

CSS Design: Going to Print

by [FSJ NFZFS](#) · May 10, 2002

Published in [CSS, Layout & Grids](#) · [109 Comments](#)

B opuf ggn uif fejpst;Uif qsjputuznf tiffuf ejdvtftf kfrpx kbt wtfe tj BMB 3/1 -k iptf GfcsvbszB112 DTT
sfeftho lifmfefv tifs tj uif n pefso DTT nqzvufso/tpn f lefubjt kfrpx kfsabj ponz up libuhbzpvu-boe bpulp BMB
4/1/Cvulif qsjpdtjnf Fsjn fzfsejdvttft tj uij bsjnf bsf bt usvf boe bthf ofsbmz bqgnjbcmf lpebzbt uifz
xfsf kifo uij bsjnf flstubbqgfbsfe tj BMB/

You've seen them before: links that say "click here for printer-friendly version" or words to that effect. Every time you follow one of those links, you load a separate document that presents exactly the same information with a different layout, and probably different markup.

That means somebody (or a script) had to take the original document and convert it to a stripped-down version that's more suitable for print output. Maybe that somebody was even you.

Uisotitubot;

Gsfodi

Svttjbo

Qsjputuznf tiffut up uif sftdvf

One of the wonderful things about CSS is that it allows authors to create media-specific styles for a single document. We're pretty used to styling for the screen, but thinking about other media isn't a habit yet. And as all the "printer-friendly" links attest, our thinking about the print medium has been limited to recreating a document in a different way.

Why bother, when the power to offer your readers a better view of your material in print is no further away than a well-structured document and a media-specific style sheet?

You can take any (X)HTML document and simply style it for print, without having to touch the markup. Worries about version skew between the web and print versions suddenly become a thing of the past. Best of all, it's simple to do. (For more information on the basic principles involved in creating media-specific stylesheets in general and print styles in particular, see "Print Different (<http://www.meyerweb.com/eric/articles/webrev/200001.html>)" at meyerweb.com.)

Let's look at how A List Apart got some new print styles that danced around a browser bug and, in the end, made the printed output look much better.

Gtyjphb -pbu-vc

As you can see by visiting Bugzilla entry #104040 (http://bugzilla.mozilla.org/show_bug.cgi?id=104040), Gecko-based browsers like Netscape 6.x or Mozilla have a problem with printing long floated elements. If a floated element runs past the bottom of a printed page, the rest of the float will effectively disappear, as it won't

be printed on the next page.

If you have a site styled like A List Apart, and the entire article content is contained in one big float, then that means readers will only get the first page of the article.

The fix, as it turns out, is to un-float the content when the page is printed. Simply put, all you have to do is assign any large floated element to have `float: none`. Doing this returns the floated element to the “normal flow” of the document, which is printed as you might expect, a page at a time until the end of the document.

So that’s what I recommended Zeldman do for ALA, and once he did, the printing problem was solved. Gecko-based browsers still have the bug (as of this writing), but at least it’s something that can easily be worked around.

Uif tubsujh qpju

Here’s the print-medium style sheet that A List Apart was using once the float bug was cleared away:

```
#menu {  
  display: none;  
}  
  
#wrapper,  
#content {  
  width: auto;  
  border: 0;  
  margin: 0 5%;  
  padding: 0;  
  float: none !important;  
}
```

It’s a good start. This style sheet removes the right-side menu completely from the document, so it isn’t printed (`display: none`), and resets the margins and padding of the article’s content so that the text will flow from one side of each page’s printable area to the other.

The problem, as I saw it, was that too many of the styles intended for the screen were leaking through to the print.

If we look at the `head` of a recent ALA article, we find (among a lot of other stuff) the following lines:

```
<style type="text/css" media="all"> @import "nucss2.css";</style>  
<link rel="stylesheet" type="text/css" media="print" href="print.css" />
```

We have a style sheet, `print.css`, which has been restricted to the print medium with the use of the attribute `media`. The style sheet `nucss2.css`, which is being imported in order to hide it from Navigator 4.x, will be used in *all* media the user agent supports. That’s screen, print, projection, aural—you name it. Unless we’re careful, we could end up with background colors and pixel-based font sizes in our printed output.

Is this a tragedy? No, but most browsers are configured not to print backgrounds; and pixel-based fonts, while nice for screen media, are not so useful in print media.

So let's do something about these and a few other layout issues.

X ijfxbtife cdblhspvoet

A lot of background colors get applied to things in the right-hand sidebar. Since that's dropped for print, we don't have to worry about changing anything within the sidebar. This removes numerous potential headaches right away.

Since printers don't print white, we want to set the page's background to be white. We also want to remove any background images that might have leaked in.

