Create vector graphics in the browser with SVG

A step-by-step guide for incorporating SVG into Web pages and applications

Skill Level: Intermediate

Uche Ogbuji (uche@ogbuji.net)
Principal Consultant
Fourthought Inc.

20 Jun 2006

Learn step-by-step how to incorporate Scalable Vector Graphics (SVG) into Web pages using real browser examples. SVG 1.1, an XML language for describing two-dimensional vector graphics, provides a practical and flexible graphics format in XML, despite the language's verbosity. Several browsers recently completed or announced built-in SVG support.

Section 1. Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this tutorial

Scalable Vector Graphics (SVG) 1.1 is an XML language for describing two-dimensional vector graphics. Developed by the World Wide Web Consortium (W3C), it has the remarkable ambition of providing a practical and flexible graphics format in XML, despite the notorious verbosity of XML. SVG's feature set includes nested transformations, clipping paths, alpha masks, raster filter effects, template objects, and, of course, extensibility. SVG also supports animation, zooming and panning views, a wide variety of graphic primitives, grouping, scripting, hyperlinks, structured metadata, Cascading Style Sheets (CSS), a specialized Document Object Model (DOM) superset, and easy embedding in other XML documents. Overall, SVG has been one of the most widely and warmly embraced XML applications.

You can develop, process, and deploy SVG in many different environments, from

mobile systems such as phones and Personal Digital Assistants (PDAs), to print environments. This tutorial focuses on SVG for Web development, offering step-by-step instruction for Web developers and designers to learn how to use SVG in practical Web sites. The lessons are built around examples that you can view and experiment with in your favorite browser. This tutorial doesn't go into a lot of detail about the SVG language. Instead, it provides a broad enough view of the language to guide you in deploying it on the Web.

Who should take this tutorial?

SVG is a technology positioned for many uses in the Web space. You can use it for presenting simple graphics (as with JPEG) or complex applications (as with Macromedia Flash). As such, it's an important weapon in every Web developer's and designer's arsenal. Programmers should take this tutorial if they deal with any Web applications. Web designers should take this tutorial to learn how to deploy efficient vector graphics on the Web.

Objectives

In this tutorial, you'll learn the basics of SVG in order to publish vector graphics on the Web using SVG. You'll learn how to render such images in a browser either stand-alone or embedded in XHTML.

Prerequisites

This tutorial assumes knowledge of XML, XML namespaces, CSS, and basic XHTML. Even though this tutorial focuses on SVG on the Web, it requires no prior knowledge of SVG and starts with the basics of the language. If you aren't familiar with XML, take the tutorial Introduction to XML. If you need to learn about XML namespaces, read the article Plan to use XML namespaces, Part 1. If you're not familiar with CSS, especially as used with XML, take the tutorial Display XML with Cascading Stylesheets: Use Cascading Stylesheets to display XML, Part 1: Basic techniques to present XML in Web browsers. This tutorial introduces the use of CSS to style XML in browsers. If you aren't familiar with XHTML, a good place to start is XHTML, step-by-step. You should also understand the basic mathematics of the two-dimensional rectilinear coordinate system, also known as the Cartesian coordinate system. You might remember this best from high school mathematics as how to specify points along X and Y axes.

System requirements

I highly recommend that you try out the examples in this tutorial. They only require a Web browser that supports SVG. Firefox 1.5 or later has such support built in, as does Opera 9. Safari has announced support for SVG in coming versions, but for now, the support is only available in nightly development snapshots where you have

no guaranteed stability. Mac OS X users might want to try the Camino Web browser for SVG support. Microsoft® Internet Explorer users will require a plug-in such as the Adobe SVG Viewer. When showing browser output examples, I show screenshots of Firefox 1.5.0.2 on Ubuntu Linux®. Firefox is a popular Web browser available on Microsoft Windows®, Mac OS X, Linux, and other platforms. It is based on Mozilla's rendering engine.

About the examples in this tutorial

This tutorial features many examples of SVG files, either stand-alone or embedded in XHTML. All the files used in this tutorial are in the zip file, x-svggraphics-tutorial-files.zip. In this package, all files start with a prefix indicating what section they're covered in and what order of examples within the section. For example, the names of files from the first example in the third section start with eg_3_1.

Files that end with .svg are stand-alone SVG. Those that end with .xhtml are XHTML. A few files use other extensions such as .css for stand-alone CSS and .xsl for XSLT transform files.

I do take care to further list the example files in each panel and how each relates to the other, so if you follow along with the tutorial, you should be able to locate and experiment with the examples easily enough.

Section 2. Basic stand-alone SVG

You can display SVG in several ways. First of all, it is an image format, just like JPEG or GIF. The most important difference is that it is a vector graphics format, meaning that you express graphics by declaring their basic elements. You define a circle merely by defining the position of the center and its radius. JPEG and GIF are raster formats, which express the properties of each point in the image. Another difference is that SVG is expressed in XML, whereas most graphics formats -- even other vector formats -- are binary formats. One thing that SVG does have in common with other formats is that you can display it as a stand-alone image, by directing the browser right at the URL of an SVG file. I'll explore this usage in the first couple of sections.

In theory, you should be able to use SVG in all the other areas where you can use Web images, such as the browser element background. In practice, however, some limitations exist, depending on your particular browser. You might consult documentation or experiment a bit to learn of any limitations.

You can also embed SVG into XML formats such as XHTML and Mozilla's XML User Interface Language (XUL -- a Mozilla-specific format for controlling the look and feel

of the browser). SVG has its own namespace, which makes it clear to the Web browser where the embedded SVG starts and ends. In this mode, you can use all the other browser facilities such as document CSS, DOM, and events.

Choose which way you'll use SVG depending on how you need it to interact with the other components of your Web page or application.

First SVG diagram

Listing 1 (eg_2_2.svg in the download file) is a complete SVG file.

