Using Variable Fonts in a WordPress theme

Variable Fonts give you a lot of flexibility in exchange of older browsers and operating systems. They reduce the number of font files required to render content and they give you options that are difficult or not possible with traditional web fonts.

Quoting Jason Pamental's [The evolution of typography with variable fonts: an introduction]:

> As described by John Hudson, a variable font is a single font that acts as many: all the variations of width and weight, slant, and even italics can be contained in a single, highly efficient and compressible font file. What's more: the format (which is technically part of the OpenType 1.8 specification) is completely extensible. The type designer has complete control over what axes are used, their ranges, and even the definition of new axes. There are currently 5 'registered' axes (width, weight, slant, italics, and optical sizing), but the designer can vary any axis they choose. Some examples include the height of ascenders and descenders, text grade, even serif shape. The possibilities are nearly limitless.

Why use them

Variable fonts improve performance in a number of ways. They reduce the number of HTTP connections we have to make for Font assets, it make the fonts smaller overall (the one file you download may be larger but it's one as opposed to 4 for each traditional font you work with).

They also allow for things that were very difficult or impossible to do before. We can animate the axes if we set them up properly giving us additional flexibility.

We'll explore Variable fonts using [Recursive] as the single font for a WordPress-based site. Along the way we'll talk about responsive typography, based on the Work of Jason Pamental, and how to work with older browsers.

Loading variable fonts

WordPress strognly suggests that you enqueue third-party scripts and stylesheets for use with a WordPress theme. However when creating a theme from scratch we don't need to enqueue the main stylesheet and that's where we'll make all our variable fonts additions.

We'll cover both methods below.

Modifying an existing theme

Assuming that we've created a stylesheet to load the font using `@font-face` syntax and all the style that override the default font size then all it takes is to enqueue the stylesheet.

We've discussed how to enqueue local stylesheets so I won't go into details aboout how the code below works, I'll just show the end product.

```
function rivendellweb_enqueue_local_fonts() {
    wp_enqueue_style( 'local_styles',
            get_stylesheet_directory_uri() . '/css/recursive-styles.css'
}
add_action( 'wp_enqueue_scripts', 'rivendellweb_enqueue_local_fonts' );
```

From Scratch

When building a theme from scratch the rules change slightly. We're not adding new resources to the theme but we're changing the existing CSS to match our design.

There is no enqueueing necessary as we're working with the default styles for the theme. We'll look at how to do it in the next section.

Example: Recursive Font from scratch

The following code will build a responsive-typography stylesheet using Recursive.

We first laod the font using `@font-face` rule with some changes to accommodate the variable fonts.

We use two different formats to support different syntaxes for the format for the attribute.

Aattributes like `font-weight`, `font-style` and `font-stretch` take two values indicating the lower and upper boundaries for the particular axis.

Finally, we use `font-display: swap` to tell the browser to swap the font once it's loaded.

```css
@font-face {
  font-family: "Recursive"Recursive VF"   url('./fonts/recursive.woff2') f
    'woff2 supports variations'url('./fonts/recursive.woff2') format('wof
    font-weight: 300 1000;
    font-display: swap;
}
```

The next block defines variables with the default values for each of the axes that the font makes available. We'll make extensive use of these variables elsewhere in the document.

```css
:root {
  --recursive-mono: 0;
  --recursive-casual: 0;
  --recursive-weight: 400;
  --recursive-slant: 0;
  --recursive-italic: 0.5;
}
```

This default selector adds the default font family and default values using the variables defined in the previous block.

font-variation-settings allows you to add the custom axes with variables.

The uppercase axes, **MONO** and **CASL**, are custom axes that will only work with Recursive.

The lowercase axes, **slnt** and **ital** are predefined axes. The reason why we don't use the equivalent CSS property is that they both match the same property so we'd have to use either one but we can't use them together.

```css
* {
  font-family:  "Recursive VF",
                Verdana,
                sans-serif;
  font-weight: var(--recursive-weight);
  font-variation-settings:
    "MONO" var(--recursive-mono),
    "CASL" var(--recursive-casual),
    "slnt" var(--recursive-slant),
    "ital" var(--recursive-italic);
}
```

The rest of the code in this post is taken and adapted from FF Meta Variable Font Demo, a pen from Jason Pamental.

We first add another `:root` block with CSS Custom Properties / Variables too define the values that we want to work with.

