Web Typography Possibilities

CSS Fonts

O'REILLY®

Eric A. Meyer

CSS Fonts

Eric A. Meyer

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Special Upgrade Offer

If you purchased this ebook directly from oreilly.com, you have the following benefits:

- DRM-free ebooks—use your ebooks across devices without restrictions or limitations
- Multiple formats—use on your laptop, tablet, or phone
- Lifetime access, with free updates
- Dropbox syncing—your files, anywhere

If you purchased this ebook from another retailer, you can upgrade your ebook to take advantage of all these benefits for just $4.99. Click here to access your ebook upgrade.

Please note that upgrade offers are not available from sample content.

Preface

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

> **TIP**
>
> This icon signifies a tip, suggestion, or general note.

> **CAUTION**
>
> This icon indicates a warning or caution.

Safari® Books Online

Safari Books Online (www.safaribooksonline.com) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and

creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://oreil.ly/css-fonts_1e.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

Chapter 1. Fonts

As the authors of CSS clearly recognized from the outset, font selection is a popular, indeed crucial, feature of web design. In fact, the beginning of the "Font Properties" section of the CSS1 specification begins with the sentence, "Setting font properties will be among the most common uses of style sheets." The intervening years have done nothing to disprove this assertion.

CSS2 added the ability to specify custom fonts for download with `@font-face`, but it wasn't until about 2009 that this capability really began to be widely and consistently supported. Now, websites can call on any font they have the right to use, aided by online services such as Fontdeck and Typekit. Generally speaking, if you can get access to a font, you can use it in your design.

It's important to remember, however, that this does not grant absolute control over fonts. If the font you're using fails to download or is in a file format the user's browser doesn't understand, then the text will be displayed with a fallback font. That's a good thing, since it means the user still gets your content, but it's worth bearing in mind that you cannot absolutely depend on the presence of a given font, and should never design as if you can.

Font Families

What we think of as a "font" is usually composed of many variations to describe bold text, italic text, and so on. For example, you're probably familiar with (or at least have heard of) the font Times. However, Times is actually a combination of many variants, including TimesRegular, TimesBold, TimesItalic, TimesBoldItalic, and so on. Each of these variants of Times is an actual *font face*, and Times, as we usually think of it, is a combination of all these variant faces. In other words, Times is actually a *font family*, not just a single font, even though most of us think about fonts as being single entities.

In order to cover all the bases, CSS defines five generic font families:

Serif fonts

> These fonts are proportional and have serifs. A font is proportional if all characters in the font have different widths due to their various sizes. For example, a lowercase *i* and a lowercase *m* are different widths. (This book's paragraph font is proportional, for example.) Serifs are the decorations on the ends of strokes within each character, such as little lines at the top and bottom of a lowercase *l*, or at the bottom of each leg of an uppercase *A*. Examples of serif fonts are Times, Georgia, and New Century Schoolbook.

Sans-serif fonts

These fonts are proportional and do not have serifs. Examples of sans-serif fonts are Helvetica, Geneva, Verdana, Arial, and Univers.

Monospace fonts

Monospace fonts are not proportional. These generally are used for displaying programmatic code or tabular data. In these fonts, each character uses up the same amount of horizontal space as all the others; thus, a lowercase *i* takes up the same horizontal space as a lowercase *m*, even though their actual letterforms may have different widths. These fonts may or may not have serifs. If a font has uniform character widths, it is classified as monospace, regardless of the presence of serifs. Examples of monospace fonts are Courier, Courier New, Consolas, and Andale Mono.

Cursive fonts

These fonts attempt to emulate human handwriting or lettering. Usually, they are composed largely of flowing curves and have stroke decorations that exceed those found in serif fonts. For example, an uppercase *A* might have a small curl at the bottom of its left leg or be composed entirely of swashes and curls. Examples of cursive fonts are Zapf Chancery, Author, and Comic Sans.

Fantasy fonts

Such fonts are not really defined by any single characteristic other than our inability to easily classify them in one of the other families (these are sometimes called “decorative” or “display” fonts). A few such fonts are Western, Woodblock, and Klingon.

In theory, every font family will fall into one of these generic families. In practice, this may not be the case, but the exceptions (if any) are likely to be few and far between, and browsers are likely to drop any fonts they cannot classify as serif, sans-serif, monospace, or cursive into the “fantasy” bucket.

