## SVG (Scalable Vector Graphics) from examples

### Introduction:

The first and most important thing to know about SVG is that it isn't a proprietary format. On the contrary, it's an XML language that describes two-dimensional graphics. SVG is an open standard, proposed by the W3C:

SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects.

SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting.

### What does this document explain?

This document explains the SVG basics and JavaScript to smooth applications. It is a step by step process, at the beginning I will explain small SVG examples. You can test these example by extracting the SVG content from this document and paste it on any text editor (like Notepad) and save it with ‘.svg’ extension. Then you ready to execute the application, and execute it with an Internet browser, make sure your Internet browser has SVG plug-in.

### Initialization

First describe to initialize an SVG file and at the same time it describes to set the view port. We specify that in the <svg> tag below. You should always include <title> and <desc> elements. SVG display programs use the title to display a tooltip, and the description is useful for search engines. Additionally your document will be more easily accessible to visually impaired users.

In this example the view port is 12cm by 9cm.

*<?xml version="1.0"?>*

*<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">*

*<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="12cm" height="9cm" viewBox="0 0 400 300">*

*<title>The first SVG example</title>*

*<desc>This is a word for searching</desc>*

*<!-- graphic specifications go here -->*

*</svg>*

When specifying width and height, you are actually establishing a **viewport** in which your drawing will be displayed. You may use em, ex, px, pt, pc, cm, mm, or in to specify dimensions. If you don't use a measurement unit, then the numbers are assumed to be pixels (px) (default).

**Unit Meaning**

**em** - a unit equal to the current font size

**ex** - the *x-height*, usually the height of a lower case letter x

**px** - pixels (the default)

**pt** - points (one point = 1/72 inch)

**pc** - picas (one pica = 1/6 inch)

**cm** - centimeters

**mm** - millimeters

**in** – inches

All numbers used to measure coordinates are assumed to be the same unit-type used to establish width and height. In our example, we'll set the width="12cm" and height="9cm". In order to avoid having to use decimals for all of the coordinates, we will establish a viewBox. A viewBox sets up a user coordinate system which is mapped into the viewport bounds, stretching or shrinking a graphic if the viewBox and viewport aren't proportional. This stretching and shrinking happens only if the preserveAspectRatio is set to none (which is not the default). In the following specification, we set up a system with ten "units" per centimeter.

*<svg version="1.1" xmlns="http://www.w3.org/2000/svg" id="body" width="12cm" height="9cm" viewBox="0 0 120 90" >*

or

*<svg id="body" width="12cm" height="9cm" viewBox="0 0 240 180" version="1.1" xmlns="http://www.w3.org/2000/svg">*

etc…

1. *width="12cm" height="9cm" viewBox="0 0 240 180"*

In which,

Width of a drawing pixel = 12 cm / 240 = 0.05cm

Height of a drawing pixel = 9 cm / 180 = 0.05cm

1. *width="12cm" height="9cm" viewBox="0 0 6 3"*

In which,

Width of a drawing pixel = 12 cm / 6 = 2cm

Height of a drawing pixel = 9 cm / 3 = 3cm

### Adding Graphic Elements

Let's start off by adding the red-bordered rectangle with the light blue interior.

*<svg version="1.1" xmlns="http://www.w3.org/2000/svg" id="body" width="12cm" height="9cm"*

*viewBox="0 0 240 180" >*

*<title>Example</title>*

*<desc>*

*Rectangle with red border and light blue interior.*

*</desc>*

*<rect x="10" y="20" width="150" height="70" fill="#eeeeff" stroke="red" stroke-width="1" />*

*</svg>*

You specify a rectangle by giving the x and y coordinates of its upper left corner and its width and height. When specifying coordinates, the positive *x* direction is to the right, and the positive *y* direction is downwards. In this case, we've also set the fill color, stroke (line) color, and stroke-width as separate attributes.

