Developer feedback needed: Frame Timing API

By Paul Lewis
Paul is a Design and Perf Advocate

Dogfood: this API is not yet implemented; we want feedback from developers.

Over the past few years browsers have made huge strides in terms of rendering performance, especially on mobile. Where previously you had no hope of hitting a smooth framerate for anything remotely complex, today it's at least achievable if you take care.

For most of us, though, there's a disconnect between what we can reasonably test on our own devices and what our users experience. If they don't get a silky smooth 60fps then their experience is impaired, and ultimately they'll go elsewhere and we'll suffer. Just as well, then, that the W3C is discussing a new API that could help us see what our users see: the Frame Timing API.

Jake Archibald and I recently recorded a video overview of the API, so if you prefer watching over reading take a look:

Uses of the Frame Timing API

There are undoubtedly a bunch of things you could do with the Frame Timing API, and, crucially, you get to measure what's important to you and for your project. Even so, here are a few ideas:

- Tracking the fps of your JavaScript and CSS animations.

- Tracking the smoothness of your page scrolls (or perhaps that nifty infinite scrolling list you have.)
- Automatically scaling back your showbiz effects based on the device's current load.
- Regression testing on runtime performance metrics.

The elevator pitch

Here's what the API currently looks like in the spec: with it you would get to pull data on renderer thread timing, where JavaScript, styles and layout run. (You may have heard the renderer thread called the main thread; it's the same thing by another name.)

```
var rendererEvents = window.performance.getEntriesByType("renderer");
```

Each of the renderer thread records you get back look roughly like this:

```
{
  sourceFrameNumber: 120,
  startTime: 1342.549374253
  cpuTime: 6.454313323
}
```

Each record is essentially an object that contains a unique frame number, a high resolution timestamp for when the frame started, and another for how much CPU time it used. With an array of these you can look at each of the **`startTime`** values and find out if the main thread is going at 60fps; essentially "does each frame's **`startTime`** go up in roughly 16ms chunks?"

But more than that, you also get the **`cpuTime`**, so you'll know if you're comfortably inside the 16ms boundary, or if you were down to the wire. If the **`cpuTime`** is right up near the 16ms boundary there's not much room for things like garbage collection kicking in and, with CPU usage high, battery consumption will also be higher.

In addition to the renderer thread, you also get to pull data on compositor thread timing, where painting and compositing happen:

```
var compositeThreadEvents = window.performance.getEntriesByType("composite");
```

Each of these also come back with a source frame number, which you can use to tie back to the main thread's events:

```
{
  sourceFrameNumber: 120,
```

```
  startTime: 1352.343235321
}
```

Because of the way compositing often works in browsers, there may be several of these records per renderer thread record, so you can use the `sourceFrameNumber` to tie those back to together. There's also some discussion as to whether there should be CPU time in these records as well, so if you feel strongly speak up on the GitHub issues.

More information

This API isn't shipping yet, but hopefully it will do soon. In the meantime here are some things you can read and do:

- **Read the explainer doc on the repo.** There is a lot of nuance about how you should best record the frame data for it to be meaningful, and the explainer gives some direction here.
- **Check out the latest draft of the spec.** It's pretty light, and it's well worth a read.
- **File issues for missing features or potential headaches.** You know what you would want to measure, so please do provide feedback if you think there's something you can't do with the API that you'd like to.

Last updated July 2, 2018.