Switching to pointer events

It is easy to think that everything will work with mouse clicks on the web. However, many devices support other types of pointing input devices, such as pen/stylus and touch surfaces so we need a way to work with all of them without writing duplicate code.

According to MDN:

> The pointer is a hardware-agnostic device that can target a specific set of screen coordinates. Having a single event model for pointers can simplify creating Web sites and applications and provide a good user experience regardless of the user's hardware. However, for scenarios when device-specific handling is desired, pointer events defines a pointerType property to inspect the device type which produced the event.
>
> The events needed to handle generic pointer input are analogous to mouse events (mousedown/pointerdown, mousemove/pointermove, etc.). Consequently, pointer event types are intentionally similar to mouse event types.
>
> Additionally, a pointer event contains the usual properties present in mouse events (client coordinates, target element, button states, etc.) in addition to new properties for other forms of input: pressure, contact geometry, tilt, etc

For this equivalency testing we'll working with basic pointer events. The idea is that these pointer events are compatible with existing code (pointer events will also fire mouse compatibility events as described in Compatibility mapping with mouse events) and will work with future code.

The following table shows the pointer events we will work with.

Event	On Event Handler	Bubbles	Cancelable	Default Action	Description
pointerover	onpointerover	Yes	Yes	None	Fired when a pointer is moved into an element's hit

Event	On Event Handler	Bubbles	Cancelable	Default Action	Description
					test boundaries.
pointerenter	onpointerenter	No	No	None	Fired when a pointer is moved into the hit test boundaries of an element or one of its descendants, including as a result of a pointerdown event from a device that does not support hover (see pointerdown).
pointerdown	onpointerdown	Yes	Yes	Varies: when the pointer is primary, all default actions of the mousedown event. Canceling this event also prevents subsequent firing of compatibility mouse events.	Fired when a pointer becomes *active*.
pointermove	onpointermove	Yes	Yes	Varies: when the pointer is primary, all default actions of mousemove	Fired when a pointer changes coordinates. This event is also used if the change in pointer state can not be reported by

Event	On Event Handler	Bubbles	Cancelable	Default Action	Description
					other events.
pointerup	onpointerup	Yes	Yes	Varies: when the pointer is primary, all default actions of mouseup	Fired when a pointer is no longer *active*.
pointercancel	onpointercancel	Yes	No	None	A browser fires this event if it concludes the pointer will no longer be able to generate events (for example the related device is deactived).
pointerout	onpointerout	Yes	Yes	None	Fired for several reasons including: pointer is moved out of the hit test boundaries of an element; firing the pointerup event for a device that does not support hover (see pointerup); after firing the pointercancel event (see pointercancel); when a pen stylus leaves the hover range detectable by the digitizer.

Event	On Event Handler	Bubbles	Cancelable	Default Action	Description
pointerleave	onpointerleave	No	No	None	Fired when a pointer is moved out of the hit test boundaries of an element. For pen devices, this event is fired when the stylus leaves the hover range detectable by the digitizer.
gotpointercapture	ongotpointercapture	Yes	No	None	Fired when an element receives pointer capture.
lostpointercapture	onlostpointercapture	Yes	No	None	Fired after pointer capture is released for a pointer.

The idea is that pointer events will work regardless of the platform. Where necessary, pointer events will fire mouse events.

Furthermore we can query the event to get more information about the type of pointer device used and react accordingly.

The HTML code

For the examples in the post, we'll use the following HTML code.

It's a simple div with a class of Box and an h1 element inside it.

```
<div class='box'>
  <h1>Click Me!</h2>
```

```
</div>
```

The first example

This first example will show some of the basics of using pointer events.

The code first checks if the the browser supports pointer events by querying for `window.PointerEvent`. If it returns true then we continue, otherwise we bail since the code will not run.