**SVG shapes (main attributes only)**

*<circle cx="70" cy="100" r="50" />*

*<rect x="150" y="50" rx="20" ry="20" width="135" height="100" />*

*<line x1="325" y1="150" x2="375" y2="50" />*

*<polyline points="50, 250 75, 350 100, 250 125, 350 150, 250 175, 350" />*

*<polygon points=" 250, 250 297, 284 279, 340 220, 340 202, 284" />*

*<ellipse cx="400" cy="300" rx="72" ry="50" />*

*<line x1="20" y1="100" x2="100" y2="20"*

*stroke="black" stroke-width="2"/>*

**Command Meaning Parameters**

**Line** Renders a line between two points x1 and y1 define first point

X2 and y2 define second point

**Polyline** Renders a sequence of lines between points points defines a sequence of x,y

Coordinates

**polygon** Renders an area defined by a sequence points defines a sequence

of lines of x,y coordinates

**rect** Renders a rectangular area width and height define size of rectangle

x and y define top left corner

rx and ry define the radii of the elliptic arc that rounds the corners

**circle** Renders a circle cx and cy define the centre

r defines the radius

**ellipse** Renders an ellipse cx and cy define the centre

rx and ry define the two radii

**SVG text element**

SVG text element has few attributes more sample format as follows

<text x="0" y="15" fill="red">University of Peradeniya</text>

<text x="20" y="25" fill="blue" transform="rotate(90, 100, 100)">Rotated Text</text>

*Advanced Settings … try*

*<text x="50%" y="50%," dx="0,0,0,0,20" dy="0,0,0,0,8" rotate="20,30,40,50,0,0,-10" textLength="..." lengthAdjust="..." font-size="30" text-anchor="middle" >Hello World</text>*

|  |  |
| --- | --- |
| **x** | A list of x-axis positions. The nth x-axis position is given to the nth character in the text. If there are additional characters after the positions run out they are placed after the last character. (0 default) |
| **y** | A list of y-axis positions. See 'x' description. (0 default) |
| **dx** | A list of lengths which moves the characters relative to the absolute position of the last glyph drawn. (see x) |
| **dy** | A list of lengths which moves the characters relative to the absolute position of the last glyph drawn. (see x) |
| **rotate** | A list of rotations. The nth rotation is performed on the nth character. Additional characters are **NOT** given the last rotation value. |
| **textLength** | A target length for the text that the SVG viewer will attempt to display the text between by adjusting the spacing and/or the glyphs. (default: The text's normal length) |
| **lengthAdjust** | Tells the viewer what to adjust to try to accomplish rendering the text if the length is specified. The two values are 'spacing' and 'spacingAndGlyphs'. |

**Path Element**

The path element is the single most useful tag for creating professional looking SVG documents and it's the most difficult tag to hand-code in SVG. It is quite hard to visualize anything but the most simple of paths.

|  |  |
| --- | --- |
| **Attribute** | **Explanation** |
| **d** | A set of commands which define the path. (See below) |
| **pathLength** | If present then the path will be scaled so that the computed path length of the points equals this value. A negative value is an error. (Note: Unsupported by Adobe's plugin) |
| **transform** | A list of transformations. |

**Path Data**

The path is defined in the 'd' attribute of the 'path' tag by a string of white space separated commands and coordinates.

Path commands are case-sensitive, an uppercase command's points use absolute positioning and a lowercase command's points are relative the last point. The one exception to this is the first point always uses absolute positioning.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Parameters** | **Repeat** | **Explanation** |
| **M** | x,y | Yes | **moveto**: Moves the pen to a new location. No line is drawn. All path data must begin with a 'moveto' command. |
| **Line Commands** | | | |
| **L** | x,y | Yes | **lineto**: Draws a line from the current point to the point (x,y). |
| **H** | x | Yes | **horizontal lineto**: Draws a horizontal line from the current point to x. |
| **V** | y | Yes | **vertical lineto**: Draws a vertical line from the current point to y. |
| **Cubic Bezier Curve Commands** | | | |
| **C** | x1 y1 x2 y2 x y | Yes | **curveto**: Draw a cubic bezier curve to the point (x,y) where the points (x1,y1) and (x2,y2) are the start and end control points, respectively. |
| **S** | x2 y2 x y | Yes | **shorthand/smooth curveto**: Draw a curve to the point (x,y) where the point (x2,y2) is the end control point and the start control point is the reflection of the last point's end control point. |
| **Quadratic Bezier Curve Commands** | | | |
| **Q** | x1 y1 x y | Yes | **Quadratic Bezier curveto**: Draw a quadratic bezier between the last point and point (x,y) using the point (x1,y1) as the control point. |
| **T** | x y | Yes | **Shorthand/smooth quadratic Bezier curveto**: Draw a quadratic bezier between the last point and point (x,y) using the reflection of the last control point as the control point. |
| **Elliptical Arc Curve Commands** | | | |
| **A** | \*\* | Yes | **elliptical arc**: Draws and arc from the current point to x, y. The actual scale factor and position of the arc needed to bridge the two points is computed by the SVG viewer. |
| **End Path Commands** | | | |
| **z** | - | No | **closepath**: Closes the path. A line is drawn from the last point to the first. |

Eg.

