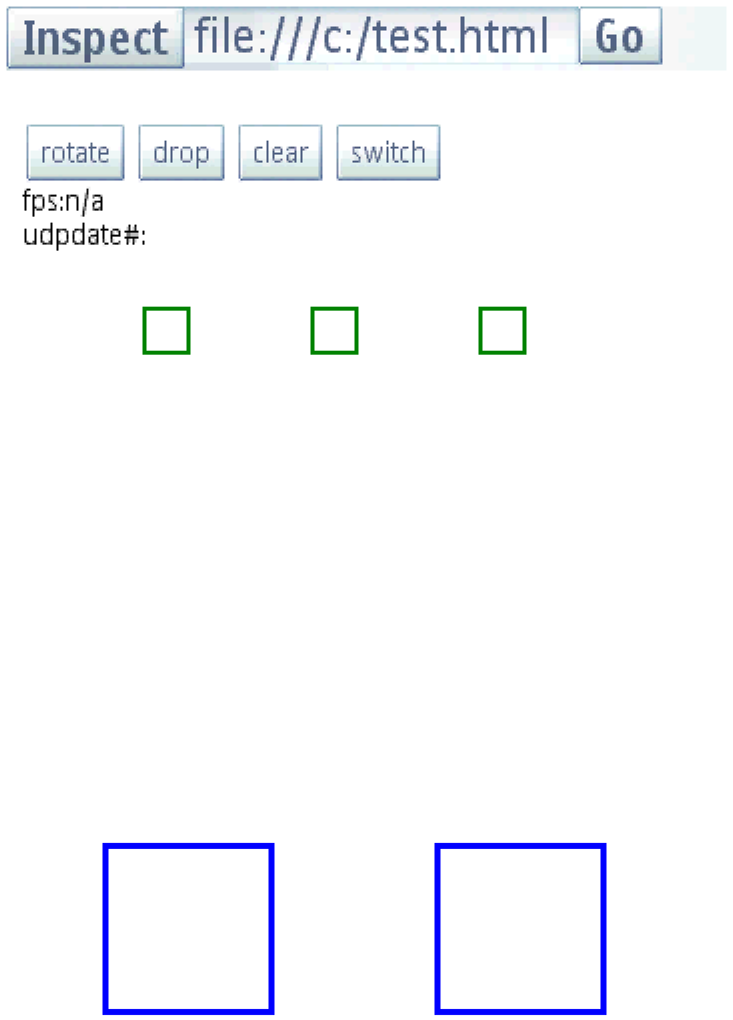
Performance JS APIs

- webkitAfterPaint
    - Slightly modified version of mozAfterPaint

```
module events {
  interface [
      GenerateConstructor,
   ] WebKitAfterPaintEvent : Event {
      readonly attribute unsigned short paintTime;
      readonly attribute [CustomGetter] Array clientRects;
   };
}
```

  - strictly for testing purposes
  - event fires at the document and bubbles up to the window
  - offers 2 attributes
    - paintTime – how long the frame takes to paint (not webcore frame, video frame)
    - clientRects – array of rectangles are being painted
  - Usage
    - window.addEventListener("webkitAfterPaint", log, false);
    - function log(e){ store.push([e.paintTime, e.clientRects]); }
    - window.removeEventListener("webkitAfterPaint", log, false);
    - result is in strore var



```
fps:n/a
udpdate#:
```

Performance JS APIs conti.

- FPS measurement
  - Profile rendering speed on WebCore level.
  - module window {

```
 interface Console {
                    ...
                    void startPaintPerformanceMeasure();
                    unsigned short stopPaintPerformanceMeasure();
                    ...
   }
}
```

  - Useage
  - window.console.startPaintPerformanceMeasure();
  - document.getElementById(“fpscounter").innerHTML = "fps:" + window.console.stopPaintPerformanceMeasure();
  - Measuring starts when the first paint event occurs after calling startPatinPefromanceMeasure() and ends on the last paint event before stopPaintPerfromanceMeasure() gets called.
  - FPS counting could be doing using webkitAfterPaint as well, but bubbling events on each paint frame could impact the fps number.

Performance JS APIs continute

- Web Timing API –next thing to do. This is mainly quoting from the link below.
  - http://dev.w3.org/2006/webapi/WebTiming/
  - User latency is an important quality benchmark for Web Apps. While JavaScript-based mechanisms can provide comprehensive instrumentations for user latency measurements within an application, in many cases, they are unable to provide a complete end-to-end latency picture. For example, the following Javascript shows the time it take to fully load a page:
  - <script type="text/javascript">

```
    var start = new Date().getTime();
    function onLoad() {
        var now = new Date().getTime();
        var latency = now - start;
        alert("page loading time: " + latency); }
```

  - The script calculates the time it takes to load the page after the first bit of JavaScript in the head is executed, but it does not give any information about the time it takes to get the page from the server.
  - This WebTiming interface allows JS to provide complete client-side latency measurements within applications. With the proposed interface, the previous example could be modified to provide information about user's perceived page load time.
  - The following script calculates how much time has elapsed since the occurance of a navigation event, such as the mouse click on a link.
  - <script type="text/javascript">

```
function onLoad() {
var now = new Date().getTime();
var latency = now - window.pageTiming.navigationTime;
alert("User-perceived page loading time: " + latency) }
```