We then add an event listener for the `pointerdown` and `pointermove` events. For each, we log a message to the console. In a more complete example, we can then run the appropriate code for the event.

```
if (window.PointerEvent) {
  const box = document.querySelector('.box');

  box.addEventListener('pointerdown', (evt) => {
    console.log('Pointer down');
  });

  box.addEventListener('pointerover', (evt) => {
    console.log('Pointer moved in');
  });
}
```

Making a function to detect pointer event support

We can also abstract the feature detection into a function. It may be overkill but I prefer a function with a name that clearly tells you what the code does rather than having to remember that `window.PointerEvent` is the feature query for pointer events.

The function wraps the feature query and returns true when the browser supports pointer events and false otherwise.

```
function supportsPointerEvents() {
  if (window.PointerEvent) {
    return true;
  }

  return false;

}
```

Detecting what type of pointer we're using

A further refinement is to check the type of decide that triggered the event.

We add a `pointerdown` event listener to the box element and associate a function (`detectInputType`) to run when the we trigger the event.

`detectInputType` uses a switch statement to check the type of pointer device that interacts with our content (`pointerType`).

The valid values for `pointerType` are:

Pointer / Device Type	pointerType Value
Mouse	mouse
Pen / stylus	pen
Touch contact	touch

Right now the code logs the type of device to console but we could just as easily write specialized code to handle special for a given type of pointer device.

```
box.addEventListener('pointerdown', detectInputType);

function detectInputType(event) {
  switch (event.pointerType) {
    case 'mouse':
```

```
      console.log('Mouse 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'
      );
  }
}
```

We could even write a single event handler that will work different depending on the target that receives event.

This code will check the `pointerdown` on the parent container of the items we want to interact with.

Instead of using the switch statement to check on the type of pointer device we check on the id attribute of the event target. In essence, we ask the browser, what's the idea of the element I just taped down on.

We then take action based on the ID of the element we taped on. The example just logs the result to console; in a real-world application we would add code to complete the necessary tasks. For example, we could store a value associated with each ID in local storage to use later or we could create a custom property to use with our styles.

```
const sizes = document.querySelector(".font-size");

sizes.addEventListener("pointerdown", handleSize);

function handleSize("pointerdown"itch (event.target.id) {
    case "font-small":
      console.log("font-small"`set small font size`ak;
```

```
    case "font-medium":
      console.log(`set medium "font-medium"`set medium font size`"font-lar
      console.log(`set large font size`);
 "font-large"`set large font size`);
      break;
  }
}
```

We could also combine both techniques where we write a single event handler and then write code specific to given pointer types.

What to do if the code relies on hover events

Because most of the code we've written has been centered on the mouse (and with compatibility layers built on pointer events) we forget that there are things that are not possible with other type of pointing devices.

For example, touch and stylus devices like what we use on phones or tablets may not hover on top of an element.

If you're starting from scratch then the obvious answer is not to rely on hover and be explicit as to what action should the user take (either a right mouse click or similar specific action) but retrofitting an existing app may be more complicated.

Modifying existing code

Most of the time, switching from mouse events to pointer events is straightfoward. Pointer events provide compatibility for mouse events.

This example builds from the example on the last section and creates a custom property in the `:root` element and stores it with its value in local storage; that way we can create an init function that will populate the settings with the values from a previous visit if they are available.

```
// root element
```

```
const root = document.documentElement;

// container for font size elements
const sizes = document.querySelector(".font-size");

sizes.addEventListener("pointerdown", handleSize);

function handleSize(eve".font-size"// switch on the target id
  switch (event.target.id) {
    case "font-small":
      root.style.setProperty("--font-size", "12pt");
      window.localStorage.setItem("font-size", "12pt");
      break;
    case "font-medium":
      root.style.setProperty("--font-size", "16pt");
      window.localStorage.setItem("font-size", "16pt");
      break;
    case "font-large":
      root.style.setProperty("--font-size", "24pt");
      window.localStorage.setItem("font-size", "24pt");
      break;
  }
}
```

Conclusion

Using pointer events facilitates a lot of things we may want do in mobile web apps without compromising the work we do in desktops.

Fully exploring what you can do with pointer events would take many posts and experiments, they will likely be explored in future content.