Building long-form content: variable fonts

Variable fonts are an evolution of the OpenType font specification that enables many different variations of a typeface to be loaded from a single file and a single `@font-face` declaration.

The single file would be larger than a single font, but in most cases smaller or about the same size as the 4 you might load for body copy. The advantage in choosing the variable font is that you have access to the entire range of weights, widths, and styles available, rather than being constrained to only the few static font files that you would have loaded separately.

You may find that some variable fonts come split into two files: one for uprights and all their variations, and one containing the italic variations. This is sometimes done to reduce the overall file size in cases where the italics aren't needed or used. It is possible to link them with a common `font-family` name so you can call them using the same font-family and appropriate font-style.

The `@font-face` declaration for variable fonts is similar to the previous example with some differences.

The values for `font-weight` and `font-stretch` take a range of two values for the respective property.

Instead of multiples of 100, you can use any value in the range. In a variable font, `font-weight: 451` and `font-stretch: 72%` are valid values in a selector.

Using Roboto Flex as an example, loading the variable font would look like this:

```
@font-face {
    font-family: "Roboto Fl"Roboto Flex Regular" url("./fonts/RobotoFlex-V
    font-weight: 100 1000;
    font-stretch: 25% 151%;
    font-style: normal;
}
```

The value for `font-style` will depend on whether the font provides italics, slant or both of them.

Creating a `@font-face` declaration for an italics-only font would look like this:

```
@font-face {
 font-family: 'MyVariabl'MyVariableFont'url('path/to/font/file/myvariable
 font-weight: 125 950;
 font-stretch: 75% 125%;
 font-style: italic;
}
```

If the font provides an oblique form for italics, loading the font looks like this:

```
@font-face {
 font-family: 'MyVariabl'MyVariableFont'url('path/to/font/file/myvariable
 font-weight: 125 950;
 font-stretch: 75% 125%;
 font-style: oblique 0deg 12deg;
}
```

Regardless of you write the `@font-face` declaration you would use it the same way, as a value in the `font-family` attribute, like this:

```
body {
  font-family: "MyVariableFont",
    arial,
    sans-serif;
}
```

Defining variation axes

The beauty of working with variable fonts is that, in addition to open type features, they provide additional customization through a set of registerd or standard axes and custom axes.

We will use Roboto Flex (again) as our example. We load the font using this `@font-face` declaration:

```
@font-face {
    font-family: "Roboto-Fl"Roboto-Flex" url("./fonts/RobotoFlex-VariableF
    font-weight: 100 1000;
    font-stretch: 25% 151%;
    font-style: normal;
}
```

Google has made the supported axes obvious by including them in the file name, but this is not always the case and even if they do provide you with the names of the axes it is seldom clear what they do just by looking at the name of the axis in question.

The registed axes (with their equivalent CSS properties) are:

Axis Tag	Equivalent CSS Property
“wght”	`font-weight`
“wdth”	`font-stretch`
“slnt” (slant)	`font-style`: `oblique + angle`
“ital”	`font-style`: `italic`
“opsz”	`font-optical-sizing`

Notes:

Capitalization matters, all registered axes are lowercase and all custom axes are uppercase.

Just because an axis is a registered axis, it doesn’t mean all fonts make it available.

Custom axes font dependent and allow font designers to to customize anything in their font, for example ascender or descender heights, the size of serifs, or anything else they can imagine.

Any axis can be used as long as the designer gives it a unique 4-character axis. Some will end up becoming more common, and may even become registered over time.

Roboto Flex defines the following custom axes (information taken from Roboto ... But Make It Flex):

Parametric Counter Width (XTRA in CSS) : The Parametric Counter Width axis alters the transparent forms inside and around glyphs in the X dimension. One application is in fine-tuning justification, as it changes the number of characters per line.

Parametric Thin Stroke (YOPQ in CSS) : The Parametric Thin Strokes axis alters stroke widths at their thinnest parts, typically in the Y dimension for the Latin script. One use-case is to improve legibility in very small sized text, because it can lower stroke contrast.

