


# Input Device Capabilities



By Paul Kinlan

Paul is a Developer Advocate

Chrome 47 has a new feature that makes it easier to understand the how users interact with your site: [InputDeviceCapabilities](#) ! Let's step back a bit and learn why this is important.

DOM input events are an abstraction above low-level input events, loosely tied to physical device input (e.g., `click` events can be fired by a mouse, touchscreen, or keyboard). However, there is a problem: there is no simple method to obtain the details of the physical device responsible for an event.

In addition, certain types of input can generate further "fake" DOM input events for compatibility reasons. One such fake DOM event happens when a user taps a touch screen (such as on a mobile phone); it not only fires touch events but, for compatibility reasons, mouse events as well.

This causes problems for developers when supporting both mouse and touch input. It's difficult to know if a `mousedown` event actually represents new input from a mouse, or is just a compatibility event for a previously-processed `touchstart` event.

The new `InputDeviceCapabilities` API provides details about the underlying sources of input events via a `sourceCapabilities` object on the `UIEvent`.

The object has a `firesTouchEvents` property that is set to `true` or `false` based on how the event was generated by the user action.

The question is: Where should this be used?

Outside of Pointer Events, many developers today handle the logic for interaction in the touch-layer, preventing Default to avoid creating "fake" mouse events in the first place. This design works well in many scenarios and doesn't need to change to take advantage of `InputDeviceCapabilities`.

But in some scenarios you really don't want to preventDefault the touch event; for example, you still want taps to send 'click' events and change focus. For these cases, the information held in the `MouseEvent.sourceCapabilities.firesTouchEvents` property allows you to start consolidating the logic for touch- and mouse-based events into a model that is *similar* to how you would manage the logic with Pointer Events. That is, you can have just one set of

code that manages the interaction logic and provides developers a simpler way to share logic among browsers that do and don't support Pointer Events.

```
function addEventListener(target, type, handler, capture) {  
    target.addEventListener(type, function(e) {  
        if (e.sourceCapabilities.firesTouchEvents)  
            return false;  
        return handler(e);  
    }, capture);  
}
```



The good news is that this has been Polyfilled by Rick Byers so that you can use it across most platforms.

Today this API is minimal, focused on solving a specific problem with identifying mouse events derived from touch events. It is even possible to instantiate an instance of `InputDeviceCapabilities`; however, it only contains `firesTouchEvents`. In the future it's expected to expand to enable you to understand more about all of the input devices on a user's system. We would love to get your feedback on use cases.

---

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

*Last updated July 2, 2018.*