WebDriver Input API Design Doc (goto/webdriver-input)

Status: Draft
Authors: samuong@chromium.org
Last Updated: 2016-09-22

Objective

Background

Overview

Detailed Design
- Gin binding
- JavaScript convenience functions

Objective

A new WebDriver Input API is currently being specified by the W3C Browser Testing and Tools Working Group. This document describes how this will be implemented in Chrome and ChromeDriver.

The new API will allow automation of keyboard, mouse and touch inputs, and replaces the previous input APIs. It will simulate input at a lower level, which means that inputs will exercise code paths in the browser process, not just the render process. It will also provide a “low-level” touch API, so that uncommon gestures (e.g. two-finger swipe) can be simulated.

Background

Selenium WebDriver is a browser automation tool that is commonly used for functional testing of web sites and web apps. ChromeDriver is Google Chrome’s implementation of the WebDriver protocol, and allows WebDriver tests to run against Chrome.

Some terminology used in the W3C spec is defined below:

> **Action** - similar to a JavaScript event, represents a single action performed by a user (e.g. a mouse movement, a touch press)

Action sequence - a set of actions from a single input source (e.g. a mouse), to be performed sequentially and in sync with animation frames
Action chain - a set of action sequences from potentially multiple sources (e.g. multiple fingers)

Overview

The features described in this document are expected to be used primarily for test automation, not by regular users of Chrome. So the functionality will be hidden behind a command-line switch (--enable-chromedriver). This is similar to the way that DevTools remote commands and Telemetry's gesture functions are hidden by default (by --remote-debugging-port and --enable-gpu-benchmarking, respectively).

The W3C spec describes a single WebDriver endpoint (POST /session/{session id}/actions) that accepts a JSON object describing a set of inputs. In Chrome, this will be implemented in a Gin extension, and exposed as a JavaScript function (chrome.chromedriver.synthesizeInputs). This function needs to accept two arguments: 1) the JSON object describing the input and 2) a callback that will be called upon completion of the gesture, or in the event of any error.

The synthesizeInputs function needs to: check for any errors in the input, group the input actions by time (in the spec, they are grouped by input source), and enqueue events onto the SyntheticGestureController.

As a convenience to testers, a small JS library will be provided to build the JSON object ActionSequenceBuilder will be implemented as a JavaScript library, rather than requiring them to hand-craft complex JSON objects describing test inputs.

Detailed Design

Gin binding

Needs to:

- Do error checking, detect errors *before* starting gesture
- Transpose matrix (2d array) so that events are grouped by time (protocol specifies a JSON object where events are grouped by input source)
- Convert to SyntheticPointerParams and queue onto SyntheticGestureController
- Call the callback when it is done

JavaScript convenience functions

Code to generate a swipe gesture could look something like this:

```
  var swipe = new Gesture();
  var firstFinger = swipe.addInputSource('pointer').withSubType('touch');
  var secondFinger = swipe.addInputSource('pointer').withSubType('touch');

  firstFinger.press(100, 110);
  secondFinger.press(100, 90);
  swipe.tick();

  for (var i = 0; i < 3; i++) {
    firstFinger.moveBy(0, 10);
    secondFinger.moveBy(0, -10);
    swipe.tick();
  }

  firstFinger.release();
  secondFinger.release();
  swipe.tick();

  chrome.chromedriver.synthesizeInputs(swipe.build());
```

The press()/moveBy()/release() functions ensure that multiple events cannot be scheduled during the same frame. The tick() function goes through each input source and, if it does not have an input event scheduled for the current frame, schedules a “pause” (no-op) event.

The call to swipe.build() above generates the following object:

```
[
  {
    "type": "pointer",
    "parameters": {"type": "touch"},
    "actions": [
      {"type": "pointerDown", "x": 100, "y": 110},
      {"type": "pointerMove", "x": 100, "y": 120},
      {"type": "pointerMove", "x": 100, "y": 130},
      {"type": "pointerMove", "x": 100, "y": 140},
      {"type": "pointerUp"}
    ]
  },
  {
    "type": "pointer",
    "parameters": {"type": "touch"},
    "actions": [
```

```
      {"type": "pointerDown", "x": 100, "y": 90},
      {"type": "pointerMove", "x": 100, "y": 80},
      {"type": "pointerMove", "x": 100, "y": 70},
      {"type": "pointerMove", "x": 100, "y": 60},
      {"type": "pointerUp"}
    ]
  }
]
```