Print Developers Community is a place for information and discussions on implementing print in software. Sponsored by HP, but open to all, find us at print-dev.com.

Using PDF generation to make beautiful webpage prints

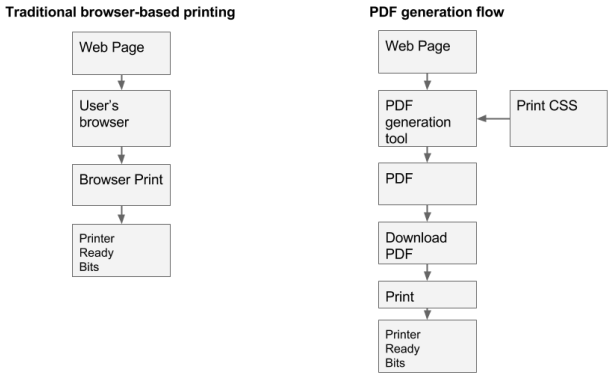
This tutorial demonstrates how to make great looking, cross-browser compatible printable forms of web content right from the HTML you already have live.

We'll be using a great PDF-generation tool called PrinceXML. PrinceXML is commercial software, although a fully-functional free version is available for download for non-commercial use.

Many of the principles demonstrated also apply to making good-looking prints right from the web browser, but using PrinceXML to generate a PDF ensures the results will be the same regardless of browser, and allow us to take advantage of extra page-layout abilities that aren't supported by all browsers.

Background

It might not be intuitive why we would convert a webpage to a PDF to print it. Let me start by showing the two different printing paths to make it clear what we're talking about.



The diagram above shows two different approaches to printing web content. The approach on the left is the traditional method. The user's web browser retrieves the HTML, the user clicks print, the browser renders the HTML into a printer-ready format using print API calls on whatever operating system it's on. Optionally, if print-specific CSS is available via a media query, that CSS is used for print.

The benefit is that it's simple and built into the browser. The downside is that most web browsers don't support the full range of CSS that's available for printing. In many cases, support for particular features might be missing (multi-column support, hyphenation) or imperfect, making it difficult to achieve high-fidelity prints. In addition, you get some of the same cross-browser problems you get with normal webpage rendering, meaning that you need different CSS for different browsers.

The path on the right is more complicated, but it's also more robust, consistent, and supports all the CSS you need to achieve high-fidelity prints.

In the second approach, the HTML is retrieved not by the user's web browser, but by a dedicated PDF rendering engine. There are two popular ones at this time: wkhtmltopdf and PrinceXML. The former is free and open-source, the latter is costly but amazing. (By the way, I have no association with PrinceXML. I just like their tool.) The PDF rendering engine can optionally override the default CSS with a special print CSS source, and it outputs a PDF. That PDF can then be consistently printed, regardless of the end user's browser, operating system, or printer.

Usually the PDF rendering would take place on a web server, with the web app responsible for generating the PDF behind the scenes, and sending the PDF through the browser for the user to print. But for the purpose of this tutorial, we'll be doing it all locally, taking advantage of the ability to override the CSS.

Downloading PrinceXML

If you want to follow along, download PrinceXML and follow their instructions to install it.

Establishing the baseline

Usually the first step is to see how a website looks when printed using the default CSS. In many cases, what looks good on a computer or mobile devices looks terrible when printed.

I'm a fan of *Wired*, so I'm going to use their website as an example throughout this article. I'm not doing this to pick on *Wired* in particular, as many websites have problems with their printed version.

In the web browser, a recent article on Europa looks great:

Wired.com article captured on 2015-02-04.

The main body of the article takes up about two screenfuls of content if I scroll down. So if I print the article, I should get about two pages of printed content, right? Unfortunately, no. I get *18* pages. Those 18 pages consist of:

- 7 pages of unusable references to other articles.
- 1 page of unusable advertisements.
- 1 page of the article, poorly formatted.
- 4 pages of comments on the article.
- 5 pages of unusable links to Wired writers.

Here's the result of the built-in browser print with default CSS: WiredEuropaBrowserPrint (PDF, 6.1MB). Here's one example page from the output:

Example of browser print with default CSS. Not only are these photos and links poorly formatted, but they are of no use on a printed page.

