- Which event to listen to
- Pointer/Mouse Event
  - PointerDown/MouseDown
  - Pointer/MouseMove monitor
  - Differentiate Touch/Pen/Mouse
- Touch/Gesture
  - Gesture Helper
  - Troubleshoot: touch not working with native elements
  - Base elements
    - input/checkbox
    - scrollable element
    - list view

Which event to listen to

Choosing the right event to listen to is a hard decision to make, as there isn't a single unified browser standard to follow. If you are using basic elements from `vs/base/browser`, you may want to only listen to predefined event listeners on them. However if you are building a plain new view component, here is a quick cheatsheet

- Do you customize overflow/scrolling, or double clicking?
  - No
    - Listen to both `scroll` for scrolling
    - Listen to both `click` for clicking.
  - Yes
    - Add the element to `Gesture`
    - Listen to both `wheel` and `GestureEvent.change` for scrolling
    - Listen to both `click` and `GestureEvent.Tap` for clicking. (`GestureEvent.tapCount == 2` for double clicking)
    - Listen to `GestureEvent.contextmenu` for content menu (right click)

For more details, please read blow:

Pointer/Mouse Event

Pointer events are DOM events fired by pointer devices, which include Mouse, Touch and Pen. Pointer events inherit from Mouse events, so we only need to adopt following generic mouse/pointer listeners without modifying out mouse handler code.

PointerDown/MouseDown

```
import * as dom from 'vs/base/browser/dom'

// dom.addDisposableListener(domNode, 'mouseDown', () => {})
```

```
dom.addDisposableGenericMouseDownListener(domNode, () => {})
// dom.addDisposableListener(domNode, 'mouseUp', () => {})
dom.addDisposableGenericMouseUpListener(domNode, () => {})
```

Pointer/MouseMove monitor

```
import * as dom from 'vs/base/browser/GlobalMouseMoveMonitor'

// no change at all
this._mouseMoveMonitor = this._register(new GlobalMouseMoveMonitor<IStandardMouseMoveEventDa
this._mouseMoveMonitor.startMonitoring(
    ...
)
```

Under the hood we will do browser feature and platform detection to decide which is the right event to listen to. It's **not suggested** to bind your own listeners to `PointerEvent` as `PointerEvent` can coexist with `MosueEvent`, browser/platform detection code is still needed to avoid the mouse event handlers not being called twice.

Differentiate Touch/Pen/Mouse

The generic mouse/pointer event will carry device information if it's from a pointer device, which can be accessed by reading `event.pointerType`

```
dom.addDisposableGenericMouseDownListner(domNode, (e) => {
      const pointerType = <any>e.pointerType;
      switch (key) {
        case 'touch':
            break;
        case 'mouse':
            break;
        case 'pen':
            break;
        default:
            break;
      }
})
```

Touch/Gesture

Gesture Helper

Gesture helper `vs/base/browser/touch#Gesture` will translate native touch events (`touchStart`, `touchEnd`, `touchMove`) to richer Gesture events based on how long users touch and how much users move their finger on the screen.

Please always use Gesture for consistence, unless you want to roll in your own touch implementation.

To register gesture event listener on a DOM node, register the DOM node to Gesture store first and listen to Gesture events the same as normal DOM events.

```
import * as dom from 'vs/base/browser/dom';
import { Gesture, EventType as GestureEventType } from 'vs/base/browser/touch';

Gesture.addTarget(domNode);
dom.addDisposableEventListner(domNode, GestureEventType.Tap, (e) => {});
```

Troubleshoot: native touch are eaten

Most of the time, if touch doesn't work with native elements in your view, please check if one of your view's ancestors is being tracked by `Gesture`. `Gesture` will prevent default behavior of touch events when a DOM node is being tracked by it, which means the DOM nodes' all descendants will no longer receive any native touch events.

When above scenario happens (and you can't change the behavior of the ancestor), do one of following

- `Gesture.ignoreTarget` to let Gesture ignore touch events happening on the element
- `Gesture.addTarget` to the element if you want to customize its touch behavior

Base elements

inputbox/checkbox `vs/base/browser/ui/inputBox` and `vs/base/browser/ui/checkbox`, by default `inputBox` and `checkbox` add themselves to Gesture's ignore list

Scrollable Element Scrollable Element scrolls when `wheel`/`mousewheel`events are triggered, but by default it's not tracked by `Gesture` to ensure it doesn't break the native touch experience on its inner elements.

To make scrollable element touch friendly, when using it, please make sure you listen to at least `Tap` and `Change` Gesture events.

```
Gesture.addTarget(domNode)
dom.addDisposableEventListner(domNode, GestureEventType.Tap, (e) => {}); // Click
dom.addDisposableEventListner(domNode, GestureEventType.Change, (e) => {}); // Scroll
```

List View List View internally listens to both scroll events and `GestureEventType.Change` events (as it's virtualized ;)). While using List View, you don't need to worry about scrolling at all but the catch is native touch down won't work on inner elements as the events are trapped by `Gesture`.

Solution:

- `Gesture.ignoreTarget` to let Gesture ignore touch events happening on the element
- `Gesture.addTarget` to the element if you want to customize its touch behavior