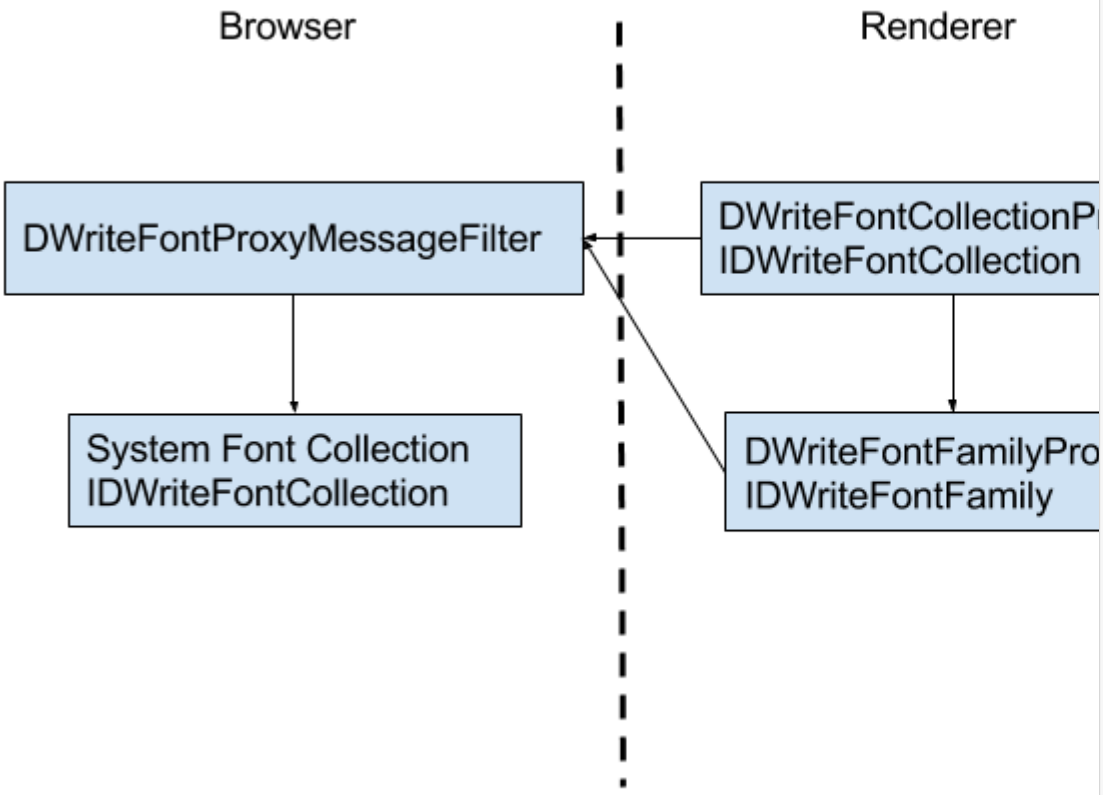
DirectWrite Font Proxy

Overview

The sandbox prevents the renderer from using the system font collection. To get around this, we use IPC to enumerate fonts from the system collection in the browser and load the corresponding font files in the renderer. The renderer is still able to leverage the functionality provided by DirectWrite by loading just the needed font files into individual collections, and presenting an IDWriteFontCollection interface that allows loading fonts on demand.

Implementation



DWriteFontProxyMessageFilter is responsible for interacting with the system font collection and processing the IPC messages. It can determine if any given font is present on the system, and return the list of corresponding font files. The message filter is also involved in font fallback.

The DWriteFontCollectionProxy implements IDWriteFontCollection, which is the primary interface the font proxy exposes to its clients (primarily Blink and Skia). The font collection proxy maintains a cache of all previously loaded font families, as well as a list of family names which it tried to load but which were not present on the system. When asked for a font family, the collection will send an IPC to determine if the family is present. If the family exists, the collection will return a DWriteFontFamilyProxy instance. Later, when the family is asked for more detailed font data, the proxy will send another IPC to get a list of needed font files from

the browser. These font files are loaded into a custom IDWriteFontCollection using IDWriteFactory::CreateCustomFontCollection, which provides the needed IDWriteFontFamily.

The DWriteFontFamilyProxy implements IDWriteFontFamily. This is mostly a pass-through implementation. The font family proxy will ensure that the font family is loaded (this involves sending an IPC to get the list of font files), and subsequently will forward calls to the underlying DirectWrite font family.

Font Fallback

Font fallback uses a similar mechanism to avoid having to load all the fonts in the renderer. When MapCharacters is called, FontFallback will send an IPC to locate the fallback font using the system font collection.

Future development

Windows 10 implments IDWriteFactory3::CreateFontCollectionFromFontSet which may provide a way to create a font collection that has all the same data as the system collection, but does not require loading all the font files. The following info was provided by a Microsoft developer and describes a potential approach:

> It sounds like you've found a pretty good solution with your IPC approach, but I'll explain more about IDWriteFontSetBuilder in case you find it helpful. In Windows 10, the font collection API is actually implemented in terms of the new font set API. Once you've created a custom font set, you can call IDWriteFactory3::CreateFontCollectionFromFontSet to create an equivalent font collection from it. There's no need to abandon your existing code that uses IDWriteFontCollection, which is still fully supported. However, you might also find it helpful to use the font set API directly in some cases – for example, to query by PostScript name.
>
> To create a custom font set:
> 1. Call IDWriteFactory3::CreateFontSetBuilder.
> 2. Call IDWriteFontSetBuilder::AddFontFaceReference once for each font you want to add.
> 3. Call IDWriteFontSetBuilder::CreateFontSet.
>
> There are two overloads of AddFontFaceReference. The one you want takes an IDWriteFontFaceReference and an array of DWRITE_FONT_PROPERTY structures. When you add a font this way, CreateFontSet just constructs the font set using the properties you specify, so it doesn't have to parse the font. The other overload just takes a font face reference, which means CreateFontSet has to parse the font file to extract the font properties.
>
> The above assumes you have a priori knowledge of all the font properties. One possible design would be for your higher-privilege process to get the system font set from DWrite, iterate over all the fonts, and serialize the font references (file names) and corresponding properties to an XML for JSON file. The lower-privilege processes can then consume that file to construct a custom font set (and font collection) in-process.
>
> To create a font face reference, you could call IDWriteFactory3::CreateFontFaceReference – no need to implement a custom loader. We won't actually open the font file unless you use the font. Note, however, that creating an IDWriteFont object counts as "using" the font in this context because not all the information exposed by that interface is stored as part of the font set. Therefore, you want to avoid patterns in which you enumerate all the fonts in the collection and create IDWriteFont objects for each. For example, you might be doing that now in order to build up a dictionary of PostScript names (something no longer necessary because PostScript name lookup is directly supported by IDWriteFontSet).

Comments

You do not have permission to add comments.