Listing 1. SVG example with one circle

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <circle cx="3cm" cy="2cm" r="1cm"/>
</svg>
```

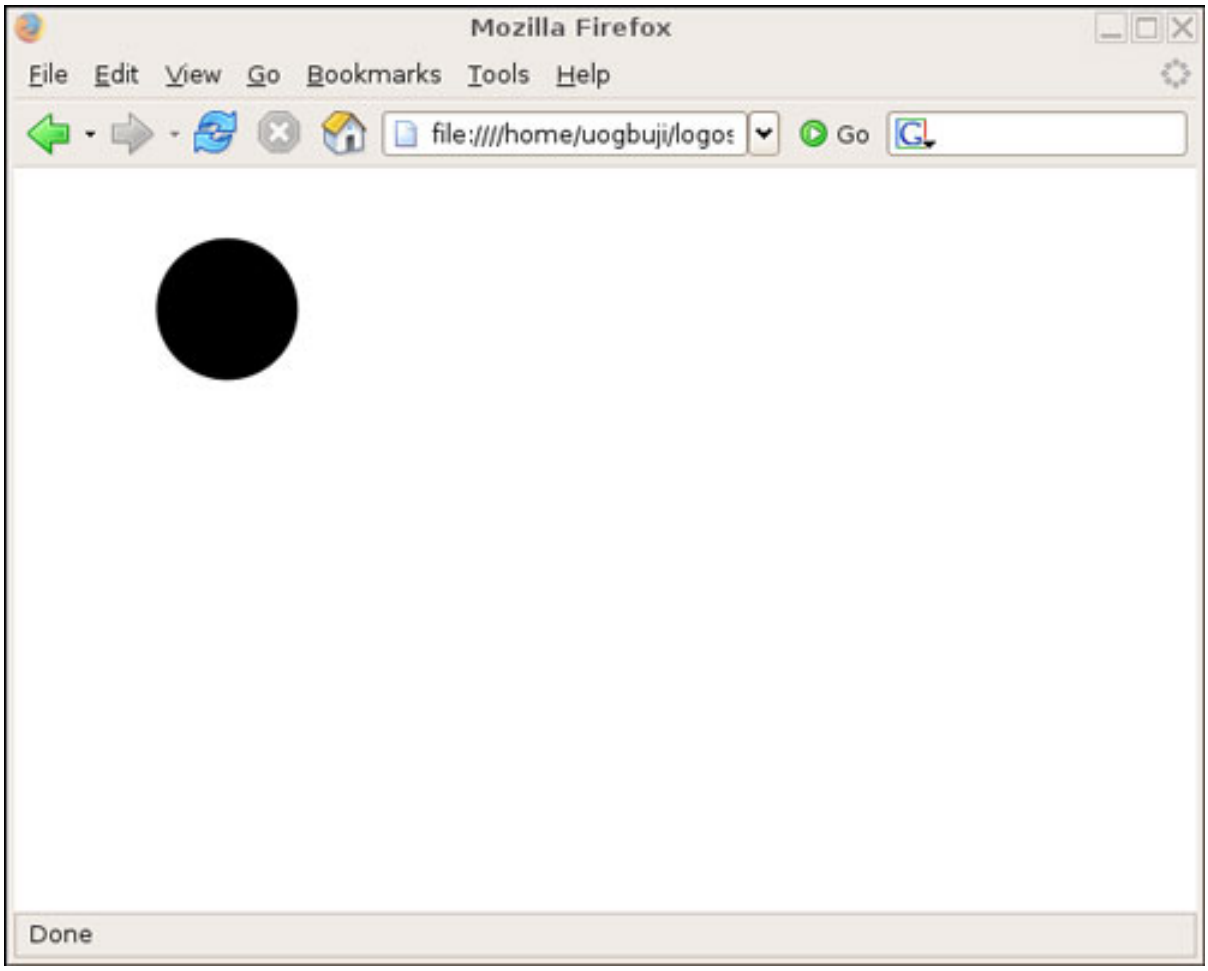
The XML declaration is optional, but I recommend never omitting the XML declaration in stand-alone SVG files. An optional document type declaration exists for stand-alone SVG files, but because of some errors and confusion across SVG versions, the SVG working group recommends that you don't use the document type declaration. Instead, you should be sure to include `version` and `baseProfile` attributes on the document element, `svg`. Some SVG experts disagree with this advice, but I have adopted it in this tutorial. The `version` attribute indicates what version of the SVG specification the file conforms to, and the `baseProfile` attribute indicates the profile. An SVG profile is a subset of the full specification, perhaps for mobile hardware processors that can't handle all the capabilities in the full SVG specification. The attributes in Listing 1 specify that the file is designed for the full SVG 1.1 specification.

The `svg` element and all its children are in the SVG namespace, `http://www.w3.org/2000/svg`, declared as the default namespace so no prefixes are necessary. `circle` is one of the many elements that you can use to define graphics primitives. The attributes `cx` and `cy` define the position of the center of the circle along the X and Y axes. The `r` attribute specifies the radius of the circle. As in many computer graph applications, the use of these axes differs from the traditional mathematical convention. The position 0,0 (the origin) is at the upper-left side of the drawing, which takes up the entire browser display space when viewing stand-alone SVG. Position 2,2 is two pixels lower and to the right of the origin. You can specify units in SVG if you don't want to worry about pixel counts. I've used centimeters as the units for all dimensions in Listing 1. You can use other units such as points, which you're already familiar with to specify font sizes and inches. SVG's CSS is just like the CSS you use on regular Web pages, except that SVG has an additional set of rules you can use.

If you view Listing 1 in the browser, you should see a display similar to Figure 1. Try loading the image, changing these values, and refreshing the browser view to see

the effect on the display. Try setting the center of the circle to the origin (position 0,0), so you can see how the browser truncates or *clips* the resulting image. You can even try to set a negative X or Y component to the location, which will end up clipping even more of the display.

Figure 1. Browser display of Listing 1



A matter of style

XML attributes control the dimensions of shapes, but matters such as the color and the border of shapes are controlled by style instructions expressed in CSS rules. Listing 2 (eg_2_3.svg in the download file) is an SVG file that defines three circles with the same radius but different appearances.

Listing 2. SVG example with three circles

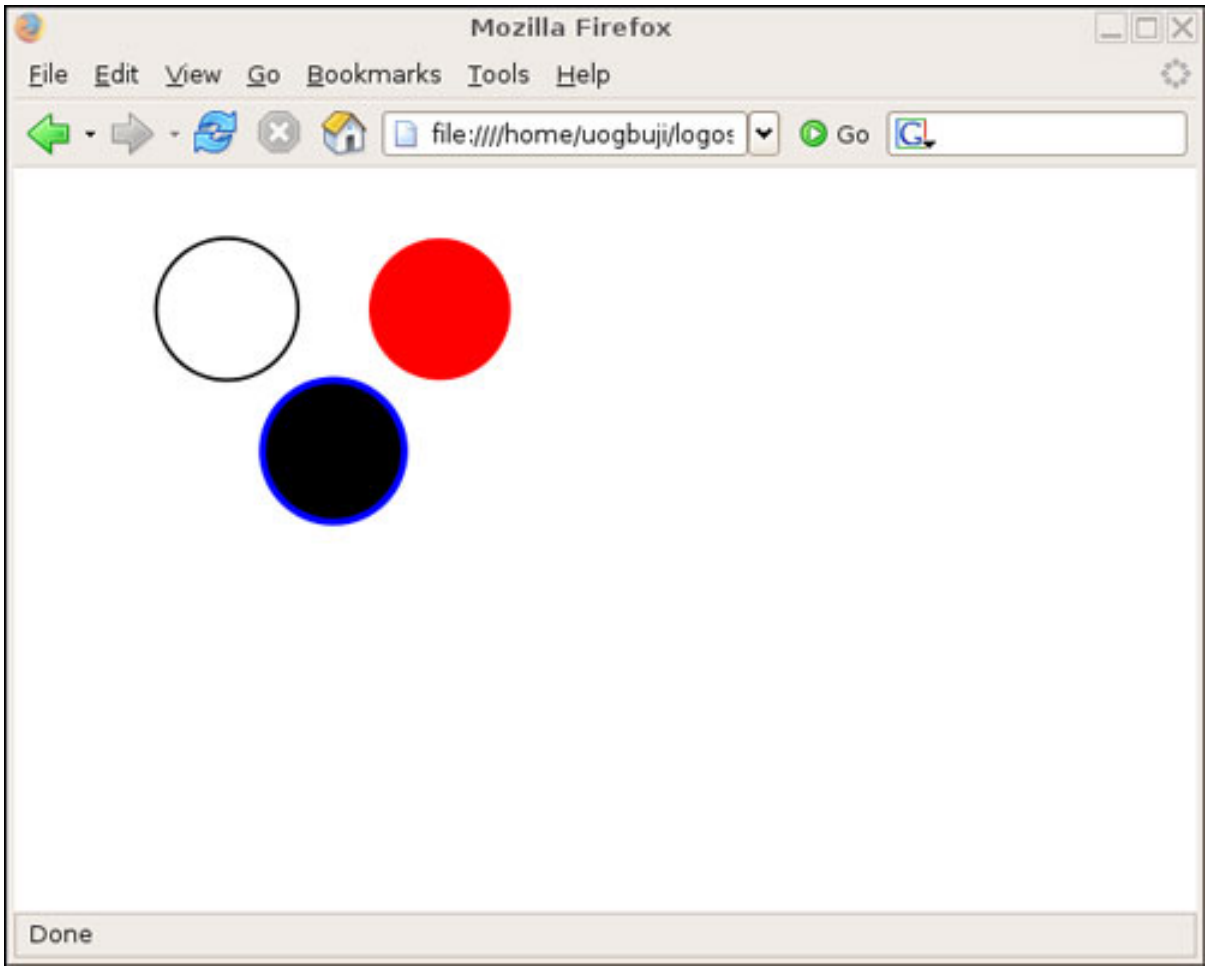
```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <circle cx="3cm" cy="2cm" r="1cm"
    style="fill: none; stroke: black; stroke-width: 0.05cm"/>
  <circle cx="6cm" cy="2cm" r="1cm"
    style="fill: red; stroke: none;"/>
  <circle cx="4.5cm" cy="4cm" r="1cm"
    style="stroke: #0000ff; stroke-width: 0.1cm"/>
```

