Algorithm for Keyboard Events

In which an algorithm for handling keyboard, input and composition events is proposed

Status: In progress draft
garykac@

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

Note: This is a proposed framework for handling keyboard, input and composition events. It is not complete, but should be detailed enough to evaluate if there are any obvious flaws that need to be addressed before fleshing out the details further.

Links:

- Tracking issue:
    - Need algorithm for how events are fired for key presses
- Related issues:
    - input-events/22
- Previous discussions:
    - DOM3 keyboard, input and composition events (pt1) (August 2013)
    - DOM3 keyboard, input and composition events (pt2) (April 2014)
- Keyboard event viewer:
    - Keyboard Event Viewer
    - Keyboard Event Viewer (for contenteditable)

Behavior observations:

- Raw key events are provided by the underlying native OS

    - key repeats are also handled by the native OS.

TODO:

- Identify the appropriate boundary between UIEvents and Input Events specifications
- Include special handling required for altgraph, see: uievents/147

Native platform assumptions

These algorithms assume that the underlying platform supports the following:

- Platforms will generate KEYDOWN and KEYUP messages when the user presses and releases keys
    - Windows: WM_(SYS)?KEYDOWN, WM_(SYS)?KEYUP, WM_(SYS)?CHAR
    - macOS: NSKeyDown, NSKeyUp, NSFlagChanged
        - NSKeyDown is a combined keydown + char message
        - NSFlagsChanged is for modifier keys (i.e., keydown without an associated char)
- Platforms MAY generate a separate CHAR message immediately after the KEYDOWN message
    - On platforms that generate separate CHAR events (like Windows), there are no intervening key messages between the keydown and the char
- Platforms will maintain a Key Repeat Threshold and will indicate in the KEYDOWN message whether it is a repeat
- Platforms will provide a mechanism to determine which modifier keys are currently pressed
    - Either via flags in the messages, or via a system call to obtain the current state
- TODO: specify platform requirements for IME (and dead keys)

Algorithm

Common entry points:

- handleNativeKeyDown
- handleNativeKeyUp
- handleNativeKeyPress

initGlobalState()

```
// TODO: What is best scoping for this: Window, Browsing Context?
keyModifierState = {}  // Set of which keys are currently being pressed.

// Used to cancel subsequent beforeinput/input events when the keydown is canceled
suppressKeyInputEvents = false

compositionMode = false  // True if IME is enabled and next input will trigger compositionstart
inComposition = false  // true if after compositionstart (but before compositionend)
```

createKeyboardEvent(eventType, target)

```
Let event = the result of creating a new KeyboardEvent
initEvent(event, eventType, target)
initUIEvent(event, target)
initKeyboardEvent(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
```

createInputEvent(eventType, target)

```
Let event = the result of creating a new InputEvent
```

```
initEvent(event, eventType, target)
initUIEvent(event, target)
initInputEvent(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
```

createCompositionEvent(eventType, target)

```
Let event = the result of creating a new CompositionEvent
initEvent(event, eventType, target)
initUIEvent(event, target)
initCompositionEvent(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
```

initKeyboardEvent(event)

```
Set event.key = ""
Set event.code = ""
Set event.location = DOM_KEY_LOCATION_STANDARD
Set event.shiftKey = false
Set event.ctrlKey = false
Set event.altKey = false
Set event.metaKey = false

Set event.repeat = false
Set event.isComposing = false

// Historical attributes
// event.keyCode
// event.charCode

// Internal
Unset the event's shift_flag
Unset the event's control_flag
Unset the event's alt_flag
Unset the event's altgraph_flag
Unset the event's meta_flag
Unset the event's capslock_flag
Unset the event's numlock_flag
// TODO: eval: ScrollLock, Hyper, Super, and (for virtual keyboards) Symbol, SymbolLock
Unset the event's suppressKeyInputEvents
```

initInputEvent(event)

```
Set event.data = null
Set event.isComposing = false
Set event.inputType = ""
```

// This matches current UA behavior, but it makes more sense for this to be false. See whatwg/html/5453 for discussion.
Set event.composed = true;

// TODO: Move this to Input events (level 1 or level 2)
Set event.dataTransfer = null
Set event.targetRanges = []

initCompositionEvent(event)

Set event.data = "" // Note: same name as attribute in InputEvent

setKeyModifiers(event)

Set the event's shift_flag if keyModifierState includes "Shift", unset it otherwise
Set the event's control_flag if keyModifierState includes "Control", unset it otherwise
Set the event's alt_flag if keyModifierState includes "Alt", unset it otherwise
Set the event's altgraph_flag if keyModifierState includes "AltGraph", unset it otherwise
Set the event's meta_flag if keyModifierState includes "Meta", unset it otherwise
Set the event's capslock_flag if keyModifierState includes "CapsLock", unset it otherwise
Set the event's numlock_flag if keyModifierState includes "NumLock", unset it otherwise
// TODO: eval: ScrollLock, Hyper, Super, and (for virtual keyboards) Symbol, SymbolLock

