Stick your landings! position: sticky lands in WebKit



By Eric Bidelman

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

position: sticky is a new way to position elements and is conceptually similar to

position: fixed. The difference is that an element with position: sticky behaves like position: relative within its parent, until a given offset threshold is met in the viewport.

Use cases

Paraphrasing from Edward O'Connor's original proposal of this feature:

Many websites have elements that alternate between being in-flow and having position: fixed, depending on the user's scroll position. This is often done for elements in a sidebar that the page author wants to be always visible as the user scrolls, but which slot into a space on the page when scrolled to the top. Good examples are news.google.com La (the "Top Stories" sidebar) and yelp.com (search results map).

Introducing sticky positioning

LAUNCH DEMO

By simply adding position: sticky (vendor prefixed), we can tell an element to be position: relative until the user scrolls the item (or its parent) to be 15px from the top:

```
.sticky {
  position: -webkit-sticky;
  position: -moz-sticky;
  position: -ms-sticky;
  position: -o-sticky;
  top: 15px;
}
```

Scroll this page.

Dignissim diam quis enim lobortis scelerisque fermentum dui. Arcu cursus vitae congue mauris rhoncus aenean vel elit scelerisque. Nunc mattis enim ut tellus elementum sagitist vitae et leo. Et molestie ac feuglat sed lectus. Egestas fringilla phasellus faucibus scelerisque eleflend donce. Sodale neque sodales ut etelan sta met nia prurus. Sed tempus uma et phaetrea phaetrea massa massa ultricies. Nunc este blandit libero volutata sed car ornare arcu. Faciliss volutpat est velt egestas dui id ornare arcu. Amet volutpat consequat mauris nunc congue nisi vitae suscipit tellus. Quis vivern bib rac pulvinar mattis nunces du. Libero enims sed faucibus trippis in. Nallamo non sis estsi atmee facilisis magna, Augue neque gravalo in fermentum et sollicitudin ac. Loreni ipsum dolor sit amet consectetur adipiscing elit duis. Tincidunt praesent semper feuglat nibh sed pulvinar proin gravid henderir. Lodi orfalisis marqis is tamet massa vita teotror. Scelerisque mauris pellentesque pulvinar pellentesque habitant. Quisque habitant. Quisque habitant. Quisque lost vivers vive congue e consequat a celle donce. Metus vulgutate eu scelerisque fella vulgare viver vive congue eu consequat a celle donce. Metus vulgutate eu scelerisque fella vulgare viver vive congue eu consequat a celle donce. Metus vulgutate eu scelerisque fella vulgare viver vive congue eu consequat a celle donce. Metus vulgutate eu scelerisque fells un de la vivera viver conque eu consequat a celle donce. Metus vulgutate eu scelerisque fella vulgare en la vivera viver conque eu consequat a celle vivera viver conque eu consequat a celle vivera vivera conque en consequat a celle vivera vivera conque en consequat a celle vivera de vivera i punt marcus. Per ordinare vivera de vivera i punt marcus en consequat a celle vivera suspendisse potenti nullam ac. Dictum sit amet justo donce min dian vulgutate. Dian phasellor subsum faucibus consequal interdum varius sit amet. Eleflend mi in null posuere sollicitudin aliquam ultrices sagittis. Bibendum est ultricies integer qu

Dignissim diam quis enim lobortis scelerisque fermentum dui. Arcu cursus vitae congue mauris rhoncus aenean vel elit scelerisque. Nunc mattis enim ut telius elementum sagittis vitae et leo. Et moleste ac feuglat sed lectus. Egestas friigilia phasellus faucibus scelerisque elefendo donce. Sodales neque sodales ut etam sit ament isal prurus. Sed tempus uma et pharetra pharetra massa massa utiriciae. Nunc eda blandit libero volutpat sed cras ornare arcu. Facilisis volutpat est veit geetas dui do rnare arcu. Amet volutpat consequat mauris nunc ecu clugue neque grovida in fermentum inhir cras pubriam mattis nunc sed. Libero en ima defiacibus trupis in. Nullam non noi set sit ai mate facilisis magna. Set alugue neque grovida in fermentum et sollicitudin ac. Lorem ipsum dolor sit amet consectetur adipiscing elit duis. Tincidunt praesent semper feuglat nibh sed pubrian proin gravida hendrerit. Odio facilisis mauris sit amet massa vitae tontro. Sederisque mentur spellentesque pubrian pellentesque habitant, Osiques (di diam vel quam elementum pubrian ertam non. Est ante in nibh mauris. Commodo nulla facilis nullam vehicula. Tellus in metus vulgutate eu scelerisque felix. Augue lacus viewar vitae congue ue consequat a felia disone. Metus vulgutate eu scelerisque felis. uctus venennatsi lectus magna fringilla uma portitor rhoncus. Fringilla ut morbi tincidunt augue interdum vellt euismod in pellentesque. Id diam maecenas ultricies mi eget mauris pharetra. Elementum nisi quis elefiend quam adipicing l'incidunt eget ultum non nis est sit amet facilisis magna. Feuglas nish petet mutace di velt ut tortor pretium vierra. Condimentum vites apple nellentesque habitant morbit trisque senectus et entess. Viliputate entim nutia aliquet optritto fraces

At top: 15px, the element becomes fixed.

To illustrate this feature in a practical setting, I've put together a <u>DEMO</u> which sticks blog titles as you scroll.

Old approach: scroll events

Until now, to achieve the sticky effect, sites setup scroll event listeners in JS. We actually use <u>this technique</u> as well on html5rocks tutorials. On screens smaller than 1200px, our table of contents sidebar changes to position: fixed after a certain amount of scrolling.

Here's the (now old way) to have a header that sticks to the top of the viewport when the user scrolls down, and falls back into place when the user scrolls up:

Try it: http://jsbin.com/omanut/2/ Z

This is easy enough, but this model quickly breaks down if you want to do this for a bunch of DOM nodes, say, every <h1> title of a blog as the user scrolls.

Why JS is not ideal

In general, scroll handlers are never a good idea. <u>Folks</u> tend to do too much work and wonder why their UI is janky.

Something else to consider is that more and more browsers are implementing hardware accelerated scrolling to improve performance. The problem with this is that on JS scroll handlers are in play, browsers may fall back into a slower (software) mode. Now we're no longer running on the GPU. Instead, we're back on the CPU. The result? User's perceive more jank when scrolling your page.

Thus, it makes a lot of sense to have such feature be declarative in CSS.

Support

Unfortunately, there isn't a spec for this one. It was <u>proposed on www-style</u> back in June and just <u>landed</u> in WebKit. That means there's no good documentation to point to. One thing to note however, according to <u>this bug</u>, if both <u>left</u> and <u>right</u> are specified, <u>left</u> wins. Likewise, if top and bottom are used at the same time, top wins.

Support right now is Chrome 23.0.1247.0+ (current Canary) and WebKit nightly.

Except as otherwise noted, the content of this page is licensed under the <u>Creative Commons Attribution 3.0</u>
<u>License</u>, and code samples are licensed under the <u>Apache 2.0 License</u>. For details, see our <u>Site Policies</u>. Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.