How to embed and subset your own webfonts

Friday, February 6th, 2015

These days many of us use Typekit[1] for serving web fonts; and should you need to create and serve them yourself, most of the advice out there will point you to use FontSquirrel's well-known Webfont Generator[2]. However, I'm a big fan of not relying on third-party sites and services if you can avoid it. What happens in five years when those sites are no longer available? It's best to know how to do get the same result yourself using basic tools, and the result will often get you better performance.

Webfonts have gotten much easier to implement. Because nearly all browsers support WOFF[3], you no longer need to supply four different file formats for each font in order to be sure it will be supported on all browsers. You can simply convert any binary font file into Base64 encoding[4] and embed the resulting text/data right into your CSS file. However, there are additional steps you should take in order to optimize the size and downloading of your webfonts.

Base64 Encoding

This part is easy. The command `base64 filename` will take any file and encode it. This works on Linux and Mac OS X.

With this info, we can create a quick shell script that will take any font file and convert it into a webfont embedded right in a snippet of CSS:

```
#!/bin/bash

fontName=$1
fontFace=$2

WOFF=`base64 $fontName`

echo @font-face {
echo font-family: \'$fontFace\'\;
echo font-weight: normal\;
echo font-style: normal\;
echo font-stretch: normal\;
echo src: url\(\'data:application/font-woff\;charset=utf-8\;base64,$WOFF\'\) format\(\'woff\'\)\;
echo }
```

1. http://typekit.com
2. http://www.fontsquirrel.com/tools/webfont-generator
3. http://caniuse.com/#search=woff
4. http://en.wikipedia.org/wiki/Base64

Save this to a file called `webfont-encode` and make it executable:

```
chmod u+x ./webfont-encode
```

You can then use it to create the webfont from any font file and append it to your CSS file, like so:

```
./webfont-encode AlegreyaSans-Reg.otf alegreyasans-1 >> fonts.css
```

On OS X, you can also copy the result to the clipboard if you wish:

```
./webfont-encode AlegreyaSans-Reg.otf alegreyasans-1 | pbcopy
```

Most times you'll need to do this four times for each typeface: once each for the regular, italic, bold, and bold italic versions of the font. For each one, edit the `font-weight` and `font-style` CSS attributes to reflect the actual attributes of that font.

Once you've done all that, you can include the above stylesheet in your HTML (make sure it comes before any other stylesheets) and reference the font in your other CSS styles.

Note: The original font file needs to have its "embeddable" flag set to `0x0000` or the font loading will fail in Internet Explorer (at least in versions 9 through 11). Other browsers do not seem to check for this value. If your font license allows embedding, you can find some more Python code for modifying the flag here: http://www.typophile.com/node/102671[5].

Subsetting your fonts

If you create your own webfonts using only the above steps, your files will be huge–likely around 1 MB per typeface once you include bold and italic versions. This is because most typefaces (the good ones, at least) include characters for Russian, Greek, Hebrew, Arabic, and many other character sets. You can speed up your site a lot by whittling each font down to only the characters you're likely to need.

For this part I'm assuming you're on OS X. You're going to need Homebrew[6] and Python installed.

First, install the fontforge python extensions with `brew install fontforge`. Then:

```
export PYTHONPATH=/usr/local/lib/python2.7/site-packages:$PYTHONPATH
```

5. http://www.typophile.com/node/102671
6. http://brew.sh

(You'll need to do this every time you open a new terminal unless you permanently add Homebrew's package folder to your Python path.)

Next, get yourself a copy of glyphIgo[7]. This lovely Python script can convert between TTF and OTF font formats, and subset fonts based on any character set. Alberto Pettarin has a great blog post[8] explaining more about its use.

If you have `git` installed (`brew install git`), the following command will download a copy into a `glyphIgo` folder:

```
git clone https://github.com/pettarin/glyphIgo.git
```

Now we need to create the character set: a file containing each of the characters you want to include in your font. I wrote a quick Python script to assist with this, and have included codes several extra typographic symbols that I use often:

```
from __future__ import print_function

charsets = [
   [0x0020,0x007F], # BASIC LATIN
   [0x00A1,0x00FF], # PUNCTUATION, LATIN-1 SUPPLEMENT, COMMON SYMBOLS
   [0xFB00,0xFB04], # STANDARD ENGLISH LIGATURES fi, fl, ffi, ffl
   [0x0100,0x017F], # LATIN EXTENDED-A
   [0x0180,0x024F], # LATIN EXTENDE0D-B
   [0x2010,0x2015], # HYPHENS AND DASHES
   [0x2018,0x2019], # LEFT/RIGHT SINGLE QUOTATION MARKS
   [0x201C,0x201D], # LEFT/RIGHT DOUBLE QUOTATION MARKS
   [0x2026,0x2026], # ELLIPSES
   [0x221E,0x221E], # INFINITY SYMBOL
   [0x2190,0x2193], # ARROWS
   [0x21A9,0x21A9], # LEFTWARDS ARROW WITH HOOK
   [0x2761,0x2761], # CURVED STEM PARAGRAPH SIGN ORNAMENT
   [0x2766,0x2767] # FLORAL HEART, ROTATED FLORAL HEART
]

for charset in charsets:
   for x in range(charset[0],charset[1]+1):
      print(unichr(x).encode('utf-8'),end='')
```

Some notes on customizing this script: If you want to include any additional character blocks in your font, simply add the hex ranges to the `charsets` list (see the complete list of Unicode blocks[9]). Single characters can be added by making the first and second numbers of the "range" identical.

Including the ligatures (if the font supports them) makes for a better result on some platforms. On Mac, for example, Safari will make use of the ligatures but Chrome doesn't

7. https://github.com/pettarin/glyphIgo
8. http://www.albertopettarin.it/blog/2014/09/16/subsetting-fonts-with-glyphigo.html
9. http://www.fileformat.info/info/unicode/block/index.htm

use them at all.

Save the above script as `makeset.py` and create your character set file like so:

```
python makeset.py > latin.set
```

Armed with our character set, you can now subset your font like so:

```
python glyphIgo.py subset -f AlegreyaSans-Regular.otf -p latin.set -o AS-R.otf
```

Now encode the new "minimized" font using the script we created above:

```
./webfont-encode AS-R.otf alegreyasans-1 >> fonts.css
```

Results

To get an idea of the size reduction, I used these methods to produce a single CSS file containing regular, italic, bold, and bold italic versions of two typefaces (a total of eight `@font-face`s).

Without subsetting the fonts, the resulting CSS file was 2.1 MB in size. When I subset the fonts before encoding them, the result was 611 KB in size, a 70% reduction.

For comparison, Typekit reports a size of 339K for a kit containing the same two typefaces, using the "Default" character set and without including OpenType features. However, I strongly suspect that this the *compressed* size of their kit. You can achieve the same result by enabling gzip compression on your web server. On my own server, according to checkgzipcompression.com[10], the 611 KB CSS file gets packed down to a 345 KB download—almost identical to the Typekit version.

I could probably save even more space by omitting the Latin Extended-A and Extended-B blocks in my character set.

Speeding it up even more

Once you have your fonts set the way you like them, you should do yourself a further favour and check out Adam Beres-Deak's post Loading webfonts with high performance on responsive websites[11]. Using the simple Javascript in his post, you can make your fonts load and perform *much* faster than they would if you were using Typekit or Google Fonts.

10. http://checkgzipcompression.com
11. http://bdadam.com/blog/loading-webfonts-with-high-performance.html