Research SVG Icons

Some initial investigation on SVG Icons was performed while investigating the viability of Font Icons. (See: Research of viability of using icon fonts in UI Options).

Why SVG Icons

In our previous exploration of Font Icons vs SVG Icons we adopted Font Icons, mainly because they provided the functionality we were looking for and were broadly supported. However, in practice we've experienced some annoyances with maintenance and usage. With font icons, you need to create a new font family and make use of the PUA value, specified at creation time, to reference the icon in the CSS. A few challenges arose around how to scope the font family. Adding in unused icons results in a font that is too large in size. However, as you need additional fonts, you are required to recreate the font, or link in an additional font. In the case where a font is extended, the creator has to be careful to keep the PUA codes for the original icons the same, else risk breaking existing implementations. In regards to PUA codes, another issue is when a platform already makes assumptions about what a PUA character is used for. (see: **FLUID-5225** - Font icon character codes collide with characters used in iOS OPEN). There are some build tools for creating the Font Icon, IcoMoon and grunt-webfoot.

The Infusion-Icons npm module can be used to simplify the process of tracking PUA codes and generating icon fonts. See: https://github.com/fluid-project/infusion-icons/blob/master/README.md

SVGs provide the same / better scalability to font icons and don't require hacking the font system to make them work. They also provide the options to support more complex icons including multiple colours. Additionally the source for font icons are SVGs anyways, so we can cut out one conversion step by using them directly. However, they are not without their own caveats. One being the fact that they don't have as broad support as font icons do. In terms of modern browsers this is no longer an issue, although there may be some rough edges if IE 11 support is required. Another issue, has to do with how the SVGs are created/exported in regards to their styles. In particular we want to be able to support easily theming them for contrast modes.

All that being said, I believe we are at or nearing a confluence of technological improvements and ideas where using SVG Icons may be the better option. (See: Inline SVG vs Icons Fonts for a comparison).

While beginning work on FLUID-6142 it was discovered that making use of `<use>` requires that a path to the external SVG sprite sheet be provided. This is okay when working directly on the page which will display them, but poses a problem for HTML templates. The issue being that the path defined in `<use>` is relative to the page that the HTML template is inserted into, rather than to the template itself. (See: http://lists.idrc.ocad.ca/pipermail/fluid-work/2017-March/010329.html)

Working with SVG Icons

See: How to work with SVG icons for a general walkthrough of using SVG icons.

Including the SVG Icon

See: SVG 'symbol' a Good Choice for Icons

In order to adjust an SVG's styles with CSS the first requirement is that the markup for the SVG is accessible in the browser. If we just add the SVG as an `<img>` tag, we won't have access to its markup. Alternatively if we embed the entire SVG markup into the html file, it may be large and difficult to work with and interpret, as well as being open for errors of accidental modification. Also, it isn't easily reusable within and between files.

The answer to this is to use `<use>, which can be used to reference an external SVG file and pull in its contents.`

```
<body>
        ...
        <svg>
                <use href="icons.svg#icon1" />
        </svg>
        ...
</body>
```

However, <use> is not available in IE 11 and there are other caveats about accessing the internal parts of the SVG. See, SVG 'use' with External Source, Take 2 for a further discussion on potential issues and work arounds. <use> does not provide sufficient cross browser support to handle complex SVG images; at the time of writing only Firefox 51 has good support. For these complex cases it is still best to use inline SVGs.

> ⚠ Issues when trying to use <use> inside of an HTML template because the URL needs to be relative to the document the template is inserted into, rather than to the template.

(See: svg-icon-spike repo for an exploration of using SVG icons)

Positioning and Layout

Setting the `viewbox` attribute in the SVG will provide information for the aspect ratio that the SVG has. This can be used in conjunction with the `preserveAspectRatio` property to define how the SVG is rendered within its viewport. For example this can be used to centre the SVG within the viewport, and define wether or not it should be clipped or scaled to fit within. Another thing to note is that it seems that browsers set a default size for the SVG's viewport. Unfortunately this will require setting the width and height if you want to have the SVG a specific size without any extra whitespace.

Creating SVGs

Care needs to be taken during the initial creation process of the SVGs. The viewBox (aspect ratio) of the SVG icon will be set based on the initial values specified or else need to be defined by the implementor when adding it to the HTML. Of even greater concern for our use case, has to do with styles. We want to be able to support multiple contrast modes, and as such, will need to be able to specify the stroke and fill colours used. Depending on how styles are exported, changing them with CSS may or may not be possible.

See the following for further discussions:

- How to work with SVG icons
- Demystifying Adobe Illustrator's Advanced Options for SVG work

Suggested Adobe Illustrator CC Settings

The following export SVG export settings seem to work well:

- Export using Artboards. To make best use of this, make sure the artboard is cropped tight to the artwork.
- SVG Options dialog
    - Styling: Internal CSS
    - Font: SVG
    - Images: Preserve
    - Object IDs: Layer Names
    - Decimal: 2
    - Minify - checked
    - Responsive - checked

SVG Sprites

For lots of icons it would be easier to work with a single SVG sprite that could be cached by the browser. However, we don't want to have to keep adding in additional icons to the sprite as new ones are required, similar to the font creation issue with font icons. Fortunately, grunt-svgstore provides a way of compiling individual SVG files into a single file. Additionally it provides options for cleaning up the SVG and adding custom IDs for the each sprite based on the original file name.

Styling the SVG

One solution for styling is to remove all of the default values for fill and stroke from the svg (this can be done as part of the build process by grunt-svgstore. In the CSS for the page, you could set the fill colour to "currentColor". This will take the value of the inherited or set color. It is possible to set the fill or stroke color in the svg itself to "currentColor" which provides for two color svgs. However, it does not seem possible to export svgs with this value from Illustrator, and would have to be manually inserted into the SVG itself; posing a maintenance burden.

Note:

- CSS stroke **will** override a presentation attribute `<path stroke="#fff" ... />`
- CSS stroke **will not** override an inline style e.g. `<path style="stroke: #fff;" ... />`

See:

- Cascading SVG Fill Color
- SVG 'use' with External Source, Take 2
- Styling SVG <use> Content with CSS (very detailed article and explains how to use CSS variables to have multi coloured SVGs styled with CSS).