SVG in html5 and JavaScript

Table of Contents

What is SVG?

There are two general ways to store an image in a digital system - pixel-based bitmaps and vector graphics.

In a pixel-based image (such as Windows bmp, gif, jpeg, png, tiff files) the image is made of a set of dots (picture elements, pixels) in rows and columns. Each pixel contains red, green and blue components, and maybe a transparency value for layered images.

Actual image file formats are like this (tiff is very simple), but may include techniques for compressing the file size (for pngs, gifs and jpegs).

In vector graphics, the image is made of a set of elements, such as lines, rectangles, circles and so on. Each element has attributes like size, position, fill colour, line colour, line thickness and so on. A vector graphics file simply lists these lines, circles and so on. In effect a vector graphics image is a set of instructions, like draw a line from here to there, draw a circle with centre.. and so on.

Vector graphics have a long history, back to the 1960s, used in applications like air traffic control and CAD. These were displayed on CRTs, and the electron beam could actually trace out the lines so it was a natural method.

The bit-mapped formats like png are binary files – byte streams with pixels represented by bytes. Vector graphics are text files – instructions to draw from here to there, and so on. So they are human and machine readable.

SVG

SVG is 'scalable vector graphics'.

This usually means W3C's specification of an XML-based vector graphics mark-up language. It has shapes named path, rect, circle, ellipse, line, polyline and polygon.

Pixel-based storage is better for photos, paintings and similar. To describe a photo in terms of simple circles, rectangles and lines would need a very large number of very small circles and so on, to get an accurate image. SVG for photos takes an enormous amount of storage.

But for line drawings, diagrams, charts and similar images SVG is much better. A chart is typically made of a few rectangles and lines, and this is just a small number of elements, so the SVG memory size is small.

Also, SVG is *scalable* - this is a key feature.

Suppose you have a jpeg image with resolution 10 by 10 pixels, and you want to enlarge it to 100 X 100 pixels. You need some way to fill in the missing pixels (interpolation) and this never works well. Or you can just enlarge the pixels you have, producing a blocky 'pixellated' image.

But for a SVG image you just scale up the sizes and positions of the lines and rectangles and other elements. It looks just as good when you zoom in or out. A third advantage of SVG in a webpage is that you can access, and modify, the picture programmatically - usually with JavaScript. So you can create and alter the image by program code inside a browser.

Documentation of SVG 1.1 from w3c is here

MDNs documentation of SVG is here.

These notes outline the SVG elements, in an html5 web page in a browser, and how these can be scripted using JavaScript.

SVG in html5

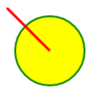
The SVG element

html5 has two graphics elements - <canvas> and <svg>. Canvas is used to display 2d and 3d pixel graphics. svg displays SVG graphics.

So a web page can have an <svg> element (or several). Inside the svg we can have a set of image elements, such as this:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
   <svg width="100" height="100">
      <!-- SVG element -->
      <!-- a circle -->
      <circle cx="50" cy="50" r="40" stroke="green" stroke-width="2" fill="yellow" />
      <!-- a line -->
      <line x1="0" y1="0" x2="50" y2="50" stroke="red" stroke-width="3" />
   </svg> <!-- end SVG -->
</body>

</html>
```

Which renders as



SVG is XML-based. A piece of SVG could be a stand-alone document (not inside html). Then the svg element needs to state the namespace it uses, like

