An event for CSS position:sticky

By Eric Bidelman

Engineer @ Google working on web tooling: Headless Chrome, Puppeteer, Lighthouse

TL;DR

Here's a secret: You may not need `scroll` events in your next app. Using an `IntersectionObserver`, I show how you can fire a custom event when `position:sticky` elements become fixed or when they stop sticking. All without the use of scroll listeners. There's even an awesome demo to prove it:

View demo | *Source*

Introducing the `sticky-change` event

An event is the the missing feature of CSS `position:sticky`.

One of the practical limitations of using CSS sticky position is that it **doesn't provide a platform signal to know when the property is active**. In other words, there's no event to know when an element becomes sticky or when it stops being sticky.

Take the following example, which fixes a `<div class="sticky">` 10px from the top of its parent container:

```
.sticky {
  position: sticky;
  top: 10px;
}
```

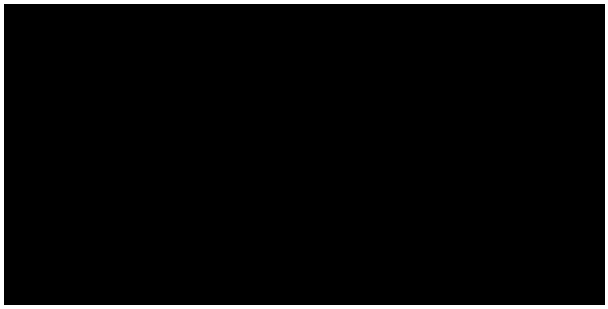
Wouldn't it be nice if the browser told when the elements hits that mark? Apparently I'm not the only one that thinks so. A signal for **`position:sticky`** could unlock a number of **use cases**:

1. Apply a drop shadow to a banner as it sticks.
2. As a user reads through your content, record analytics hits to know their progress.
3. As a user scrolls the page, update a floating TOC widget to the current section.

With these use cases in mind, we've crafted an end goal: create an event that fires when a **`position:sticky`** element becomes fixed. Let's call it the **`sticky-change`** event:

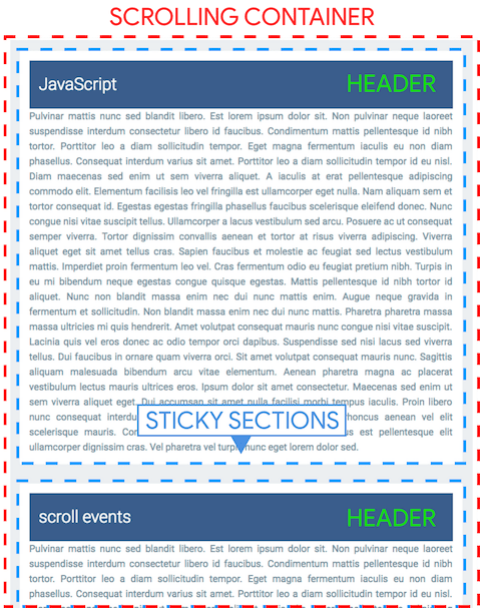
```
document.addEventListener('sticky-change', e => {
  const header = e.detail.target;  // header became sticky or stopped sticking.
  const sticking = e.detail.stuck; // true when header is sticky.
  header.classList.toggle('shadow', sticking); // add drop shadow when sticking.

  document.querySelector('.who-is-sticking').textContent = header.textContent;
});
```

The demo uses this event to headers a drop shadow when they become fixed. It also updates the new title at the top of the page.



In the demo, effects are applied without scrollevents.

Scroll effects without scroll events?



Structure of the page.

Let's get some terminology out of the way so I can refer to these names throughout the rest of the post:

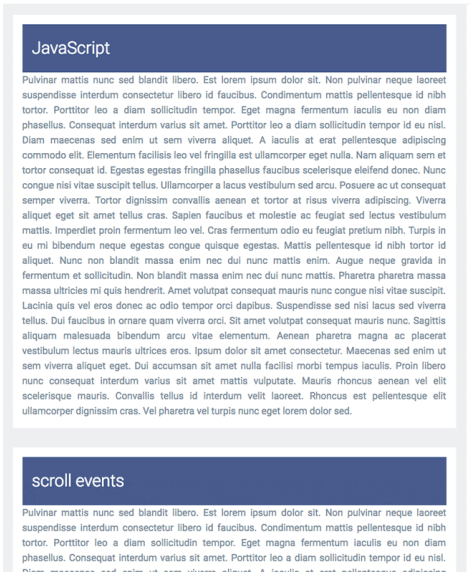
1. **Scrolling container** - the content area (visible viewport) containing the list of "blog posts".
2. **Headers** - blue title in each section that have `position:sticky`.
3. **Sticky sections** - each content section. The text that scrolls under the sticky headers.
4. **"Sticky mode"** - when `position:sticky` is applying to the element.