This is a simplified version that considers only p and h1 elements. The full version has additional entries for h2 through h4.

This code only deals with font size, line height and their relationship when screen size changes using media queries. It also takes advantage of Fontface Observer to add styles for when the font fails to load.

```css
:root {
  /* Breakpoint variables */
  --bp-small: 24.15;
  --bp-medium: 43.75;
  --bp-large: 60.25;
  --bp-xlarge: 75;
  /* Paragraph variables */
  --p-line-height-min: 1.25;
  --p-line-height-max: 1.4;
  --p-font-size-min: 1.0;
  --p-font-size-max: 1.25;
  /* H1 variables */
  --h1-line-height-min: 1.1;
  --h1-line-height-max: 1.1;
  --h1-font-size-min: 2.5;
  --h1-font-size-max: 4;
  --h1-vf-wght-multiplier-s: 0.75;
  --h1-vf-wght-multiplier-m: 0.75;
  --h1-vf-wght-multiplier-l: 0.75;
}
```

The default rule for paragraphs sets the size to 16px, the font size to 400 and the line height to 1.

All the media queries play with what values to use and how to combine them together.

```
p, li {
  font-size: calc( var(--p-font-size-min) * 1rem );
  font-weight: var(--recursive-weight);
  line-height: var(--p-line-height-min);
}
@media screen and (min-width: 24.15em) {
  p, li {
    line-height: calc(( var(--p-line-height-min) * 1em ) + ( var(--p-line-
  }
}
@media (min-width: 60.25em) {
  p, li {
    font-size: calc(( var(--p-font-size-min) * 1em ) + ( var(--p-font-size
    line-height: var(--p-line-height-max);
  }
}
@media (min-width: 75em) {
  p, li {
    font-size: calc( var(--p-font-size-max) * 1em );
  }
}
```

We do something similar with h1 with the corresponding h1 variables and one additional change.

We leverage the `.fonts-failed` class generated by FontFace Observer and style elements when our variable font is not available.

```
h1 {
  font-weight: calc( var(--recursive-weight) * var(--h1-vf-wght-multiplier
  font-size: calc( var(--h1-font-size-min) * 1em );
  font-style: normal;
  line-height: var(--h1-line-height-min);
}
.fonts-failed h1 {
  font-family:  Georgia,
                "New Times Roman",
                serif;
  margin: 2em 0;
  letter-spacing: -.5px;
}
"New Times Roman"@media screen and (min-width: 24.15em) {
  h1 {
    line-height: calc(( var(--h1-line-height-min) * 1em ) +
      ( var(--h1-line-height-max) - var(--h1-line-height-min) ) * ((100vw
    font-size: calc(( var(--h1-font-size-min) * 1em ) + ( var(--h1-font-s
  }
}
@media screen and (min-width: 43.75em) {
  h1 {
    font-weight: calc( var(--recursive-weight) * var(--h1-vf-wght-multipl
  }
  .fonts-failed h1 {
    letter-spacing: normal;
  }
}
```

```
@media (min-width: 75em) {
  h1 {
    font-size: calc( var(--h1-font-size-max) * 1em );
    font-weight: calc( var(--recursive-weight) * var(--h1-vf-wght-multipl
    line-height: var(--h1-line-height-max);
  }
  .fonts-failed h1 {
    letter-spacing: -1px;
  }
}
```

Yes, this is a lot of code but it will keep text readable and easy to change. Whenever we need to change something, we change the corresponding variables at the top.

One of the Recursive font's custom axes is Casual. I use it to create distinctive headers in combination with both Slant and Italic axes.

The code looks something like this:

```
h1.casual {
  --recursive-casual: 1;
  --recursive-slant: -15;
  --recursive-italic: 1;
}
```

We've done the same thing with styles. This is the modified styles for Prism.js used on my project.

We change the font to monospaced and add slashed 0 to fully distinguish them from lowercase and uppercase o.

```
code[class*="language-"],
pre[class*="language-"] {
    --recursive-mono: 1;
    --recursive-zero: "zero" on;
    color: #657b83;
```

```css
    font-family:  "Recursive VF",
                  Consolas, Monaco,
                  'Andale Mono',
                  'Ubuntu Mono',
                  monospace;
    font-size: 1.1em;
    text-align: left;
    white-space: pre;
    word-spacing: normal;
    word-break: normal;
    word-wrap: normal;
    line-height: 1.5;
    tab-size: 4;
    hyphens: none;
}
```