Generate a PDF from PrinceXML

Now that we know what the default behavior is, let's switch to generating PDFs from PrinceXML. This will allow us to create our own CSS and rapidly iterate to improve the output. Here's an example command line to generate a PDF:

```
prince --no-author-style \
           -s wired.css \
           http://www.wired.com/2015/02/white-house-wants-go-europa \
           -o europa.pdf
```

We haven't created wired.css yet, so you'll see a warning message, but the PDF will still be generated. The output is similar to the built-in browser print.

Stage one: Eliminate Junk

By using Chrome's Inspect Element feature, we can examine the page's html structure, and start to turn off elements that don't add value on printed output. A bit of quick perusing shows us that much of the noise comes from nav#global-nav and aside#sidebar. Neither of these navigational elements makes sense in the context of print, so let's turn them off.

Start with this in wired.css:

```
1
2 aside#sidebar, nav#global-nav {
3   display:none;
4 }
5
```

Then regenerate the PDF using Prince. It shrinks from the original 18 pages and 6MB to a far smaller 5 pages: prince-1-europa (PDF, 136KB).

It takes a little more work investigating what's generating the remainder of unclickable links and photos, but eventually we can get it down to a very clean, if ugly, 3 pages with this CSS:

```
1
2 /* top nav and side-nav*/
3 #global-nav, #leaderboard, #sidebar, #footer-ad, #post-nav, #disqus_thread, #social-bottor
4   display: none;
5 }
6
7 .advertisement, .OUTBRAIN, .entry-extra, .entry-tags {
8   display: none;
9 }
10
11 footer {
12   display: none;
13 }
14
15 noscript {
16   display: none;
17 }
18
```

This is the resulting, cleaned up 3 page PDF: prince-2-europa (PDF, 103KB)


The White House Wants to Go to Europa

Page 1 of the resulting file. It's cleaned up of extraneous details, but not yet styled.

Reminder #1: Everything we've done so far using Prince can also be done in the browser using media queries to detect the 'print' media type. We do this by creating a media query in the primary CSS file:

```
@media print {
    …
}
```

Ideally, site owners should be creating their own @media print CSS. With about an hour of work, *Wired* could dramatically improve how their articles print.

Reminder #2: For the purposes of this demo, we've been aggressive about turning off everything that isn't essential to the printed article. But in a real-world context, it may make sense to keep some of that content. What is kept will need to be specially formatted for print. For example, a small number of banner ads and navigation elements can be displayed, but they should be actionable from the printed page. That implies either printing a short, keyboard-friendly URL below it, or using image watermarking technology so that a smartphone can automatically follow it. (*HP plug*: we have a web service called Link Technology that implements both, and can make it easier for you to make great website prints.) In the *Wired* example, we can imagine that they can be strategic about choosing the one or two most valuable ads, and including them on the printed page.

Stage two: Make It Pretty

Once we've eliminated what we don't need, we can focus on making the rest of the content attractive while conforming to good visual design principles for print.

This next bit of CSS will make the headline and byline look better on the printed page. We're changing the font to match what Wired uses, and eliminating the bulleted list.

```
1
2  /* headers and meta-data */
3  .entry-header, .product-info {
4    color: #333;
5    font-family: brandon-grotesque, brandon-grotesque-1, brandon-grotesque-2, HelveticaNeue
6    font-size: 12px;
7    line-height: 15px;
8    margin: 10px 0;
9    text-transform: uppercase;
10 }
11
12 .entry-header li {
13   display: none;
14   white-space: nowrap;
15   list-style: none;
16 }
17
18 /* specific to wired articles - not really ideal for other sites */
19 .entry-header li.author, li.entryDate, li.entryTime {
20   display: inline-block;
21   white-space: nowrap;
22   list-style: none;
23 }
24 h1, h2, h3 {
25   font-family: brandon-grotesque, brandon-grotesque-1, brandon-grotesque-2, HelveticaNeue
26   font-size: 24px;
27   line-height: 32px;
28   margin-bottom: -12px;
29 }
30
31 h1#headline {
32   font-family: brandon-grotesque, brandon-grotesque-1, brandon-grotesque-2, HelveticaNeue
33   font-size: 28px;
34   line-height: 30px;
35   margin-bottom: 10px;
36 }
37
```