```
<svg viewBox="0 0 300 100" xmlns="http://www.w3.org/2000/svg">
```

then that means that the name ‘circle’, for example, is as in that namespace, and enables a parser to check it.

But in html, the namespace is not needed.

The defs element

An SVG element will contain things like a <circle> element, to draw shapes.

But some things – such as an arrow shape or a text style – might be re-used several times in a drawing.

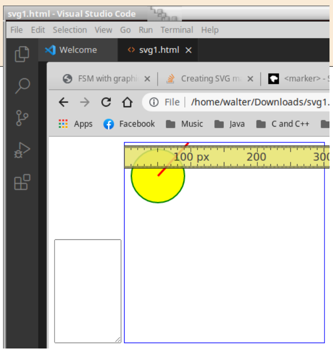
In this case a <defs> element can contain definitions of such items, which can be re-used in items actually drawn. Examples follow.

Circles and lines

SVG has primitives to draw circles, ellipses, rectangles, straight lines, polygons and paths (Bezier curves). There are documented here at w3c and here at MDN.

For example:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
   <textarea rows=10 cols=10></textarea>
   <svg width="300" height="300" style="border:thin blue solid">
      <!-- SVG element -->
      <!-- a circle -->
      <circle cx="50" cy="50" r="40" stroke="green" stroke-width="2" fill="yellow" />
      <!-- a line -->
      <line x1="1in" y1="0" x2="50" y2="50" stroke="red" stroke-width="3" />
   </svg> <!-- end SVG -->
</body>

</html>
```



svg1.html - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Welcome
svg1.html
FSM with graph
Creating SVG m
<marker> -
File /home/walter/Downloads/svg1.
Apps
Facebook
Music
Java
C and C++
100 px
200
300

which renders as

Each primitive, like ‘circle’, has a set of attributes to control size, position, colour and so on.

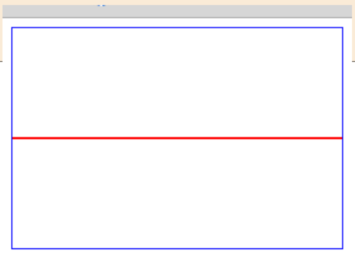
Co-ordinates are measured from the top left of the SVG element (and not from the browser window corner, or the screen corner.

Units default to pixels, but others are possible. The line starts 1 inch in, and CSS defines an inch to be 96 pixels as shown.

Viewbox

An SVG element can have a *viewBox* attribute. The idea of this is to make sure the relative sizes and positions of components will stay correct even if the overall size of the SVG element changes. This works by scaling co-ordinates, as this example:

```
<body>
   <svg viewBox="0 0 100 100" width="300" height="200"
   preserveAspectRatio="none"
   style="border:thin blue solid">
      <line x1="0" y1="50" x2="100" y2="50" stroke="red" />
   </svg>
</body>
```



which produces this:

The actual SVG size is 300 by 400 pixels. But the viewBox scales this so that 0,0 is top left, and 100,100 is bottom right.

So the line is drawn from 0,50 to 100,50, which is left edge halfway down to right edge halfway down.

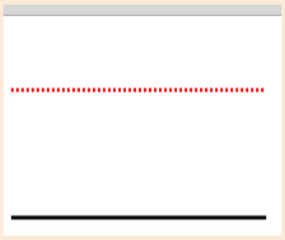
The preserveAspectRatio="none" allows to modify this in certain ways, such as scaling x only, or y only, or keeping it centered, and so on.

Styles

As with html elements, we can set display properties in two ways – either setting attributes on each element, or defining a style, and applying the style to the element. As for html, SVG elements can have id and class, for use in a style selector.

For example:

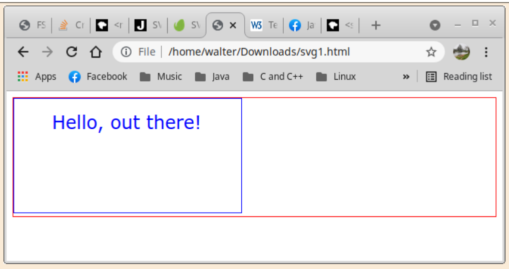
```
 <!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <style>
      line.Red {
         stroke: red;
         stroke-width: 3;
         stroke-dasharray: 2;
      }

      line.Black {
         stroke: black;
         stroke-width: 3;
      }
   </style>
</head>

<body>
   <svg width="200" height="200">

      <line x1="0" y1="50" x2="200" y2="50" class="Red" />
      <line x1="0" y1="150" x2="200" y2="150" class="Black" />

   </svg>
</body>

</html>
```



Text

Like this:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
</head>

<body style="border: thin red solid">

   <svg style="border:thin blue solid">
      <text x="50" y="40" font-family="Verdana" font-size="24" fill="blue">
         Hello, out there!
      </text>
   </svg> <!-- end SVG -->
</body>

</html>
```

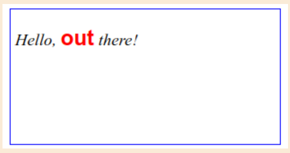


Hello, out there!

tspan

The idea is to style a section of text differently from the rest of the line. It is like <span> in html:

```
 <!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
</head>

<style>
   text  { font: italic 14pt serif; }
   tspan { font: bold 18pt sans-serif; fill: red; }
</style>

<body >

   <svg style="border:thin blue solid">
      <text x="5" y="40" >
         Hello,
         <tspan> out</tspan> there!
      </text>
   </svg> <!-- end SVG -->
</body>

</html>
```



Hello, **out** *there!*

Paths

A path is a 2D path, which can be used to

- draw or fill a shape made of straight line segments,

- or draw or fill a curved line, as a Bezier curve
- or display text on that curve

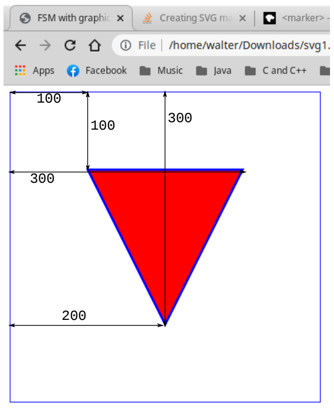
The element has expected attributes like fill colour, stroke colour, stroke thickness and so on. It also has a parameter named ‘d’, which is a string containing a sequence of commands. See w3 docs.

Straight-edge path

For example

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
</head>

<body >
   <svg width="400" height="400" style="border:thin blue solid">
      <path d="M 100 100 L 300 100 L 200 300 z"
        fill="red" stroke="blue" stroke-width="3" />
   </svg>
</body>

</html>
```

produces (with measurements added)



The d attribute d="M 100 100 L 300 100 L 200 300 z" means

Move to 100, 100 (from the SVG element corner, y down)

Draw a line to 300, 100

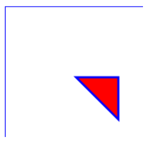
Draw a line to 200, 300

z means close the shape.

If we use lower case letters, the co-ordinates are relative to the last point, so

```
d="m 100 100 l 60 0 l 0 60 z"
```

means line 60 right, 0 down, then 0 right, 60 down, so we get

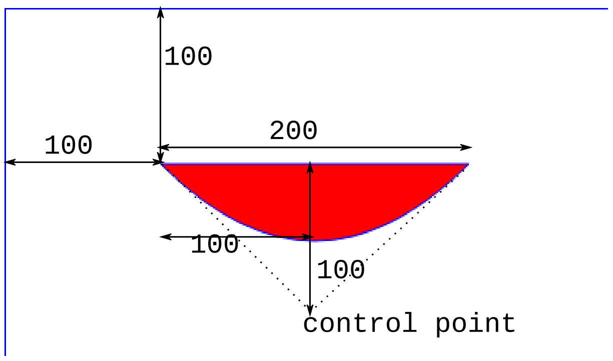


Quadratic Bezier curve

A quadratic Bezier curve has one ‘control point’. The curve is drawn from a start point to an end point. The lines to the control point form the tangents at the two end points. Q or q is the command for a quadratic Bezier. For example

```
   <svg width="800" height="400" style="border:thin blue solid">
      <path d="M 100 100 q 100 100 200 0 z"
        fill="red" stroke="blue" stroke-width="1" />
   </svg>
```

produces



the "M 100 100 q 100 100 200 0” means

move to 100, 100, then the control point is 100 right and 100 down, and the end of the curve is at 200 right and down. It is q not Q so these are relative to the first point.

Cubic Bezier

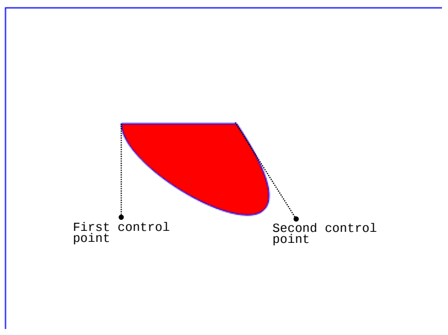
This has 2 control points, one for the start point and one for the end. The d command is

c x1 y1 x2 y2 x y

where the control points are x1,y1 and x2, y2, and x,y is the end. So

```
<path d="M 100 100 C 100 150 300 250 200 100 z"
```

gives

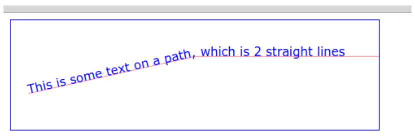


Text on a path

This example defines a path in a <defs> section, then uses it:

```
<body>
   <svg width="12cm" height="3.6cm" viewBox="0 0 1000 300" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <defs>
    <path id="MyPath"
          d="M 50 200
             L 500 100 1000 100
```

```
        " />
  </defs>

  <use href="#MyPath" fill="none" stroke="red"  />
  <text font-family="Verdana" font-size="33" fill="blue" >
    <textPath href="#MyPath">
      This is some text on a path, which is 2 straight lines
    </textPath>
  </text>

  <rect x="1" y="1" width="998" height="298"
        fill="none" stroke="blue" stroke-width="2" />
</svg>
   </svg>
</body>
```
which gives:

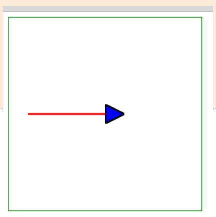


Markers

A marker is a shape which can be drawn at the start, end or middle of a line or path. It is often used to draw a pointer by putting an arrow-head on a line.

The geometry of this is not simple. Here is a version:

```
   <svg width=300 height=300 style="border: thin green solid" viewBox="0 0 100 100">
      <defs>
         <marker id="arrow" refX="0" refY="5" markerWidth="10" markerHeight="10"
         markerUnits="userSpaceOnUse" >
          <path d="M 0 5 L 0 10 10 5 0 0 z" stroke="black" fill="blue"/>
         </marker>
      </defs>

      <line x1='10' y1='50' x2='50' y2='50'
      fill="none" stroke="red" stroke-width="1"
      marker-end="url(#arrow)"
      />
   </svg>
```
The result is:



Some notes on this:

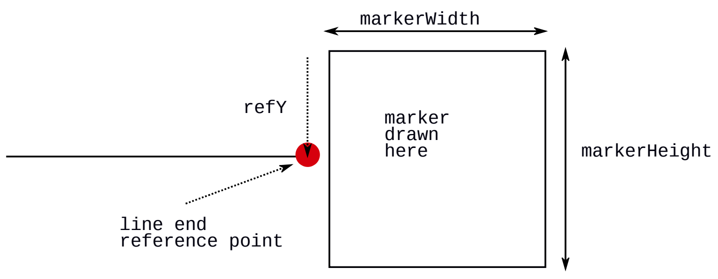
1. The SVG element has viewBox="0 0 100 100". So we take co-ordinates as if the whole SVG was 100 X 100.

2. The structure of the SVG element is
<defs>.. </defs>
<line .. />

In defs we define a marker with id="arrow".

Then tell the line to use this marker with marker-end="url(#arrow)"

3. The marker is drawn relative to some fixed point – in this case, the end of the line. The refX and refY attributes let us adjust the top left of the marker relative to that, like this



4. refX and refY locate the reference point from the top left of the marker. So a positive value for refY puts the marker corner above the reference point.

5. markerWidth and markerHeight size a rectangle, which clips the marker. If something is drawn outside this, it does not appear.

6. markerUnits="userSpaceOnUse" means to use dimensions given. The alternative (and default) is markerUnits="strokeWidth", and this has the effect of giving a larger marker on larger lines.

