Pointer and Keyboard Events

Having to code for touch, mouse and keyboard events is a pain. There has to be a way to make the code easier to work with events. The code will not be shorter (we have to work around the lack of pointer event support in mobile Safari) but it'll be easier to reason through and, in time, will become shorter when mobile Safari implements the API.

In this post we will discuss both pointer and keyboard events, how to use them together to provide a better user experience, and alternatives for browsers that don't support the API (particularly Safari)

Pointer events

Pointer events provide a unified interface for different pointing devices in different devices. So instead of adding events for touch and mouse events we can run a single pointer event that will work in all supported devices.

Current versions of Safari for both desktop and moibile do not support Pointer Events. caniuse.com indicates that Safari Technical Preview for desktop supports the API. That leaves mobile Safari as the only browser that doesn't support the API now or in the projected future.

The events available to the api are listed in the table below.

event	description
pointerover	The pointer has entered the bounding box of the element. This happens immediately for devices that support hover, or before a pointerdown event for devices that do not.
pointerout	The pointer has left the bounding box of the element or screen. Also after a pointerup, if the device does not support hover.
pointerenter	Similar to pointerover, but does not bubble and handles descendants differently. Details on the spec.

event	description
pointerleave	Similar to pointerout, but does not bubble and handles descendants differently. Details on the spec.
pointerdown	The pointer has entered the active button state, with either a button being pressed or contact being established, depending on the semantics of the input device.
pointerup	The pointer has left the active button state.
gotpointercapture	Element has received pointer capture.
lostpointercapture	Pointer which was being captured has been released.
pointermove	The pointer has changed position.
pointercancel	Something has happened and it's unlikely the pointer will emit any more events. You should cancel any in-progress actions and go back to a neutral input state.

The idea is that we replace click or hover with the equivalent pointer events so that we code the event only once. If needed, we can create different event responses based on the `pointerType` attribute if we need different responses.

At the mmost basic the script to handle responses would look like this:

```
const myButton = document.getElementById('myButton');

if (window.PointerEvent) {
  myButton.addEventListener("pointerdown", function(evt) {
    "pointerdown"// add the pointer down code here
  });
} else {
  // fall back on touch and mouse eventsddEventListener('touchstart', func
      // prevent co'touchstart'// prevent compatibility mouse events and c
      evt.preventDefault();
      // do what you need for touchstarttListener('mousedown', function(ev
    evt.preventDefault();
    // whatever you need to d'mousedown'// whatever you need to do with mo
  });
}
```

We can further refine the pointer event handler by adding cutom code based on the type of pointer device that triggers the event.

```
const myButton = document.getElementById('myButton');

if (window.PointerEvent) {
  myButton.addEventListener("pointerdown", function(evt) {
    switch(evt.pointerType) {
      case "mouse":
        console.log('mouse input detected');
        break;
      case "pen":
        console.log('pen/stylus input detected');
        break;
      case "touch":
        console.log('touch input detected');
        break;
      default:
        console.log('pointerType is empty or could not be detected');
    }
  });
}
```

This level of detail may not always necessary but it’s nice to know that we have the flexibility of targeting different types of devices in the same code.

This barely scratches the surface of what you can do with Pointer Events. Until there is consistent support for the API across browsers I will hold off doing any further work other than replace touch and mouse events.

Polyfill

Until Safari implements Pointer Events we have to polyfill the API to make sure that it works consistently in all our target browsers.

There are many polyfills for Pointer Events like PEP from the jQuery Foundation, Points from Rich Harris, and others.

I’ve chosen to work with PEP.

For the demo I've chosen to link to the jQuery CDN. In a production site I'd likely download it and link to it locally.

The first step is to link to the script so it'll be available on the page.

```
<script src="https://code.jquery.com/pep/0.4.3/pep.js"></script>
```

We then setup the elements that we want to use the polyfill.

According to the spec, the touch-action CSS property controls whether an element will perform a "default action" such as scrolling, or receive a continuous stream of pointer events.

The polyfill uses a `touch-action` attribute instead of the CSS property. For PEP to work correctly, you will therefore need to include `touch-action` attributes in your HTML that mirror any `touch-action` propertiesin your CSS.

The button below has been configured not to accept any touch events.

The output element (id = o) will hold the results of the script.

```
<button id="b" touch-action="none">Test button!</button>
<p id="o"></p>
```

Finally, the script element wull run all out code. The idea is that when a `pointerdown` event is triggered on the button, the browser insert information about the event:

- What type of device it was (`pointerType`)
- The type of event it was (`type`)
- The element that received the event (`target.nodeName`)

```
<script>
myButton = document.getElementById("b");

myButton.addEventListener( "pointerdown", (e) => {
  document.getElementById("o").innerHTML = "that was a " +
    e.pointerType + " " + e.type + " on a "+ e.target.nodeName;
} );
</script>
```

You can see the code in action in this pen

Links and Resources

- Pointer Events Level 2 Spec
- Pointing the Way Forward
- Using Pointer Events