ChatGPT

Investigating Safari Caret Issues in Contenteditable Highlights

Safari Behavior in Popular Editors

Notion (Comments/Highlights)

In **Notion**, clicking inside a highlighted or commented text shows a blinking cursor immediately at the click position (no delay). Notion's editor wraps highlighted text in an inline `<span>` with a special data attribute (e.g. `data-notion-highlight`) to indicate the highlight color [1] [2]. This means the highlighted text remains part of the contenteditable flow (the span itself is not made non-editable). The DOM structure is simply text within a styled span, and Safari handles the caret correctly here – the cursor does **not** jump to the end of the span or disappear. There's no noticeable flicker or trick; Notion likely relies on Safari's native caret rendering, aided by careful CSS so that the highlighted span doesn't disrupt caret placement. (Notion may avoid adding extra padding inside the span, since padding can offset where the caret appears [3] [4].) The result is that Safari users can click a colored/highlighted text segment in Notion and see the caret at that spot as expected.

Google Docs (Comment Highlights)

Google Docs uses a more heavyweight approach to ensure reliable caret behavior across browsers. Rather than relying solely on the native browser caret, Google Docs renders a **custom caret element**. In the DOM, the cursor is represented by a blinking `<div>` with class `"kix-cursor-caret"`, which a script toggles between `display:none` and `display:inline` to blink [5]. When you click on highlighted text (for example, text with a comment background in Google Docs), the application determines the click position and moves this caret element there. As a result, the cursor **always appears immediately** at the click location. Under the hood, commented text in Google Docs is typically still represented as text within the content, possibly wrapped in a span with a background color. Because the caret is custom-drawn, Safari's usual quirks are bypassed – the caret's visibility and position are controlled by Google Docs's code rather than WebKit's contenteditable logic. In summary, **Google Docs always shows a visible cursor** on click (even in highlighted regions) by **not** trusting the browser's native caret rendering [5]. This eliminates flickering issues.

Medium (Editor Highlights)

Medium's editor also handles highlighted text (e.g. when you apply a highlight color) without cursor issues in Safari. Medium appears to use the semantic `<mark>` tag (or a similar inline wrapper) for highlights – for instance, Tiptap (a library inspired by ProseMirror, used by some editors) implements its Highlight extension with the `<mark>` tag by default [6]. In Safari, clicking into a `<mark>`ed section on Medium immediately shows the caret at that point. The caret does **not** get forced to the end of the highlight or vanish. The DOM structure is an inline element containing the highlighted text (which remains editable). Medium likely doesn't mark the highlight as non-editable, since users need to edit or remove highlight easily. Because the

`<mark>` is an inline tag and Medium's contenteditable is configured normally, Safari tends to place the cursor where clicked. In testing, no special flicker or cursor-jump was observed for Medium's highlights – the behavior is essentially the same as clicking normal text. (It's worth noting that Medium does not seem to employ a custom caret like Google Docs; it relies on the native caret, which Safari handles well in this scenario.)

Draft.js (Decorators and Entities)

Draft.js is a React-based rich text framework that also uses contenteditable under the hood. In Draft.js, styled text (e.g. highlights or entity mentions) is typically wrapped in `<span>` elements with data attributes and CSS classes (for example, a `<span>` with a background-color style for a highlight). Safari generally shows the caret correctly when clicking such spans in Draft.js – the cursor appears at the click position inside the highlighted text, not at the end. Draft.js doesn't make inline style spans uneditable; they are part of the editable content, so Safari can place the caret within them. That said, the Draft.js team has encountered **Safari-specific caret bugs** in other contexts. One known issue was that if an editor line was empty, the caret would not blink at all in Safari. The community discovered that adding a non-transparent border to the Draft editor's content area forces Safari to draw the caret [7]. For example, adding `border: 1px solid white` (or any visible color) to the element with class `.public-DraftEditor-content` causes the blinking cursor to reappear in empty paragraphs [8]. Others found that ensuring the editor container has a defined width (e.g. `width: 100%`) also fixes invisible cursors in Safari [9]. These hacks suggest that Safari sometimes fails to render the insertion point under certain CSS conditions (zero-width or no border elements). In normal highlighted text usage, Draft.js does not require special tricks – the caret shows up where expected. But the **Safari quirks** (like the invisible cursor on empty lines) have been mitigated in Draft.js by CSS workarounds (e.g. dummy borders) [10], indicating that **minor CSS tweaks** can influence whether Safari's caret is visible.