Using Generic Font Families

You can employ any font family available by using the property `font-family`.

FONT-FAMILY

Values:

[*<family-name>* | *<generic-family>*]# | `inherit`

Initial value:

User agent-specific

Applies to:

All elements

Inherited:

Yes

Computed value:

As specified

If you want a document to use a sans-serif font, but you do not particularly care which one, then the appropriate declaration would be:

```
body {font-family: sans-serif;}
```

This will cause the user agent to pick a sans-serif font family (such as Helvetica) and apply it to the `body` element. Thanks to inheritance, the same font family choice will be applied to all the elements that descend from the `body`—unless a more specific selector overrides it, of course.

Using nothing more than these generic families, an author can create a fairly sophisticated style sheet. The following rule set is illustrated in Figure 1-1:

```
body {font-family: serif;}
h1, h2, h3, h4 {font-family: sans-serif;}
code, pre, tt, kbd {font-family: monospace;}
p.signature {font-family: cursive;}
```

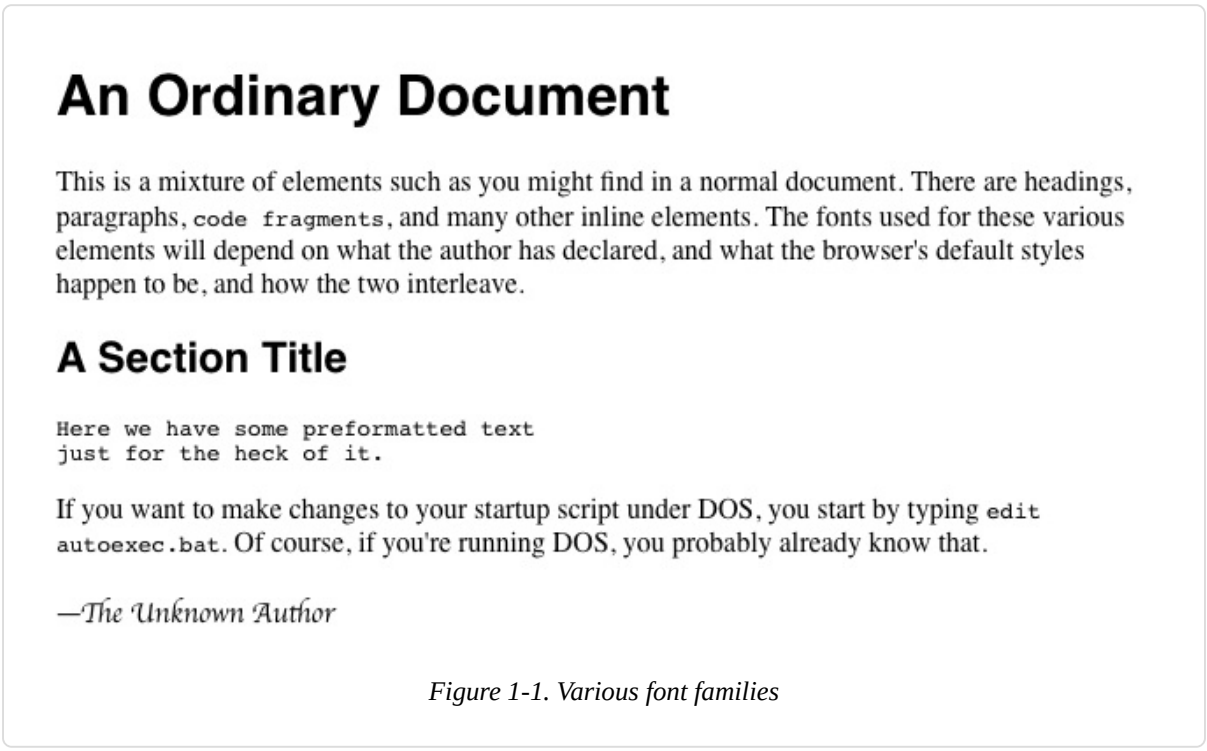

Figure 1-1. Various font families

Thus, most of the document will use a serif font such as Times, including all paragraphs except those that have a `class` of `signature`, which will instead be rendered in a cursive font such as Author. Heading levels 1 through 4 will use a sans-serif font like Helvetica, while the elements `code`, `pre`, `tt`, and `kbd` will use a monospace font like Courier.

Specifying a Font Family

An author may, on the other hand, have more specific preferences for which font to use in the display of a document or element. In a similar vein, a user may want to create a user style sheet that defines the exact fonts to be used in the display of all documents. In either case, `font-family` is still the property to use.

Assume for the moment that all `h1`s should use Georgia as their font. The simplest rule for this would be the following:

```
h1 {font-family: Georgia;}
```

This will cause the user agent displaying the document to use Georgia for all `h1`s, as shown in Figure 1-2.

Figure 1-2. An h1 element using Georgia

Of course, this rule assumes that the user agent has Georgia available for use. If it doesn't, the user agent will be unable to use the rule at all. It won't ignore the rule, but if it can't find a font called "Georgia," it can't do anything but display `h1` elements using the user agent's default font (whatever that is).

All is not lost, however. By combining specific font names with generic font families, you can create documents that come out, if not exact, at least close to your intentions. To continue the previous example, the following markup tells a user agent to use Georgia if it's available, and to use another serif font if it's not.

```
h1 {font-family: Georgia, serif;}
```

If a reader doesn't have Georgia installed but does have Times, the user agent might use Times for `h1` elements. Even though Times isn't an exact match to Georgia, it's probably close enough.

For this reason, I strongly encourage you to always provide a generic family as part of any `font-family` rule. By doing so, you provide a fallback mechanism that lets user agents pick an alternative when they can't provide an exact font match. Here are a few more examples:

```
h1 {font-family: Arial, sans-serif;}
h2 {font-family: Charcoal, sans-serif;}
p {font-family: 'Times New Roman', serif;}
address {font-family: Chicago, sans-serif;}
```

If you're familiar with fonts, you might have a number of similar fonts in mind for displaying a given element. Let's say that you want all paragraphs in a document to be displayed using Times, but you would also accept Times New Roman, Georgia, New Century Schoolbook, and New York (all of which are serif fonts) as alternate choices. First, decide the order of preference for these fonts, and then string them together with commas:

```
p {font-family: Times, 'Times New Roman', 'New Century Schoolbook', Georgia,
      'New York', serif;}
```

Based on this list, a user agent will look for the fonts in the order they're listed. If none

of the listed fonts are available, then it will simply pick an available serif font.

Using quotation marks

You may have noticed the presence of single quotes in the previous example, which we haven't seen before. Quotation marks are advisable in a `font-family` declaration only if a font name has one or more spaces in it, such as "New York," or if the font name includes symbols such as # or $. Thus, a font called Karrank% should probably be quoted:

```
h2 {font-family: Wedgie, 'Karrank%', Klingon, fantasy;}
```

If you leave off the quotation marks, there is a chance that user agents will ignore that particular font name altogether, although they'll still process the rest of the rule.

Note that the quoting of a font name containing a symbol is not actually required any more. Instead, it's recommended, which is as close to describing "best practices" as the CSS specification ever really gets. Similarly, it is recommended that you quote a font name containing spaces, though again, this is generally unnecessary in modern user agents. As it turns out, the only required quotation is for font names that match accepted `font-family` keywords. Thus, if you call for a font whose actual name is "cursive," you'll definitely need to quote it in order to distinguish it from the value keyword `cursive`:

```
h2 {font-family: Author, "cursive", cursive;}
```

Obviously, font names that use a single word (that doesn't conflict with any of the keywords for `font-family`) need not be quoted, and generic family names (`serif`, `monospace`, etc.) should never be quoted when they refer to the actual generic families. If you quote a generic name, then the user agent will assume that you are asking for a specific font with that name (for example, "serif"), not a generic family.

As for which quotation marks to use, both single and double quotes are acceptable. Remember that if you place a `font-family` rule in a `style` attribute, which you generally shouldn't, you'll need to use whichever quotes you didn't use for the attribute itself. Therefore, if you use double quotes to enclose the `font-family` rule, then you'll have to use single quotes within the rule, as in the following markup:

```
p {font-family: sans-serif;}  /* sets paragraphs to sans-serif by default */

<!-- the next example is correct (uses single-quotes) -->
<p style="font-family: 'New Century Schoolbook', Times, serif;">...</p>

<!-- the next example is NOT correct (uses double-quotes) -->
<p style="font-family: "New Century Schoolbook", Times, serif;">...</p>
```

If you use double quotes in such a circumstance, they interfere with the attribute syntax, as you can see in Figure 1-3.