Set event.shiftKey = true if the event's shift_flag is set, false otherwise
Set event.ctrlKey = true if the event's control_flag is set, false otherwise
Set event.altKey = true if the event's alt_flag or altgraph_flag is set, false otherwise
Set event.metaKey = true if the event's meta_flag is set, false otherwise

setKeyAttributes(event, nativeKeyInfo)

```
event.key = the key attribute value for this key press  // TODO more formal description of non-printable characters
event.code = the key code attribute value corresponding to the physical key that was pressed.
event.location = calcKeyLocation()
if nativeKeyInfo indicates that this is a repeat key event
    Set event.repeat

setKeyModifiers(event)

// Historical attributes
// event.charCode
// event.keyCode
// event.which
```

calcKeyLocation(code)

```
if the code is any of [ "AltLeft", "ControlLeft", "MetaLeft", "ShiftLeft"], then
    return DOM_KEY_LOCATION_LEFT
if the code is any of [ "AltRight", "ControlRight", "MetaRight", "ShiftRight"], then
    return DOM_KEY_LOCATION_RIGHT
if the code matches any of the values in the numpad section, then
    return DOM_KEY_LOCATION_NUMPAD
return DOM_KEY_LOCATION_STANDARD
```

getModifierState(modifier)

```
if modifier == "Shift", then return true if shift_flag is set
if modifier == "Control", then return true if control_flag is set
if modifier == "Alt", then return true if alt_flag is set
if modifier == "AltGraph", then return true if altgraph_flag is set
if modifier == "Meta", then return true if meta_flag is set
if modifier == "CapsLock", then return true if capslock_flag is set
if modifier == "NumLock", then return true if numlock_flag is set
return false
```

handlePreBrowserKey(key)

```
// Returns true if the browser handles this key *before* the keydown event is dispatched

// TODO: This is a placeholder. If there is agreement on KeyboardLock, then this would be checkKeyboardLock()
// and the definition would be in that spec.
if KeyboardLock enabled, then return false

// TODO: create list of pre-keydown browser keys
if key is ctrl-t, then create new tab, return true
if key is ctrl-w, then close tab, return true
...

// TODO: list OS-level keys here when they affect browser state?
if key is <key to enter IME>, then beginComposition() and return true
...

return false
```

handlePostBrowserKey(key)

```
// Returns true if the browser handles this key *after* the keydown event is dispatched

// TODO: This is a placeholder. If there is agreement on KeyboardLock, then this would be checkKeyboardLock()
// and the definition would be in that spec.
if KeyboardLock enabled, then return false

if key combination is:
    // TODO: list of post-keydown browser keys
    the UA shortcut key for cut (e.g., ctrl-x), then handleCut()
    the UA shortcut key for copy (e.g., ctrl-c), then handleCopy()
    the UA shortcut key for paste (e.g., ctrl-v), then handlePaste()
    if tab then change focus  // TODO: verify this is post-keydown
    ...
    return true
return false
```

handleNativeKeyDown(key)

```
// Update global modifier key state
If key is left or right Shift key, then add "Shift" to keyModifierState
If key is left or right Control key, then add "Control" to keyModifierState
if key is left or right Alt key, then add "Alt" to keyModifierState
if key is AltGraph key, then add "AltGraph" to keyModifierState
if key is left or right Meta key, then add "Meta" to keyModifierState
```

```
// Chrome/Firefox handle some BrowserKeys before generating keydown events.
// Tested with ctrl-t (to create new tab)
if handlePreBrowserKey(key), then exit

Let nativeKeyInfo = info extracted from native key event // TODO - expand this
Let target = currently focused area of a top-level browsing context
Let event = createKeyboardEvent(keydown, target)
setKeyAttributes(event, nativeKeyInfo)
result = dispatch event at target

// Note: when in composition, canceling the keydown has no effect (tested: macOS Chrome; TODO: test other configs)
if inComposition, then handleCompositionKey() and exit

if result = false, then set suppressKeyInputEvents

// TODO: Is this check necessary? Can native CHAR events be ignored or do they contain useful data?
if native platform has separate CHAR event, then exit  // because CHAR event will trigger handleNativeKeyPress()

handleNativeKeyPress()
```

handleNativeKeyUp()

```
If key is left or right Shift key, then remove "Shift" from keyModifierState
If key is left or right Control key, then remove "Control" from keyModifierState
if key is left or right Alt key, then remove "Alt" from keyModifierState
if key is AltGraph key, then remove "AltGraph" from keyModifierState
if key is left or right Meta key, then remove "Meta" from keyModifierState

Let nativeKeyInfo = info extracted from native key event // TODO - expand this
```

