Leveraging local fonts

Note

This feature needs an Origin Trial Key for your origin.

If you choose not to go for an Origin Trial then you and your users will need to enable the Font Access APIs flag (`#font-access`) in `chrome://flags` and restart the browser.

The web has had custom fonts capabilities from very early on the history of CSS. CSS introduced `@font-face` in 1998 as part of CSS 2. It allowed developers to link to font files and use them to display content on the web.

As with many things in the early days of the web, different browser vendors decided that their format was the best one and started pushing their own formats for web fonts. Doing a little bit of historical digging I came up with all the possible combinations shown below.

```
@font-face {
  font-family: 'myAwesome'myAwesomeFont'/* IE9 Compat Modes */: url('webfo
  /* IE6'webfont.eot'/* IE6-IE8 */
  src:  url('webfont.eot?#iefix') format('embedded-opentype'),
  /* Super Modern Browsers */nt.woff2') format('woff2'),
        /* Pretty Modern Br') format('/* Pretty Modern Browsers */ormat('w
        /* Safari, Android, iOS */
   'woff'/* Safari, Android, iOS */'truetype'),
        /* Legacy iOS */
        url('webfon'truetype'/* Legacy iOS */('svg');
'svg'url('webfont.svg#svgFontName') format('svg');
}
```

Things have gotten better. Unless you’re required to provide support for ancient browsers, you’re fine keeping only WOFF and WOFF2 as the formats for your font. This will work in most modern browsers

```
@font-face {
  src:  /* Super Modern Browsers */
        url('webfont.woff2') format('woff2'),
        'woff2'/* Pretty Modern Browsers */  url('webfont.woff') format('w
}
'webfont.woff'
```

While @font-face introduced the ability to use fonts on the web either hosted locally on the same server or in an external server accessed via a URL, it was not without its problems.

@font-face presented an interesting challenge. On one side web developers who wanted to use fonts on the web and on the other side were type foundries that were afraid of piracy and IP theft since there was no way to restrict what fonts people used on their sites and refused to provide licenses for using their fonts.

SO while the technology was available it wasn't used a lot because, other than the 'web safe' fonts that Microsoft made available, there wasnt really any good font to use.

It wasn't until the CSS3 Web Fonts Working Draft that provided a consistent way to load fonts. While the specification was under development people began to push for a re-introduction of @font-face into modern browsers. These people knew that some browsers hesitated to implement a Microsoft proprietary font format (EOT), so they began to push for different font formats, such as the newer TrueType and OpenType, to be used instead.

People were using custom fonts anyway, using techniques like cuffon and image replacement. These techniques are not accessible or future-proof as using actual fonts.

Font foundries began to see the value of the web and digital licenses, but still lacked a way to package and distribute their fonts for use on the web. The licensing issues were still present but had taken a back seat.

In 2009, web fonts became a thing (again). There wwere three things that helped this happen:

- Firefox and Sadary finally implemented @font-face using the TrueType and OpenType formats, not EOT.

        - This was enough for developers to start experimenting with web fonts across all browsers using things like the the Bulletproof @font-face syntax (now hopefully retired)
- On June 18, 2009 the CSS Fonts working group at the W3C released a full and cohesive specification for Web Fonts. There would, of course, be further revisions as browsers continued their implementations, but the draft represented something stable.
- Typekit was first released. Typekit is a font-hosting service that connected font foundries to designers and developers with easy to understand licensing terms and cross-browser compatibility. Typekit (now Adobe Fonts) was the first of many such font hosting services.

So now we have web fonts that work across browsers but there’s one thing missing: We still can’t work with fonts that are installed by the operating system or are desktop fonts.

If you see software like the Adobe Creative Cloud or Microsoft Office you will see that the font menus list the fonts available from the Operating System.

The Font Access API seeks to address this. It provides tools for browsers to work with local fonts installed at the Operating system level. This would allow applications and pages to use fonts that would not be available to the browser otherwise.

The most basic use is to query the local fonts and get a list of fonts available from the operating system. The example below uses the Font Access API to query the available local fonts and then logs metadata information about each font to the console.

Warning

The permission dialogue for the Font Access API is still under development. I've left it in the examples for completeness but have commented it out to prevent permission errors.

```
// Async block
(async () => {
  // const status = await navigator.permissions.query({ name: "font-acces
  // if (status.state === "denied")
  //   throw new Error("Cannot enumerate local fonts");

const fonts = navigator.fonts.query();

  try {
    // May prompt the user:
    for await (const metadata of fonts) {
      console.log(metadata.postscriptName);
      console.log(metadata.fullName);
      console.log(metadata.family);
    }
  } catch(e) {
    // Handle error. It could be a permission error.
    throw new Error(e);
  }
}
})();
```

The following code populates a drop-down selection form element with the available local fonts, and could be used as part of the user interface for an editing application.

Right now, the example will query for the available local fonts and build a select pull-down list. When you select a font it will display the example text in the selected font.

It uses the PostScript font name as the unique identifier when loading the font using `@font-face`. This is a safe option as it's supposed to be unique and is acceptable as the value of a `@font-face` declaration's local src artribute.

```
(async () => { // Async block
  // const status = await navigator.permissions.query({ name: "font-acces
  // if (status.state === "denied")
  //   throw new Error("Cannot continue to style with local fonts");

  const exampleText = document.createElement("p");
  exampleText.id = "exampleText";
  exampleText.innerText = "The quick brown fox jumps over the lazy dog";
  exampleText.style.fontFamily = "dynamic-font";

  const textStyle = document.createElement("style")"p" const fontSelect =
  fontSelect.onchange = e => {
    console.log("selected:", fontSelect.value);
    textStyle.textContent = `
      @font-face {
        font-family: "dynamic-font";
        src: local("${postscriptName}");
      }`;
  };

  try {
    "select"`
      @font-face {
        font-family: "dynamic-font";
        src: local("${postscriptName}");
      }`// May prompt the user:
    for await (const metadata of navigator.fonts.query()) {
      const option = document.createElement("option");
      option.text = metadata.fullName;

      option.value = metadata.postscriptName;
      fontSelect.append(option);
    }

    document.body.appendChild(textStyle);
    document.body.appendChild(exampleText);
    document.body.appendChild(fontSelect);
  } catch(e) {
    // Handle error. It could be a permission error.
    throw new Error(e);
  }
})();
```

There API surface will likely change before the Origin Trial is completed but even with these basics we can picture other areas and applications.

For example, we could build a word processor that can take advantage of locally installed fonts to display the content in the font that we want to use.

One question still remaining is how will this work with variable fonts and many other elements that I don’t know if they will be part of this API.