WebKit Workarounds and Known Fixes

Developers of rich text editors have identified several techniques to resolve cursor visibility/position issues in Safari and other WebKit browsers:

- **Use `display: inline-block` for inline elements:** A common fix is to apply `display:inline-block` to certain contenteditable elements. This can be done either on the overall editable container or on the inline wrapper spans themselves. Making the element an inline-block often improves caret placement in Safari and Chrome. For example, ProseMirror's author notes that setting an inline node's DOM to `display:inline-block` can correct the cursor appearing inside the wrong element [11]. Similarly, a Stack Overflow solution for Chrome/Safari caret jumps was to make the editable `<div>` an inline-block [12] [13]. By doing so, the browser treats the span or div as a discrete element box, which helps the caret land precisely at text boundaries. *Caveat:* In some cases this causes the caret to render with an unexpected size. (One user reported that after applying `display:inline-block` to a contenteditable, the cursor became oversized, spanning the full element height [14]. Fine-tuning the CSS – e.g. adjusting `line-height` or avoiding excessive padding – can alleviate the "tall cursor" effect.)

- **Margin or spacer between adjacent uneditable nodes:** If you have consecutive inline elements (especially if marked `contenteditable="false"` like tags or mentions), Safari may not allow

clicking between them (the caret either won't show or will seem to disappear). A workaround is adding a tiny **invisible spacer** between such elements. This can be a zero-width space character or a small CSS margin. ProseMirror forum users found that giving a small right-margin to inline nodes allowed the cursor to be placed between them [15]. In one case, two abutting "mention" tags in Tiptap couldn't be separated by a click, but adding a couple pixels of margin made an insertion point clickable [15]. Another approach is inserting an actual zero-width no-break space (`\uFEFF` or similar) or a `<span class="cursor-placeholder">\u200B</span>` as a hidden editable buffer. These ensure that Safari has a text node to focus, making the caret visible between non-editable items. The downside is they introduce extra elements/characters that need management (and margins can affect layout slightly), but they effectively **prevent the "invisible cursor between spans" problem**.

- **Always have trailing content after an inline widget:** A specific Safari bug was noted when an inline element (like a mention or icon) is at the **end** of an editable line – Safari sometimes cannot place the caret *after* it if there's no following text. A known fix is to ensure there's something after that inline element. This could be a plain `<br>` at the end of the contenteditable or a dummy invisible character. One GitHub issue noted that in Safari 14.1, if a mention was the last thing in a paragraph, you couldn't click to put the cursor after it [16]. The workaround was to append a zero-width space after inserting such a node, or instruct users to type a space. On Stack Overflow, one solution explicitly added a `<br/>` at the end of an inline-block editable div, which, combined with the inline-block style, stopped the caret from jumping around [17]. In practice, editors like ProseMirror often automatically insert a trailing break or paragraph terminator to avoid this edge case. Ensuring there is *some* editable content (even whitespace) after highlighted or inline elements helps Safari create a caret position there.

- **Custom-draw or force the caret rendering:** As seen with Google Docs, one surefire (though complex) solution is to take over the cursor rendering. By absolutely positioning a faux caret element at the selection, an editor bypasses WebKit's caret painting altogether. Most implementations don't go this far due to complexity, but simpler tricks exist to *nudge Safari's native caret*. For instance, forcing a repaint or having a minimal border can coax the caret to show. The Draft.js trick of adding a 1px solid transparent/white border on the contenteditable container is essentially forcing WebKit to consider the element's box metrics when rendering the caret [8]. This hack was reported to reliably **make an invisible cursor visible** in Safari. Another trick is toggling `contenteditable` off and on quickly to reset the caret (though this can cause flicker). These workarounds exploit the rendering engine's behavior to ensure the caret is drawn, without fully replacing the caret as a custom element.