To know which *header* enters "sticky mode", we need some way of determining the scroll offset of the *scrolling container*. That would give us a way to calculate the *header* that's currently showing. However, that gets pretty tricky to do without `scroll` events :) The other problem is that `position:sticky` removes the element from layout when it becomes fixed.

So without scroll events, we've **lost the ability to perform layout-related calculations** on the headers.

Adding dumby DOM to determine scroll position

Instead of `scroll` events, we're going to use an `IntersectionObserver` to determine when *headers* enter and exit sticky mode. Adding two nodes (aka sentinels) in each *sticky section*, one at the top and one at the bottom, will act as waypoints for figuring out scroll position. As these markers enter and leave the container, their visibility changes and Intersection Observer fires a callback.

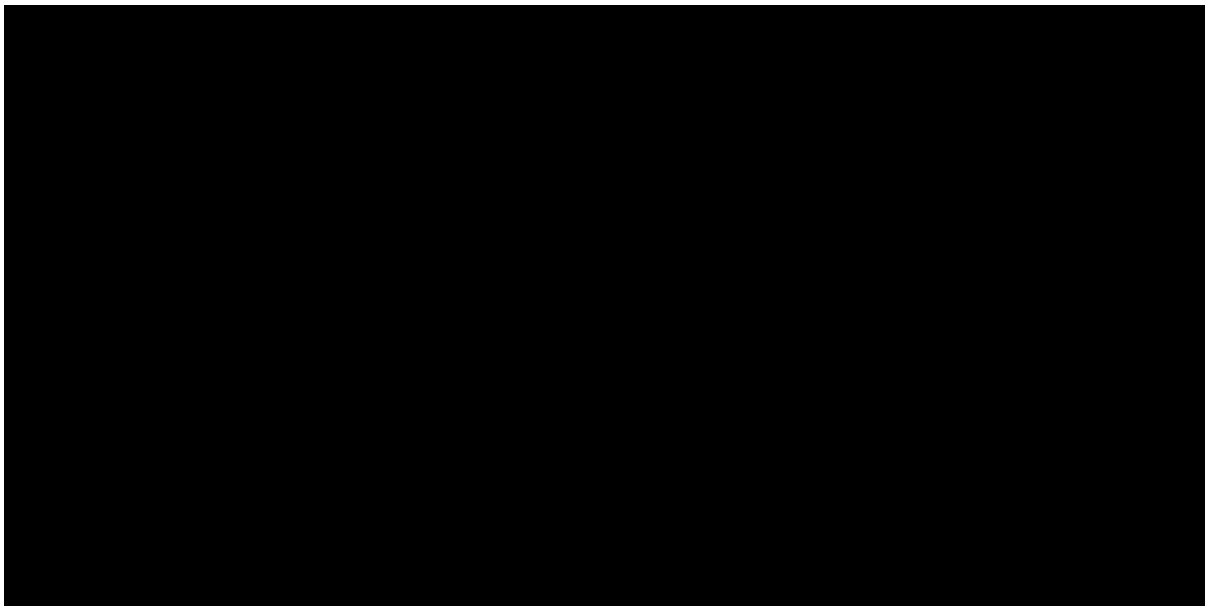


The hidden sentinel elements.

We need *two* sentinels to cover four cases of scrolling up and down:

1. **Scrolling down** - *header* becomes sticky when its top sentinel crosses the top of the container.
2. **Scrolling down** - *header* leaves sticky mode as it reaches the bottom of the section and its bottom sentinel crosses the top of the container.
3. **Scrolling up** - *header* leaves sticky mode when its top sentinel scrolls back into view from the top.
4. **Scrolling up** - *header* becomes sticky as its bottom sentinel crosses back into view from the top.