```
</svg>
```

As you can see, one `circle` element exists for each object. Each has a different center location, so you can see each of them, but they all have the same radius. They all look different because of the differences in the `style` attributes, which specify CSS style rules for the object. One circle is rendered with a thin black stroke and no fill. The stroke is the outline of the circle shape; the fill is the way the body of the circle is painted. Notice that the circle in Figure 1 is filled in black. This is the default if no style is specified. In Figure 2, another circle is filled in red with no stroke. The third one has a stroke color specified as `#0000ff`. This is standard CSS code for a color with no red or green components and a maximum blue component. This is the equivalent to specifying `blue`.

If you view Listing 2 in the browser, you should see a display similar to Figure 2.

Figure 2. Browser display of Listing 2



Note that `style` attributes are not the best way to apply CSS to SVG elements. If possible, it's better to use separate stylesheets (introduced in a later subsection), which attach rules to SVG elements through selectors based on element name, `class` and `id` attributes, and so forth. I demonstrate `style` attributes in this tutorial because they're extremely common in real-world SVG, so you should at least be familiar with them.

Working with groups

In the examples so far, I've thrown the SVG object elements right into the document at the top level, but in most real-world uses of SVG, you'll find them at least bundled into groups. Groups are a way of organizing sets of objects. One useful property of a group is that you can apply certain characteristics to all objects within a group. This is similar to XHTML `divs`, which you often use to control cascading style for contained objects. Listing 3 (eg_2_4.svg in the download file) is an SVG file that defines three circles within a group, using some degree of common style.

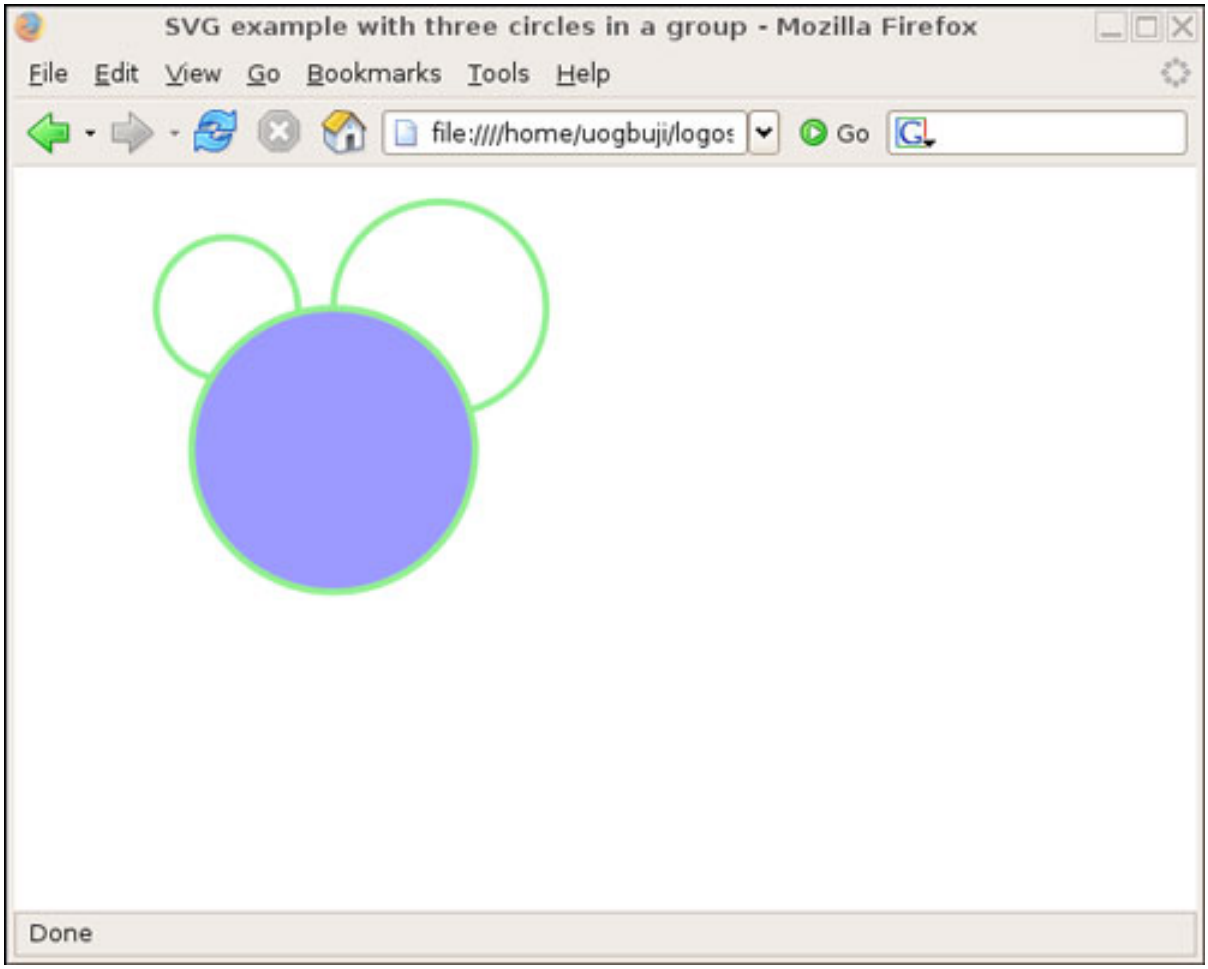
Listing 3. SVG example with three circles in a group

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>SVG example with three circles in a group</title>
  <g style="fill: none; stroke: lightgreen; stroke-width:0.1cm">
    <circle cx="3cm" cy="2cm" r="1cm"/>
    <circle cx="6cm" cy="2cm" r="1.5cm"/>
    <circle cx="4.5cm" cy="4cm" r="2cm" style="fill: #9999ff;"/>
  </g>
</svg>
```

The group is defined by the g element, which encloses members of the group. Characteristics specified on the g element are inherited by all members of the group. In this case, a style is specified for the stroke and fill. Any group member can override inherited characteristics. In this case, the third circle has a specified fill that overrides the group value.

Listing 3 also introduces the `title` element, which you can use on almost any SVG element. You should always have a title for a stand-alone SVG document. The `desc` element can also occur within almost every SVG element as a description passage. The content of these elements doesn't usually appear within the image, but it's usually available in some way through a viewer. Top-level titles appear in the title bar of SVG loaded in Firefox, much as the title of HTML documents do. If you view Listing 3 in the browser, you should see a display similar to Figure 3. Don't forget to check out the title bar.

Figure 3. Browser display of Listing 3



Using overlap

Notice how in Figure 3, the third circle overlaps the first two and obscures them. In SVG, objects specified later in the document order obscure objects specified earlier. If you imagine laying down the shapes as paper cutouts, the XML document order is analogous to the order in which you drop the cutouts onto the surface. You can use this fact to create some simple effects. For example, Listing 4 (eg_2_5.svg in the download file) is an SVG file that draws a crescent moon.

Listing 4. SVG example creating a crescent-moon effect

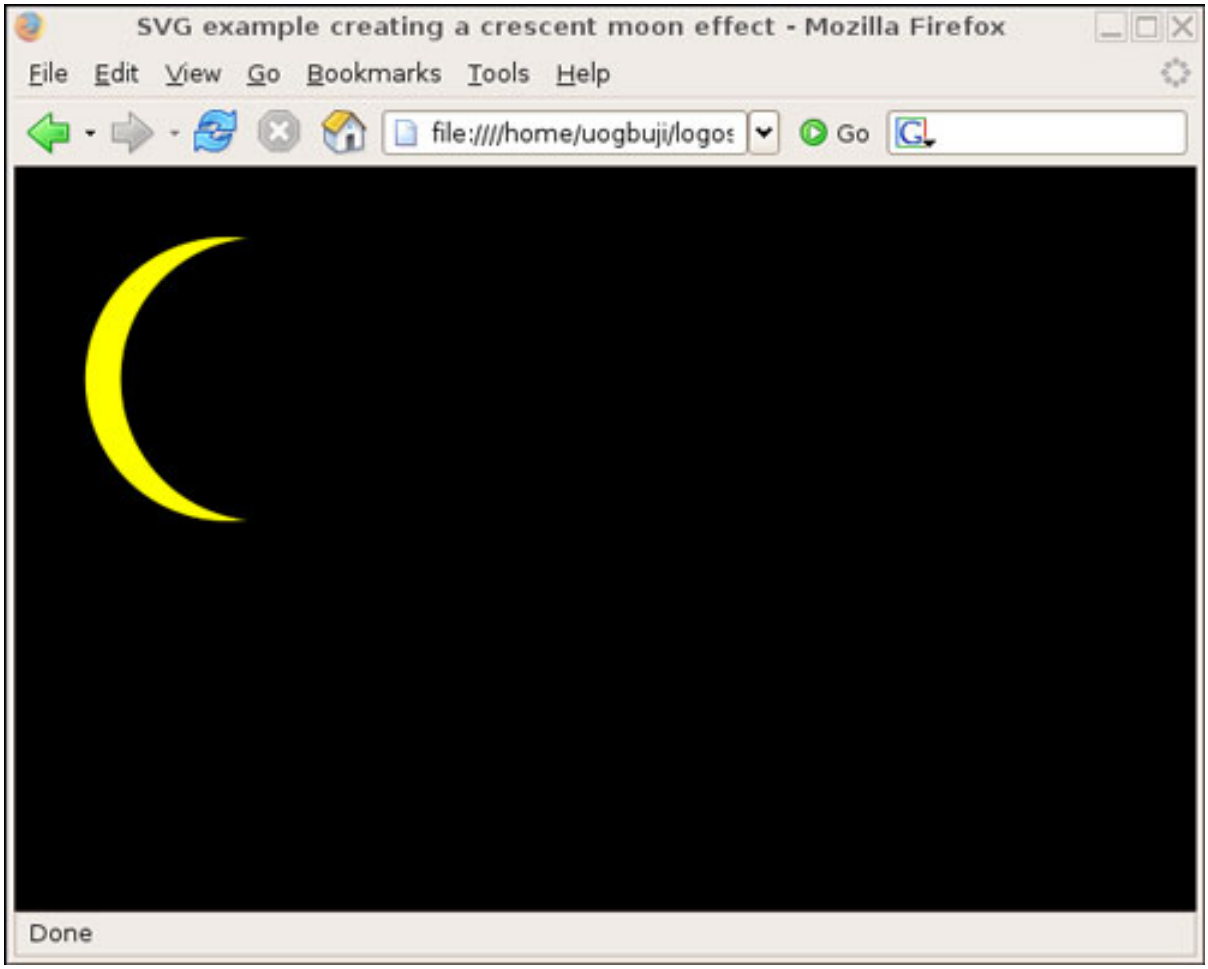
```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>SVG example creating a crescent-moon effect</title>
  <defs>
    <style type="text/css"><![CDATA[
      svg { background-color: black; }
      #moon { fill: yellow; }
      .shadow { fill: black; }
    ]]></style>
  </defs>
  <circle id="moon" cx="100" cy="100" r="50"/>
  <circle class="shadow" cx="120" cy="100" r="50"/>
</svg>
```

First, the browser draws a circle for the body of the moon, and then another circle, shifted a bit to the right, which covers the first and serves as the shadow, leading to the crescent effect. Interestingly, this listing doesn't use `cm` units. In fact, I don't specify units at all, defaulting to one pixel per unit. I use the `defs` element, which is a wrapper section for elements that aren't graphics objects themselves, but which provide information for reference by other elements. This includes global `style` elements. Such an element can provide a stylesheet inline or by reference to an external file. In this case, I include an inline stylesheet with the following instructions:

- Set the background of the entire document to black.
- Set the fill of the element with ID `moon` to yellow.
- Set the fill of any element with class `shadow` to black.