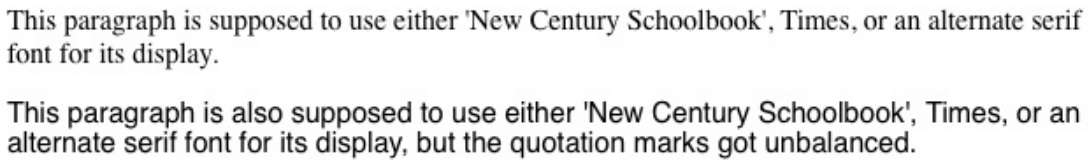

Figure 1-3. The perils of incorrect quotation marks

Using @font-face

A feature that originally debuted in CSS2 but wasn't implemented until late in the first decade of the 2000s, `@font-face` lets you use custom fonts in your designs. While there's no guarantee that every last user will see the font you want, this feature is very widely supported and (as of early 2013) gaining a lot of currency in web design.

Suppose you want to use a very specific font in your style sheets, one that is not widely installed. Through the magic of `@font-face`, you can define a specific family name to correspond to a font file on your server. The user agent will download that file and use it to render the text in your page, the same as if it were installed on the user's machine. For example:

```
@font-face {
   font-family: "SwitzeraADF";
   src: url("SwitzeraADF-Regular.otf");
}
```

This allows the author to have conforming user agents load the defined `.otf` file and use that font to render text when called upon via `font-family: SwitzeraADF`.

> **NOTE**
>
> The examples in this section refer to SwitzeraADF, a font face collection available from the Arkandis Digital Foundry.

The intent of `@font-face` is to allow "lazy loading" of font faces. This means that only those faces needed to render a document will actually be loaded, with the rest being left alone. In fact, a browser that downloads all declared font faces without considering whether they're actually needed is considered to be buggy.

Required Descriptors

All the parameters that define the font you're referencing are contained within the `@font-face { }` construct. These are called *descriptors*, and very much like properties, they take the format `descriptor: value;`. In fact, most of the descriptor names refer directly to property names, as will be explained in just a moment.

There are two required descriptors: `font-family` and `src`.

FONT-FAMILY

Value:

<family-name>

Initial value:

Not defined

SRC

Values:

[[*<uri>* [format(*<string>*#)]?] | *<font-face-name>*]#

Initial value:

Not defined

The point of `src` is pretty straightforward: it lets you define one or more sources for the font face you're defining, using a comma-separated list if there are in fact multiple sources. You can point to a font face at any URI, but there is a restriction: font faces can only be loaded from the same origin as the style sheet. Thus, you can't point your `src` at someone else's site and download their font; you'll need to host a local copy on your own server, or use a font-hosting service that provides both the style sheet(s) and the font file(s).

NOTE

There is an exception to the same-origin restriction, which is that servers can permit cross-site loading using the HTTP header `Access-Control-Allow-Origin`.

You may well be wondering how it is that we're defining `font-family` here when it

was already defined in a previous section. The difference is that this `font-family` is the font-family *descriptor*, and the previously-defined `font-family` was the font-family *property*. If that seems confusing, stick with me a moment and all should become clear.

In effect, `@font-face` lets you create low-level definitions that underpin the font-related properties like `font-family`. When you define a font family name via the descriptor `font-family: "SwitzeraADF";`, you're setting up an entry in the user agent's table of font families for "SwitzeraADF." It thus joins all the usual suspects like Helvetica, Georgia, Courier, and so forth, as a font you can just refer to in your `font-family` property values.

```
@font-face {
   font-family: "SwitzeraADF";   /* descriptor */
   src: url("SwitzeraADF-Regular.otf");
}
h1 {font-family: SwitzeraADF, Helvetica, sans-serif;}  /* property */
```

Note how the `font-family` descriptor value and the entry in the `font-family` property match. If they didn't match, then the `h1` rule would ignore the first font family name listed in the `font-family` value and move on to the next. As long as the font has cleanly downloaded and is in a format the user agent can handle, then it will be used in the manner you direct, as illustrated in Figure 1-4.

