Event Breakpoints PRD

Intro Goal

Events are the binding glue of the web and need first-class debugging methods.

Related

- UX
  - Victoria's initial mockups (M1, but no event handlers - the 3rd level)
  - Matt: https://mozilla.invisionapp.com/share/F5OAI2MM3QY
- Callstacks & Stacktraces PRD
- DOM Mutation Breakpoints PRD

Resources

- Dynamic and Graphical Web Page Breakpoints (pdf)
- Pause Your Code With Breakpoints (post)

Stakeholders

DevTools Product	Harald Kirschner
DevTools Frontend Lead	David Walsh
DevTools Server Lead	Logan Smyth
DevTools UX	Matt Croud
Sponsor	Mike Taylor

Why? Problem Statement

1. **Blocks removal of old debugger**. We still get reports that power users enable the old debugger to get the old event breakpoints panel.
2. **Web compat priority**. Events is what that team debugs for breakfast, lunch and dinner.

Who benefits? Target Audience/Market

1. **Web Compat team**. Main target customer for this and informs success.
2. **Frontend developers**. Frequently asking about this and possibly cause of DevTools losing power users.

What? Functional Requirements

See Use Cases for longer stories.

Milestone 1: Pause on Event Types in Debugger

Why only event types vs handlers: Pausing on event types solves all the event pausing use cases while only adding minor nuisance of stepping through multiple handlers to find the right one. On the other side, pausing on specific event handlers depends on handlers persisting across time and sessions. In highly dynamic apps, handlers can come and go quickly within interactions like hover or timed animations; making finding and selecting the handler in time impossible.

Must Have: MVP Goals. *Should Have*: Stretch Goals. *Could Have*: Future Backlog

Requirement	**Job Story**	**Priority**	**Notes**
Pause on Event Type	When trying to understand click issues, I want to pause on all click events, so that I can step through every single handler.	Must Have	Covers most use cases as it provides the basic pausing. Bug 1549999 - Pause on Event Type Notes, see also Bug 1451594
Skip Blackboxed Scripts	When pausing on a click event, I want to skip blackboxed code that handles event delegation, so that I pause directly in the actual code.	Should Have	
Group Event Types	When browsing event types, I want to have them grouped thematically,	Should Have	MDN has event categories. Chrome has these event categories.

	so that the list becomes easier to browse.		
Selectable Groups	When debugging mouse behaviour, I want to select and deselect all events in the Mouse group, so that I don't have to manually enable them all.	Should Have	
Selection represented in top-level groups	When having one of many sub event selected, I want to see that the group is "active", so that I don't have to expand it to see.	Should Have	Chrome's intermediate checkbox state
Persist selected events	When reloading and reopening the Debugger, I want the selected events to persist, so that debugging continues	Must Have	Bug 1550001 - Persist selected events
MDN Links	When browsing events, I want links to their MDN docs, so I can learn how they work.	Should Have	Right click or question mark icon on hover

Milestone 2: Parity & Usability

Requirement	**Job Story**	**Priority**	**Notes**
Filter Types	When trying to find a specific event type, I want to filter the list to specific types (like "mouse"), so that I can find the types I am looking for.	Should Have	
Pause on Script/Scheduling/Messaging Lifecycle *(aka non-events)*	When I want to debug side effects caused by script execution or timers, I want to pause on those lifecycle points	Should Have? *(needs to be broken down)*	Chrome comparison Scheduling (request & callback) setTimeout/setInterval requestAnimationFrame call/callback requestIdleCallback call/callback Parse

			Set innerHTML Document.write Canvas Canvas context creation WebGL Error Fired WebGL Warning Fired Script Script First Statement Script Blocked by Content Security Policy Web Audio Create AudioContext Close AudioContext Resume AudioContext Suspend AudioContext
Map Framework Handlers *(aka blackbox frameworks)*	When I use frameworks to attach & delegate events, I want pausing to happen in my code's event handlers, so that I don't have to step through the framework event handler to my handler.	Should Have?	Top Frameworks How can this support unknown frameworks, maybe by allowing users blackboxing internals?

Milestone 3: Event Handler Inspector

This part is Inspector specific and has its own PRD.

Milestone X: Future Backlog

Requirement	Job Story	Priority	Notes
Pause on Web Extension Lifecycle	When I want to debug web extension content script, I want to pause on content script lifecycle and message hooks, so that I can	Could Have	Content script cues: tabs.executeScript() content_scripts: document_start, document_end, document_idle contentScripts.register()
Indicate used Event Types	When diagnosing event handler issues, I want to see which event types are actively used,	Could Have	Clicking the "used" indicator should link to a filtered view of Inspector Events panel.

	so that I can understand which scripts control events on the page and how.		
Constrain Event Breakpoints to specific with by CSS rules		?	
Change Event Breakpoints to Logpoints		?	**Web Compat P1**
Console command for adding event listeners		?	
Debugger Event panel links to Inspector Events Panel	When looking at event types, I want a quick way to jump to the Inspector's event handler panel, so that I can switch tracks to inspect and toggle event handlers.	Could Have	

What **not?** Constraints

1. Solve breakpoint issues

How? Technical Requirements

Events and Categories

Chrome & MDN (Events only)

MDN	**Chrome**
Resource cached error abort load	Media play pause playing

- beforeunload
- unload

Network
- online
- offline

Focus
- focus
- blur

WebSocket
- open
- message
- error
- close

Session History
- pagehide
- pageshow
- popstate

CSS Animation
- animationstart
- animationend
- animationiteration

CSS Transition
- transitionstart
- transitioncancel
- transitionend
- transitionrun

Form
- reset
- submit

Printing
- beforeprint
- afterprint

Text Composition
- compositionstart
- compositionupdate
- compositionend

View
- fullscreenchange
- fullscreenerror
- resize
- scroll

Clipboard
- cut
- copy
- paste

Keyboard
- keydown
- canplay
- canplaythrough
- seeking
- seeked
- timeupdate
- ended
- ratechange
- durationchange
- volumechange
- loadstart
- progress
- suspend
- abort
- error
- emptied
- stalled
- loadedmetadata
- loadeddata
- waitingaudio
- video

Clipboard
- copy
- cut
- paste
- beforecopy
- beforecut
- beforepaste

Control
- resize
- scroll
- zoom
- focus
- blur
- select
- change
- submit
- reset

Device
- deviceorientation
- devicemotion

DOM Mutation
- DOMActivate

- keypress
- keyup

Mouse
- auxclick
- click
- contextmenu
- dblclick
- mousedown
- mouseenter
- mouseleave
- mousemove
- mouseover
- mouseout
- mouseup
- pointerlockchange
- pointerlockerror
- select
- wheel

Drag & Drop
- drag
- dragend
- dragenter
- dragstart
- dragleave
- dragover
- drop

Media
- audioprocess
- canplay
- canplaythrough
- complete
- durationchange
- emptied
- ended
- loadeddata
- loadedmetadata
- pause
- play
- playing
- ratechange
- seeked
- seeking
- stalled
- suspend
- timeupdate
- volumechange
- waiting

- DOMFocusIn
- DOMFocusOut
- DOMAttrModified
- DOMCharacterDataModified
- DOMNodeInserted
- DOMNodeInsertedIntoDocument
- DOMNodeRemoved
- DOMNodeRemovedFromDocument
- DOMSubtreeModified
- DOMContentLoaded

