auxclick is Coming to Chrome 55

By Jeff Posnick
Web DevRel @ Google

When is a click not a `click`? For a web developer working on a complex user interface, that's not an abstract philosophical question. If you're implementing custom mouse input behavior, it's crucial to keep user intent in mind. If a user clicks on a link with their mouse's middle button, for instance, then it's reasonable to assume that they wanted to open a new tab with that link's contents. If a user middle-clicks on a random UI element, then it you may want to assume that it was inadvertent, and ignore that input, while a primary button click would be expected to trigger a response from the UI.

It's possible, if a bit cumbersome, to model these nuanced interactions via a single `click` event listener. You would have to explicitly check the `button` property of the `MouseEvent`, to see whether it was set to `0`, representing the primary button, versus anything else, with `1` usually representing the middle button, and so on. But not many developers go as far as explicitly checking the `button` property, leading to code that handles all `click`s identically, regardless of which button was pressed.

Starting with Chrome 55, a new type of `MouseEvent`, called `auxclick`, is fired in response to any clicks made with a non-primary button. Accompanying this new event is a corresponding change in behavior of the `click` event: it will only fire when the primary mouse button is pressed. We hope that these changes will make it easier for web developers to write event handlers that respond to only the type of clicks that they care about, without having to specifically check the `MouseEvent.button` property.

Reduce false positives

As mentioned, one motivation for creating `auxclick` was to avoid deployment of custom `click` handlers that mistakenly override the "middle-click-opens-a-tab" behavior. For example, imagine that you've written a `click` event handler that uses the History API to rewrite the location bar, and implement custom single-page navigations. It might look something like:

```
document.querySelector('#my-link').addEventListener('click', event => {
  event.preventDefault();
  // ...call history.pushState(), use client-side rendering, etc....
});
```

Your custom logic might work as intended when triggered by a mouse's primary button, but if that code runs when a middle button is clicked, it's effectively a false positive. Prior to the new behavior, you'd end up preventing the default action of opening a new tab, which runs counter to your user's expectations. While you could explicitly check for `event.button === 0` at the start of your handler, and only execute the code if that's the case, it's easy to forget, or never realize it's necessary to do that.

Only run the code you need

The flip side of fewer false positives is that `auxclick` callbacks will only run when there's actually a non-primary mouse button clicked. If you have code that needs to, for example, calculate an appropriate destination URL before opening a new tab, you can listen for `auxclick` and include that logic in your callback. It won't incur the overhead of being run when the primary mouse button is clicked.

Browser support and compatibility

This new behavior is currently only implemented in Chrome 55. As mentioned in the initial proposal, feedback (both positive and negative) from the web developer community is appreciated. Filing a GitHub issue is the best way to share that feedback with the folks who are working on the standardization process.

In the meantime, developers don't have to wait for `auxclick` to be widely available to follow some best practices for handling mouse events. If you take the time to check the value of the `MouseEvent.button` property at the start of your `click` event handler, you can ensure that you take appropriate action. The following pattern will handle primary and auxiliary clicks differently, whether or not there's native support for `auxclick`:

```
function handlePrimaryClick(event) {
  // ...code to handle the primary button click...
}

function handleAuxClick(event) {
  // ...code to handle the auxiliary button click….
}

document.querySelector('#my-link').addEventListener('click', event => {
  if (event.button === 0) {
    return handlePrimaryClick(event);
  }

  // This provides fallback behavior in browsers without auxclick.
  return handleAuxClick(event);
});

// Explicitly listen for auxclick in browsers that support it.
document.querySelector('#my-link').addEventListener('auxclick', handleAuxClick);
```

Last updated July 2, 2018.