Let target = currently focused area of a top-level browsing context
Let event = createKeyboardEvent(**keyup**, target)
setKeyAttributes(event)
result = dispatch *event* at *target*

handleNativeKeyPress()

// May be called directly if the native platform generates CHAR events that are separate from keydown
// Or implicitly from handleNativeKeyDown

// Note: macOS Chrome, ctrl-x/c/v disabled when IME is active
if compositionMode, then handleCompositionKey() and exit

if handlePostBrowserKey(key), then exit

Let target = currently focused area of a top-level browsing context
Run fireKeyInputEvents(key, target)

fireKeyInputEvents(key, target)

if suppressKeyInputEvents == true, then exit // if keydown was canceled

// Historical keypress event fires here (see uievents/220)

inputType = null
data = null

// TODO: Consider moving some or all of this into input events spec

```
if key == "Backspace" then inputType = deleteContentBackward
if key == "Delete" then inputType = deleteContentForward
// TODO: more…
otherwise
    inputType = insertText
    data = the keydown event's key attribute

if inputType != null
    // COMPAT: Firefox does not fire the beforeinput event
    result = fireInputEvent(beforeinput, inputType, data)  // Cancelable
    if result == true
        // TODO: Perform DOM updates here
        Insert key in DOM target element

        // COMPAT: for insertFromPaste, Chrome has data=null, Firefox has data = same as beforeinput
        fireInputEvent(input, inputType, data)  // Not cancelable
```

fireInputEvent(eventType, inputType, data)

```
Let target = currently focused area of a top-level browsing context
Let event = createInputEvent(eventType, target)
Set event.inputType = inputType
Set event.data = data
result = dispatch event at target

return result
```

fireCompositionEvent(type, data)

```
Let target = currently focused area of a top-level browsing context
Let event = createCompositionEvent(type, target)
Set event.data = data
result = dispatch event at target

return result
```

handleCut()

```
// TODO: define when the "cut" event is fired
data = null
result = fireInputEvent(beforeinput, deleteByCut, data)
if result == true
    Copy selected text to clipboard
    Remove selected text from DOM target element
    fireInputEvent(input, deleteByCut, data)
```

handleCopy()

```
// No input events fired
// TODO: define when the "copy" event is fired -- call into clipboard spec?
Update clipboard with current text selection
```

handlePaste()

```
// This is intended to be called when the UA triggers a paste action (via user action).
// TODO: define when the "paste" event is fired
data = The text on the clipboard that is being pasted.
result = fireInputEvent(beforeinput, insertFromPaste, data)
if result == true
    Paste clipboard contents into DOM target element
    fireInputEvent(input, insertFromPaste, data)
```

enterCompositionMode()

```
compositionMode = true
```

exitCompositionMode()

```
endComposition()
compositionMode = false
```

handleCompositionKey()

```
// This is called when a key is added to the composition buffer, or when the user navigates between composition options.

// This is where the IME handles the UI part to navigate the candidates or accept all or part of the text.
// Not sure how much (if any) of this makes sense to "specify" here since it is controlled by the IME.
if special IME key, then
    if arrow, then select new candidate
    if enter, then accept the current candidate, run endComposition() and exit

// TODO: Agree on event order for compositionupdate and beforeinput
// Chrome currently does bi -> cu, whereas Safari does cu -> bi. Firefox doesn't yet produce bi, but prefers cu -> bi.
// See input-events/49 and uievents/66

Let data = the current composition candidate string
fireCompositionEvent(compositionupdate, data)
// TODO: verify compositionupdate cannot be canceled

result = fireInputEvent(beforeinput, insertCompositionText, data)
// TODO: Not all beforeinput events can be canceled - need to encode that here
if result == true

    Update the DOM with the composition text

    fireInputEvent(input, insertCompositionText, data)
    // Not cancelable
```

beginComposition()

```
if inComposition == true, then exit

data = currently selected text
result = fireCompositionEvent(compositionstart, data)
// TODO: document what happens compositionstart is canceled
inComposition = true

handleCompositionKey()
```

endComposition()

if inComposition == false, then exit

// COMPAT: Chrome sends out beforeinput/compositionupdate/input sequence before compositionend (tested macOS)
//handleCompositionKey()

Let data = the current composition candidate string
result = fireCompositionEvent(**compositionend**, data)

// COMPAT: Firefox sends input event after compositionend. Not sure when DOM is updated relative to compositionend (tested on macOS)
//result = fireInputEvent(**input**, **insertCompositionText**, data)

Set inComposition = false