Drag / drop
- drag
- dragstart
- dragend
- dragenter
- dragover
- dragleave
- drop

Keyboard
- keydown
- keyup
- keypress
- input

Load
- load
- beforeunload
- unload
- abort
- error
- hashchange
- popstate

Mouse
- auxclick
- click
- dblclick
- mousedown
- mouseup
- mouseover
- mousemove
- mouseout
- mouseenter
- mouseleave

Progress abort error load loadend loadstart progress Timeout More from MDN.	mousewheel wheel contextmenu Pointer pointerover pointerout pointerenter pointerleave pointerdown pointerup pointermove pointercancel gotpointercapture lostpointercapture Touch touchstart touchmove touchend touchcancel Worker message messageerror XHR readystatechange load loadstart loadend abort error progress timeout

Implementation

Old Debugger / Jason's version

- Enumerate handlers
- Enabling would add a breakpoint at the location of the handler

Chrome

- Event handlers are added internally
- The pane more is more category oriented - pause on the JS that is run

Cases

- We don't know about new listeners
- Do we want to pause in onClick for non click events

Logan's Approach

- Listen to event dispatches, and then pause in the "click" events handler
- Equivalent to listening to DOM mutations
- Similar to XHR, which means we can improve XHR if we invest in this approach

Milestone 1

New Events (Pause on Type) panel in the Debugger.

Old Victoria Mockups

Question	**Outcome**
Cost of Blackboxing	
Name of the Debugger panel	Rational: Chrome parity **Decision:** "Event Breakpoints"
Translate category titles or not?	Maybe? **Decision: No** until we get requests
Event list stored in backend or frontend	Chrome has frontend Frontend would make it easier to add new events to older targets Depends on how generic the backend is. David will follow up with Logan.
How reactive do we want we want the event listing to be? Do we need dynamic events (event listeners added via console, or within other event listeners) to be part of milestone 1? A few notes:	

https://gist.github.com/darkwing/1d779e15667e316e2c8857e6e638242a	

Milestone 3

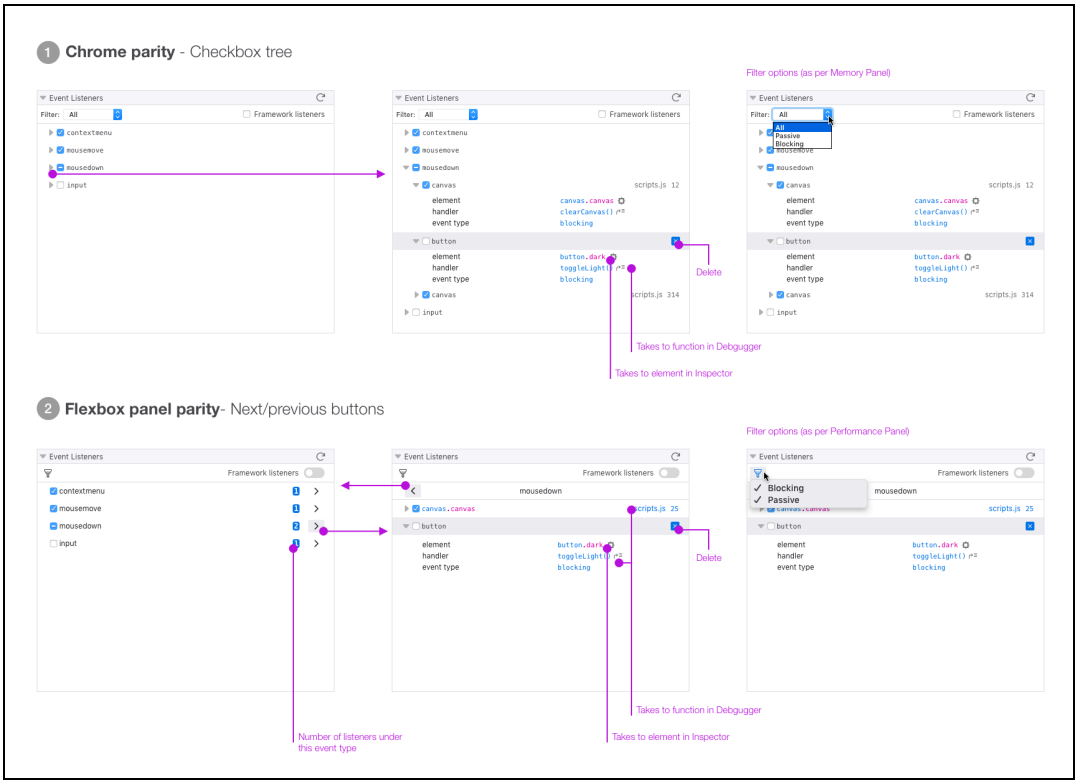
New Events (Inspect Handlers) panel in Inspector

Question	Outcome
Will pretty-printing be considered?	**Decision:** Yes! (PRD)
What additional event listener properties do we wish to expose i.e. `useCapture`, `once`?	Assumptions … **Needs decision.**

Design 1

Design 1 presents two different approaches. A checkbox tree similar to Chrome, and a next/previous panel mechanism similar to Firefox flexbox panel, designed to reduce the “chaotic checkbox tree” which concerns were raised about. There are other variations in concepts 1 and 2 such as different filter methods etc.

Questions:



Large Image | Feedback

Search

What's **success**? Success Metrics

1. 5% of debugger DAU use the event breakpoints.
2. Web compat team successfully applies event breakpoints in the wild

Why **now**? Market Window

P1 issue for **web compat**.
Often reported from users as badly lacking feature.

What must be **true**? Assumptions

1. Breakpoint quality and mapping must be excellent for the breakpoints to work and correctly step.
2. We need to have a good update cycle for un-mapping/blackboxing framework's event delegation for the relevant.

What could go **wrong**? Risks

Bad performance. Mitigate by measuring

Who else? Competitive Landscape & Product Inspiration

Chrome: Event Listener Breakpoints

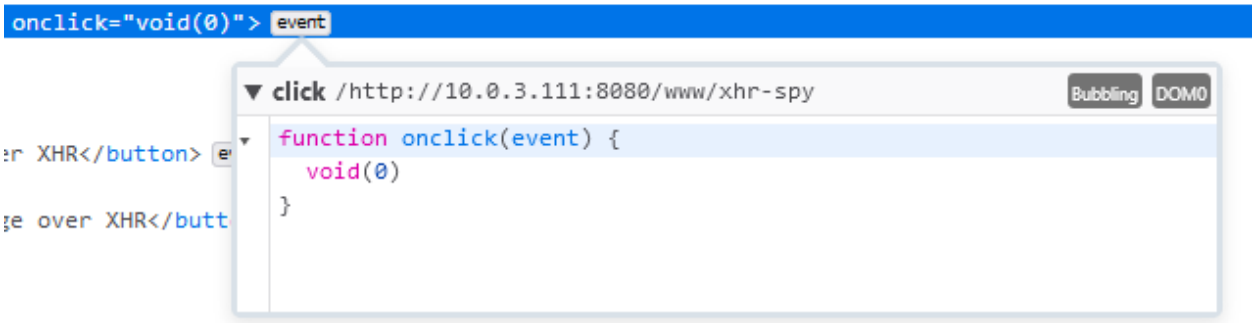
- This panel shows all available event types grouped by event-category (e.g. *mouse*) and event-type (e.g. *click*)
- Every group (i.e. category and type) has a checkbox allowing to break on an event type. No extra breakpoint entry in the Breakpoints panel if a checkbox in this list is marked.