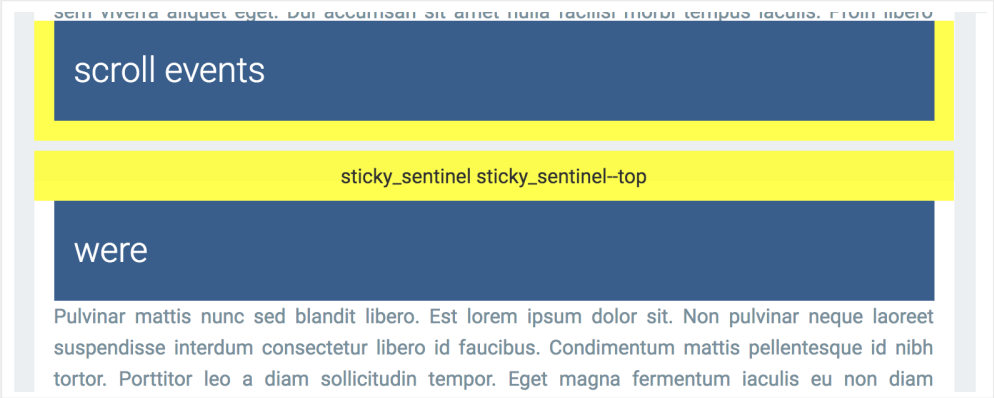
It's helpful to see a screencast of 1-4 in the order they happen:



Intersection Observers fire callbacks when the sentinels enter/leave the scroll container.

The CSS

The sentinels are positioned at the top and bottom of each section. **`.sticky_sentinel--top`** sits on the top of the header while **`.sticky_sentinel--bottom`** rests at the bottom of the section:



Position of the top and bottom sentinel elements.

```
:root {
  --default-padding: 16px;
  --header-height: 80px;
}
.sticky {
  position: sticky;
  top: 10px; /* adjust sentinel height/positioning based on this position. */
  height: var(--header-height);
  padding: 0 var(--default-padding);
}
.sticky_sentinel {
  position: absolute;
  left: 0;
  right: 0; /* needs dimensions */
  visibility: hidden;
}
.sticky_sentinel--top {
  /* Adjust the height and top values based on your on your sticky top position.
  e.g. make the height bigger and adjust the top so observeHeaders()'s
  IntersectionObserver fires as soon as the bottom of the sentinel crosses the
  top of the intersection container. */
  height: 40px;
  top: -24px;
}
.sticky_sentinel--bottom {
  /* Height should match the top of the header when it's at the bottom of the
  intersection container. */
  height: calc(var(--header-height) + var(--default-padding));
```

```
  bottom: 0;
}
```

Setting up the Intersection Observers

Intersection Observers asynchronously observe changes in the intersection of a target element and the document viewport or a parent container. In our case, we're observe intersections with a parent container.

The magic sauce is `IntersectionObserver`. Each sentinel gets an `IntersectionObserver` to observer its intersection visibility within the *scroll container*. **When a sentinel scrolls into the visible viewport, we know a header become fixed or stopped being sticky**. Likewise, when a sentinel exits the viewport.

First, I set up observers for the header and footer sentinels:

```
/**
 * Notifies when elements w/ the `sticky` class begin to stick or stop sticking.
 * Note: the elements should be children of `container`.
 * @param {!Element} container
 */
function observeStickyHeaderChanges(container) {
  observeHeaders(container);
  observeFooters(container);
}

observeStickyHeaderChanges(document.querySelector('#scroll-container'));
```

