Finally Understanding SVG

SVG is a powerful vector graphics format that works particularly well for icons and other line artwork in web pages. This is my attempt at documenting what I understand of SVG and its component elements and children

The W3C began development of SVG in 1999 and released as a W3C specification in 2001. It is an XML-based vector image format for two-dimensional graphics and supports interactivity and animation.

SVG images and their behaviors are defined in XML text files and can be best thought of as a set of instructions for drawing shapes and defining behavior. The instructions in SVG read something like this: “Go to such-and-such coordinate; draw a rectangle that is so tall and so wide.

The fact that it’s text means that they can be searched, indexed, scripted, and compressed. As XML files, SVG images can be created and edited with any text editor, as well as with drawing software.

Let’s look at the biggest source of confustion for me: vieports and viewBoxes and

Viewport versus viewBox

The basic SVG canvas element defines the viewport of the image with the origin at the top left corner, point (0, 0), the x-axis points towards the right, and the positive y-axis points down. One unit in the initial coordinate system equals one “pixel” in the viewport.

To create an 800px by 600px SVG element use the following code:

```
<svg  width="800px" height="600px"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <!-- the content goes here -->
</svg>
```

It will produce an SVG element like the one below, captured from Sara Soueidan’s

Interactive SVG Coordinate System demo

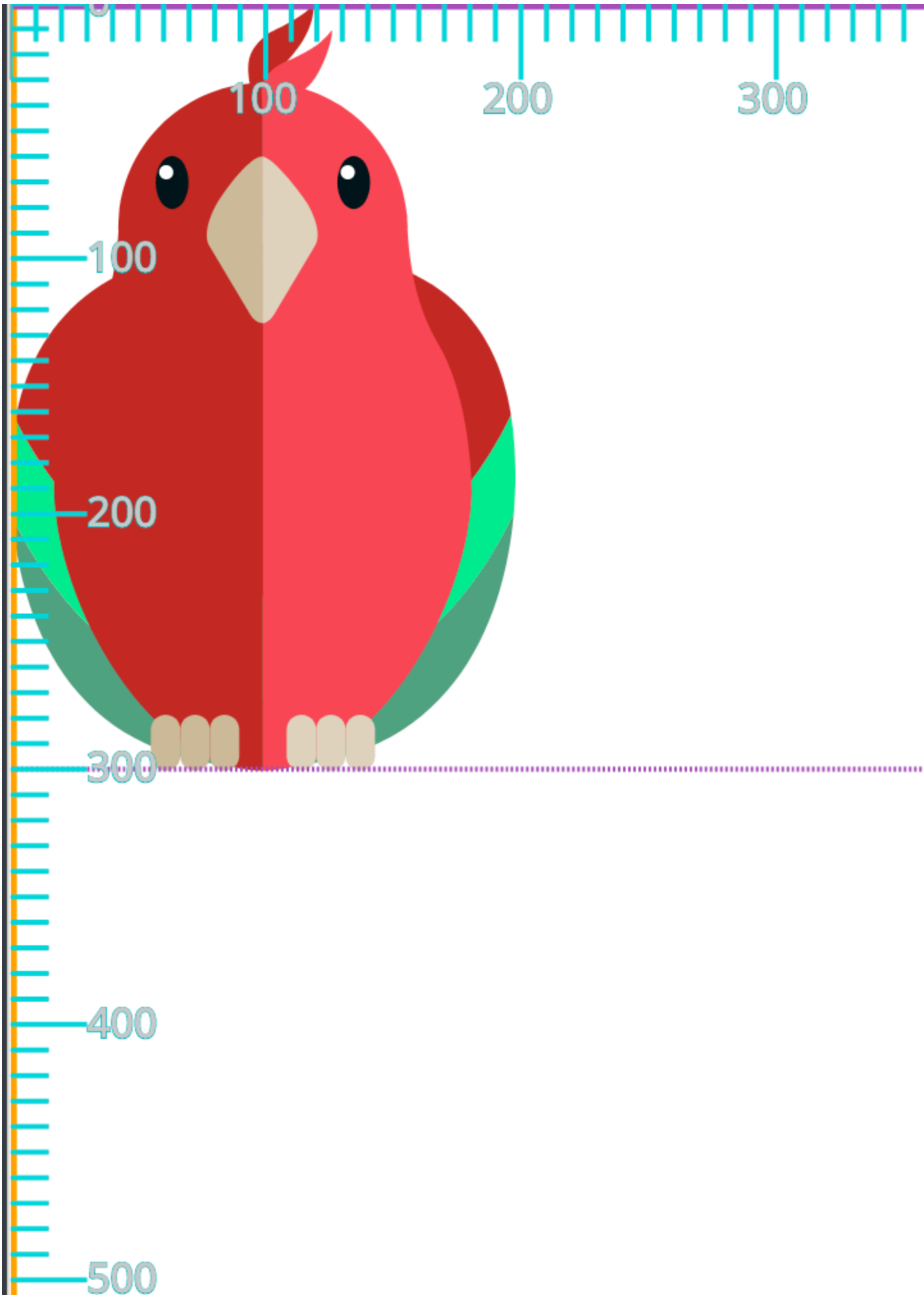
0
100
200
300
100
200
300
400
500

Figure 1: SVG viewport with only height and width

Size attributes are optional. We'll look at how to manage sizing and controlling the aspect ratio of SVG images later in the post. One thing I'm not sure about is whether we need the dimensions to prevent unwanted reflow of the page... more research needed there.

In addition to deciding what size you want your SVG to be, you're also going to have to decide how you want your graphic to scale to fit that size.

Viewbox

One of the hardest areas for me to wrap my head around is the fact that we can define our own viewports inside the svg element using the `viewBox` attribute of the SVG element and how it interacts with the SVG element dimensions and the `preserveAspectRatio`.

The viewBox is an attribute of the `<svg>` element. Its value is a list of four numbers, separated by whitespace or commas: x, y, width, height.

The width is the width in user coordinates/px units, within the SVG code, that should be scaled to fill the width of the area into which you're drawing your SVG (the viewport in SVG lingo). Likewise, the height is the number of px/coordinates that should be scaled to fill the available height. Other units, such as inches or centimeters, will also be scaled to match the scale created by the viewBox.

The x and y numbers specify the coordinate, in the scaled viewBox coordinate system, to use for the top left corner of the SVG viewport. For simple scaling, you can set both values to 0. However, the x and y values are useful for two purposes:

- Create a coordinate system with an origin centered in the drawing (this can make defining and transforming shapes easier)
- Crop an image tighter than it was originally defined.

Once you add a viewBox to your `<svg>` document you can use that SVG file as an image, or as inline SVG code, and it will scale perfectly to fit within whatever size

you give it, it will not be stretched or distorted if you give it dimensions that don't match the aspect ratio.

Preserve Aspect Ratio

The `viewBox` attribute has a sidekick: `preserveAspectRatio`. It has no effect unless there's a viewBox to define the aspect ratio of the image. When there is a viewBox, preserveAspectRatio describes how the image should scale if the aspect ratio of the viewBox doesn't match the aspect ratio of the viewport. **The default behavior works well most of the time: the image is scaled until it just fits both the height and width, and it is centered within any extra space.**

`preserveAspectRatio` default behavior can be explicitly set with preserveAspectRatio="xMidYMid meet". The first part, xMidYMid tells the browser to center the scaled viewBox region within the available viewport region, in both the x and y directions.

The second half of the default preserveAspectRatio, `meet`, is the part that tells the browser to scale the graphic until it just fits both height and width.

The attribute packs a lot in the definition of its possible values. The table in MDN's preserveAspectRatio Syntax, explains the possible values and what they do.

Be careful with capitalization: SVG, like XML, is case sensitive. The x is lowercase but the Y is capital.

So, now that we have our SVG graphic complete with `viewBox` and `preserveAspect`, how do we scale the graphic without distorting it?

The viewBox attribute is really all you need here, although you can use preserveAspectRatio to adjust the alignment. the correct `viewBox` values will vary from image to image.

When an SVG file has a viewBox, and it is embedded within an <img>, browsers will (nearly always) scale the image to match the aspect ratio defined in the viewBox, with the notable exception of Internet Explorer.

SVG images are nice, but in many cases you'll prefer to use inline SVG. Inline SVG reduces the number of HTTP requests, allows user interactions, and can be

modified by the CSS in your main web page.

They will only scale if you're using the latest Firefox or Blink browsers. Just set the viewBox on your <svg>, and set one of height or width to auto. The browser will adjust it so that the overall aspect ratio matches the viewBox.

Test for compatibility with your target browsers and that the result will work as intended.

Links and resources

- Specifications
  - SVG 1.1 (second edition)
  - SVG 2
- The SVG 2 Conundrum
- The difference between size and viewport
  - Understanding SVG Coordinate Systems and Transformations (Part 1) — The viewport, viewBox, and preserveAspectRatio
  - Understanding SVG Coordinate Systems and Transformations (Part 2) — The transform Attribute
  - Understanding SVG Coordinate Systems and Transformations (Part 3) — Establishing New Viewports
  - How to scale SVG