The result is that we change from unstyled to styled article title and byline:

Before styling

Here's the styled version:

After styling

Now that our headline is done, we have to address the body of the article. We still have a few problems

to deal with:

1. Print design principles tell us that shorter lines are easier for the eye to track than longer lines. That's why print magazines and newspapers use narrow columns. So we'll switch to two-column layout.
2. Short line-lengths only look good when hyphenation is turned on. Otherwise we get distracted by the pattern of the right side of the column.
3. Big photos look great online, but when printed they consume a lot of ink, which people don't like.
4. Printed links don't do us much good. We can turn off the link or display the destination url. We'll save the latter approach for another tutorial, because we can use HP Link Technology to display the destination url. For today, we'll just turn off the blue underlined links.

Fortunately PrinceXML supports both multiple columns and hyphenation.

This CSS will address all four items:

```
1
2  /*Two column layout*/
3  div.entry {
4    columns: 2;
5    column-gap: 1.5em;
6  }
7
8  /* Use hyphenation and an appropriate print font */
9  p {
10   font-family: 'Exchange SSm 4r', Georgia, serif;
11   font-size: 14px;
12   font-style: normal;
13   font-variant: normal;
14   font-weight: normal;
15   line-height: 18px;
16   text-align: justify;
17   hyphens: auto;
18   prince-hyphenate-lines: 2;
19   prince-hyphenate-before: 3;
20   prince-hyphenate-after: 2;
21   orphans: 2;
22   widows: 2;
23 }
24
25 /* adjust the image and caption to fit within a column */
26 p.wp-caption-text img {
27   height: 10px;
28   width: auto;
29 }
30
31 p.wp-caption-text {
32   font-family: proxima-nova, proxima-nova-1, proxima-nova-2, Helvetica, Arial, sans-serif
33   font-size: 11px;
34   text-align: center;
35   color: #444;
36   margin-top: 0px;
37 }
38
39 .size-660-single-full, .size-full {
```

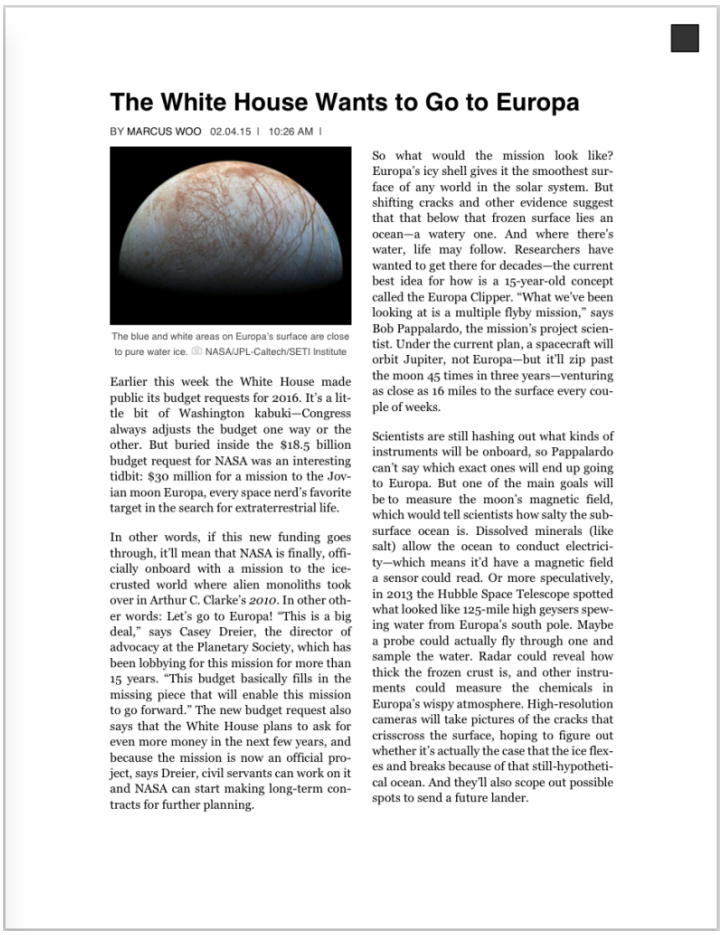
```
  width: 100%;
  height: auto;
}

/* hide those ugly links... */
a:link {
  color: #000000;
  text-decoration: none
}