As I alluded to earlier, this is the proper way to specify style in SVG. The presentation details of the objects are now properly separated from the substance of the objects. Remember that more than one element can have a given class (even elements of different type), but only one can have a particular ID. If you view Listing 4 in the browser, you should see a display similar to Figure 4. You can see the crescent effect of the black circle drawn on the offset yellow.

Figure 4. Browser display of Listing 4



Transforms

SVG allows you to declare transforms on objects. In this case, the browser draws the object normally and then changes it in some way, such as by moving it, rotating it, resizing it, or otherwise distorting it. Listing 5 (eg_2_6.svg in the download file) draws a crescent moon to match Listing 4, but it uses a transform to offset the second circle.

Listing 5. SVG example creating a crescent moon using a transform

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>SVG example creating a crescent moon using a transform</title>
  <defs>
    <style type="text/css"><![CDATA[
      svg { background-color: black; }
      #moon { fill: yellow; }
      .shadow { fill: black; }
    ]]></style>
  </defs>
  <g>
    <title>Crescent</title>
    <circle id="moon" cx="100" cy="100" r="50"/>
    <circle class="shadow" cx="100" cy="100" r="50" transform="translate(20, 0)"/>
  </g>
```

```
</svg>
```

First, notice that I've put both circles in a group, and I've given that group a title. This underscores the fact that the group acts as a virtual, composite object. I've declared both circles with the same original positions and radii, but the second one has the attribute `transform="translate(20, 0)`, which shifts the object 20 units along the X axis before rendering it. You cannot use units such as `cm` with the numbers expressed in transforms. You'll encounter some other sorts of transforms later in this tutorial. If you view Listing 5 in the browser, you should see the same display as for Listing 4.

Manipulating opacity

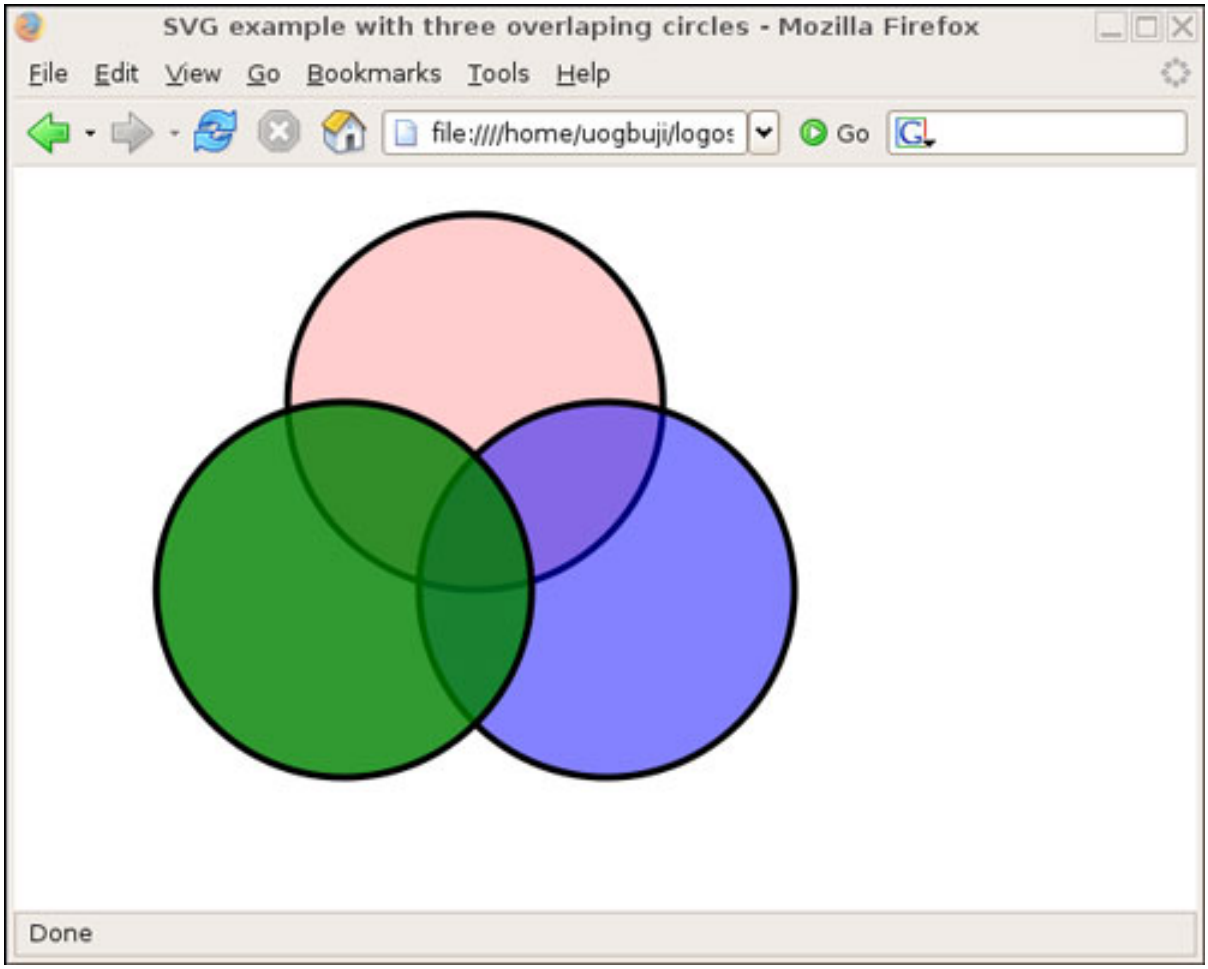
So far, the objects sharing a space have completely obscured part of other objects, because SVG fills shapes with 100 percent opacity by default. You can make objects more transparent by modifying the opacity. Listing 6 (eg_2_7.svg in the download file) draws three overlapping circles of varying color and transparency.

Listing 6. SVG example creating three overlapping circles

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>SVG example with three overlapping circles</title>
  <defs>
    <style type="text/css"><![CDATA[
      circle { stroke:black; stroke-width:0.1cm; }
      #top { fill:red; fill-opacity:0.2; }
      #left { fill:green; fill-opacity:0.8; }
      #right { fill:blue; fill-opacity:0.5; }
    ]]></style>
  </defs>
  <g>
    <title>Translucent circles</title>
    <circle id="top" cx="6.5cm" cy="2cm" r="100"
        transform="translate(0,50)" />
    <circle id="right" cx="6.5cm" cy="2cm" r="100"
        transform="translate(70,150)" />
    <circle id="left" cx="6.5cm" cy="2cm" r="100"
        transform="translate(-70,150)"/>
  </g>
</svg>
```

The `fill-opacity` CSS property controls transparency. It's a value from 0 for fully transparent to 1 for fully opaque. I use one stylesheet rule to set the stroke properties for all circles, and then I use one rule for each circle to set its unique color and opacity. You might notice that the opacity also affects the appearance of color. The red circle is the most translucent, so it's more of a very light salmon color. The green circle is the most opaque, and thus is the deepest color. Notice that once again, I use transforms to offset the circles. One more interesting point is that I use a combination of units -- most are in `cm`, but the circles' radii don't have units specified and thus default to pixels. If you view Listing 6 in the browser, you should see a display similar to Figure 5.

Figure 5. Browser display of Listing 6



Section 3. More complex drawings

Circles are the only shapes that I've demonstrated, but SVG supports many more. In this section, you'll learn the building blocks for more ambitious drawings.

Squares and polygons

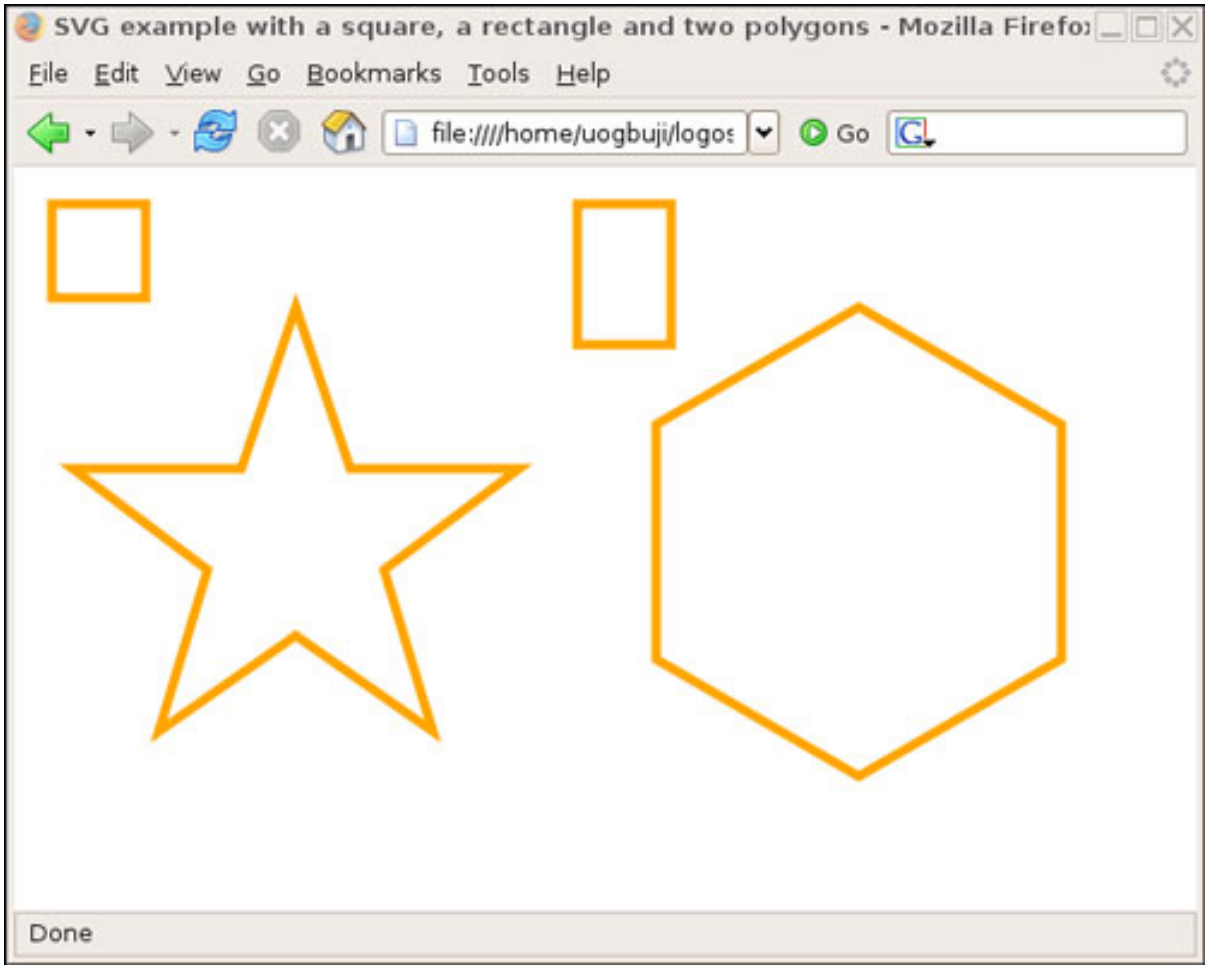
So far, I've stuck with circles, but you can use many other graphics primitives in SVG, including those for squares and other polygons. You can create a square by declaring its position and size. Polygons are a bit trickier. You define an array of points by their locations, and the result is a closed polygon with those points. Listing 7 (eg_3_1.svg in the download file) is an SVG file that defines a square, a rectangle, and a couple of polygons.

Listing 7. SVG example with a square, a rectangle, and two polygons

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>SVG example with a square, a rectangle, and two
polygons</title>
  <defs>
    <style type="text/css"><![CDATA[
      .polygons { fill: none; stroke: orange; stroke-width: 5; }
    ]]></style>
  </defs>
  <g class="polygons">
    <title>Polygons</title>
    <rect x="20" y="20" width="50" height="50"/>
    <rect x="300" y="20" width="50" height="75"/>
    <polygon points="150,75  179,161 269,161 197,215
                     223,301 150,250  77,301 103,215
                      31,161 121,161"/>
    <polygon points="450,75  558,137.5 558,262.5
                     450,325 342,262.6 342,137.5"/>
  </g>
</svg>
```

Of course, a square is just a special case of a rectangle, and a rectangle is just a special case of a polygon. You could use the `polygon` element to make all the shapes in Listing 7, but SVG provides the `rect` element as a convenience. You just specify the location (as `x` and `y` attributes) of the top left-hand corner and the size. If the `height` and `width` are the same, then you have a square. The `polygon` element takes a different approach. You specify the coordinates of each vertex in the polygon in the `points` attribute. All the vertices are crammed into this attribute as a series of X and Y values for the location of each. This is not an ideal approach to XML design, but it is necessitated by the need for SVG to be reasonably compact. If you view Listing 7 in the browser, you should see a display similar to Figure 6.

Figure 6. Browser display of Listing 7



Lines and ellipses

Two more of the many simple object types you can express in SVG are lines and ellipses. You express lines as locations of a start and end point. You express ellipses in much the same way as circles, except that the radius has both X and Y axis components. Listing 8 (eg_3_2.svg in the download file) is an SVG file that defines a pair of lines and a pair of ellipses.

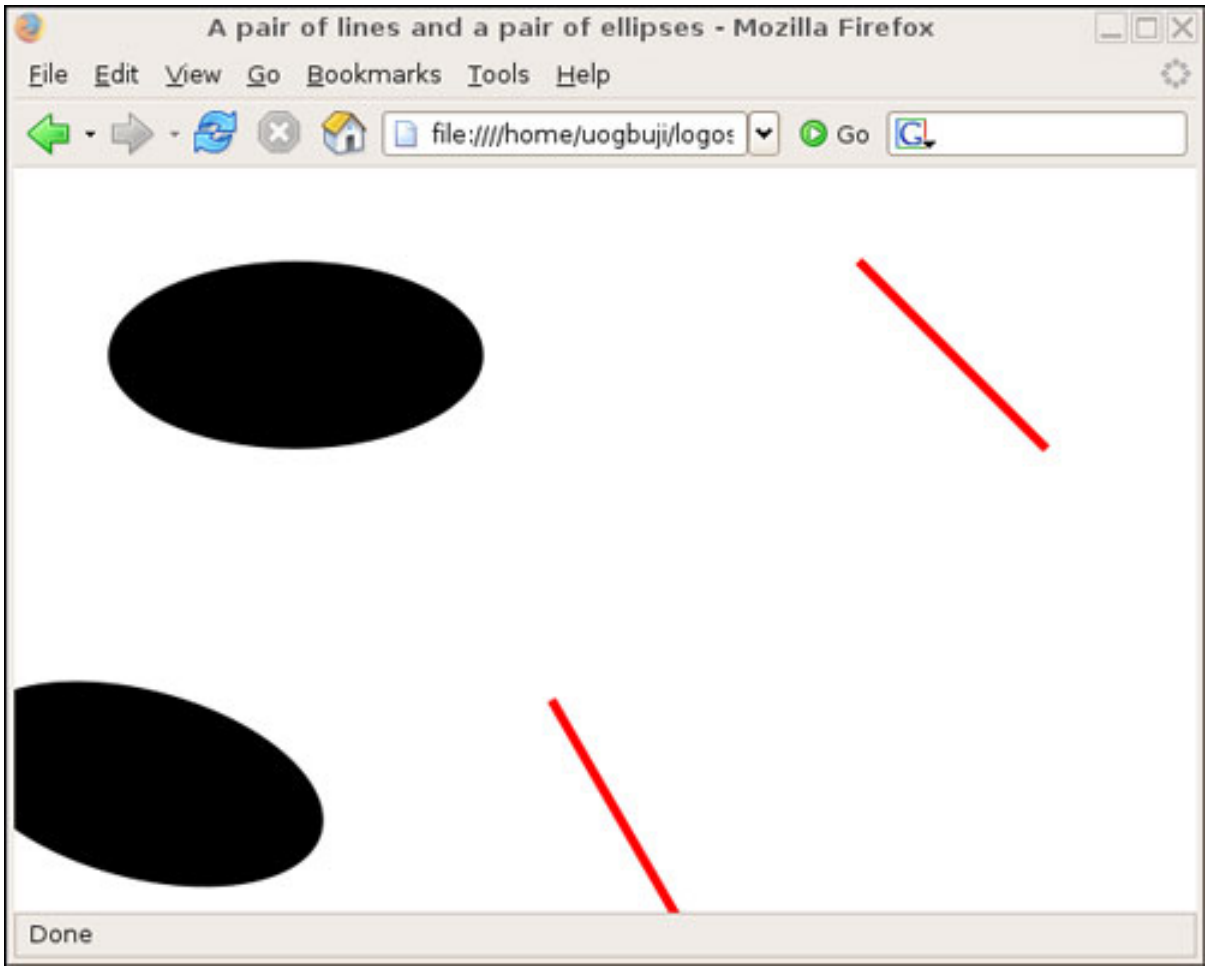
Listing 8. SVG example with a pair of lines and a pair of ellipses

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg">
  <title>A pair of lines and a pair of ellipses</title>
  <defs>
    <style type="text/css"><![CDATA[
      line {
        stroke: red;
        stroke-width: 5;
      }
    ]]></style>
  </defs>
  <g>
    <title>Plain shapes</title>
    <ellipse cx="150" cy="100" rx="100" ry="50"/>
    <line x1="450" y1="50" x2="550" y2="150"/>
```

```
  </g>
  <g>
    <title>Rotated shapes</title>
    <ellipse cx="150" cy="300" rx="100" ry="50"
        transform="rotate(20)"/>
    <line x1="350" y1="200" x2="450" y2="300"
        transform="rotate(20)"/>
  </g>
</svg>
```

