Anne: for painting purposes and for computing final text, does it have to be discontinuous?

… can't we, once we have those points in the flat tree, paint that bit?

Ryosuke: but then, what does it mean to copy that text?
… you have to walk in the flat tree to copy the text?

Sanket: selection would be hard to do
… we'd have to do something much more difficult to make that possible

Ryosuke: consider A, B, C; swap A, B
… if you try to select from A to C, in logical DOM order, that is two discontiguous nodes
… they are visually next to each other, so if you select in flat tree, they're selectable
… but in DOM, you need two different ranges pointing at A and C

Mason: or you add the intervening node, and just accept more stuff may become part of the selection

Sean: then you don't need to have multiple ranges

Megan: but then you need a flat tree range

Sean: that's the argument – using flat tree provides better user experience

Ryosuke: but the flat tree version of the range cannot represent the DOM state, because DOM is in the composed tree, not the flat tree

Sanket: not too different from positioned stuff, where it's hard to do such a selection
… similar concept, just in shadow DOM

Ryosuke: bi-di and flexbox can also change order

Megan: with bi-di, if you want a bi-di selection, that would require multiple selections, unless we have another concept of flat tree selection, which is a different concept

Ryosuke: definitely not easy to implement

Sanket: within this concept of one contiguous range in the composed tree, is there a solution that would be meaningful for isPointInRange?

Ryosuke: I don't see a use case for it. All the copy operations etc are done in DOM order, not flat tree order.

Megan: can you give a concrete example, or is this abstract?

Sean: I asked because I assumed we were going to change to flat tree order. If we continue to use composed tree order, there's no need to change.

Megan: maybe flat tree selection is something we should think about because it would be a better user experience? but it's a giant pot of spaghetti. I dream of fixing this.

Ryosuke: using flat tree for selection endpoints is an interesting idea – it avoids the problem of multiple range objects to manage, but looks visually contiguous to the user

Anne: except you have to translate it back to composed tree

Megan: have to figure out what the interface is

Ryosuke: selection is really on the flat tree, then if you are trying to toString, you just walk the render tree

smaug: looks like Chromium is using flat tree for copy/paste, but stringifying uses composed tree
… the browsers are inconsistent

Megan: using flat tree for selection and copy?

smaug: seems that way

Anne: doesn't that mean Selection toString leaks the shadow tree?
… toString also needs to be amended to accept shadow root parameters?

Mason: [showing example with slot reordering]

Megan: would love for that to work consistently, but everything to this point has been using DOM ranges
… I guess Blink has already switched? Can talk to them about it. Want selection on the web to be more intuitive.

Mason: getComposedRanges, if we do that, the point is to allow shadow roots to work correctly
… I'm hoping we don't go halfway, and do it right

Sanket: one point was to put isPointInRange on StaticRange, no specific use case, correct?
… proposed resolution: don't need to do that?

Anne: need to file an issue on Selection toString, because it might leak the shadow tree

Ryosuke: toString in general needs to be specced more

Mason: maybe there is opportunity to make it work correctly?

Megan: +1

Anne: what are the next steps here? are we going to investigate flat tree selection?

Megan: would like to do homework, interesting to know that Blink is doing that
… need to understand more about what's going on with shadow roots

Ryosuke: if we're solving this problem for shadow roots, might as well solve it for bidi

Megan: there are so many things, where if you could just have visual selection work on the web, it would be good

Johannes: column selection in tables? would that also be affected?

Ryosuke: wouldn't work, because start and end are part of separate sections, even in the flat tree

Simon: would need multiple ranges for that

Mason: same is true of flexbox reordering

Ryosuke: could start from one point, and what's visually in between them

Anne: related Range object ends up stringifying to something nonsensical

Ryosuke: sure, but that's the least of the issues

Johannes: sounds good, but if we overhaul the whole thing and still don't cover column selection or flexbox, that's something to be aware of

Ryosuke: for column selection, have a way to select a <col> element?

Sanket: custom element tables, things which look like tables but aren't?

Ryosuke: then what you need is geometry-based selection

Megan: don't know if you can do that without multiple selections

Anne: also with columns, you don't know if someone is trying to select all the rows, …
… need some kind of different input

Megan: need to keep in mind all of the problems we need to solve

… but if we can advance it without precluding solutions to the more advanced questions, we have to start somewhere

Anne: flat tree thing, follows the model we have today and solves a problem

Megan: want to spend more time thinking about it, understand what is going on with Blink

Ryosuke: when you do editing operations, that's where the problems lie

smaug: do we need to modify getComposedRanges to deal with flat tree ranges?

Ryosuke: would need something dramatic like disabling contenteditable in such places

Simon: there's a proposal for CSS reading flow; have we discussed how selection works when you reorder with reading flow

Anne: only influences focus direction

Mason: and a11y tree, but same spirit as selecting across flexbox

Simon: shouldn't affect selection?

Mason: agreed we shouldn't boil the ocean; the flat tree thing is useful; the rest can wait

[adjourned]