If we simply use the property `background`, we can accomplish both tasks with a single value. In order to leave ourselves a little flexibility, we'll set the `body` background to be white, and the content and wrapper elements to have a transparent background (thus letting the page's white shine through):

```
body {
  background: white;
}

#menu {
  display: none;
}

#wrapper,
#content {
  width: auto; margin: 0 5%;
  padding: 0;
  border: 0;
  float: none !important;
  color: black;
  background: transparent;
}
```

Setting the foreground color to go with the background colors wasn't absolutely necessary, but it's always a good idea. Now we have two non-floated elements (`#wrapper`, `#content`) with no visible background, and a page with a white background.

You might be worried that Navigator 4.x does terrible things with the value `transparent`, but be of good cheer: NN4.x only pays attention to style sheets that have a `media` value of `screen`. Just like with `@import`, the styles in our print style sheet are forever hidden from Navigator 4's rheumy eyes. So no worries there.

Tj{ jh uif gpougsqsjou

The all-media style sheet is sizing our printed text to be 11px tall, which seems like a poor choice. The font family (Georgia to start, and serif fonts thereafter) is fine for print, where serif fonts are usually preferred. Only the size needs a change. Thus:

```
body {
```

```
background: white;
font-size: 12pt;
}
```

“Wait!” you’re probably exclaiming. “Heretic! Everyone knows points are evil! Todd Fahrner said so (http://style.cleverchimp.com/font_size/points/font_wars.GIF)!”

Well, yes, he did, and he was entirely correct—for screen media, points are a horrible choice. In print, points make just as much sense as they have for decades now. Since we *are* styling for print, setting our `body` font size to 12 points is actually a good thing.

You can of course pick any size you like, but 12 points is a very common size. And since Zeldman’s all-media styles size elements in the article content in relation to the `body`, we’re all set there.

N bshjbmbj

The existing styles for the wrapper and content `div`s sets their left and right margins to be 5%. That means that there will be “blank space” to either side of the article, and each blank area will be 10% the width of the page’s printable area. That’s because the content `div` is inside the wrapper, and each one has 5% on each side. Add ’em up and you get 10%.

The original all-media styles set a 15% right padding on the content `div`. The margins already give us 10%, so we just need an extra 5%. That’s easy enough to do:

```
div#content {
  margin-left: 10%;
}
```

Another option would have been to leave the margin alone and give the left padding a value of 5%. Since the content `div` doesn’t have a visible background, either approach should have the same effect.

Unfortunately, some browsers have trouble handling padding appropriately, so it’s currently a better idea to move things around with margins whenever possible.

Qspufe mjlt

One tricky question was what to do about hyperlinks. Obviously they aren’t going to be as useful in print as they are onscreen, but it’s often important to provide some clue that there were links in the original. So here’s what I devised:

```
a:link,
a:visited {
  color: #520;
  background: transparent;
  font-weight: bold;
  text-decoration: underline;
}
```

This gives the links a color dark enough to be close to black in grayscale output, while still using a dark red that will show up on a color printout. The boldfacing and underlining ensure that the text of the links will stand out.

In a fully CSS2-conformant browser, we can parenthetically insert the URLs of the links after each one, thus making them fairly useful to anyone who has a copy of the printout and a web browser handy. Here's the rule, which restricts this effect to the "content" div and thus avoids sticking a URL in the masthead:

```
#content a:link:after,  
#content a:visited:after {  
    content: " ( " attr(href) " ) ";  
    font-size: 90%;  
}
```

Try it out in a Gecko-based browser, like Mozilla or Netscape 6.x. After every link in the printout, you should see the URL of the link in parentheses.

In any browser that doesn't understand the rule, there should be no ill effects—the links will still be dark red, underlined, and boldfaced. They just won't have a URL appear after the text of the link, that's all.

Note that the spaces before and after the parentheses are actually part of the rule above—the spaces have to appear in the rule in order to be inserted into the document.

There is one aesthetic problem with this new rule, precisely because it causes the value of a link's href attribute to be inserted into the document verbatim.

If we look at the code of ALA pages, we'll quickly notice there are a lot of "rooted" URLs like /issues/144/. Those will be dropped into the document exactly as they are, which makes them less useful than if they were displayed as an absolute URL.

As it happens, the CSS3 selectors draft offers us an out. Any attribute selector that uses the operator ^= selects elements based on the beginning of their attribute values. Thus we can select any href that starts with a slash and insert enough text to fill out the value.

```
#content a[href^="/"]:after {  
    content: " (http://www.alistapart.com" attr(href) " ) ";  
}
```

This rule transforms a value like /issues/144/ into http://www.alistapart.com/issues/144/. It won't help with relative URLs that don't start with slashes, but fortunately ALA doesn't use those kinds of URLs.