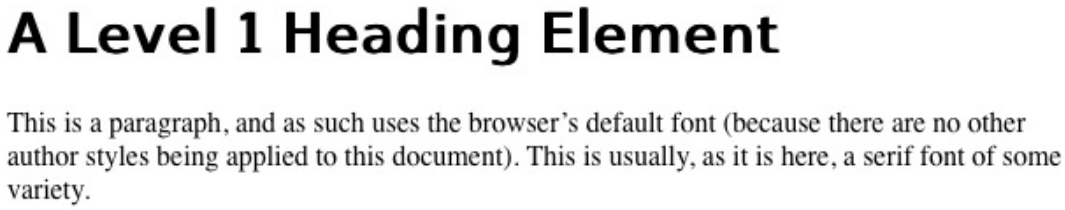

Figure 1-4. Using a downloaded font

In a similar manner, the comma-separated `src` descriptor value provides fallbacks. That way, if (for whatever reason) the user agent is unable to download the first source, it can fall back to the second source and try to load the file there.

```
@font-face {
   font-family: "SwitzeraADF";
   src: url("SwitzeraADF-Regular.otf"),
        url("/fonts/SwitzeraADF-Regular.otf");
}
```

Remember that the same-origin policy generally applies in this case, so pointing to a copy of the font some other server will usually fail, unless of course said server is set up to permit cross-origin access.

If you want to be sure the user agent understands what kind of font you're telling it to use, that can be done with the optional `format()`.

```
@font-face {
   font-family: "SwitzeraADF";
   src: url("SwitzeraADF-Regular.otf") format("opentype");
}
```

The advantage of supplying a `format()` description is that user agents can skip downloading files in formats that they don't support, thus reducing bandwidth use and loading speed. It also lets you explicitly declare a format for a file that might not have a common filename extension and thus be unfamiliar to the user agent.

```
@font-face {
   font-family: "SwitzeraADF";
   src: url("SwitzeraADF-Regular.otf") format("opentype"),
        url("SwitzeraADF-Regular.true") format("truetype");
}
```

THE FLASH

If you're a designer or developer of a certain vintage, you may remember the days of FOUC: the Flash of Unstyled Content. This happened in earlier browsers that would load the HTML and display it to the screen before the CSS was finished loading, or at least before the layout of the page via CSS was finished. Thus, what would appear was a split-second of "plain ol' text" (using the browser's default styles) before it was replaced with the CSS-decorated layout.

As of early 2013, there is a cousin to this problem, which is the Flash of Un-Fonted Text, or FOUFT. This happens when a browser has loaded the page and the CSS and displays the laid-out page before it's done loading custom fonts. This causes text to appear in the default font, or a fallback font, before being replaced by text using the custom-loaded font.

Since the replacement of text with the custom-loaded font face can change its layout size, authors should take care in selecting fallback fonts. If there is a significant height difference between the font used to initially display the text and the custom font eventually loaded and used, significant page reflows are likely to occur. There's no automated way to enforce this, though `font-size-adjust` (covered later) can help. You simply have to look at your intended font and find other faces that have a similar height.

The core reason for the "flash" behavior is pretty much the same now as it was then: the browser is ready to show something before it has all the resources on hand, so it goes ahead and does so, replacing it with the prettier version once it can. The FOUC was eventually solved, and it's likely that some day we'll look back at the FOUFT the same way we do at the FOUC now. Until then, we'll have to take comfort in the fact that the FOUFT isn't usually as jarring as was the FOUC.

Table 1-1 lists all of the allowed format values (as of early 2013).

Table 1-1. Recognized font format values

Value	Format
`embedded-opentype`	EOT (Embedded OpenType)
`opentype`	OTF (OpenType)
`svg`	SVG (Scalable Vector Graphics)
`truetype`	TTF (TrueType)
`woff`	WOFF (Web Open Font Format)

In addition to the combination of `url()` and `format()`, you can also supply a font family name (or several names) in case the font is already locally available on the user's machine, using the aptly-named `local()`.

```
@font-face {
   font-family: "SwitzeraADF";
   src: local("Switzera-Regular"),
        local("SwitzeraADF-Regular "),
        url("SwitzeraADF-Regular.otf") format("opentype"),
        url("SwitzeraADF-Regular.true") format("truetype");
}
```

In this example, the user agent looks to see if it already has a font family named "Switzera-Regular" or "SwitzeraADF-Regular" available. If so, it will use the name `SwitzeraADF` to refer to that locally installed font. If not, it will use the `url()` value to try downloading the remote font.