7. The marker path goes

```
 <path d="M 0 5 L 0 10 10 5 0 0 z"
```

so this is move to 0,5, half-way down the lefthand edge, to match the reference point,

0,10 line to bottom left

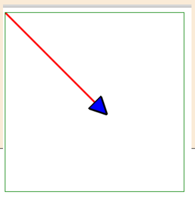
10,5 right edge, halfway up

0,0 top left corner

z close the shape to make a triangle

This is the simple case where the line if horizontal left-right. If the line is not, then orient="auto" rotates the marker to appear in the correct direction:

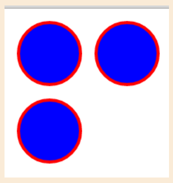
```
   <svg width=300 height=300 style="border: thin green solid" viewBox="0 0 100 100">
      <defs>
         <marker id="arrow" refX="0" refY="5" markerWidth="10" markerHeight="10"
         markerUnits="userSpaceOnUse" orient="auto"
         >
          <path d="M 0 5 L 0 10 10 5 0 0 z" stroke="black" fill="blue"/>
         </marker>
      </defs>

      <line x1='0' y1='0' x2='50' y2='50'
      fill="none" stroke="red" stroke-width="1"
      marker-end="url(#arrow)"
      />
   </svg>
```



Groups

The <g> element groups other elements, and attributes assigned in it apply to all:

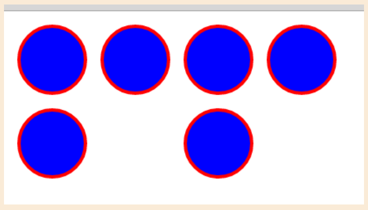
```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

</head>

<body>
   <svg width="200" height="200">
      <g stroke="red" stroke-width="4" fill="blue">
         <circle cx=50 cy=50 r=40 />
         <circle cx=150 cy=50 r=40 />
         <circle cx=50 cy=150 r=40 />
      </g>
   </svg>
</body>

</html>
```



<use>

<use> means to repeat some content with a given id, possibly a group:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

</head>

<body>
   <svg width="400" height="200">
      <g id="3circles" stroke="red" stroke-width="4" fill="blue">
         <circle cx=50 cy=50 r=40 />
         <circle cx=150 cy=50 r=40 />
         <circle cx=50 cy=150 r=40 />
      </g>
      <use href="#3circles" x=200 />
   </svg>
</body>

</html>
```



Scripted SVG

JavaScript – basic technique

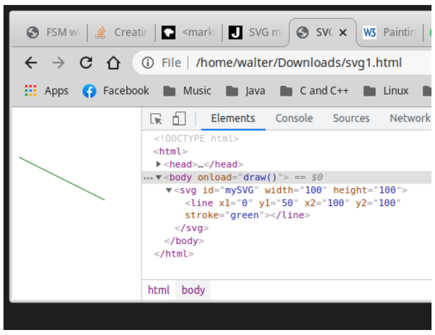
The SVG element is in the DOM, and JavaScript lets us manipulate the DOM using standard techniques.

So we can create suitable shapes, set its attributes, then append it to the DOM in the appropriate place.

For example:

```
<!DOCTYPE html>
<html>

<head>
<title>SVG</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<script>
function draw() {
var svgPart = document.getElementById("mySVG");

var svgns = "http://www.w3.org/2000/svg"; // XML namespace

shape = document.createElementNS(svgns, "line");
shape.setAttributeNS(null, "x1", 0);
shape.setAttributeNS(null, "y1", 50);
shape.setAttributeNS(null, "x2", 100);
shape.setAttributeNS(null, "y2", 100);

shape.setAttributeNS(null, "stroke", "green");
svgPart.appendChild(shape);

}
</script>
</head>

<body onload="draw()">
<svg id="mySVG" width="100" height="100" >
</svg>

</html>
```

which produces:



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
…▼<body onload="draw()"> == $0
   ▼<svg id="mySVG" width="100" height="100">
      <line x1="0" y1="50" x2="100" y2="100"
      stroke="green"></line>
    </svg>
  </body>
</html>
```

This shows the green line drawn, and also what we see when we 'inspect' the html – showing the script produces just the same as if the SVG had been in the html itself.

About namespaces

DOM Level 1 knew nothing about SVG.

