Algorithm for Mouse Click Events

In which an algorithm for handling "mouse click" events is proposed

Status: In progress draft
garykac@

Moved to UIEvents repo: https://w3c.github.io/uievents/event-algo.html

Links:

- Need algorithm for how mouse click events are fired (tracking issue)
- UI Events spec: Mouse Event Order examples
- Testing:
  - Mouse event viewer
  - Mouse event viewer (with Shadow DOM)

Behavior observations:

- User Agents don't determine clicks, i.e., they don't measure time from mouseup to mousedown to determine whether a click event should fire. Instead, the UA relies on click and double-click events from the native OS
  - Note: the native OS may provide mechanisms for the user to adjust the exact timing for double-clicks (for accessibility)
  - Ubuntu - Chrome/Firefox:
    - 2 rapid clicks: click - click - dblclick
    - 3 rapid clicks: click - click - dblclick - click
    - 4 rapid clicks: click - click - dblclick - click - click
    - 5 rapid clicks: click - click - dblclick - click - click - click - dblclick

        - … need 8 rapid clicks for 3rd dblclick
    - macOS - Chrome/Firefox/Safari
        - 2 rapid clicks: click - click - dblclick
        - 3+ rapid clicks: click - click - dblclick - click+
        - NSResponder mouseUp method has a clickCount attribute, which is typically compared with 2 for double-click
    - Windows
        - Separate WM_(NC)?[LMR]BUTTONDBLCLICK messages

Assumptions:

- The native events will always be given in the correct order
    - So we don't need to have the algorithms verify and enforce it

initGlobalState()

// TODO: What is best scoping for this: Window, Browsing Context?
mouseButtonBitMask = 0

createMouseEvent(eventType, target)

Let event = the result of creating a new MouseEvent
initEvent(event, eventType, target)
initUIEvent(event, target)
initMouseEvent(event)

If this event represents the result of a user action, then
    Set event's **due to user interaction** flag // See uievents/270
    Set event.isTrusted = true

return event

initEvent(event, type, target)

// TODO Move to / merge with DOM spec Event interface
// For reference (from DOM): initialize an event, list of event flags
Set event.type = type
Set event.target = target
Set event.currentTarget = null // Will be set appropriately during dispatch
Set event.eventPhase = NONE (0) // Will be set appropriately during dispatch
Set event.bubbles = true
Set event.cancelable = true
Set event.defaultPrevented = false
Set event.composed = false // See COMPAT note for mouseenter and mouseleave
Set event.isTrusted = false
Set event.timeStamp = Number of milliseconds relative to the time origin

// Historical attributes
// event.srcElement = target
// event.cancelBubble = alias for stopPropagation
// event.returnValue = alias for !canceled_flag

// Internal event state
Unset the event's **stop propagation** flag
Unset the event's **stop immediate propagation** flag
Unset the event's **canceled** flag
Unset the event's **in passive listener** flag
Unset the event's **composed** flag
Unset the event's **initialized** flag

Unset the event's **dispatch** flag
Unset the event's **due to user interaction** flag // TODO: this is a proposal. See uievents/270

initUIEvent(event, target)

Set event.view = the target's node document's Window object, if any, and null otherwise.
Set event.detail = 0

// Historical attributes
// event.which // Used by Mouse and Keyboard events

initMouseEvent(event)

Set event.screenX = the x-coordinate of the position where the event occurred relative to the origin of the desktop
Set event.screenY = the y-coordinate of the position where the event occurred relative to the origin of the desktop
Set event.clientX = the x-coordinate of the position where the event occurred relative to the origin of the viewport
Set event.clientY = the y-coordinate of the position where the event occurred relative to the origin of the viewport

setEventModifiers(event) // Set shift, ctrl, alt, meta flags

Set event.button = 0
Set event.buttons = *mouseButtonBitMask*

// PointerLock attributes (TODO: Move into PointerLock spec)
Set event.movementX = 0
Set event.movementY = 0

// CSSOM attributes (TODO: Move into CSSOM spec)

Set event.pageX = pageX
Set event.pageY = pageY
Set event.x = x
Set event.y = y
Set event.offsetX = offsetX
Set event.offsetY = offsetY

calcMouseEventButtonAttribute(mbutton):

if mbutton is the primary mouse button, then return 0
if mbutton is the secondary mouse button, then return 2
if mbutton is the auxiliary (middle) mouse button, then return 1
if mbutton is the X1 (back) button, then return 3
if mbutton is the X2 (forward) button, then return 4
return 0

handleNativeMouseDown(mbutton)

Update the *mouseButtonBitMask* as follows:
    if mbutton is the primary mouse button, then set the 0x01 bit
    if mbutton is the secondary mouse button, then set the 0x02 bit
    if mbutton is the auxiliary (middle) mouse button, then set the 0x04 bit
    (if supported, other buttons may be represented by setting the other bits starting with 0x08)

// Unlike mousedown, pointerdown events are not nested when multiple buttons are pressed. The PE spec should define that.
Call PointerEvents spec to (possibly) generate pointerdown events

target = hitTest(viewport_pos)

Let event = createMouseEvent(**mousedown**, target)
Set event.button = calcMouseEventButtonAttribute(mbutton)
result = dispatch *event* at *target*

// Default action - update focus
if result is true AND target is a focusable area that is click focusable, then
    // focusable:
    // Input, Select, TextArea, Anchor, Area. Button. IFrame
    // any element with a tabindex
    // any element with isContentEditable == true
    run the focusing steps on *target*

if mbutton is the secondary mouse button:
    Let menuevent = createMouseEvent(**contextmenu**, target)
    result = dispatch *menuevent* at *target*
    if result is true:
        Show UA context menu

handleNativeMouseUp(mbutton)

// Note: Other mouse events can occur between mousedown and mouseup: out, leave, over, enter, move

Update the *mouseButtonBitMask* as follows:
    if mbutton is the primary mouse button, then clear the 0x01 bit
    if mbutton is the secondary mouse button, then clear the 0x02 bit
    if mbutton is the auxiliary (middle) mouse button, then clear the 0x04 bit
    (if supported, other buttons may be represented by clearing the other bits starting with 0x08)

Call PointerEvents spec to (possibly) generate pointerup events

target = hitTest(viewport_pos)
Let event = createMouseEvent(**mouseup**, target)
Set event.button = calcMouseEventButtonAttribute(mbutton)
result = dispatch *event* at *target*

handleNativeMouseClick(mbutton)

// ASSERT previous mouse event was mouseup

target = hitTest(viewport_pos)
if mbutton is the primary mouse button, then
    Let event = createMouseEvent(**click**, target)
otherwise
    Let event = createMouseEvent(**auxclick**, target)
// Note: the event's button attribute is not set for this event
result = dispatch *event* at *target*

// Note: any "default action" is handled during dispatch by triggering the activation behavior algorithm for the target
if result is true AND target is not disabled AND target has a default action:
    // clickable:
    // any element with an onclick handler
    // Anchor elements - navigate to url target
    // TODO: form submission?

handleNativeMouseDoubleClick(mbutton)

// ASSERT previous mouse event was click

// Only left clicks generate double click events
if mbutton is not the primary mouse button, then return

target = hitTest(viewport_pos)
Let event = createMouseEvent(**dblclick**, target)
// Note: the event's button attribute is not set for this event
result = dispatch *event* at *target*