Note that this capability allows an author to create custom names for locally installed fonts. For example, you could set up a shorter name for Helvetica (or, failing that, Helvetica Neue) like so:

```
@font-face {
   font-family: "H";
   src: local("Helvetica"),
        local("Helvetica Neue");
}

h1, h2, h3 {font-family: H, sans-serif;}
```

As long as the user has Helvetica installed on their machine, then those rules will cause the first three heading levels to be rendered using Helvetica. It seems a little gimmicky, but it could have a real impact on reducing style sheet file size in certain situations.

On Being Bulletproof

The tricky part with `@font-face` is that different browsers of different eras supported different font formats. (To the insider, Table 1-1 reads as a capsule history of downloadable font support.) In order to cover the widest possible landscape, you should turn to what is known as the "Bulletproof `@font-face` Syntax." Initially developed by Paul Irish and refined by the chaps at FontSpring, it looks like this:

```
@font-face {
    font-family: "SwitzeraADF";
    src: url("SwitzeraADF-Regular.eot");
    src: url("SwitzeraADF-Regular.eot?#iefix") format("embedded-opentype"),
         url("SwitzeraADF-Regular.woff") format("woff"),
         url("SwitzeraADF-Regular.ttf") format("truetype"),
         url("SwitzeraADF-Regular.svg#switzera_adf_regular") format("svg");
}
```

Let's break it down piece by piece. The first bit, assigning the `font-family` name, is straightforward enough. After that, we see:

```
src: url("SwitzeraADF-Regular.eot");
    src: url("SwitzeraADF-Regular.eot?#iefix") format("embedded-opentype"),
```

This supplies an EOT (Embedded OpenType) to browsers that understand only EOTs —IE6 through IE9. The first line is for IE9 when it's in "Compatibility Mode," and the second line hands the same file to IE6-IE8. The `?#iefix` bit in that line exploits a parsing bug in those browsers to step around another parsing bug that causes them to 404 any `@font-face` with multiple formats listed. IE9 fixed its bugs without expanding its font formats, so the first line is what lets it join the party.

```
url("SwitzeraADF-Regular.woff") format("woff"),
```

This line supplies a Web Open Font Format file to browsers that understand it, which includes most modern browsers. At this point, in fact, you'll have covered the vast majority of your desktop users.

```
url("SwitzeraADF-Regular.ttf") format("truetype"),
```

This line hands over the file format understood by most iOS and Android devices, thus covering most of your handheld users.

```
url("SwitzeraADF-Regular.svg#switzera_adf_regular") format("svg");
```

Here, at the end, we supply the only font format understood by old iOS devices. This covers almost all of your remaining handheld users.

Obviously, this gets a bit unwieldy if you're specifying more than a couple of faces,

and typing it in even once is kind of a pain in the wrists. Fortunately, there are services available that will accept your font faces and generate all the `@font-face` rules you need, convert those faces to all the formats required, and hand it all back to you as a single package. One of the best is Font Squirrel's "@Font-Face Kit Generator". Just make sure you're legally able to convert and use the font faces you're running through the generator (see the next sidebar, , for more information).

Other Font Descriptors

In addition to the required `font-family` and `src` descriptors, there are a number of optional descriptors that can be used to associate font faces with specific font property values. Just as with `font-family`, these descriptors (summarized in Table 1-2) correspond directly to CSS properties (explained in detail later in this chapter) and affect how user agents respond to the values supplied for those properties.

CUSTOM FONT CONSIDERATIONS

There are two things you need to keep in mind when using customized fonts. The first is that you have the rights to use the font in a web page, and the second is whether it's a good idea to do so.

Much like stock photography, font families come with licenses that govern their use, and not every font license permits its use on the web. You can completely avoid this question by only using FOSS (Free and Open-Source Software) fonts only, or by using a commercial service like Fontdeck or Typekit that will deal with the licensing and format conversion issues so you don't have to. Otherwise, you need to make sure that you have the right to use a font face in the way you want to use it, just the same as you would make sure you had the proper license for any images you bought.

In addition, the more font faces you call upon, the more resources the web server has to hand over and the higher the overall page weight will become. Most faces are not overly large—usually 50K to 100K—but they add up quickly if you decide to get fancy with your type, and truly complicated faces can be larger. Of course, the same problems exist for images. As always, you will have to balance appearance against performance, leaning one way or the other depending on the circumstances. Furthermore, just as there are image optimization tools available, there are also font optimization tools. Typically these are "subsetting" tools, which construct fonts using only the symbols actually needed for display. If you're using a service like Typekit or Fonts.com, they probably have subsetting tools available, or do it dynamically when the font is requested.