Parametric Lowercase Height (YTLC in CSS)

: The Parametric Lowercase Height axis alters the volume of all space within the lowercase's vertical alignment zone: the vertical space inside counters of glyphs and the space in their sidebearings. (This adds more space vertically instead of changing the side bearings like tracking does). : This is not truly an "x-height axis," since it only adjusts space vertically, and x-height proportions are formed by the relationships of uppercase, lowercase, ascenders and descenders, as well as horizontal forms. There are parametric axes for all of these.

Parametric Uppercase Height (YTUC in CSS) : Similarly, the Parametric Uppercase Height axis alters the volume of all space within the uppercase's vertical alignment.

Parametric Ascender Height (YTAS in CSS) : The Parametric Ascender Height axis alters the volume of all space within the ascenders' vertical alignment zone.

Parametric Descender Depth (YTDE in CSS) : The Parametric Descender Depth axis alters the depth of the space within the descenders' vertical alignment zone, which has a negative value being below the baseline.

Parametric Figure Height (YTFI in CSS) : The Parametric Figure Height axis alters the vertical space of figures. It can be used for aligning figures to other vertical zones.

Using variation axes in CSS

Using the registered axes is fairly straightfoward. You can use the associated CSS properties.

These registered axes are also defined with four-letter values as follows:

Axis	four-letter value
Weight	“wght”
Stretch / Width	“wdth”
Slant	“slnt”
Italic	“ital”
Optical Size	“opsz”

This will become important when we talk about updating values for `font-variation-settings`, later in the post.

Using custom axes are more difficult since we don’t have a common list of axes available for the font. We can use tooks like Wakamaifondue will give you a list of the available axes, both registered and custom, available for the font you’re working with.

The biggest issue with variable fonts, as described in Boiling eggs and fixing the variable font inheritance problem, is that whenever you update one axis of a variable font the ones that are not updated and the default values are used.

As the article indicates, the best solution is to use CSS Custom Properties. One possible way to do it is like this:

The following block sets custom properties for the :root pseudo-element, essentially turning them into global variables.

```
:root {
--roboto-wght: 400;
--roboto-wdth: 100;
--roboto-opsz: 14;
```

```
--roboto-grad: 0;
--roboto-slnt: 0;
--roboto-xtra: 468;
--roboto-xopq: 96;
--roboto-yopq: 79;;
--roboto-ytlc: 514;
--roboto-ytuc: 712;
--roboto-ytas: 750;
--roboto-ytde: -203;
--roboto-ytfi: 738;
}
```

We first use them in the universal selector to create our document default font styles

```
* {
  font-variation-settings:
    "wght" var(--roboto-wght),
    "wdth" var(--roboto-wdth),
    "opsz" var(--roboto-opsz),
    "GRAD" var(--roboto-grad),
    "slnt" var(--roboto-slnt),
    "XTRA" var(--roboto-xtra),
    "XOPQ" var(--roboto-xopq),
    "YOPQ" var(--roboto-yopq),
    "YTLC" var(--roboto-ytlc),
    "YTUC" var(--roboto-ytuc),
    "YTAS" var(--roboto-ytas),
    "YTDE" var(--roboto-ytde),
    "YTFI" var(--roboto-ytfi);
}
```

With this in place we can override it everywhere that we need to make a change.

We first set Roboto Flex and sans-serif as the `font-family` for the body and then we can tweak the defaults for whatever element needs it.

```
body {
  font-family: "Rob"Roboto-Flex"ns-serif;
}

strong, b {
  --roboto-wght: 700;
}

em, i {
  --roboto-slnt: -10;
}
```

Although, semantically, `strong` and `b` and `em` and `i` are intended for different purposes, browsers style them the same way so I did the same.

This is just a basic exercise. You can create additional classes and experiment with the custom axes available to the font.

A working example is in this pen: