Creating svg sprites

The idea behind SVG sprites is similar to sprite sheets using CSS to place the images on the page.

The structure of the demonstration project is as follows:

```
svg-sprites
├── gulpfile.js
├── icons
└── package.json
```

We'll use Gulp to create the sprite and the `gulpfile.js` contains the instructions for the build. The details of the process will be described in the next section.

The `icons` directory will hold the individual icons that we'll use in the sprite

`package.json` contains all NPM-related information necessary for the Gulp process to run.

Creating the sprite sheet

As with all Node projects, first we need to initialize the project's `package.json` file. I choose to automate the file creation and then, if necessaary, change it.

```
npm init --yes
```

The next step is to install the packages we need. I've chosen to do this as a standalone project but it can be just as easily included in a larger Gulp-based project.

The packages we will use are

- Gulp to drive the build process. I am running version 3, not the latest version so the syntax will be different than what you see for Gulp 4 tutorials
- run-sequence allows Gulp to run sequences of files in the specified order
- gulp-cheerio allows for document manipulation using a jQuery-like syntax

- gulp-svgminminifies SVG using SVGO
- gulp-svgstore combines SVG files into a single file using <symbol>

```
npm i -D gulp@3.9.1 \
run-sequence \
gulp-cheerio \
gulp-svgmin \
gulp-svgstore
```

We now start building the tasks that will create the sprite.

The first step is to require the packages we want to use. These are the same as the packages we installed in the previous step.

```
const gulp = require('gulp');
const runSequence = require('run-sequence');
const cheerio = require('gulp-cheerio');
const svgmin = require('gulp-svgmin');
const svgstore = require('gulp-svgstore');
```

`svgstore` is the main task of this build.

It takes all the svg files in the `icons` directory, minimizes them using `svgmin`, creates the sprite using `svgstore`, removes the fill attribute (if present) using `Cheerio` and the pushes the resulting `sprite.svg` file to the includes directory.

```
gulp.task('svgstore', () => {
  return gulp
    .src('icons/*.svg')
    .'icons/*.svg')
    .pipe(
      svgstore({
        fileName: 'sprite.svg',
      }),
    )
    .pipe(
      cheerio({
```

```
        run: function($) {
          $('[fill]').removeAttr('fill');
        },
        parserOptions: {
          xmlMode: true,
        },
      }),
    )
    .pipe(gulp.dest('includes/'));
});
```

The `default` task links to `svgstore` to make sure that using the gulp command without parameters doesn't give an error

```
gulp.task('default', () => {
  runSequence(['svgstore']);
});
```

Using the sprites

Using SVG images for the icons requires much less work to implement. We can use a single `svg` element to hold our icons and one additional `svg` element to insert the icons we referenced in our definition element.

To make the explanation easier I've put all the code inside the body of the HTML document.

We start with the SVG sprite we generated with our build process. I've manually copied it to the document. It looks like this (shortened to make it easier to read)

```
<svg xmlns="http://www.w3.org/2000/svg" id="icons">
   <symbol id="codepen" viewBox="0 0 24 24">
     <pat<symbol id="codepen" viewBox="0 0 24 24">5c-.01-.024-.018-.05-.0
   </symbol>
   </symbol><!-- Other symbols go here -->
```

```
</svg>
```

The next block is the CSS. In this section I want to highlight three areas:

First how we set the container for the icons, the `social-media-bar` element to be a flex container that takes half the width of the screen and is centered using margins.

The second is a peculliarity of SVG. While we can use CSS to style SVG the primary attribute names are different and we need to be careful not to confuse them. The `fill` attribute handles the background of an SVG element, whatever shape it has.

The last is to make sure we remove the underline for the links inside SVG elements.

Here to, I've removed selectors and attributes to make sure that it's readable without confusing you.

```
<style>
   .social-media-bar {
     display: flex;
     flex-flow: row wrap;
     justify-content: space-around;
     width: 50%;
     margin: 0 auto;
   }

   .codepen-icon {
     fill: #000;
   }

   .facebook-icon {
     fill: #4267b2;
   }

   a {
     text-decoration: none;
```

```
  }
</style>
```

The last section is the combination of HTML and SVG that actually builds the navigation bar.

Some of the attributes in the svg element look different than those in HTML.

xmlns provides a default namespace, an association between a string and an XML vocabulary, `xmlns:xlink` associates the string `xlink` with the xlink vocabulary, a way to link between XML vocabularies.

title and desc.

Height and width are presented without units to make sure it scaled appropriately.

The use element links to the reference the symbols we added earlier in the document. It's the XML way to link to an anchor... just like HTML's.

```
<div class="social-media-bar">
  <a href="https://codepen.io">
    <svg
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      title="Codepen"
      desc="Codepen Logo"
      height="50"
      width="50"
      class="icon codepen-icon"
    >
      <use xlink:href="#codepen" />
    </svg>
  </a>

  <a href="https://facebook.com/">
    <svg
      xmlns="http://www.w3.org/2000/svg"
```

```
      xmlns:xlink="http://www.w3.org/1999/xlink"
      title="Facebook"
      desc="Facebook Logo"
      height="50"
      width="50"
      class="icon facebook-icon"
    >
      <use xlink:href="#facebook" />
    </svg>
  </a>
</div>
```

So, after all the work, was it worth it?

> SVG has advantages and disadvantages. Advantages first

- It eliminates network requestsby inlining the resources
- Vector graphics scale to whatever screen size you need or want. No more responsive images
- You can style them with CSS

Disadvantages

- Vector graphics are only good for line drawings and icons
- They require more work to produce and display
- The format is not well supported among older browsers

With those advantages and limitations I would definitely consider using SVG for icons and other line drawings on the page.