[Articles](http://fvsch.com/articles/)[Code](http://fvsch.com/code/)[Photos](http://fvsch.com/photos/)

***How to work with SVG icons***

March 2016

There are many ways to use SVG icons in HTML and CSS, and I haven’t tried them all. This is how we do it in our small front-end team at [Kaliop](http://www.kaliop.com/). It works well for our needs, which include:

* Content and communication websites, often based on big CMSes, rather than full-JS web apps.
* Icons which are often simple, single-color icons (each potentially used in several different colors depending on context and user interactions). Two-color icons are possible too.
* IE9+ support.

*Table of contents*

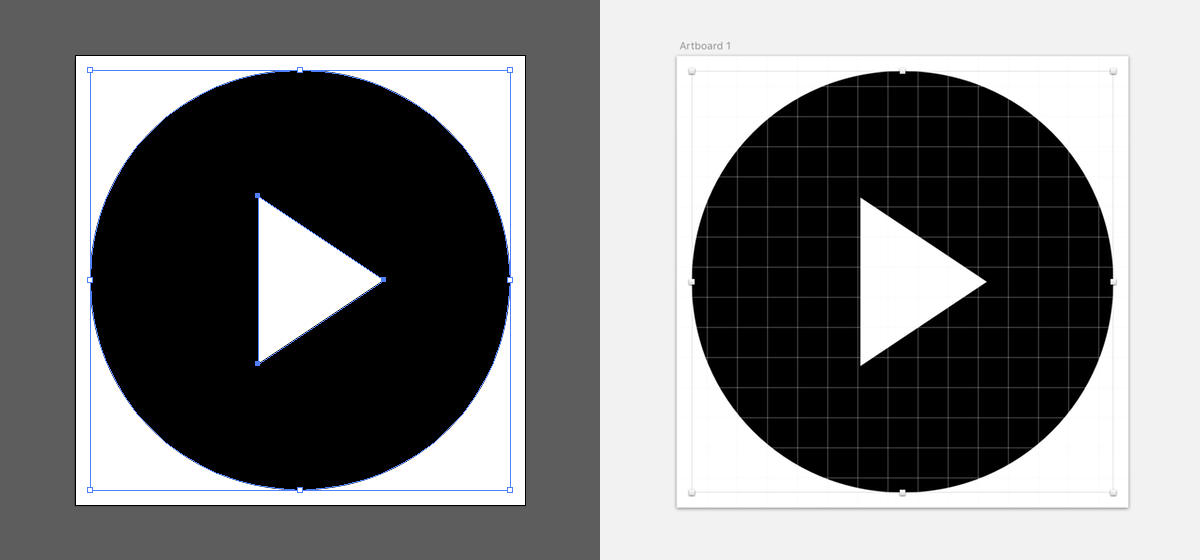
1. [Preparing your icons](http://fvsch.com/code/svg-icons/how-to/#section-preparing)
2. [Making a SVG sprite](http://fvsch.com/code/svg-icons/how-to/#section-sprite)
3. [Adding icons to your pages](http://fvsch.com/code/svg-icons/how-to/#section-adding)
4. [Styling icons with CSS](http://fvsch.com/code/svg-icons/how-to/#section-styling)
5. [Advanced topics and tricks](http://fvsch.com/code/svg-icons/how-to/#section-advanced)
6. [Conventions inspired by browser bugs](http://fvsch.com/code/svg-icons/how-to/#section-bugs)

*Translations*

This article is available [in French](https://la-cascade.io/comment-travailler-avec-des-icones-en-svg/) and [in Chinese](http://zhuanlan.zhihu.com/p/20753791).  
Many thanks to the translators!

PREPARING YOUR ICONS

When you get a SVG icon from a designer or from a graphics tool you use (Illustrator, Adobe Assets, Sketch, Inkscape, etc.), it’s tempting to just throw it into your project. Yet I find that reworking it a little bit in your favorite tool to make sure it’s *just right* can save you some headaches and improve the result.