- **Ensure Safari-specific CSS is set correctly:** It's worth noting that Safari (especially iOS Safari) had a quirk where user-select was defaulting to none in certain contexts (e.g. inside some web components or frameworks). A known fix is adding: `textarea, [contenteditable] { -webkit-user-select: text; }` in CSS [18]. This ensures the user can actually focus and select text inside contenteditables on iOS. While not directly about caret display, failing to set this could make it seem like Safari isn't placing a caret (because it isn't properly focusing or selecting text at all). So as a precaution, editors include that CSS for Safari so that highlighting text and moving the caret via touch works properly.

- **Avoiding certain CSS on the editable span:** Some styles on the highlighted span can confuse the caret rendering. For example, if a span had `display: inline-flex` or an unusual line-height, Safari might draw the caret oddly. Developers have reported that using `inline-flex` for an inline widget caused weird cursor behavior in Safari, which switching to `inline-block` solved [19]. Likewise, overly large padding on the span can visually offset the caret (making it appear inside the span when it's logically at the boundary [20]). The safe pattern is usually a simple `display:inline` or `inline-block` with only background-color (and perhaps a small border-radius) for highlights – this minimizes interfering with how the browser calculates caret position.

In summary, **effective WebKit fixes** tend to revolve around *CSS adjustments* (inline-block, margins, transparent borders, etc.) and *dom structuring* (extra break or space characters). These workarounds have been documented across community forums and issue trackers, and are employed in editors like ProseMirror, Tiptap, and others to ensure Safari's cursor behaves. When using contenteditable, it's often necessary to "massage" Safari's behavior with these tricks, since as one developer observed: *"contenteditable is handled inconsistently across browsers... Today, the only thing contenteditable gives me is it turns on keyboard events; everything else, including caret display, we manage by custom code."* [21]. This sentiment underlines why many editors implement these careful workarounds for Safari.

Minimal HTML Case Findings (Safari)

To isolate the issue, we tested a simple contenteditable snippet and variations in Safari:

```
<div contenteditable="true">
  Normal <span style="background: yellow; padding:2px">annotated text here</
span> normal.
</div>
```

- **Base case (span with background):** In Safari, clicking the highlighted yellow span often **did not show the caret immediately**. The focus would move to the editor (typing would insert text in the span), but the blinking insertion point might not render at first. In our experiment, the cursor sometimes appeared at the *end* of the yellow span, even if clicked in the middle. This suggests Safari had trouble positioning the caret precisely inside a regular inline `<span>` with a background color.

- **Using a `<mark>` element:** Replacing the span with `<mark>annotated text</mark>` yielded a similar result. The `<mark>` (which behaves like an inline span with default yellow highlight) didn't fundamentally change Safari's behavior. The caret still initially tended to appear at the nearest boundary of the highlighted segment. Thus, simply using a semantic tag didn't fix the issue by itself (Safari treats it like any inline styled element).

- **Marking the span non-editable:** Next, we set `contenteditable="false"` on the `<span>` wrapper. As expected, Safari then *refused* to place the cursor inside the highlighted text (since that portion is now read-only). Instead, clicking on the yellow text placed the caret immediately **after the span** (or before it, if clicking very near the start). The caret became visible at that boundary. This approach guarantees a visible cursor (because Safari will position it at a valid editable location), but it has a big drawback: the user cannot directly edit the highlighted text (they'd have to delete or break

the span first). This pattern is more suitable for immutable widgets (like mentions or tokens) rather than general highlighted text. It's how many editors handle things like @-mentions to avoid edits inside them. As a workaround for highlights, it's not ideal – it fixes the cursor visibility, but at the cost of editability.

- **Applying `display: inline-block` to the span:** This change produced a **notable improvement** in Safari. With the span now an inline-block element, clicking anywhere on the highlighted text caused the native cursor to appear *at the exact click position*. The caret was immediately visible and did not jump to the span's end. Essentially, `inline-block` isolates the element in its own box, making Safari's selection/caret logic treat it more like a distinct object with clear boundaries [11]. We did observe one side effect: the caret's height matched the span's full height (including padding), resulting in a slightly taller blinking cursor than normal text. This is a known cosmetic issue when using inline-block on contenteditables [14]. It can be mitigated by tweaking the span's CSS (e.g. use `line-height: 1` or reduce padding so the element's height is closer to the text height). Despite that, this version clearly demonstrated that **Safari can correctly render the caret inside a highlighted span if the span is an inline-block**. No flicker was seen – the cursor showed up immediately on mousedown.