The `ellipse` elements are similar to `circle`, except that rather than a single attribute for radius, there are `rx` and `ry`. If both these values are the same, the result is a circle. The `line` elements have `x1` and `y1` to define the starting point for drawing the line, and `x2` and `y2` to define the ending. By default, you have to specify stroke properties in order to see a line, but I take a different approach to doing so in this example. Once again, you have a top-level stylesheet, which defines rules for `line` elements. You apply these rules to all lines in the drawing, although you can override them, of course. I also introduce a new transform in this example. The second set of shapes are rotated 20 degrees. If you view Listing 8 in the browser, you should see a display similar to Figure 7.

Figure 7. Browser display of Listing 8



Notice how the second ellipse and line aren't just rotated about the center of each, but are also rotated with respect to their position within the entire canvas. This is

important to understand about SVG transforms. They are applied to all aspects of positioning and size of the object.

Text

In SVG, you can use all sorts of objects to work up an image, including text and other images. Listing 9 (eg_3_3.svg in the download file) is an SVG file that uses both, throwing in a few neat tricks for good measure.

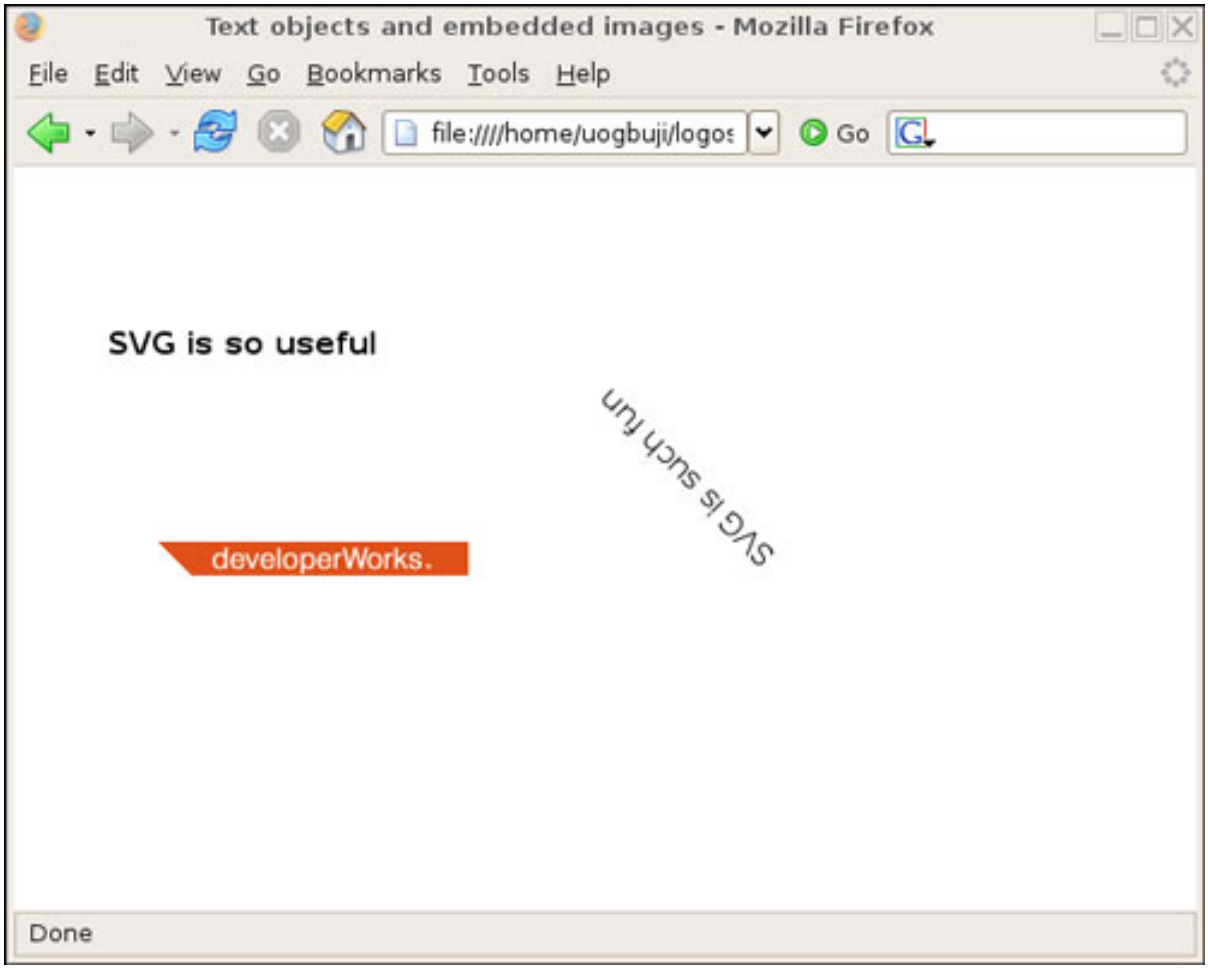
Listing 9. SVG example with text objects and embedded images

