JS font loading API

CSS Font Loading Module Level 3 presents a Javascript API for loading fonts in a similar way to how Fontface Observer works.

This API allows you to do more things than third-party tools can and it's setup to interact directly with the browser's font loading process.

In its most basic application, the API will load the font, add it to the list of available fonts can can, optionally, add it to the document's stylesheet and cause it to render right away. This may harm CLS but it's the only way to get the font to render right away.

The FontFace constructor takes three parameters, the name of the font, the URL for the font wrapper using the url() format and an array of options.

```
var f = new FontFace(
  "Roslindale Text",
  "url(newfont.woff)",
  {}
);
```

The optional array of options mirros the options of the CSS `@font-face` rule.

`FontFace.ascentOverride` : retrieves or sets the ascent metric of the font

`FontFace.descentOverride` :retrieves or sets the descent metric of the font

`FontFace.display` : determines how a font face is displayed based on whether and when it is downloaded and ready to use.

`FontFace.family` retrieves or sets the family of the font.

`FontFace.featureSettings` : retrieves or sets infrequently used font features that are not available from a font's variant properties. It is equivalent to the font-feature-settings descriptor.

`FontFace.lineGapOverride` : retrieves or sets the line-gap metric of the font

`FontFace.stretch` : retrieves or sets how the font stretches

`FontFace.style` : retrieves or sets the style of the font. This controls the font italics or oblique values

`FontFace.unicodeRange` : retrieves or sets the range of unicode codepoints encompassing the font

`FontFace.variant` : retrieves or sets the variant of the font

`FontFace.variationSettings` : retrieves or sets the variation settings of the font.

`FontFace.weight` : contains the weight of the font

Once the font has been set up we can call the `load()` method to start the font loading process. The method returns a promise that will resolve when the font finishes loading.

When the promise resolves we add the font to the document's fonts array so it's available to the page and then we use it by setting the FontFamily attribute on the body element or in any element where we want to use the font.

```
f.load()
  .then(function (loadedFace) {
    document.fonts.add(loadedFace);
    document.body.style.fontFamily = "Roslindale Text, serif";
  });
```

Rather than set the font family as soon as the promise resolves we can use the `ready()` method of the FontFace object to wait for the font to complete loading before we take on any action.

In the following example, the load() method work aas before and is called to load the font, add the font to the fonts array and then set the fontFamily style attribute on the body element.

The `document.fonts.ready()` method assumes that the content inside the element with ID of `content` is hidden by default. Once the font has finished loading and the ready method is called and the content is revealed to the user by changing the visibility of the content.

```
f.load()
  .then(function (loadedFace) {
    document.fonts.add(loadedFace);
    document.body.style.fontFamily = "Roslindale Text, serif";
  });


document.fonts.ready
  .then(function () {
    const content = document.getElementById("content");
    content.style.visibility = "visible";
  });
```

We can also use `document.fonts.ready()` to query the fonts available to the browser. The following example will run after the fonts complete loading. It will list the number of fonts available to the browser and for each font it'll list is attributes.

```
document.fonts.ready
  .then(function () {
  console.log("fonts are ready");
  console.log('There are', document.fonts.size, 'FontFaces loaded.\n');

    'There are'// document.fonts has a Set-like interface.
    // Here, we iterate over its values.ar fontFace of document.fonts.valu
      console.log('FontFace:');
      for (var property in fontFace) {
        console.log('  ' + property + ': ' + fontFace[property]);
      }
      console.log('\n');
    }
  });
'FontFace:'
```

It is important to note that the fonts available to the browser are most likely not the same fonts that are available to your desktop applications. There are fingerprinting and security concerns for this.

For example, if we know that a company uses a font called “company font” and it’s only available to employees. If a website queries the list of available fonts and all fonts are available, the site may be able to guess if a user works at the company (or got the font illegally).

There is one more method on the API, `check()`. The method checks if a particular font is available to the browser.

The example below, the variable `checked` holds the results of testing if the `Roslindale Text` font is available in 12px size.

```
document.fonts.ready
  .then(function () {
    let checked = document.fonts.check('12px "Roslindale Text"');
    if (checked) {
      console.log('Roslindale Text is available');
    } else {
      console.log('Roslindale Text is not available');
    }
});
```

We could also populate a font properties sheet with information we get after we ensure the font is available.