What I Learned This Month: Don’t Wait to Respond to the User

Scott Chapman
mikagika, ltd.

As you may remember, along with being a mainframe sysprog (can I still use that term today?), performance analyst and capacity planner, I’m also a webOS developer. This month’s lesson comes from my webOS experience, but I want to share it because of a couple of ideas that apply beyond webOS. Next month I’ll return to talking about the mainframe, I promise!

If you’re not familiar with it, webOS was the operating system developed by Palm for their new smart phones released in 2009. Since then, HP has purchased Palm and is now talking about bringing webOS not just to smart phones and tablets, but eventually to PCs as well. What is that going to be like? I have no idea! But it sure sounds interesting.

The genius of webOS is that it is completely built upon web technology: HTML, CSS, and JavaScript. So if you know something about writing web applications, then you know something about writing webOS applications and vice-versa.

One of my webOS apps (a calculator) has had intermittent performance problems since I released it—it occasionally would not respond to the user tapping the buttons. And it would consistently stutter and pause when scrolling through the preferences. While testing on a new Pre2 phone with the latest webOS 2.0, I discovered that instead of occasionally ignoring my input, it was ignoring one out of every 3 or 4 taps! That’s clearly not acceptable. Strangely, others with the same hardware and OS reported little or no problem.

It seems that the Pre2 is slightly more sensitive to me rolling my finger as I pressed the on-screen buttons, which meant I was lifting my finger from a very slightly different position, which it interpreted as a drag, not a click. The fix was simple: change the event I use to detect that the user has pressed a button from “onclick” to “onmousedown”. In normal web parlance, when the user presses the mouse button when the pointer is over a particular element, the element will trigger the “onmousedown” event. When the user releases the button, the “onmouseup” and “onclick” events are typically triggered. So in retrospect, this seems perfectly obvious: when do I want to detect that the user has pressed the calculator button? When they press the button of course (onmousedown), not when they lift their finger away from it! D’oh!

Flush with the excitement of fixing that problem, I became determined to fix the preferences scene. (A “scene” in WebOS is analogous to a dialog in Windows or a page in a traditional web app. In fact, the format and layout of a scene is defined in an HTML file.) I suspected that it had something to do with the complexity of the HTML, although it wasn’t really that complicated at all. So I started removing elements until it magically started working. It turns out that the problem was a <div> element that had a background set to an image to provide a nice background gradient. Switching the background to a solid color instead of an image immediately fixed the problem: scrolling was as smooth as silk! I don’t really understand why, but probably the issue related to scaling the image to fit the size of the element vs. simply filling the element with a solid color.

So the lessons this month are pretty basic, but apparently I needed to relearn them. Always provide your users with feedback as soon as possible: if you can trigger an action on multiple events, use the one that happens first. And when faced with an HTML performance problem that you can't localize, the old trick of commenting out code until it performs acceptably works with HTML as well as any other language.

So that's what I think I learned this month. If you think I didn't learn my lesson correctly, please email me and let me know! This month, since we're talking about webOS, feel free to contact me at geek@mikagika.com.