[mockup]

Firefox: Event Listener Bubble in the Inspector Panel

The inspector panel show a little bubble next to an element in case there is an existing listener registered.

Clicking on this little bubble shows a popup window:



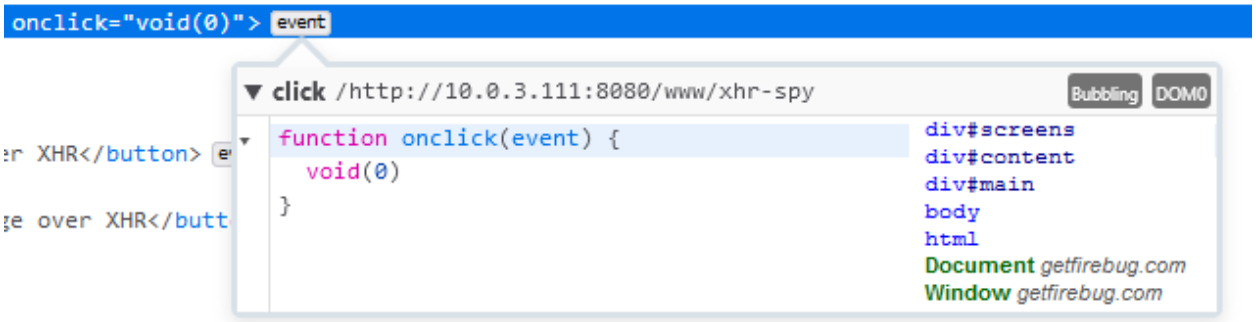
- The user can see URL of the parent file where the listener lives
- It's also possible to see the source code of the event listener
- It should be possible to switch on Framework support - skip event handlers bound by frameworks and show user source code.

Possible improvements:

- Offer a link navigating the user to the source code.

- Removing the listener
- Disabling the listener
- Showing event targets

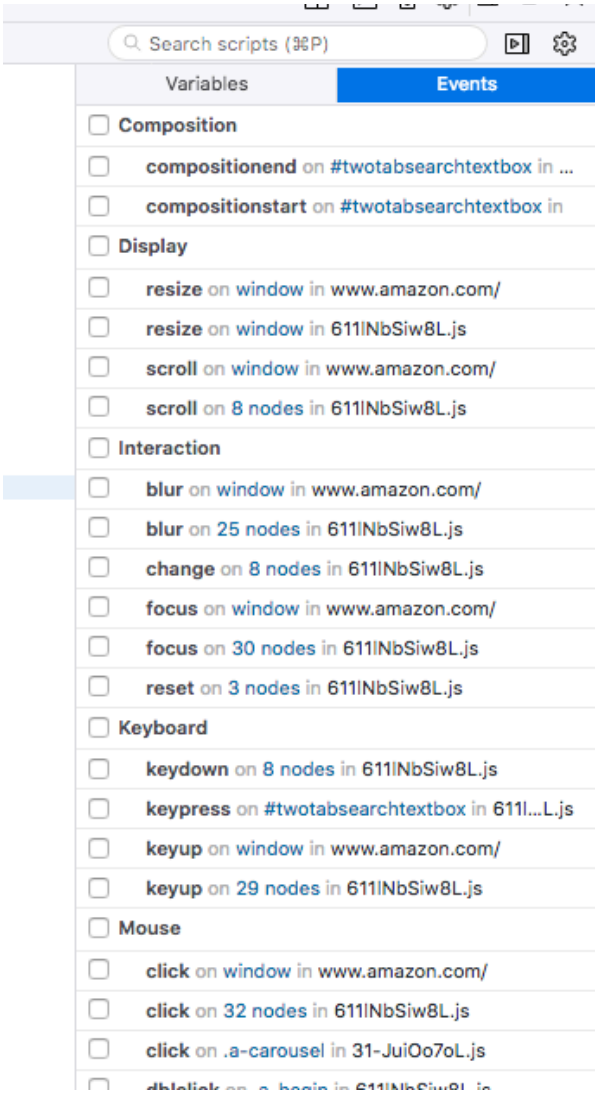
See the next screenshot showing how we could display event targets for specific event handler (see also screenshot from EventBug extension at the bottom of this doc):



- Mockup

Firefox DevTools (Previous UI)

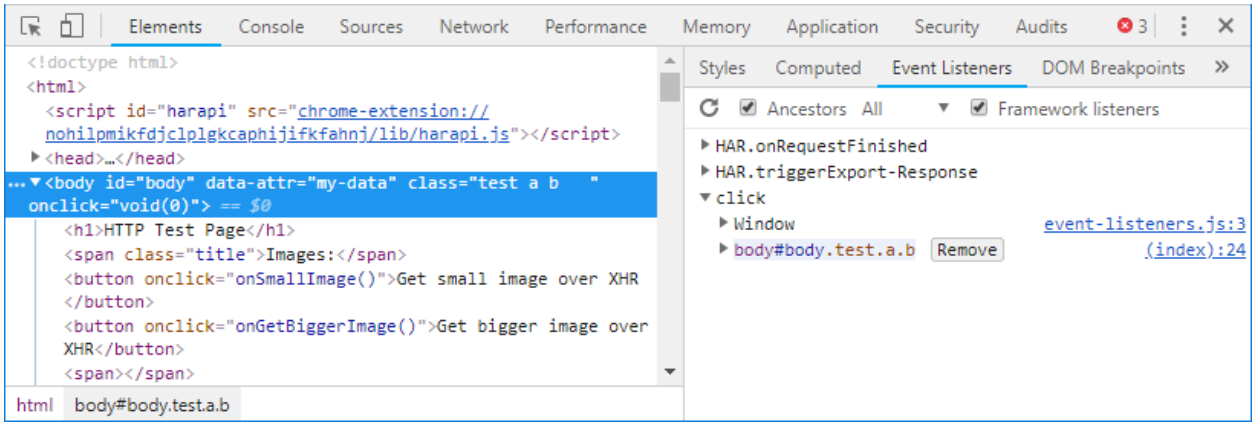
Pause on Event Type was previously known as *Break on DOM Event* (mdn) and was offered through an *Events* panel available on the right side of the Debugger panel. See the screenshot taken from the previous (XUL based) debugger UI:



- The events panel lists all events for which you have assigned a listener (together with element count and file name)
- The user can check any checkbox to break in the event handler when executed.

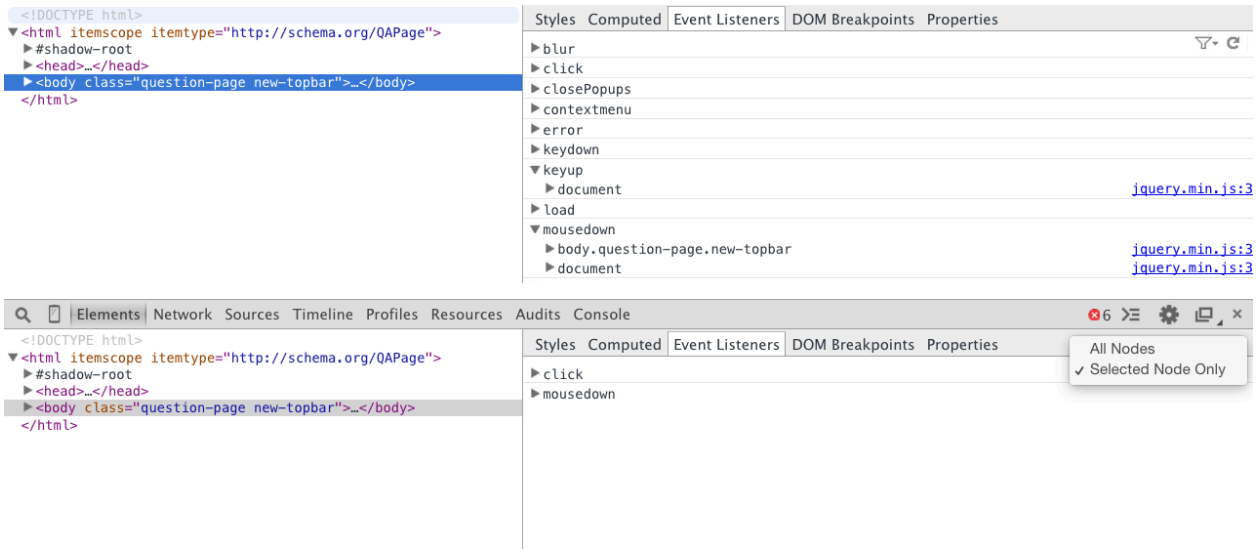
Chrome: Event Listeners

Chrome implements quite complex and exhaustive UI to offer features related to event breakpoints. Let's start with the *Elements* panel.

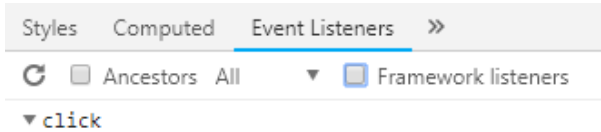


- The list of event handlers registered for selected element is displayed in the side panel.
- Ancestor are optional (on by default)
- All, Passive or Blocking listeners are displayed
- Hovering over element (in the side panel) highlights the element on the page and also displays a *Remove* button for unregistering the event handler.
- There is no way to create a breakpoint from this panel directly. The user needs to use the source link to go to the Sources panel. The proper line is auto highlighted, so the breakpoint can be easily created there.
- Handler instances can be inspected
- Framework listeners - resolve event listeners bound with framework (see below)
- Collects ancestor handlers by default

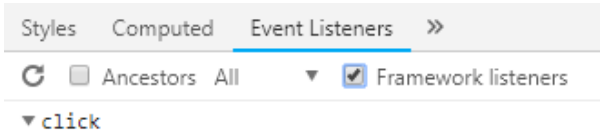
For reference, the [old events panel] had a filter UI for all events:



Here is couple more screenshots showing the difference when toggling framework listeners.

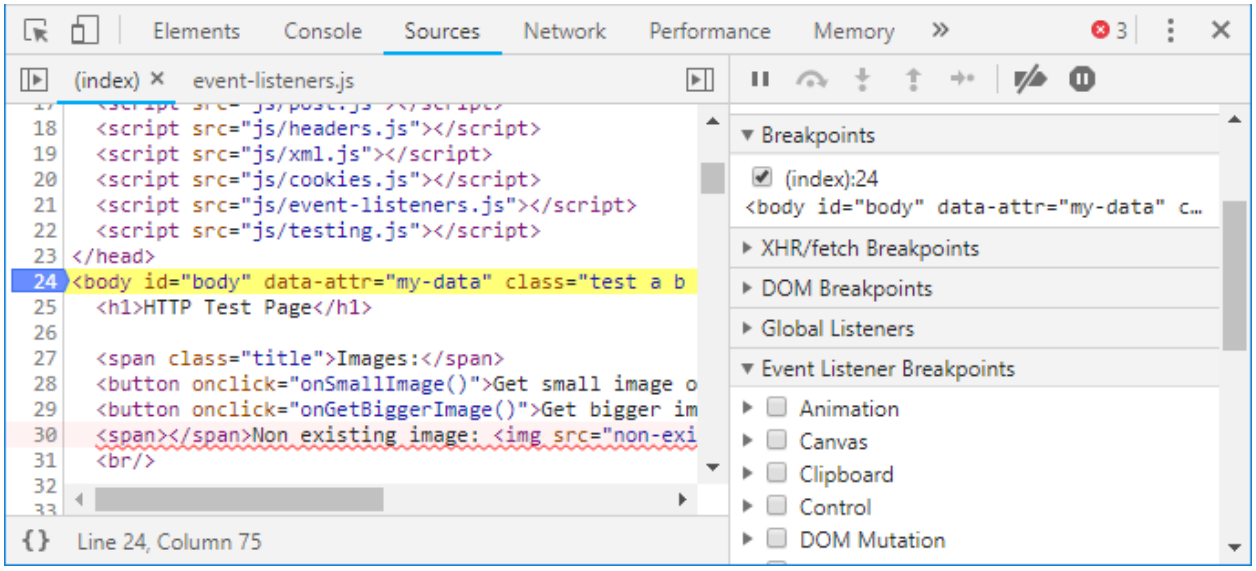


- The location link points to jQuery code base to the place where a listener is added.



- The location link points to user code base to the place where a listener is added. Note that there are actually two listeners created by the user.

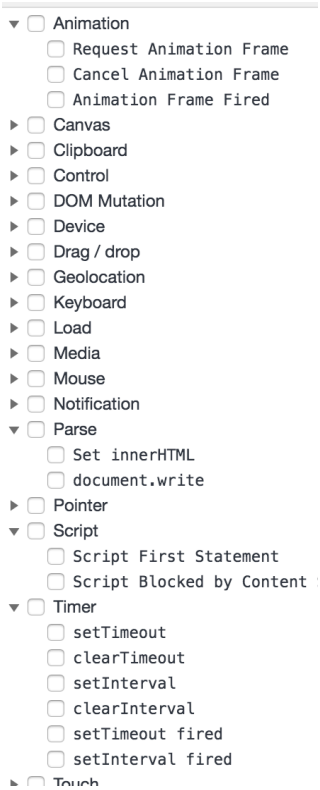
The *Sources* panel has *Event Listener Breakpoints* section. It allows to break on event type.



- Breaking on any event (for which there is an event handler) is possible through *Event Listener Breakpoints* panel.