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1" baseProfile="full"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Text objects and embedded images</title>
  <text x="50" y="100" style="font-weight:bold">SVG is so useful</text>
  <image x="50" y="200" width="192px" height="18px"
         xlink:href="http://www-128.ibm.com/developerworks/i/dw.gif">
    <title>IBM developerWorks logo</title>
  </image>
  <svg x="300" y="100" width="200" height="200">
    <title>Nested SVG</title>
    <text transform="rotate(-135) translate(-150)">SVG is such fun</text>
  </svg>
</svg>
```

To use the `text` element, you express the text to be rendered in character content. The `x` and `y` attributes specify the position of the first character (the default for each is 0 if not specified). These attributes have this function in all the objects covered in this example. I also specify a style `font-weight:bold` to render the text in bold face. The first embedded image is a GIF, indicated as a link (the `xlink:href` attribute). Notice that the `xlink` prefix is declared on the top `svg` element. Notice also that I provide a `title` for the image, which you should always do. The second embedded image is a neat surprise: It's SVG nested within SVG.

The only object in the nested SVG is another `text` element. This one features a transform in order to flip it around. It's rotated 135 degrees counter-clockwise and then moved. I chose this sequence of transforms deliberately. The rotation without the translation would render the text invisible. To understand why, remember that the text originates at the default position of 0,0, determined within the box taken up by the nested SVG. After the rotation, the body of the text would be entirely above and to the left of its boundary, and would thus be clipped and invisible. The translate transform pushes the text back into the bounding box. I express only a translation along the X axis, but it's important to understand that the browser applies this to the distorted version of the axes after it has applied the rotate transform. If you view Listing 9 in the browser, you should see a display similar to Figure 8.

Figure 8. Browser display of Listing 9

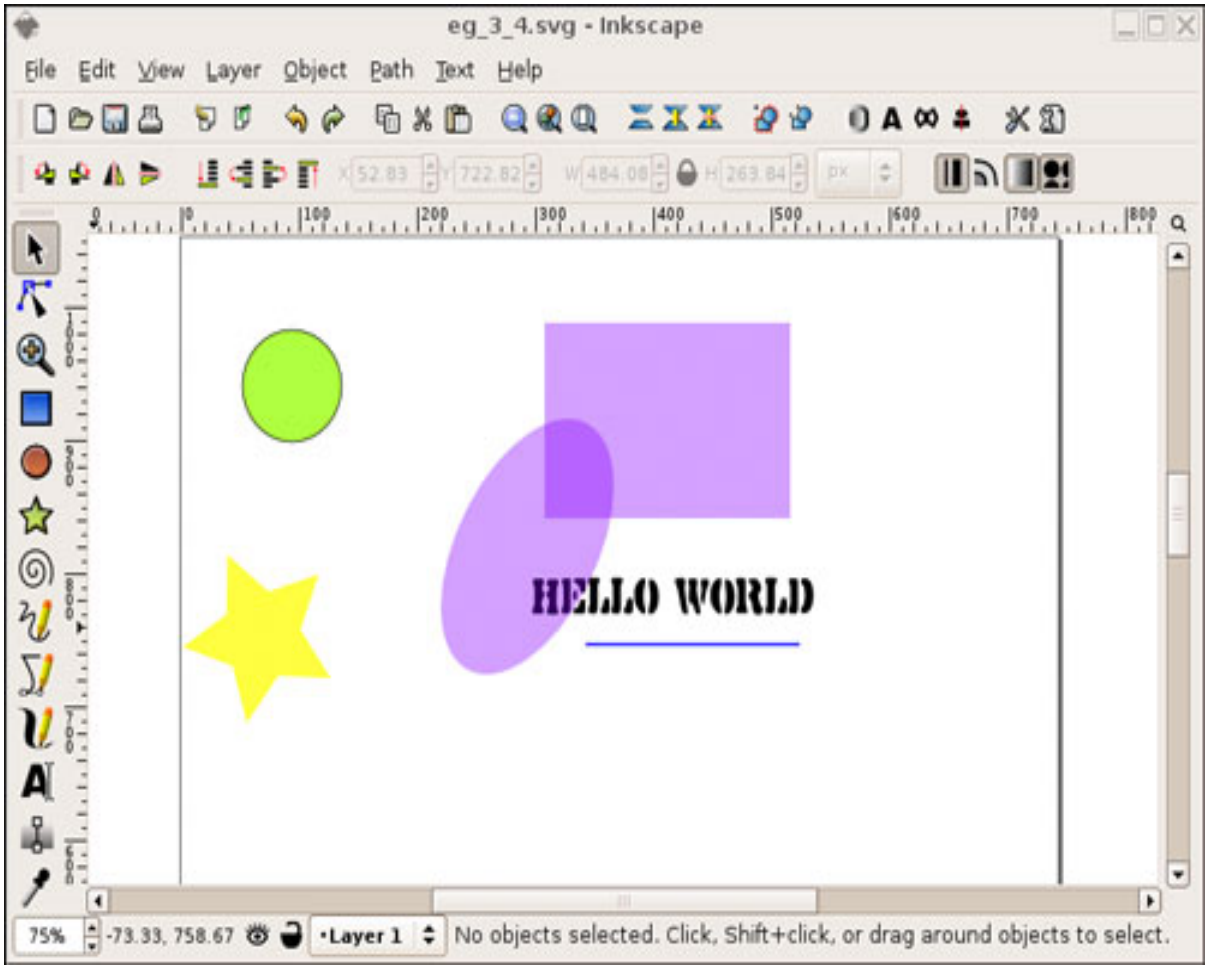


I strongly suggest you experiment with this arrangement until you understand these concepts clearly. Transforms and the coordinate system can be quite tricky. See what happens when you change the amount or order of the transforms, add a Y axis component to the rotation, and so on.

Using drawing tools to create images

You can imagine from the last subsection how complex it can be to put together interesting SVG drawings by hand. You have to wrack your brain a bit to make sense of how transforms fit into the coordinate system, and again to visualize how to use series of points to create a complex shape such as a polygon. It's much easier to use a drawing tool for SVG. A growing number of drawing tools are built around SVG, and many established tools now have SVG export. The SVG Wiki (see Resources) has a list of such tools in the Design Tools section. I use an open source tool called Inkscape. Figure 9 is a screenshot of an Inkscape session.

Figure 9. Creating SVG with Inkscape



One problem with using any such tool is that you end up with SVG that might not be exactly what you had in mind. The diagram might look exactly as intended, but the XML source might not. I've already mentioned how you can use the `polygon` element to create a square or rectangle rather than `rect`. There are many other ways you can decide how to choose between SVG objects to gain the same effect. You can create curved paths in SVG using a mechanism I won't cover in this tutorial, and you could use this approach to create a crescent rather than the overlapping circles approach you saw earlier. As a matter of fact, Inkscape defines the lime-green circle in Figure 9 as such a path object rather than using the `circle` element. Tools also tend to add annotations and additional information to the SVG document, generally in extension namespaces, in order to keep information specific to that tool. If such idiosyncrasies are a problem for you, consider using such tools just for the convenience of defining the more complex shapes, and then copying the necessary SVG elements by hand into a more general template. It takes some practice to get this approach right, and it's beyond the scope of this tutorial, but as with everything SVG-related, all it takes is a little practice and you'll be amazed at what you can craft.

One note of interest with regard to the SVG produced by Inkscape is that the choice of more complex constructs to represent simpler shapes as well as the Inkscape-specific extensions leads to larger file sizes. So far, the largest SVG file in this tutorial has been much less than 1 kilobyte (I'm not counting the size of the

linked image in the previous subsection's example). The file produced by Inkscape saving the session in Figure 9 (eg_3_4.svg in the download file) is more than 4Kb in size. Then again, exporting that SVG to PNG format using the default options results in a file of almost 14Kb. SVG file sizes can vary greatly, but for the sort of shape-driven images to which SVG is suited, the format is quite efficient.

The importance of well-formedness

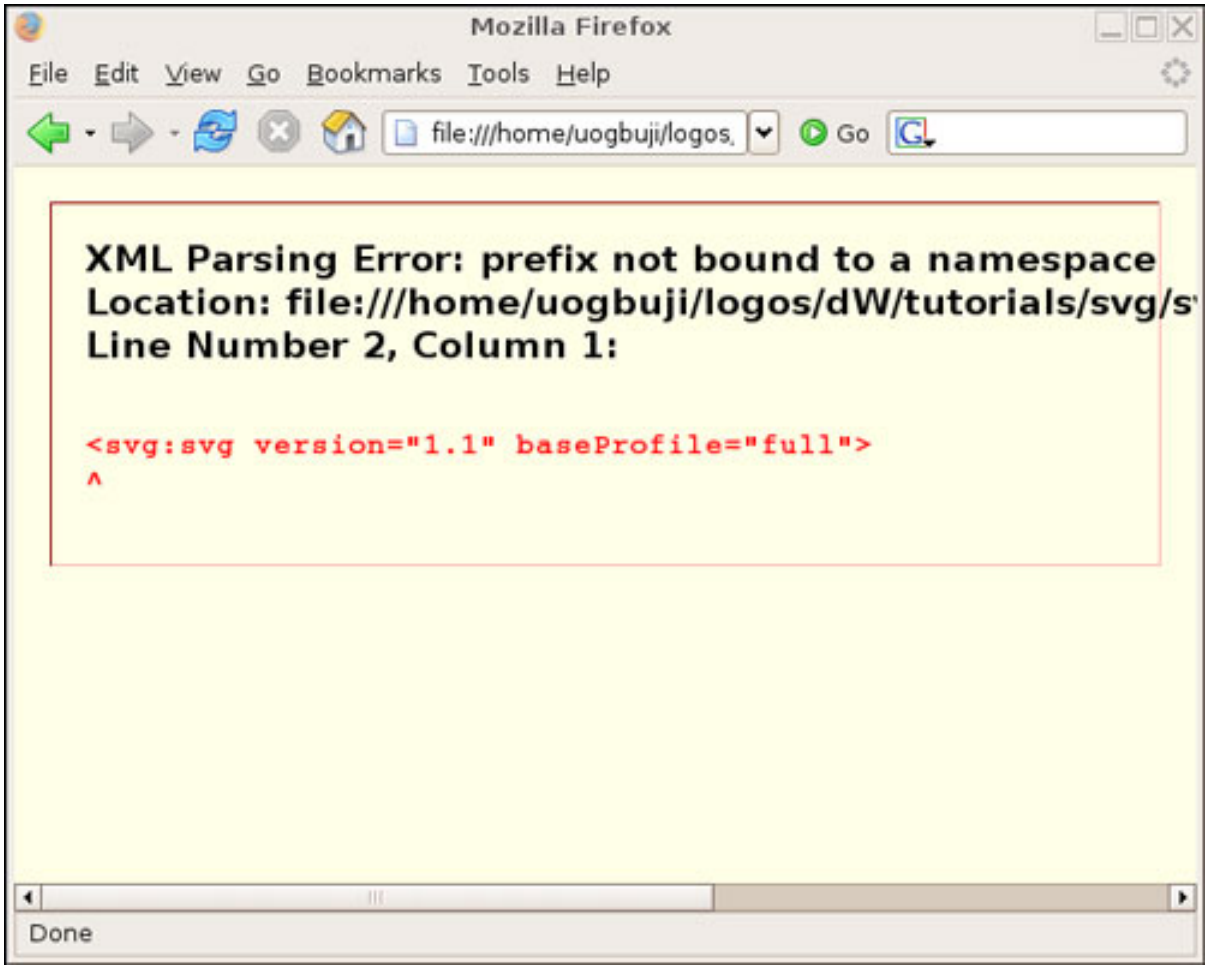
As the XML you use in SVG becomes more complex, errors in well-formedness become more likely (unless you're using well-tested tools to generate all the SVG). But well-formedness in SVG is of no less importance than with other XML applications. You might be used to lenience by Web browsers in the HTML world, but they are very strict in reading SVG. Listing 10 (eg_2_8_bad.svg in the download file) is based on Listing 1, but is not well-formed according to the XML namespaces spec. It is well-formed according to XML 1.0, but SVG is a namespace-aware application and requires namespace well-formedness as well.

Listing 10. Ill-formed SVG