Table 1-2. Font descriptors

Descriptor	Default value	Description
`font-style`	`normal`	Distinguishes between normal, italic, and oblique faces
`font-weight`	`normal`	Distinguishes between various weights (e.g., bold)
`font-stretch`	`normal`	Distinguishes between varying degrees of character widths (e.g., condensed and expanded)
`font-`	`normal`	Distinguishes between a staggeringly wide range of potential variant

`variant`		faces (e.g., small-caps); in most ways, a more “CSS-like” version of `font-feature-settings`
`font-feature-settings`	`normal`	Permits direct access to low-level OpenType features (e.g., enabling ligatures)
`unicode-range`	`U+0-10FFFF`	Defines the range of characters for which a given face may be used

Because these font descriptors are optional, they may not be listed in a `@font-face` rule, but CSS does not allow descriptors to go without default values any more than it does for properties. If an optional descriptor is omitted, then it is set to the default value. Thus, if `font-weight` is not listed, the default value of `normal` is assumed.

Restricting Character Range

There is one font descriptor, `unicode-range`, which (unlike the others in Table 1-2) has no corresponding CSS property. This descriptor allows authors to define the range of characters to which a custom font can be applied. This can be useful when using a symbol font, or to ensure that a font face is only applied to characters that are in a specific language.

UNICODE-RANGE

Values:

<urange>#

Initial value:

U+0-10FFFF

By default, the value of this property covers the entirety of Unicode, meaning that if a font can supply the glyph for a character, it will. Most of the time, this is exactly what you want. For all the other times, you’ll want to use a specific font face for a specific kind of content. To pick two examples from the CSS Fonts Module Level 3:

```
unicode-range: U+590-5FF;  /* Hebrew characters */
unicode-range: U+4E00-9FFF, U+FF00-FF9F, U+30??;  /* Japanese kanji, hiragana,
katakana */
```

In the first case, a single range is specified, spanning Unicode character code point 590 through code point 5FF. This covers the characters used in Hebrew. Thus, an author might specify a Hebrew font and restrict it to only be used for Hebrew characters, even if the face contains glyphs for other code points:

```
@font-face {
   font-family: "CMM-Ahuvah";
   src: url("cmm-ahuvah.otf" format("opentype");
   unicode-range: U+590-5FF;
}
```

In the second case, a series of ranges are specified in a comma-separated list to cover all the Japanese characters. The interesting feature there is the `U+30??` value, which is a special format permitted in `unicode-range` values. The question marks are wildcards meaning "any possible digit," making `U+30??` equivalent to `U+3000-30FF`. The question mark is the only "special" character pattern permitted in the value.

Ranges must always ascend. Any descending range (e.g., `U+400-300`) is treated as a parsing error and ignored. Besides ranges, you can also declare a single code point, which looks like `U+221E`. This is most often useful in conjunction with other code points and ranges, like so:

```
unicode-range: U+4E00-9FFF, U+FF00-FF9F, U+30??, U+A5;
  /* Japanese kanji, hiragana, and katakana, plus yen/yuan currency symbol*/
```

Of course, you could use a single code point to declare that a specific face only be used to render one, and only one, character. Whether or not that's a good idea is left to you, your design, the size of the font file, and your users' connection speeds.

Because `@font-face` is designed for "lazy loading" optimization, it's possible to use `unicode-range` to download only the font faces a page actually needs. Suppose that you have a website that uses a mixture of English, Russian, and basic mathematical operators, but you don't know which will appear on any given page. There could be all English, a mixture of Russian and math, and so on. Furthermore, suppose you have special font faces for all three types of content. You can make sure a user agent only downloads the faces it actually needs with a properly-constructed series of `@font-face` rules.

```
@font-face {
   font-family: "MyFont";
   src: url("myfont-general.otf" format("opentype");
}
@font-face {
   font-family: "MyFont";
   src: url("myfont-cyrillic.otf" format("opentype");
   unicode-range: U+04??, U+0500-052F, U+2DE0-2DFF, U+A640-A69F, U+1D2B-1D78;
}
@font-face {
   font-family: "MyFont";
   src: url("myfont-math.otf" format("opentype");
   unicode-range: U+22??;       /* equivalent to U+2200-22FF */
}
```