Simple icon on an artboard in Illustrator (left) and Sketch (right)

*Work with a new document or artboard*

Create a new document or new artboard in your favorite tool, and copy-paste your icon in the center. It’s a great way to make sure your icon is clean and doesn’t have a ton of hidden paths lying around.

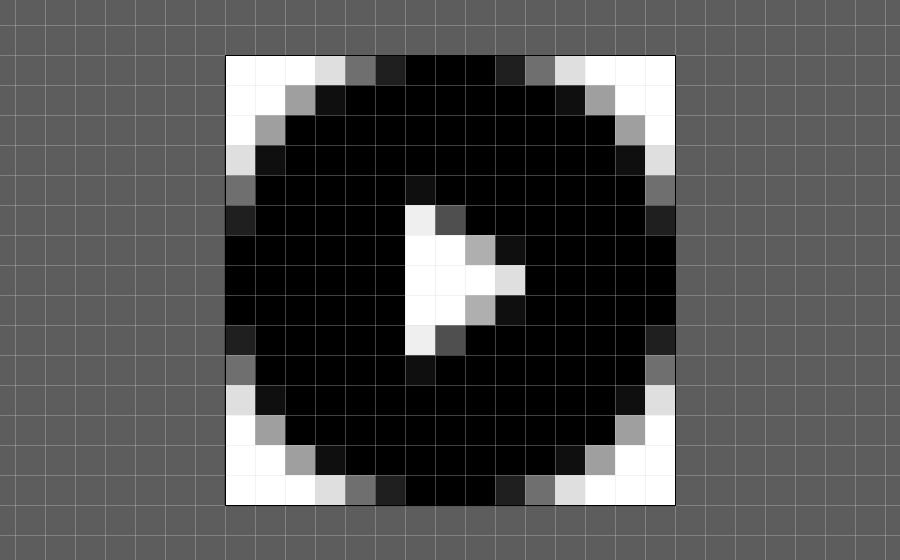
*Square is easier*

Your icon *doesn’t have to be square*, but square icons are easier to work with (unless your icon or graphic is really wide or really tall).

Exact dimensions only matter if you want to micromanage pixel fitting (to get the sharpest possible results on low dpi screens). For example if all your icons can fit on a 15 by 15 pixel grid, and are mostly used with those exact dimensions, go ahead and work with 15×15 artboards or documents. When I’m not sure, I like setting stuff on a 20×20 artboard.

*Breezy on the sides*

Leave a little bit of space near the edges, especially for round shapes. Browsers use anti-aliasing when rendering SVG shapes, but sometimes the extra pixels from the anti-aliasing are rendered outside of the viewBox and they’re cut off.

We didn’t leave any space around the icon, so there’s a risk that it will be rendered with squarish sides. And if the browser doesn’t render the SVG perfectly, it can get worse.

As a rule of thumb, in a 16px or 20px icon, leave 0.5px or 1px of empty space on each side. Also, remember to export the whole artboard, not the selected paths at the center, or you will lose that white space in the export.

*Export to SVG*

* In Illustrator I just use “Save As” and pick “SVG” for the format. (It might be better to use “Export as…” and pick SVG for an optimized result.)
* In Sketch you can select an artboard, click “Make Exportable” on the bottom right, and pick “SVG” for the format.
* In Inkscape you can “Save As” and pick “Optimized SVG”.

*Learn some SVG*

You definitely should learn some SVG basics, and be able to read and understand the structure of simple SVG files. At the very least you should know about these:

* Elements: <svg>, <symbol>, <g>, <path>.
* Attributes: d, fill, stroke, stroke-width.

Note that when you export SVG from a design tool, it will often have a little bit or a lot of unnecessary markup, metadata and such. It can also have excessively precise path data (in the d attribute). Try using a tool such as [SVGOMG](https://jakearchibald.github.io/svgomg/) and compare the before and after code to see what gets removed or simplified.

*Remove color data*

For single-color icons, make sure that:

1. In your source file, the paths are black (#000000).
2. In the exported code, there are no fill attributes.

If we have hardcoded fills in the SVG source, we won’t be able to change those colors from our CSS code. So it’s generally best to remove them, at least for single-color icons.

Illustrator doesn’t output fill attributes for path that are fully black (#000000). Sketch does, so you may have to open the exported SVG code and manually remove the fill="#000000" attributes.

MAKING A SVG SPRITE

This part has a lot of code, but it’s actually not complex at all. We want to create a SVG document containing <symbol> elements. Each<symbol> must have an id attribute, a viewBox attribute, and will contain the icon’s <path/> elements (or other graphical elements).

I’m calling this SVG document a sprite (in reference to [sprites in computer games](https://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29) and CSS), but it may also be called a sprite sheet, or a symbol store.

Here’s a sprite with just one icon:

<svg xmlns="http://www.w3.org/2000/svg">

<symbol id="cross" viewBox="0 0 20 20">

<path d="M17.1 5.2l-2.6-2.6-4.6 4.7-4.7-4.7-2.5 2.6 4.7 4.7-4.7 4.7 2.5 2.5 4.7-4.7 4.6 4.7 2.6-2.5-4.7-4.7"/>

</symbol>

</svg>

*Adding an icon to our sprite*

Say that Illustrator gave us this code for the icon show above:

<?xml version="1.0" encoding="utf-8"?>

<!-- Generator: Adobe Illustrator 19.2.0, SVG Export Plug-In . SVG Version: 6.00 Build 0) -->

<svg version="1.1" id="Layer\_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"

viewBox="0 0 15 15" style="enable-background:new 0 0 15 15;" xml:space="preserve">

<path id="ARROW" d="M7.5,0.5c3.9,0,7,3.1,7,7c0,3.9-3.1,7-7,7c-3.9,0-7-3.1-7-7l0,0C0.5,3.6,3.6,0.5,7.5,0.5 C7.5,0.5,7.5,0.5,7.5,0.5L7.5,0.5L7.5,0.5z M6.1,4.7v5.6l4.2-2.8L6.1,4.7z"/>

</svg>

We can simplify it quite a bit (manually or using [SVGOMG](https://jakearchibald.github.io/svgomg/)), only keeping the viewBox attribute and essential data:

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 15 15">

<path d="M7.5,0.5c3.9,0,7,3.1,7,7c0,3.9-3.1,7-7,7c-3.9,0-7-3.1-7-7l0,0C0.5,3.6,3.6,0.5,7.5,0.5 C7.5,0.5,7.5,0.5,7.5,0.5L7.5,0.5L7.5,0.5z M6.1,4.7v5.6l4.2-2.8L6.1,4.7z"/>

</svg>

Now we can copy-paste it in our sprite. We need to transform the<svg viewBox="…"> element into a <symbol id="…" viewBox="…">element and inserted it manually into our sprite:

<svg xmlns="http://www.w3.org/2000/svg">

<symbol id="cross" viewBox="0 0 20 20">

<path d="M17.1 5.2l-2.6-2.6-4.6 4.7-4.7-4.7-2.5 2.6 4.7 4.7-4.7 4.7 2.5 2.5 4.7-4.7 4.6 4.7 2.6-2.5-4.7-4.7"/>

</symbol>

<symbol id="play" viewBox="0 0 15 15">

<path d="M7.5,0.5c3.9,0,7,3.1,7,7c0,3.9-3.1,7-7,7c-3.9,0-7-3.1-7-7l0,0C0.5,3.6,3.6,0.5,7.5,0.5 C7.5,0.5,7.5,0.5,7.5,0.5L7.5,0.5L7.5,0.5z M6.1,4.7v5.6l4.2-2.8L6.1,4.7z"/>

</symbol>

</svg>

You can do this manually for all your icons, but there are tools that automate the process. We use [gulp-svg-sprite](https://github.com/jkphl/gulp-svg-sprite) (here’s an [example gulpfile.js](https://gist.github.com/fvsch/8a407c04156093c5f661) that we use, if you’re curious), but there are several graphical and command-line tools that export SVG symbol sprites, including [Icomoon](https://icomoon.io/).

*Pro tip: Keep a folder with your source icons*

If you make your sprite manually, I recommend keeping a folder with individual SVG icons:

assets/

icons/

cross.svg

play.svg

search.svg

...

public/

sprite/

icons.svg

Then if you need to rebuild the icons.svg or change an individual icon, you have the right sources to work with. Be careful to not let your sprite and your source folder get out of sync.

Of course if you use a build process (with Grunt or Gulp or something else), you can feed it your source folder and let it build the sprite directly.

ADDING ICONS TO YOUR PAGES

The bad news is that in order to use our SVG icons, we have to put them in the HTML. No CSS backgrounds, no ::before pseudo-elements. The good news is that it’s not too verbose:

<svg><use xlink:href="/path/to/icons.svg#play"></use></svg>

*Providing alt text for your icons*

There are a few possible solutions for adding accessible text to an icon. Based on our own screen reader tests, this is what we use.

First, if we don’t want to add any alt text (often because there’s already relevant text in context), we use aria-hidden="true" to make sure screen readers don’t try reading the icon out loud:

<a href="/news/">

<svg aria-hidden="true">

<use xlink:href="/path/to/icons.svg#newspaper"></use>

</svg>

Latest News

</a>

The second use case we have a lot is: there’s a link or a button whose only content is an icon. That’s when we use aria-label, preferably on the <a> or <button> element:

<a href="/news/" aria-label="Latest News">

<svg aria-hidden="true">

<use xlink:href="/path/to/icons.svg#newspaper"></use>

</svg>

</a>

Another option is using the <title> element. It’s especially useful outside of interactive elements (where aria-label might not be read by some screen readers). For instance, if you show yes/no markers in a table column, you could have:

<td>

<svg>

<title>Yes</title>

<use xlink:href="/path/to/icons.svg#tick"></use>

</svg>

</td>

Finally, remember that:

* Your accessible text *may* change depending on context. (A “magnifying glass” icon can mean “Show the search form” in one context, and “Submit my search query” in a second context.)
* Your accessible text *must* change when the page’s language changes.

This is why your alt text must live *in the HTML where your icon is inserted*. Some articles recommend putting <title> elements inside your sprites, but that doesn’t work in practice (also, most screen readers ignore it anyway).

*External and inline sprites*

Up until now we’ve shown examples for an *external* sprite. But some older browsers — namely, older WebKits and any version of Internet Explorer (before Edge 13) — only support *inline* references for <use xlink:href="#some-id"/>.

This can be polyfilled with some JavaScript ([svg4everybody](https://github.com/jonathantneal/svg4everybody), [svgxuse](https://github.com/Keyamoon/svgxuse)). Or you can decide to include your sprite *in the HTML code of every page*.

<body>

<!-- Hidden icon data -->

<svg aria-hidden="true" style="display:none">

<symbol id="icon-play">…</symbol>

<symbol id="icon-cross">…</symbol>

<symbol id="icon-search">…</symbol>

</svg>

<!-- A visible icon: -->

<button aria-label="Start playback">

<svg aria-hidden="true"><use xlink:href="#icon-play"/></svg>

</button>

</body>

Each method has its pros and cons.

|  | **Inline sprite** | **External sprite** |
| --- | --- | --- |
| **Browser support** | Native support in IE9+.  Older Safari/WebKit [needs the sprite at the beginning of the page](http://fvsch.com/code/svg-icons/symbol-sprite/) (before any reference). | Native support in Edge 13+, Safari 9+.  IE and older Safari/WebKit need a JS polyfill such as[svg4everybody](https://github.com/jonathantneal/svg4everybody) or [svgxuse](https://github.com/Keyamoon/svgxuse).  Loading an external sprite from a different domain doesn’t work for now (even with CORS, see [Chromium bug](https://bugs.chromium.org/p/chromium/issues/detail?id=470601) for instance). This can be polyfilled with JS using svgxuse. |
| **Caching** | No caching.  The weight of your sprite (5KB, 15KB, 50KB…) is added to every page. | Sprite is a separate file and can be cached by the browser.  Also it does not bloat your server-side HTTP cache. |
| **Rendering speed** | Icons are rendered instantly.  On slow networks, rendering of the page structure and content may be delayed if you have a heavy SVG sprite inserted before your content. | Possible FONSI! (Flash of No SVG Icons, but frankly I’m making an acronym as a joke so don’t reuse it okay?)  Icons can be shown a bit late because a) they require a separate HTTP request and b) the browser did not prioritize loading this SVG file (using its [look-ahead pre-parser](http://andydavies.me/blog/2013/10/22/how-the-browser-pre-loader-makes-pages-load-faster/)). |
| **Updating** | When you need to update your sprite, you need to make sure it’s added to every single page. This regenerating all HTML pages if you use a static generator, or invalidating all caches. | Only one public file is changed, so managing updates and server-side caching is a bit easier. |

I like mixing both methods, building two sprites:

1. A small one with essential icons (e.g. main icons used in the page header), to be inlined on each page. Target size: 5KB or less.
2. A bigger one with all the project’s icons, kept as an external file. Target size: 50KB or less.

On bigger projects, we might add more external sprites when some icons can be grouped together and are only used in one part of the site or for a specific feature.

STYLING ICONS WITH CSS

Okay now we have a SVG icons and a SVG sprite and we know how to add icons to our HTML and ***it took an awful lot of work***, can we finally style our icons or what?

*Sure, let me add some classes real quick*

You could select all <svg> elements in CSS but that’s not great if you use SVG for more than just icons. Also there’s a Firefox bug that could bite us in the ass if we do that ([details below](http://fvsch.com/code/svg-icons/how-to/#fx-use-selector-bug)), so let’s not.

Instead, I recommend adding two classes for each icon: a generic one, and one with the symbol’s name.

<svg class="Icon Icon--arrow" aria-hidden="true">

<use xlink:href="/path/to/icons.svg#arrow"></use>

</svg>

We’re using [SUIT CSS naming conventions](https://github.com/suitcss/suit/blob/master/doc/naming-conventions.md), but you can use a different style if you want (for instance class="icon-arrow" is shorter, and can be targeted with a selector such assvg[class\*="icon-"]).

*Default icon style*

I recommend this default style for your icons:

.Icon {

/\* Allows sizing by changing the icon’s font-size \*/

width: 1em; height: 1em;

/\* Nice visual alignment for icons alongside text \*/

vertical-align: -0.15em;

/\* Default path fill = value of the color property \*/

fill: currentColor;

/\* Paths and strokes that overflow the viewBox can show in IE.

If you use normalize.css, it already sets this. \*/

overflow: hidden;

}

Icons with our default style. The only change between the top and bottom row is the font-size and color of the container.

Then if you want to customize an icon in a specific context, you can have styles that look like:

.MyComponent-button .Icon {

/\* Change the width and height \*/

font-size: 40px;

/\* Change the fill color \*/

color: purple;

/\* Change the vertical-align if you need more precision

(sometimes needed for pixel-perfect rendering) \*/

vertical-align: top;

}

With that code, by default your SVG icons should be small and use the parent’s text color.

If the icon’s shapes do not inherit the parent’s text color (currentColor), check that you don’t have hardcoded fill attributes in your icon’s code.

*Inherited SVG styles*

Many SVG style properties are inherited. For instance when we set thefill CSS property on our containing <svg> element, it trickles down to our <path>, <circle> and other graphic elements.

We can use this technique for other SVG CSS properties as well. For instance the stroke properties:

.Icon--goldstar {

fill: gold;

stroke: coral;

stroke-width: 5%;

stroke-linejoin: round;

}

Star icon with default and custom styling

Most of the time we’re not going to change a lot of stuff: only the fillproperty for the main color, and sometimes we’ll add or tweak astroke (kinda like a border).

*Two fill colors per icon*

There’s a fairly simple technique which allows an icon to have two set of paths with two different fill values (aka two colors).

<symbol id="check" viewBox="0 0 20 20">

<!-- Will inherit the value of the fill CSS property -->

<path d="…" />

<!-- Will inherit the value of the color CSS property -->

<path fill="currentColor" d="…" />

</symbol>

.Icon--twoColors {

fill: rebeccapurple;

color: mediumturquoise;

}

Two color icon

*Leave some room for strokes*

Remember when we said to leave some room around your shapes? It’s especially important if you plan to use strokes.

.Icon--strokespace {

fill: none;

stroke: currentColor;

stroke-width: 5%;

}

In SVG, strokes are always painted on both sides of the path. If your path touches the limits of the viewport, half the stroke will be cut off.

In this example, the first icon has no reserved blank space on the sides, and the second icon has a small one (0.5px, for a 15px viewport).

*Use percentages for stroke-width*

Giving the right size to a stroke can be challenging. Look at these two examples where we set stroke-width:1px on the <svg> element:

What’s happening? The stroke-width property takes a “length” value, but that value is relative to the *local coordinates* of your icon. In the examples above:

1. The first icon has a 20px by 20px viewBox. So a 1px stroke is 1/20th of the icon’s size, big but not too big.
2. The second icon has a 500px by 500px viewBox. So a 1px stroke is 1/500th of the icon’s size, and that really really small.

If all your icons use a similar viewBox, that’s not a problem. But if they can differ wildly, using pixel or unitless values (stroke-width:1) is out. What should we do?

Percentages can be good. Same example, with stroke-width:5%:

Now that’s better. (For square icons, stroke-width:N% will work perfectly, but note that it can behave a bit differently [for large or tall SVG elements](http://fvsch.com/code/svg-icons/how-to/#percent-stroke-width).)

*Not everything is an icon*

Just because something is SVG, it doesn’t mean it should go inside your SVG sprite. For instance:

* Illustrations that you don’t need to style: consider using the <img>element.
* Illustrations that you want to animate: consider inlining the whole<svg> in your page, so you can select and style specific groups or paths or shapes, animate some parts, etc.

As a rule of thumb, if it’s a big illustration that you need to show at 100px by 100px or much bigger, or if it contains dozens of elements, it might not be an “icon”.

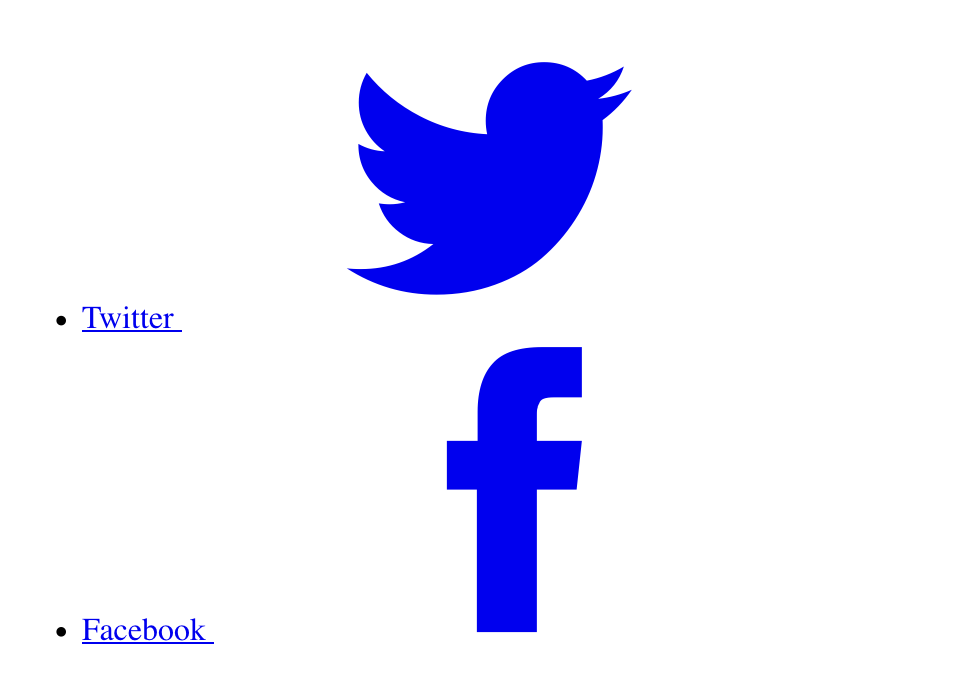
ADVANCED TOPICS AND TRICKS

With the previous sections you should have everything you need to know to use SVG icons. This next section adds some more advanced information, with perhaps fewer real-world applications.

*Avoid really big unstyled icons*

What happens if your main stylesheet fails to load because the user is on a flaky connection, maybe on a train (this happens to me all the time)? The page might still render without styles.

If you have a decent HTML structure, the page will still be readable, but your icons will end up really big.

Recent browsers size the <svg> element at 300 by 150 pixels by default. Other browsers might give it a gigantic 100% width!

I recommend putting this in the <head> of your pages:

<style>.Icon{width:1em;height:1em}</style>

Short and sweet.

*Preloading external sprites*

In the “Adding icons to your pages” section we said that icons from an external sprite can show up a bit late, partly because of the browser’s pre-load scanner (or look-ahead pre-parser or whatever) not picking up that <use xlink:href="/path/to/icons.svg#something"></use> means there is an important file to load early.

What can we do about this?

* Standard and future solution: add <link rel="preload" href="/path/to/icons.svg" as="image"> in the <head> of the page ([details on preload](https://www.smashingmagazine.com/2016/02/preload-what-is-it-good-for/), support coming to a Chrome near you and hopefully in more browsers in the future).
* Old school solution: add <img style="display:none" alt="" src="/path/to/icons.svg"> at the start of the <body>.

I haven’t actually tested those solutions, generally the “inline + external” compromise gives good enough performance that we don’t have to rely on preloading. But it’s worth investigating.

*Selecting individual shapes or paths*

We’ve looked at ways to customize fills, strokes etc. for all paths from a <symbol>, and 2 or more colors from different paths. But what if we could select specific paths (using classes maybe) directly in the*instance* of the <symbol>? Is it possible?

Right now the answer is: yes and no.

1. If you use external sprites, you can’t select individual paths (or other elements) inside a used <symbol>.
2. If you inline your sprite, you can select and style elements inside the sprite, but those styles will apply to *all instances* of the symbols.

So even with an inlined sprite, you could do this:

#my-symbol .style1 {

/\* Styles for one group of paths \*/

}

#my-symbol .style2 {

/\* Styles for another \*/

}

But you can’t do this:

.MyComponent-button .Icon .style1 {

/\* For 1 group of paths for this icon in this context \*/

}

.MyComponent-button .Icon .style2 {

/\* For another group \*/

}

Except in Firefox! Turns out that in Firefox, selecting inside an*instance* of the symbol works perfectly. The only problem is that it’s a non-standard behavior, so there’s no chance that other browsers will gain the same capabilites. It would actually be a good thing if Firefox fixed this bug they have.

In the future, there might be a standard way that allows selecting through a Shadow DOM, but that’s not sure at all (there used to be the/deep/ combinator, but it was removed).

*More than two colors with CSS Custom Properties*

So we can easily change colors of our SVG icons from CSS, for single-color icons (easy) and two-color icons (takes some preparation). Is there a way we could have multicolor icons with *more than two customizable colors*?

We might be able to do that with [CSS Custom Properties](https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables) (aka CSS Variables). This requires a lot of preparation on the SVG side:

<symbol id="iconic-aperture" viewBox="0 0 128 128">

<path fill="var(--icon-color1)" d="…" />

<path fill="var(--icon-color2)" d="…" />

<path fill="var(--icon-color3)" d="…" />

<path fill="var(--icon-color4)" d="…" />

<path fill="var(--icon-color5)" d="…" />

<path fill="var(--icon-color6)" d="…" />

</symbol>

For this demo I stole an icon from the excellent [Iconic](https://useiconic.com/), which offers responsive, multicolor SVG icons (powered by CSS and some JS, as I understand). I tried to mimick their own multicolor example for this icon, I hope they don’t mind.

  One symbol using 6 different CSS custom properties.  
See in Firefox, Chrome, or Safari 9.1+

It works fairly well [in supported browsers](http://caniuse.com/#feat=css-variables). There’s only one icon: the first instance doesn’t declare the expected variables, so it falls back to currentColor; the next two instances each declare a set of variable values.

*How are percentage stroke-width computed?*

What does the percentage correspond to in stroke-width:N%? Is it the width, the height of the icon? Turns out it’s relative to the diagonal, but with a [funky formula (spec)](https://svgwg.org/svg2-draft/coords.html#Units) (diagonal length divided by the square root of 2, which is close to 1.4).

What does it mean? Well for square icons the result of that formula is the side of the square. So 1% means “one percent of the width or one percent of the height”. Nice and simple.

For wider or taller icons, though, the result can change a bit:

In the second icon (aspect ratio of 2:1, shown with the same height and twice as large), the stroke-width:5% gives us a stroke that is roughly: 7.91% of the height and 3.95% of the width.

All things considered, I still recommend using percentage values forstroke-width. If you stick to square or sharish icon, you can use percent values with the understanding that they mean, roughly, “percentage of the width of the icon”.

*Sorry, no gradient fill*

With all the possibilities for fill colors, surely we can do something simple like use a gradient as a fill?

Sadly, we can’t. The fill property doesn’t accept image values, and CSS’s linear-gradient() function generates an image value.

SVG has its own syntax for coding and using gradients. But using it would take us rather far from *simple* SVG icons, so I’m going to just say: it might be done, but it’ll take same work and you’ll have to hardcode at least some parameters. Give it a try if you want. :)

CONVENTIONS INSPIRED BY BROWSER BUGS

*Safari: avoid width and height attributes*

In order to avoid gigantic icons in unstyled pages, we started by relying on the <svg> element’s width and height attributes:

<svg width="20" height="20">

<use xlink:href="…"></use>

</svg>

Then we tested our first “SVG icons-based” websites in Safari on iOS and half of the icons were broken! Why?!??

Turns out that Safari/WebKit doesn’t like having width and heightattributes with one size, and CSS code that tries to change that size later on. Especially when making the icons smaller, the icon’s box would get smaller, but not the content!

Our solution was to ditch those attributes and rely on CSS only for sizing icons.

Note that this bug may have been fixed in the most recent releases of Safari (9.1 on desktop, iOS 9.3).

*Safari: avoid padding on the <svg> container*

If you want a background color, borders, padding, etc., you should try to style the element containing the icon, and not the <svg> element itself. Although it seems to work well in the latest browsers, there are known rendering issues in older WebKit browsers, so I recommend styling a wrapping element (<span>, <button>, <a>, etc.).

Box styles applied on the <svg> element directly or on a wrapper element. Most browsers should render the elements identically, but slightly older WebKit versions will take offense.

*Firefox: avoid targeting the svg element*

It will create issues in Firefox. Why? When we use the <use> element, browsers create a Shadow DOM where they duplicate the contents of the <symbol> we’re using. Schematically, it looks like this:

<svg class="Icon Icon--something" aria-hidden="true">

<use xlink:href="#something">

<svg viewBox="0 0 20 20">

<path d="…" />

</svg>

</use>

</svg>

As explained in the previous section, Firefox currently allows selection in the Shadow DOM created by the <use> element. So if you have this CSS:

svg {

fill: red;

}

.Icon--something {

fill: green;

}

In Firefox the styles resolve as:

<svg class="Icon Icon--something" aria-hidden="true" fill="green;">

<use xlink:href="#something">

<svg viewBox="0 0 20 20" fill="red;">

<path d="…" />

</svg>

</use>

</svg>

In other browsers the the icon will be green (as you expect), but in Firefox it will be red because the inner <svg> element will get styles from the first rule-set (fill: red).

Another way to avoid this bug is by using this selector:

:not(use) > svg { … }