From DOM 2, the requirement was to have more than one namespace (set of elements and attributes) in the same document. For example, we might use both MathML and SVG in the same web page. So some way was needed to say which namespace we were using.

So DOM 2 replaced methods like createElement with createElementNS, taking another parameter, saying which namespace was in use. So

```
var svgns = "http://www.w3.org/2000/svg"; // XML namespace
shape = document.createElementNS(svgns, "line");
```

establishes and uses a namespace.

The "http://www.w3.org/2000/svg" has the format of a URL – but it is not used to link to that location. URLs are only used as namespace names because they are unique.

When using the default namespace, the first parameter to setAttributeNS should be null – and so,

```
         shape = document.createElementNS(svgns, "line");
         shape.setAttributeNS(null, "x1", 0);
```

Deleting all SVG elements

To 'clear' all elements from a SVG element, we can say something like

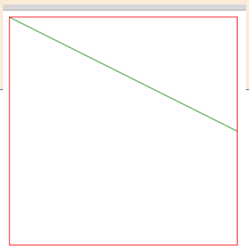
```
   svgns = "http://www.w3.org/2000/svg"; // XML namespace
   svgPart = document.getElementById("mySVG");
   // remove anything already there
   while (svgPart.firstChild) {
      svgPart.removeChild(svgPart.firstChild);
   }
```

Drawing simple primitives

For example:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <script>
      function draw() {
         const SVGWIDTH=1000; // set as viewBox
         const SVGHEIGHT=1000; // everything scaled to this size
         const svgPart = document.getElementById("mySVG");
         svgPart.setAttribute("viewBox", "0 0 "+SVGWIDTH+" "+SVGHEIGHT)
         const svgns = "http://www.w3.org/2000/svg"; // XML namespace
         shape = document.createElementNS(svgns, "line");
         shape.setAttributeNS(null, "x1", 0); // top corner
         shape.setAttributeNS(null, "y1", 0);
         shape.setAttributeNS(null, "x2", SVGWIDTH); // halfway down right edge
         shape.setAttributeNS(null, "y2", SVGHEIGHT/2);

         shape.setAttributeNS(null, "stroke", "green");
         shape.setAttributeNS(null, "stroke-width", SVGWIDTH/300);
         svgPart.appendChild(shape);
      }
   </script>
</head>

<body onload="draw()">

   <svg id="mySVG" width="300" height="300" style="border: thin red solid">
      <!--actual size is 300 X 300 pixels
      red border so we can see it-->
   </svg>

</html>
```

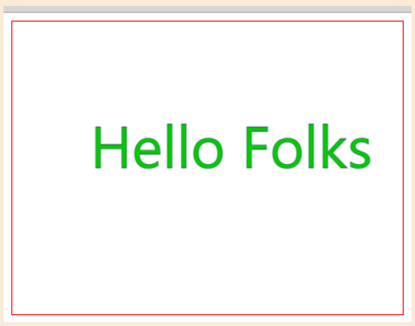
produces:



Text

We can use CSS to set styles:

```
<!DOCTYPE html>
<html>

<head>
   <title>SVG</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <style>
      text.myText {
         stroke: #000066;
         fill: #00cc00;
         font-size: 200px;
         font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

      }
   </style>
   <script>
      function draw() {
         const SVGWIDTH = 1000; // set as viewBox
         const SVGHEIGHT = 1000; // everything scaled to this size
         const svgPart = document.getElementById("mySVG");
         svgPart.setAttribute("viewBox", "0 0 " + SVGWIDTH + " " + SVGHEIGHT)

         const svgns = "http://www.w3.org/2000/svg"; // XML namespace
         shape = document.createElementNS(svgns, "text");
         shape.setAttributeNS(null, "x", SVGWIDTH / 10); // top corner
         shape.setAttributeNS(null, "y", SVGHEIGHT / 2);
         shape.setAttributeNS(null, "class", "myText");
         shape.innerHTML = "Hello Folks";

         svgPart.appendChild(shape);
      }
   </script>
</head>

<body onload="draw()">

   <svg id="mySVG" width="400" height="300" style="border: thin red solid" version="1.1">
   </svg>

</html>
```


Hello Folks