!IMPORTANT: As mentioned, ^= is a CSS3 selector. The W3C CSS validator can only test for compliance with CSS1 and CSS2. Unable to understand the CSS3 selector, the W3C validator will report it as an error, even though it is perfectly valid per the CSS3 Selectors Candidate Recommendation.

Fyufoejph uif nbtuifbe

In doing a final review of the print styles, I realized that the masthead of the site bugged me. It wasn't that it existed so much as it had an "underline" as part of the graphic. That line, of course, ended with the edge of the graphic. It occurred to me that it would be possible to pull a little sleight-of-hand and make the line extend to the width of the article text.

The first step was to give the content `div` a top border that was, like the underline in the graphic, a solid one-pixel dark red line. Since I was going to create a visible top border for the content, it also seemed a good idea to insert some padding between the border on the content.

```
div#content {  
    margin-left: 10%;  
    padding-top: 1em;  
    border-top: 1px solid #930;  
}
```

By itself, that would draw a line just underneath the masthead graphic. It turns out that said graphic is wrapped in a `div` element all by itself, so the bottom margin of that `div` is used to pull the rest of the content up into itself.

```
div#mast {  
    margin-bottom: -8px;  
}  
  
div#mast img {  
    vertical-align: bottom;  
}
```

These simple rules cause the content `div`'s top border to line up with the underline in the masthead graphic. The border is actually overlaying the graphic, but since the colors match it's difficult (if not impossible) to realize that's what is happening. Even if a browser decided to put the graphic on top of the content's top border, the graphic and border would still line up and create the illusion.

The second rule shown, the one with `vertical-align`, works around an interesting Mozilla behavior that is only seen when a document is rendered in "standards" mode, as are all ALA articles. See my Netscape DevEdge article "Images, Tables, and Mysterious Gaps (<http://devedge-temp.mozilla.org/viewsource/2002/img-table/>)" for more information on this behavior, and various workarounds (including the one used above). Also see Better Living Through XHTML (</articles/betterliving/>) in ALA 137.

As for the background color of the masthead graphic, there's really no way to remove it short of actually setting those pixels to be transparent in the graphic itself. We could set a matching background color on the masthead `div`, but that would probably be overkill for so light a background.

X ibufmtf@

That's all we did for this redesign, but it certainly isn't the end of what's possible.

Fiddling with colors to be more printer-friendly is one possibility, as is tweaking the column's margins so they're based on points (or picas, or even inches) instead of percentages. In the print world, you have almost as much room to design as you do on-screen.

One area we didn't scratch as deeply as we could have is font sizing. A number of classes, like `superfine`, were left to use the all-media styles based on pixel sizes. Resizing them for print is as simple as what we did for the `body` element—just write a rule that addresses the element(s) in question, and assigns a new value to `font-size`.

To pick one example, we might have written `.superfine {font-size: 9pt;}` ... or any other value that seemed appropriate. There are other elements, like `footers` and `pre`, that could benefit from similar work, and they may appear in future versions of the ALA print styles.

Uif foe sftvnu

Here's the print style sheet we ended up with after making the changes:

```
body {
  background: white;
  font-size: 12pt;
}

#menu {
  display: none;
}

#wrapper,
#content {
  width: auto;
  margin: 0 5%;
  padding: 0;
  border: 0;
  float: none !important;
  color: black;
  background: transparent none;
}

div#content {
  margin-left: 10%;
  padding-top: 1em;
  border-top: 1px solid #930;
}

div#mast {
  margin-bottom: -8px;
}

div#mast img {
  vertical-align: bottom;
}

a:link,
a:visited {
  color: #520;
  background: transparent;
  font-weight: bold;
}
```

```

text-decoration: underline;
}

#content a:link:after,
#content a:visited:after {
  content: " (" attr(href) " ) ";
  font-size: 90%;
}

#content a[href^="/"]:after {
  content: " (


---



```

As minimal and (in some ways) crude as this style sheet may be, the effect it has should be obvious to anyone holding a printout next to the same article online. Similarly useful and dramatic changes are possible for almost any design, from the simple to the sublime, and these sorts of media-specific styles free authors from ever having to create another “printer-friendly” copy of a document.

Bcpvuuf Bvuips



Fsjd N fzf s

Eric Meyer has been working on the web since late 1993 and is the founder of [Complex Spiral Consulting](#), co-founder of the [microformats](#) movement, and co-founder (with [Jeffrey Zeldman](#)) of [An Event Apart](#). A widely read [author](#), Eric enjoys a good meal whenever possible and considers almost no type of music to be worthless. He lives in Cleveland Heights with his wife and their three children.

N PSF GSPN UIJT BVUIPS

[The Vendor Prefix Predicament: ALA's Eric Meyer Interviews Tantek Çelik](#), [Prefix or Posthack](#), [From Switches to Targets: A Standardista's Journey](#), [ALA's New Print Styles](#)