<path d="M100,10 L100,10 40,180 190,60 10,60 160,180 z" stroke="blue" fill="darkblue" stroke-width="4" />

<path d="M40,140 L40,100 10,100 C10,10 90,10 90,100 L60,100 60,140 M140,50 C70,180 195,180 190,100 " stroke="darkgreen" stroke-width="4" fill="#f00" />

### Transformations

Let's add the gray drop-shadow rectangle. We want it three units to the right and below the red rectangle. Rather than do the addition of the x and y coordinates ourselves, we can specify that SVG should perform a transform on the graphic; it should translate the rectangle by three units in both dimensions.

*<svg version="1.1" xmlns="http://www.w3.org/2000/svg" id="body" width="12cm" height="9cm"*

*viewBox="0 0 240 180" >*

*<title>Example</title>*

*<desc>*

*Rectangle with red border and light blue interior, with (intended) gray shadow rectangle.*

*</desc>*

*<rect x="10" y="20" width="150" height="70" fill="#eeeeff" stroke="red" stroke-width="1" />*

*<rect x="10" y="20" width="150" height="70" transform="translate(3, 3)"*

*fill="#999999" stroke="#999999" stroke-width="1" />*

*</svg>*

**About SVG transform**

All SVG graphic elements have a "transform" attribute. This attribute can be set and changed to various values in order to move and distort the element. A transform attribute takes a list of transformations, which are applied in reverse order. The available transformations include:

**matrix(<a> <b> <c> <d> <e> <f>)** defines a 2x3 matrix (or a six-number array) that is multiplied to the element's coordinates. See the comments for the other transformations below for the meaning of each member of this matrix.

**translate(<tx> [<ty>])** will move the element <tx> units along the x-axis and <ty> units along the y-axis. In the full six-number array, <tx> and <ty> are the last 2 numbers of the array.

**scale(<sx> [<sy>])** will scale the element by factor of <sx> and <sy> from the centerpoint (0,0). If <sy> is not given, it is assumed to be the same as <sx>. For example, if the value of <sx> is "0.5," the element will be scaled to 50% of the original width. In the full six-number array, <sx> and <sy> are the first and fourth numbers of the array.

To scale using a centerpoint other than (0,0), you must translate the element so that the centerpoint becomes (0,0), perform the scale, then translate the element back to its original location. For example, to scale an element by 50% centered on (150,100), you must set the transform attribute to "translate(150 100) scale(.5) translate(-150 -100)" (note the reverse order).

**rotate(<rotate-angle>)** will rotate the element <rotate-angle> degrees around the centerpoint (0,0). For example, if the value of <rotate-angle> is "45," the element will be rotated 45 degrees clockwise. As in the scale transformation, you can rotate using other centerpoints by enclosing the rotate inside translate transformations.

**skewX(<angle>)** will skew the x-axis of the element <angle> degrees from the centerpoint (0,0). In the full six-number array, <angle>, expressed in radians, is the third number of the array.

**skewY(<angle>)** will skew the y-axis of the element <angle> degrees from the centerpoint (0,0). In the full six-number array, <angle>, expressed in radians, is the second number of the array.

**Animations**

**Type 1: Morphology**

  <rect x="50" y="0" fill="#f00" width="10" height="100">

       <animate attributeName="height" from="0" to="100" dur="0.5s" fill="freeze" />

    </rect>

     <rect x="70" y="0" fill="#0f0" width="10" height="100">

       <animate attributeName="height" from="0" to="100" dur="1.0s"  fill="freeze" />

    </rect>

    <rect x="90" y="0" fill="#f00" width="10" height="100">

       <animate attributeName="height" from="0" to="100" dur="1.0s" repeatCount="indefinite" />

    </rect>

**Type 2: Color**

<circle cx="70" cy="100" r="50" fill="transparent"

        stroke="#800" stroke-width="5">

        <animate

            attributeName="stroke"

            values="#800;#f00;#080;#800"

            dur="0.8s"

            repeatCount="indefinite"/>

    </circle>

<path

fill="#FFFFFF" stroke="#000"

d="M160,100 Q200,100,200,300 Q100,300,100,200 Z">

<animate

attributeName="fill"

values="#800;#f00;#800;#800"

dur="1.2s"

repeatCount="indefinite"/>

</path>

**Groups**

<g fill="red">

        <rect x="50" y="50" width="100" height="100" />

        <rect x="30" y="40" width="50" height="50" />

</g>

**SVG with Scripts**

<svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="200px" height="150px">

<defs>

<script language="Javascript"><![CDATA[

function myAction()

{

alert("alert");

}

]]>

</script>

</defs>

<rect width="100" height="25" x="30" ry="12" y="40" fill="black" onclick="myAction()"/>

</svg>