Non-Event Events Breakpoints

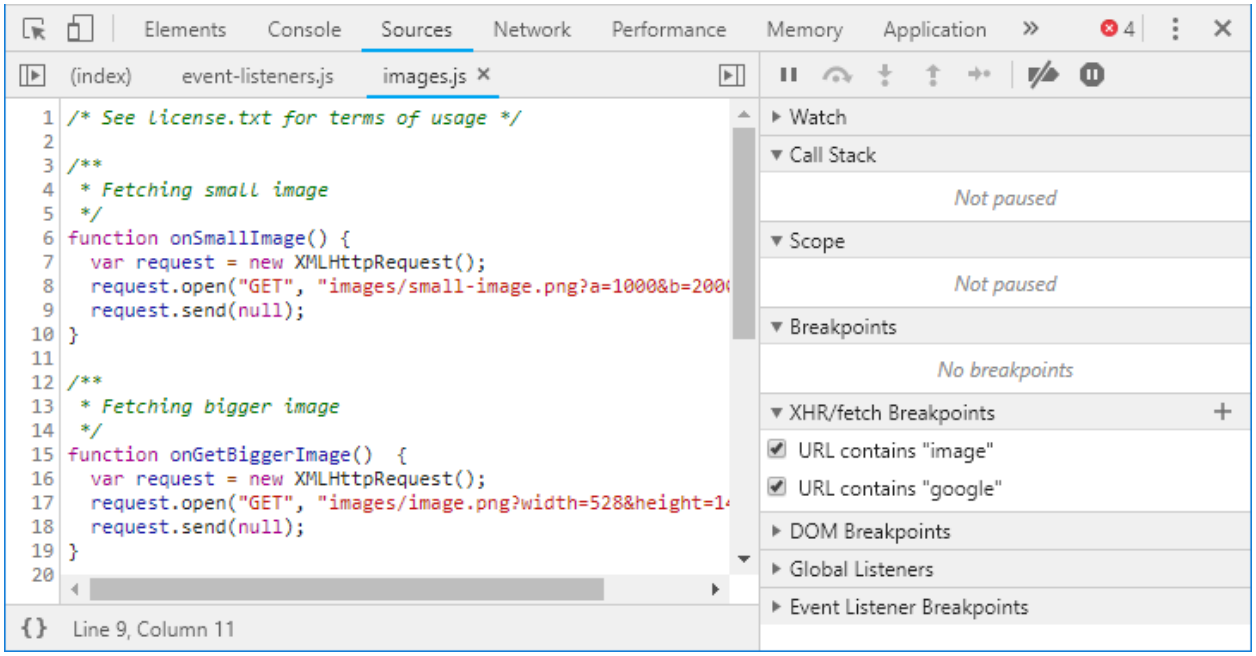
Chrome mixes the Event Listener panel with more non-web-exposed browser events:

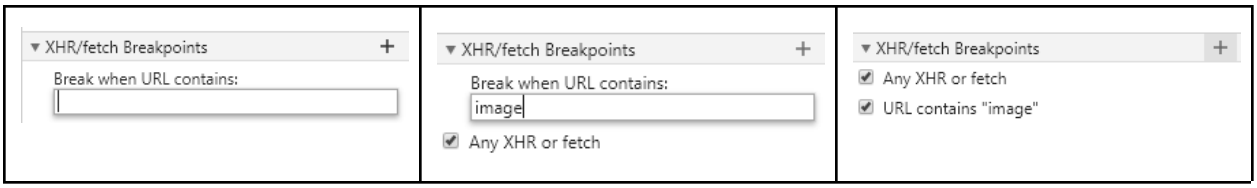


Note that existing event breakpoints are displayed together with standard JS breakpoints (within *Breakpoints* side panel)

Chrome: XHR Breakpoints

List of XHR breakpoints is displayed in its own section in the *Source* panel at the right side.



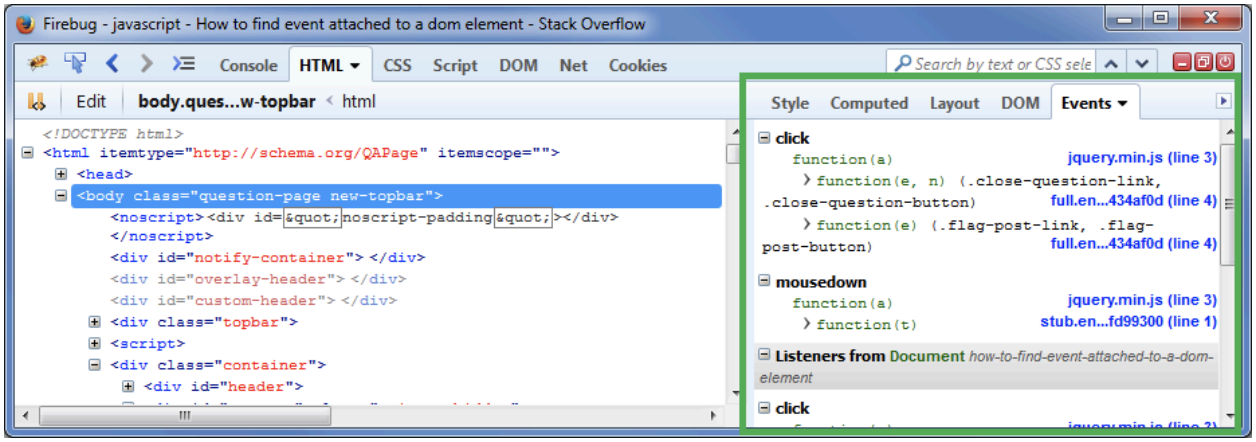


- Adding a new XHR breakpoint is done through a + button.
- An input field is displayed for specifying keyword to break on
- Leaving that field empty results with “Break on any XHR or fetch event”

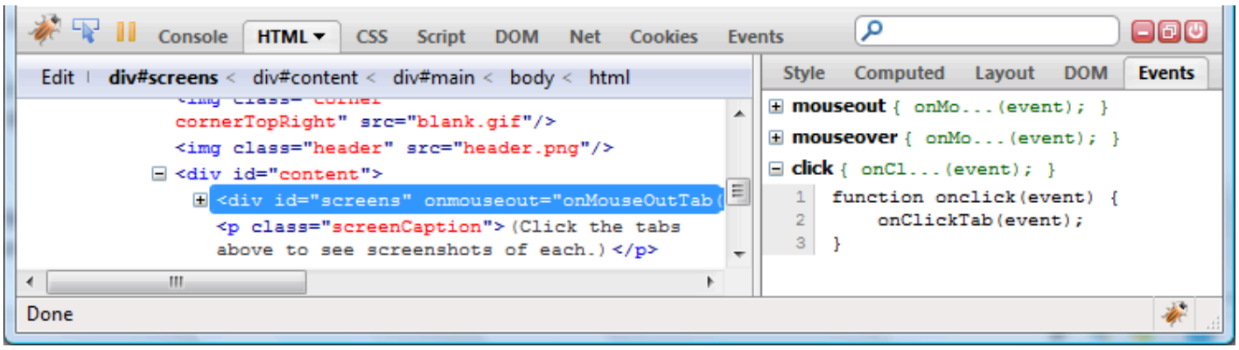
Firebug

Firebug had also UI for creating a breakpoint in an event handler.