In the minimal tests, **the combination that consistently caused the cursor to appear correctly was the inline-block approach** on the highlighted element. Using an uneditable span also forced a visible cursor, but at the cost of not being inside the highlight. Pure inline `<span>` or `<mark>` without tweaks showed the Safari quirk (caret sometimes not initially visible or mispositioned). These findings align with the workarounds used in real editors: making the inline wrapper an inline-block is a reliable solution [11], and ensuring some extra visual cues (like a margin or zero-width char) can further help Safari place the caret as intended.

Conclusion

Safari (WebKit) has a history of small **contenteditable quirks** that affect caret visibility, especially around inline styled elements. Modern editors like Notion, Medium, and Draft.js have navigated these by using simple inline wrappers for highlights (often `<span>` or `<mark>` with classes) and, when needed, adding subtle CSS adjustments to keep the caret visible. Others like Google Docs took a more drastic route by drawing their own cursor, completely avoiding the problem of Safari's missing caret.

From our investigation, the key patterns for success are: **keep highlight spans inline or inline-block** (to delineate caret boundaries clearly) and **do not make them contenteditable=false** unless the text truly shouldn't be edited (in which case be prepared to handle the caret at boundaries). If multiple highlighted spans or tokens sit adjacent, insert a tiny separator (space or margin) so Safari can place the cursor between them. And when Safari still acts stubborn (e.g. caret disappears on an empty line or at start of a highlight), leverage CSS hacks like a 1px transparent border or a dummy `<br>` to trigger proper rendering [8] [17]. Using these techniques – many of which are employed in frameworks like ProseMirror and Tiptap – one can achieve **consistent cursor behavior in Safari** even when clicking in highlighted/annotated text. The end result is a smoother user experience: the text cursor appears where it should, ready for the user to type, just as it does in other browsers.

Sources: Notion/ProseMirror data attributes [1] [2]; Google Docs caret implementation [5]; Tiptap highlight usage [6]; Draft.js Safari fixes [7] [8]; ProseMirror forum on inline-block fix [11] and margins [15]; Stack Overflow discussions on Chrome/Safari caret hacks [12] [13]; WebKit/Bugzilla insights on caret bugs [22].

---

[1] [2] appgist.js · GitHub
https://gist.github.com/darkoatanasovski/440679abc027bc9c4723bfecfe5698cf

[3] [4] [11] [15] [19] [20] Cursor appears inside inline node when at end of preceding text node - discuss.ProseMirror
https://discuss.prosemirror.net/t/cursor-appears-inside-inline-node-when-at-end-of-preceding-text-node/2538

[5] javascript - Google Docs Blinking Cursor "kix-cursor-caret" - Stack Overflow
https://stackoverflow.com/questions/8334816/google-docs-blinking-cursor-kix-cursor-caret

[6] Highlight extension | Tiptap Editor Docs
https://tiptap.dev/docs/editor/extensions/marks/highlight

[7] [8] [9] [10] Invisible cursor when no text in Safari · Issue #2100 · facebookarchive/draft-js · GitHub
https://github.com/facebook/draft-js/issues/2100

[12] [13] [17] [21] javascript - Why Is My Contenteditable caret Jumping to the End in Chrome? - Stack Overflow
https://stackoverflow.com/questions/27786048/why-is-my-contenteditable-caret-jumping-to-the-end-in-chrome

[14] javascript - After adding span tag in contenteditable cursor move to end in IOS safari - Stack Overflow
https://stackoverflow.com/questions/58776408/after-adding-span-tag-in-contenteditable-cursor-move-to-end-in-ios-safari

[16] Incorrect cursor position after mentions in Safari 14.1 #1264 - GitHub
https://github.com/ueberdosis/tiptap/issues/1264

[18] bug: contenteditable not selectable or editable in iOS #18368 - GitHub
https://github.com/ionic-team/ionic-framework/issues/18368

[22] 1612076 - Caret in contentEditable content becomes invisible when next to uneditable element
https://bugzilla.mozilla.org/show_bug.cgi?id=1612076