Then, I added an observer to fire when `.sticky_sentinel--top` elements pass through the top of the *scrolling container* (in either direction). The `observeHeaders` function creates the top sentinels and adds them to each section. The observer calculates the intersection of the sentinel with top of the container and decides if it's entering or leaving the viewport. That information determines if the section header is sticking or not.

```
/**
 * Sets up an intersection observer to notify when elements with the class
 * `.sticky_sentinel--top` become visible/invisible at the top of the container.
 * @param {!Element} container
 */
function observeHeaders(container) {
  const observer = new IntersectionObserver((records, observer) => {
    for (const record of records) {
      const targetInfo = record.boundingClientRect;
```

```
      const stickyTarget = record.target.parentElement.querySelector('.sticky');
      const rootBoundsInfo = record.rootBounds;

      // Started sticking.
      if (targetInfo.bottom < rootBoundsInfo.top) {
        fireEvent(true, stickyTarget);
      }

      // Stopped sticking.
      if (targetInfo.bottom >= rootBoundsInfo.top &&
          targetInfo.bottom < rootBoundsInfo.bottom) {
       fireEvent(false, stickyTarget);
      }
    }
  }, {threshold: [0], root: container});

  // Add the top sentinels to each section and attach an observer.
  const sentinels = addSentinels(container, 'sticky_sentinel--top');
  sentinels.forEach(el => observer.observe(el));
}
```

The observer is configured with **`threshold: [0]`** so its callback fires as soon as the sentinel becomes visible.

The process is similar for the bottom sentinel (**`.sticky_sentinel--bottom`**). A second observer is created to fire when the footers pass through the bottom of the *scrolling container*. The **`observeFooters`** function creates the sentinel nodes and attaches them to each section. The observer calculates the intersection of the sentinel with bottom of the container and decides if it's entering or leaving. That information determines if the section header is sticking or not.

```
/**
 * Sets up an intersection observer to notify when elements with the class
 * `.sticky_sentinel--bottom` become visible/invisible at the bottom of the
 * container.
 * @param {!Element} container
 */
function observeFooters(container) {
  const observer = new IntersectionObserver((records, observer) => {
    for (const record of records) {
      const targetInfo = record.boundingClientRect;
      const stickyTarget = record.target.parentElement.querySelector('.sticky');
      const rootBoundsInfo = record.rootBounds;
      const ratio = record.intersectionRatio;

      // Started sticking.
      if (targetInfo.bottom > rootBoundsInfo.top && ratio === 1) {
```

```
        fireEvent(true, stickyTarget);
      }

      // Stopped sticking.
      if (targetInfo.top < rootBoundsInfo.top &&
          targetInfo.bottom < rootBoundsInfo.bottom) {
        fireEvent(false, stickyTarget);
      }
    }
  }, {threshold: [1], root: container});

  // Add the bottom sentinels to each section and attach an observer.
  const sentinels = addSentinels(container, 'sticky_sentinel--bottom');
  sentinels.forEach(el => observer.observe(el));
}
```

The observer is configured with **`threshold: [1]`** so its callback fires when the entire node is within view.

Lastly, there's my two utilities for firing the **`sticky-change`** custom event and generating the sentinels:

```
/**
 * @param {!Element} container
 * @param {string} className
 */
function addSentinels(container, className) {
  return Array.from(container.querySelectorAll('.sticky')).map(el => {
    const sentinel = document.createElement('div');
    sentinel.classList.add('sticky_sentinel', className);
    return el.parentElement.appendChild(sentinel);
  });
}

/**
 * Dispatches the `sticky-event` custom event on the target element.
 * @param {boolean} stuck True if `target` is sticky.
 * @param {!Element} target Element to fire the event on.
 */
function fireEvent(stuck, target) {
  const e = new CustomEvent('sticky-change', {detail: {stuck, target}});
  document.dispatchEvent(e);
}
```

That's it!

Final demo

We created a custom event when elements with `position:sticky` become fixed and added scroll effects without the use of `scroll` events.

View demo | *Source*

Conclusion

I've often wondered if `IntersectionObserver` would be a helpful tool to replace some of the `scroll` event-based UI patterns that have developed over the years. Turns out the answer is yes and no. The semantics of the `IntersectionObserver` API make it hard to use for everything. But as I've shown here, you can use it for some interesting techniques.

Another way to detect style changes?

Not really. What we needed was a way to observe style changes on a DOM element. Unfortunately, there's nothing in the web platform APIs that allow you to watch style changes.

A `MutationObserver` would be a logical first choice but that doesn't work for most cases. For example, in the demo, we'd receive a callback when the `sticky` class is added to an element, but not when the element's computed style changes. Recall that the `sticky` class was already declared on page load.

In the future, a "Style Mutation Observer" extension to Mutation Observers might be useful to observe changes to an element's computed styles. `position: sticky`.

Last updated July 2, 2018.