Later version with event delegation

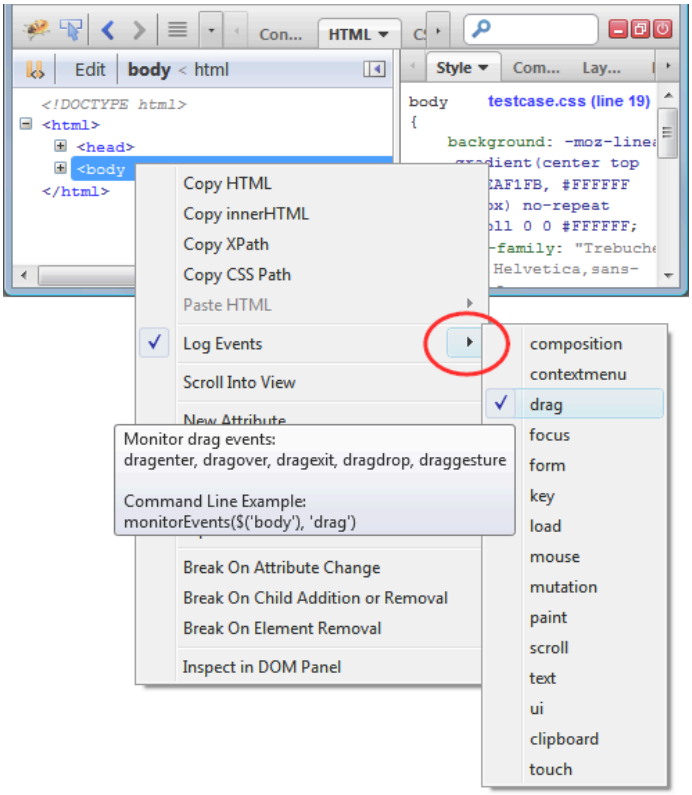


Earlier version



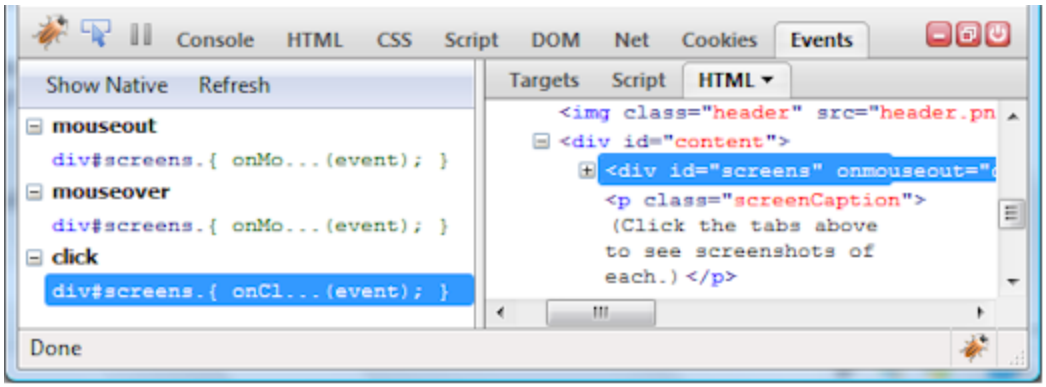
- The list of event handlers registered for selected element is displayed in the side panel.
- Source code is visible after expand offering also breakpoint gutter.
- Note that Firebug allowed to create event breakpoint through the Command line (post, docs) using `getEventListener` and `debug` commands.

Firebug also has `monitorEvents` *exposed as console command and menu:*

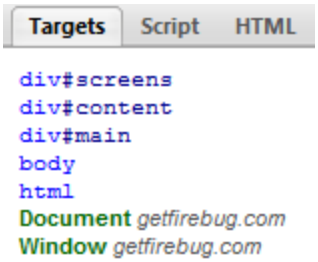


- Brian wrote about it here and Chrome documents it here.

Firebug was also having EventBug extension:

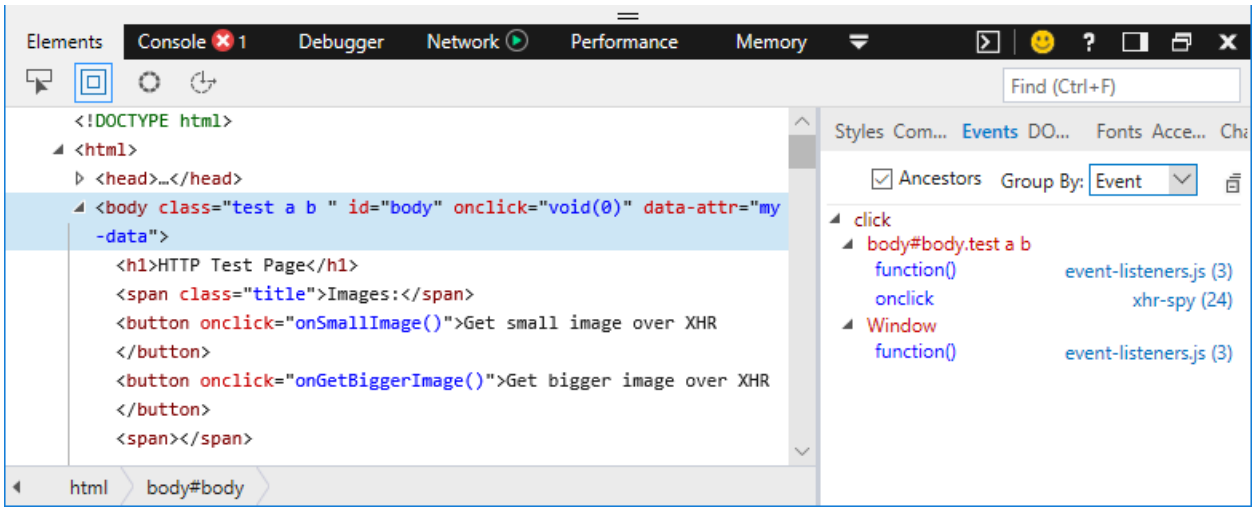


- The extension introduced a new panel listing all event handlers on the left side.
- The right side offered details for selected event handler.
- The details tab displayed e.g. Targets - Showing list of event targets that would be used as *DOMEvent.currentTarget* when event bubbles. All targets are clickable and navigate the user to the HTML panel.

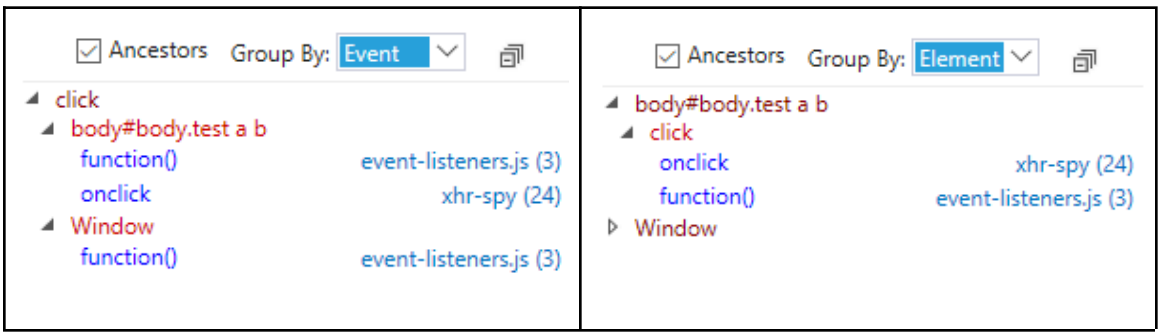


Edge

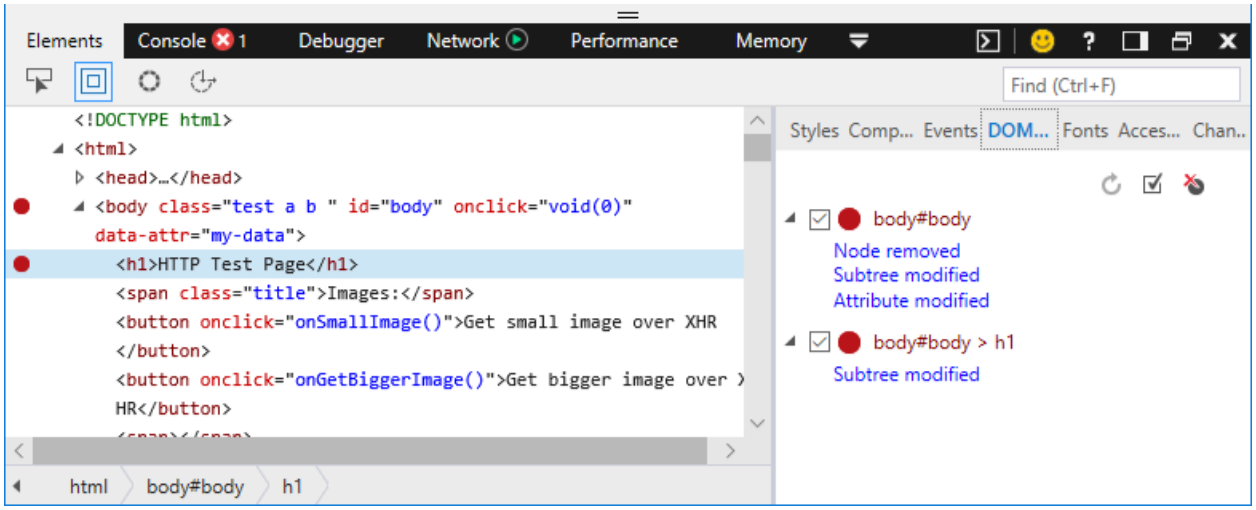
Edge implements offers very similar user experience like Chrome. The *Elements* panel has a list of events available on the right side:



- It's possible to see existing event handlers for selected node
- Displaying listeners for node ancestors is optional
- It's possible to group by event/element

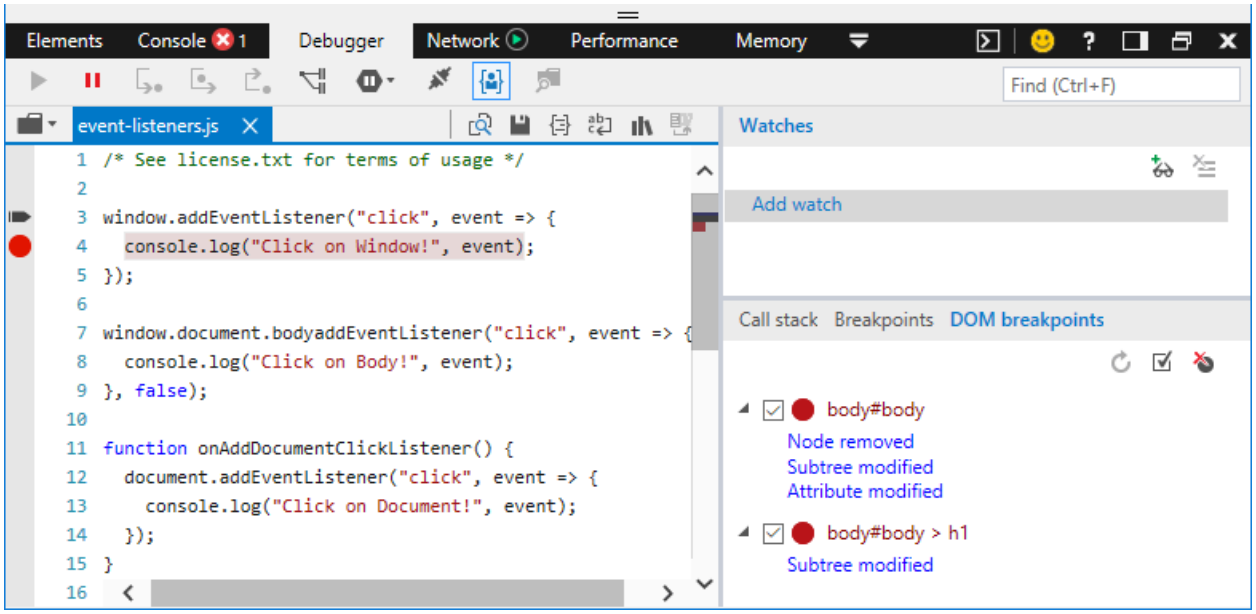


The *Elements* panel can also show list of DOM breakpoints for selected node:



- It's possible to remove/disable an existing breakpoint

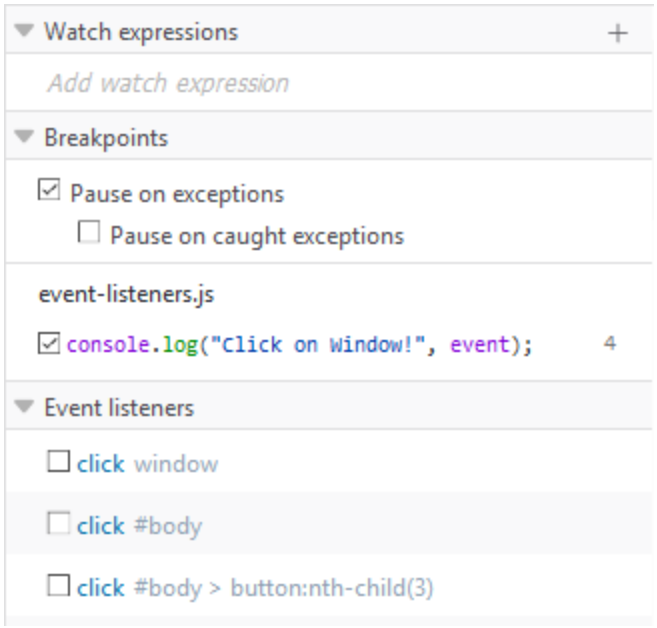
The *Debugger* panel has exactly the same *DOM breakpoints* side-panel:



Firefox DevTools

Firefox DevTools already have a support for event handler breakpoints hidden behind the following pref: `devtools.debugger.features.event-listeners`

The user can see a new *Event Listener* side panel after enabling this feature.



- The panel shows all event listeners registered in the current page.
- Clicking on the checkbox displayed in the list creates a breakpoint in the *Breakpoints* panel and collapses the *EventListener* panel

Appendix

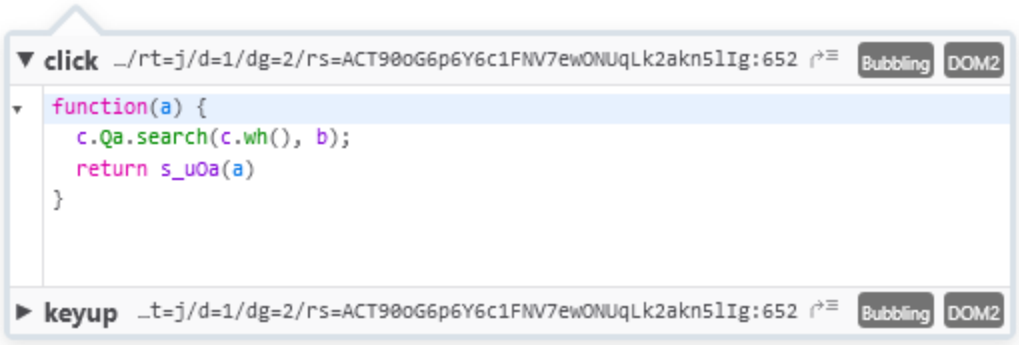
Use Cases

Use Case #1: Searching for event listener implementation

The goal is to find implementation (location in the source) of a listener registered for specific element.

