SitePoint Design View

Issue 79: January 19, 2011 The Useful and the Usable in Web Design

http://www.sitepoint.com/newsletter/viewissue.php?id=5&issue=79#5

Hey, Hey! Ho, Ho! 960 Grid Has Got to Go

Let's travel in our minds way back to the heady days of 2007: Apple has announced (*gasp!*) a "phone," Mika is chirping manically about Grace Kelly, and a handful of well-designed CSS frameworks arrive on the scene. The two attracting the most buzz are Blueprint and 960 Grid System.

By this time, *most* web developers have embraced the concept of CSS layouts, but many are tired of reinventing the wheel for each new project. CSS frameworks begin winning fans by handling a lot of the grunt work for users, while offering familiar class names and structures.

Of course, when you standardize something, you tend to lock in certain foundations, and one of these givens was a set-width layout. When you knew the precise layout width, it was much easier to calculate columns and gutters.

According to W3Counter's figures for June 2007, more than half of all users (50.54%) are viewing the Web on 1024x768px displays, and almost 94% of users are using displays no larger than 1280px. With most displays within such a limited range, it's easy to understand why both Blueprint and 960.gs (not to mention the many frameworks based on them) settled on a fixed 960 pixel-wide grid.

StephenCaver.com. A great responsive layout.

Now, this grid has served us faithfully, but we all know four years is a long time in web development.

Fast forward to 2011, and there has *never* been more variation in the display ecosystem. Today, almost 30% of displays are larger than 1280 pixels—and 8% of these displays are wider than 1680 pixels. On the other hand, 960 pixels is often too wide to be truly useful on a slew of smartphones and smaller tablet devices.

Unfortunately, old-school fluid (or liquid) layouts aren't the answer. What single layout could work at 600 pixels and 1920 pixels?

What we need are layouts that respond to the viewports they find themselves in. Last year, the term "responsive

web design" was coined for this new breed of self-adjusted layouts. Ethan Marcotte wrote a great primer on *A List Apart*, and a few wonderful working examples began to appear—notably from Stephen Caver and Jon Hicks.

Okay, but where do you start?

Most of these responsive layouts use CSS media queries. If you haven't used them before, media queries let us set simple tests that determine which parts of our CSS are rendered (and which aren't). For instance, you might say "if this is a monochrome display, use 'stylesheet X'".

Here's the syntax:

```
<link rel="stylesheet" href="small.css" media="only screen and (max-width:
1023px)" />
```

This line will attach the "small.css" stylesheet to our page, but *only* to devices with browser screens that are smaller than 1024 pixels. The moment the user resizes their browser window to a width larger than 1023px, the "small.css" is removed from the document.

We can just as easily set minimum widths for specified styles and even these chain media queries together. For example:

```
@import url(/mid.css) screen and (min-width:800px) and (max-width:1280px);
```

The code above lets us set a range of 800 pixels to 1280 pixels where "mid.css" will be applied. We now have the ability to reflow and restructure our layouts based on the available real estate. Powerful stuff, indeed.

If you have yet to work with these responsive layouts, they take a little more consideration than standard set-width layouts. Where do you start?

Last week, Louis Simoneau covered the great starting point—the Less Framework. Today we're going to look at another responsive CSS framework from Andy Taylor: CSSGrid.net.

CSSGrid.net

CSSGrid goes a little further than Less Framework by building in a complete 12-column layout framework.

Andy's approach works like this:

Wide screen: On large displays the layout scales fluidly up to 1139 pixels. At higher screen widths, the layout locks at 1140 pixels and centers your content.

Medium screen: On smaller desktop displays, the layout scales down fluidly until it reaches 768 pixels.

Smaller screen: On any device less than 768 pixels-wide (most often, but not exclusively, phones and smaller tablets), the layout automatically breaks down to a stacked single-column layout.

Here's a demo page I've put together that lets you see the switching in action.

I think there are lots of reasons to like this approach. First, we can let people make better use of their large displays without the risk of our content being strung out like a meandering ant trail on super-wide monitors. This issue was always the killer blow to traditional fluid layouts.

On medium-sized desktops, I believe we have a much better experience than static 960-pixel grids. Sure, they can use almost all of their 1280-pixel display if that's what they want to do.

However, it's not uncommon for users to run launch bars, Twitter clients, or instant message apps on the right side of the screen. I know on my laptop, I'm unable to read any 960-pixel layout without it overlapping my Twitter client.

My Twitter client will always overlap any 960px layout

Allowing users to tailor your layouts to whatever space they have available helps both them and you.

There's not much argument that mobile users are becoming more widespread every day and Andy's CSSGrid provides an elegant solution out of the box. Content will stack vertically on any device less than 768 pixels wide. You write it once, and no device detection is required.

Browser support is good, with all current browsers supporting media queries. It's true that older IEs won't allow you to set a maximum width, but I'm willing to risk an angry letter from that one guy running IE6 on a 1920-pixel monitor. But I'm crazy like that.

It certainly looks like responsive layouts will be one of the big web development trends of 2011. Will you be changing your approach?

Or perhaps you already have.