```

Results

The final result of about an hour of CSS design work is a reasonably attractive two page article (prince-4-europa PDF, 100KB):



The White House Wants to Go to Europa

BY MARCUS WOO 02.04.15 | 10:26 AM |

The blue and white areas on Europa's surface are close to pure water ice. NASA/JPL-Caltech/SETI Institute

Earlier this week the White House made public its budget requests for 2016. It's a little bit of Washington kabuki—Congress always adjusts the budget one way or the other. But buried inside the $18.5 billion budget request for NASA was an interesting tidbit: $30 million for a mission to the Jovian moon Europa, every space nerd's favorite target in the search for extraterrestrial life.

In other words, if this new funding goes through, it'll mean that NASA is finally, officially onboard with a mission to the ice-crusted world where alien monoliths took over in Arthur C. Clarke's *2010*. In other other words: Let's go to Europa! "This is a big deal," says Casey Dreier, the director of advocacy at the Planetary Society, which has been lobbying for this mission for more than 15 years. "This budget basically fills in the missing piece that will enable this mission to go forward." The new budget request also says that the White House plans to ask for even more money in the next few years, and because the mission is now an official project, says Dreier, civil servants can work on it and NASA can start making long-term contracts for further planning.

So what would the mission look like? Europa's icy shell gives it the smoothest surface of any world in the solar system. But shifting cracks and other evidence suggest that that below that frozen surface lies an ocean—a watery one. And where there's water, life may follow. Researchers have wanted to get there for decades—the current best idea for how is a 15-year-old concept called the Europa Clipper. "What we've been looking at is a multiple flyby mission," says Bob Pappalardo, the mission's project scientist. Under the current plan, a spacecraft will orbit Jupiter, not Europa—but it'll zip past the moon 45 times in three years—venturing as close as 16 miles to the surface every couple of weeks.

Scientists are still hashing out what kinds of instruments will be onboard, so Pappalardo can't say which exact ones will end up going to Europa. But one of the main goals will be to measure the moon's magnetic field, which would tell scientists how salty the subsurface ocean is. Dissolved minerals (like salt) allow the ocean to conduct electricity—which means it'd have a magnetic field a sensor could read. Or more speculatively, in 2013 the Hubble Space Telescope spotted what looked like 125-mile high geysers spewing water from Europa's south pole. Maybe a probe could actually fly through one and sample the water. Radar could reveal how thick the frozen crust is, and other instruments could measure the chemicals in Europa's wispy atmosphere. High-resolution cameras will take pictures of the cracks that crisscross the surface, hoping to figure out whether it's actually the case that the ice flexes and breaks because of that still-hypothetical ocean. And they'll also scope out possible spots to send a future lander.

Final results of our CSS work. (The square in the upper-right corner is an artifact created by the trial version of PrinceXML.)

We've cut output from eighteen pages to two, eliminating useless and unneeded information. We've designed our page for print, choosing an appropriate font, line length, and layout that is optimized for the printed page. In short, we've made the output both more useful and more attractive, making it likely that this printed article still stick around longer and be read by more people.

Much of what I've shown today can also be achieved in the browser using print media queries. However, browser support of multiple column layout and hyphenation varies. To get the highest fidelity prints,

using an approach where PDFs are generated on the server will guarantee consistency across all browsers, even for mobile printing.

Resources

- PrinceXML has loads of great examples of what can be done with their PDF generator on their samples page.
- I also mentioned HP Link Technology, HP's server side tool for generating short, keyboard-friendly URLs, QR codes, and scannable images.
- Tips and Tricks for Print Style Sheets is another great tutorial that goes into print media queries.

If you have any questions, drop by the print developers forum.

Edit