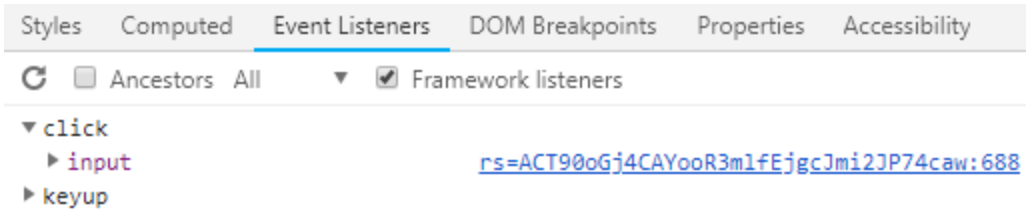
The user is asking where is the click listener for specific button on the page. Here is what should happen from the user perspective:

1. The user inspects the button on the page and opens the Inspector panel highlighting the inspected <button> element.
2. Clicking on a little *event* bubble displayed next to the element opens a tooltip-panel with a list of all event listeners added to the element.
3. In order to jump into the right location in the source code (Debugger panel) where the listener is implemented, the user can click an icon displayed at the right side



There is yet another way how to find the location:

1. The user inspects the <button> element using Inspector.
2. The inspector panel has *Event Listeners* side panel showing list of event listeners added to the selected element.
3. Clicking on a link within that panel navigates the user to the source code.



Use Case #2: Breaking in event listener

The goal is to break in specific event listener when it's executed.

The user wants to debug the listener without the knowledge where it's implemented in the code. The user could use case #1 to find the location, but if the Debugger panel is currently opened it should be possible to do it right away (and avoid switching between panels).

Here is list of steps the user needs to do:

1. The user opens *Event Listeners* side panel (available within the Debugger panel). This panel shows all existing event listeners on the page grouped by type (e.g. *Mouse*).
2. The user expands desired group (e.g. *Mouse*) and tries to locate the listener.
3. Clicking on a link displayed next to the listener entry navigates the user to the location of the listener in the source, and so it's the matter of a second click on breakpoint gutter to create the breakpoint.

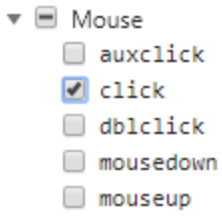
Use Case #3: Breaking on event type

The goal here is to break on specific event type.

It might be hard to locate specific listener if there are too many of them. But, knowing at least the type of the event (e.g. *click*) might be enough to break at the right place.

Steps:

1. The user is in the Debugger panel and wants to break in the first click listener that is executed when click on the page.
2. The user opens *Event Listeners Breakpoints* side panel and checks *click* event type
3. The user performs the user action - click on the page and JS execution is automatically halted in the debugger on the first statement of the first *click* listeners that is executed.
4. Resuming debugger should break in the next *click* listener (if there is one).



Use Case #4: Breaking in Framework event listener

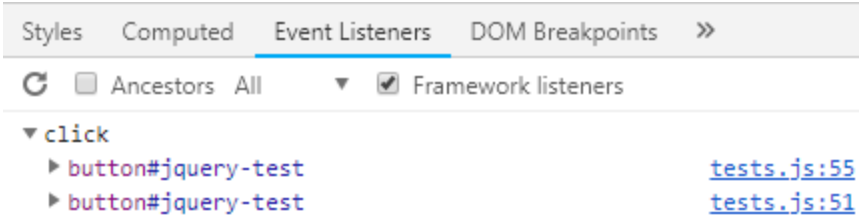
Some JavaScript frameworks (React, jQuery, ...) can use helper functions that are bound to DOM instead of the user provided functions.

In such case the user might want to break in a listener bound by the framework (in framework code base) or the user defined listener (in user code base).

Break in user defined listener

1. The user inspects an element using the Inspector and selects the element in the Inspector panel.
2. The user checks *Framework listeners* checkbox in *Event Listeners* side panel to make sure that navigation links point to the user code base
3. The user clicks one of the navigation links to get to the Debugger panel.
4. The user clicks on breakpoint gutter to create a breakpoint.

The location links point to the user defined code base.



Use Case #5: Inspecting chain of targets

The goal is visualizing the list of parent ancestor listeners.

The user wants the see all event targets (only those with a listener) that would be used as DOMEvent.currentTarget when an event bubbles.

1. The user highlights proper element in the Inspector panel and selects the Event listeners side panel.
2. The user checks *Ancestor* checkbox to show listeners on ancestors.

- Only listeners registered on selected button element are displayed.

- Event listeners registered on ancestors of the button element are also displayed.

Use Case #7: Pause on XHR/Fetch

The goal is to break when XHR/Fetch happens on the page.

The user might be asking what line of code is responsible for triggering a network request. For example, what line is responsible for appending a new <img> element or into the page.

Here is what needs to happen:

1. The user opens the Debugger panel and clicks a little + icon in the XHR/fetch side panel.
2. An input field appears allowing the user to specify what keyword the network URL needs to include for the break to happen.

The use can also check `*Pause on all requests*` checkbox to pause on all requests (with JS callstack) similarly to what we know as `*Pause on exceptions*`.

Event Listener Breakpoint

There is already a **Event Listeners** panel in the Debugger (hidden behind `devtools.debugger.features.event-listeners` pref). This panel displays list of existing event handlers on the page.

The current state:

- Showing all event handlers in flat list
- Every item in the list has a checkbox that can be used to create a breakpoint.

Improvements:

- Individual event handlers displayed in this panel should be grouped by event type. E.g. all *click* handlers should be in one group collapsed by default.
- Removing a breakpoint in the *Breakpoints* panel should not collapse the *Event Listeners* Panel
- The panel shows CSS selector for the associated element (takes quite a bit of space). There should rather be a link to source location or both (if enough space).
- It should be possible to filter by event type: All/Passive/Blocking
- It should be possible to switch on Framework support - skip event handlers bound by frameworks and show user defined functions.

Improvements when the panel is in the Inspector panel:

- The same panel should also be available in the Inspector panel and filtered according to the currently selected element (i.e. show only handlers attached to the selected element)
- It should be possible to show/hide ancestor event handlers (for selected element)

To Discuss:

- Creating a breakpoint using the event handler checkbox feature should be removed. It doesn't work well when the panel is displayed in the Inspector panel. The breakpoint existence could be indicated by an icon in front of the event handler.

Decisions

1. **Decision:** Show types of events, each event type with event handlers
2. **Decision:** Enable breaking on all click events
3. How does clicking an event type affect the child event handlers
    a. Event handlers list can be noisy, added and removed a lot (even more so with delegated events)
    b. **Decision:** Keep it simple. Disabled+checked state for event handler checkboxes
4. **Decision:** Interactions on event handlers:
    a. Click on DOM element to navigate to Inspector
    b. Highlight DOM element on hover
    c. Click on event handler source
    d. Discussion
        i. Remove event handler?
5. Should framework events be mapped (original vs generated FOREVER)
    a. Honza will investigate state of Events Bubble
    b. Follow up: Should we allow switching between original/delegated and unmapped event handlers
6. Live-updating events or manual refresh
    a. The platform allows registration of a callback for event listener changes. See `nsIListenerChangeListener` interface

7. Filtering logic, filter categories, types, sources & nodes
    a. Explore limitations
8. Show web compat warnings for events with known issues
9. Define MVP:
    a. Maybe not filtering
    b. Maybe not live-updating
    c. Maybe not framework mapping

TODO: mockup

Meeting Notes

September 13

- Make sure the UI offers the user events (not the frameworks handlers)
    - We should support jQuery and React
    - Some frameworks can add one handler for document root to get all e.g. mouse events. E.g. React has its own internal map with all users events, so the browser sees just one event listener.
    - jQuery has it’s own event handler that calls user callback.