```
<?xml version="1.0" encoding="utf-8"?>
<svg:svg version="1.1" baseProfile="full">
  <svg:circle cx="3cm" cy="2cm" r="1cm"/>
</svg:svg>
```

The listing uses the `svg` prefix, but does not declare it. If you view Listing 10 in the browser, you should see a browser error display along the lines of Figure 10.

Figure 10. Browser display of Listing 10



Section 4. SVG in XHTML

Now that you have a grasp of SVG basics, it's time to learn another important way to use SVG within a Web browser, by integrating it into XHTML rather than loading it as a stand-alone file. This allows you to incorporate the magic of SVG into your Web pages.

External SVG files referenced from XHTML

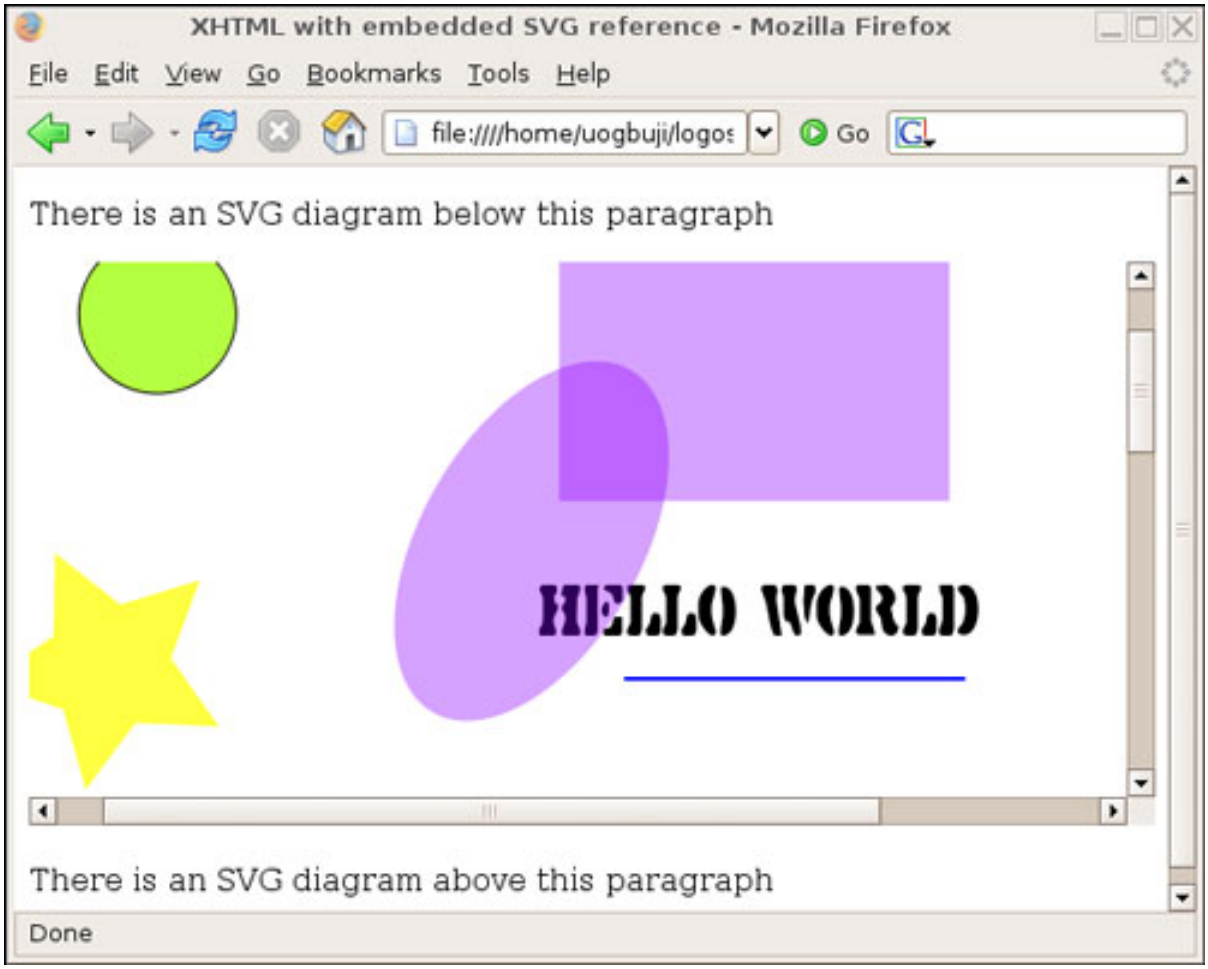
You can include a reference to an SVG file to be rendered inline using the XHTML `object` element, which is a lot like the `img` element. In fact, you can also use `img` (just as you can use `object` for embedding PNG, JPEG, and GIF images), but I recommend using `object` for embedding SVG. Listing 11 (eg_4_1.svg in the download file) is an XHTML file with an embedded reference to an SVG file.

Listing 11. XHTML file with an embedded reference to an SVG file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
  <head>
    <title>XHTML with embedded SVG reference</title>
  </head>
  <body>
    <p>There is an SVG diagram below this paragraph</p>
    <object data="eg_3_4.svg" type="image/svg+xml" width="600"
height="300"
            title="SVG created using Inkscape">
      If you're seeing this text your browser set-up does not
support SVG.
      You may need a plug-in.
    </object>
    <p>There is an SVG diagram above this paragraph</p>
  </body>
</html>
```

This regular XHTML 1.0 file conforms to the Strict DTD. The `object` element references an SVG file through the `data` parameter, which is a URI (a simple reference in this case, since the SVG file is in the same location as the XHTML). The `width` and `height` attributes specify the area the browser will set aside for the entire image. The text within the `title` attribute is informational and might be available from the browser view in some way -- for example, as a tool-tip when the mouse hovers over the object. The text within the element is fall-back text to be presented if the browser cannot handle the referenced object. If you view Listing 11 in the browser, you should see a display similar to Figure 11.

Figure 11. Browser display of Listing 11



Notice how the embedded SVG image is not shown directly in its entirety, and it has scrollbars to allow you to pan across the image. This is because the space I allowed for the `object` in the `width` and `height` attributes is smaller than the natural space taken up by the image. This is different from how Firefox deals with raster images, which are scaled up or down to fit the specified size.

Using `object` to embed a reference to SVG works fine in HTML as well as XHTML, so if you cannot deploy XHTML, this is your best bet. The more interesting techniques I'll discuss next are XHTML-specific.

SVG embedded directly into XHTML

If all you want is to generate images for inclusion in a Web page, the embedding approach discussed previously is sufficient, but some of the most exciting things SVG has in store for Web designers come with a more intimate pairing between SVG and other Web languages such as XHTML. Thanks to XML namespaces, you can embed an SVG image directly into XHTML as a section of foreign markup. Listing 12 (eg_4_2.svg in the download file) is an XHTML file with an embedded SVG image.

Listing 12. XHTML file with an embedded SVG image

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>XHTML with embedded SVG image</title>
  </head>
  <body>
    <p>There is an SVG diagram below this paragraph</p>
    <svg:svg version="1.1" width="600" height="220">
      <svg:title>A pair of lines and a pair of ellipses</svg:title>
      <svg:defs>
        <svg:style type="text/css"><![CDATA[
          line {
            stroke: red;
            stroke-width: 5;
          }
        ]]></svg:style>
      </svg:defs>
      <svg:g>
        <svg:title>Plain shapes</svg:title>
        <svg:ellipse cx="150" cy="50" rx="100" ry="50"/>
        <svg:line x1="450" y1="0" x2="550" y2="100"/>
      </svg:g>
      <svg:g>
        <svg:title>Rotated shapes</svg:title>
        <svg:ellipse cx="150" cy="150" rx="100" ry="50"
            transform="rotate(20)"/>
        <svg:line x1="350" y1="50" x2="450" y2="150"
            transform="rotate(20)"/>
      </svg:g>
    </svg:svg>
    <p>There is an SVG diagram above this paragraph</p>
  </body>
</html>
```

Notice the namespace declaration for SVG right on the `html` document element. This establishes the `svg` prefix for the embedded SVG, and sets it apart from all prior examples, which have avoided prefixes by using the default namespace. It is valid XML namespace usage to redefine the default namespace on the embedded `svg` element (as in Listing 13 (eg_4_2_2.svg in the download file), but experience with namespaces indicates that namespace redefinition within a document is generally not a good practice. I recommend the pattern in Listing 12 over that in Listing 13, despite the small annoyance of the necessary prefixes. Notice also that I define the `height` and `width` attributes as I did with the nested `svg` element in Listing 9. Again, the browser uses this to establish the space the SVG image will take up. In all other respects, the embedded SVG image is similar to what you've seen in this tutorial already.

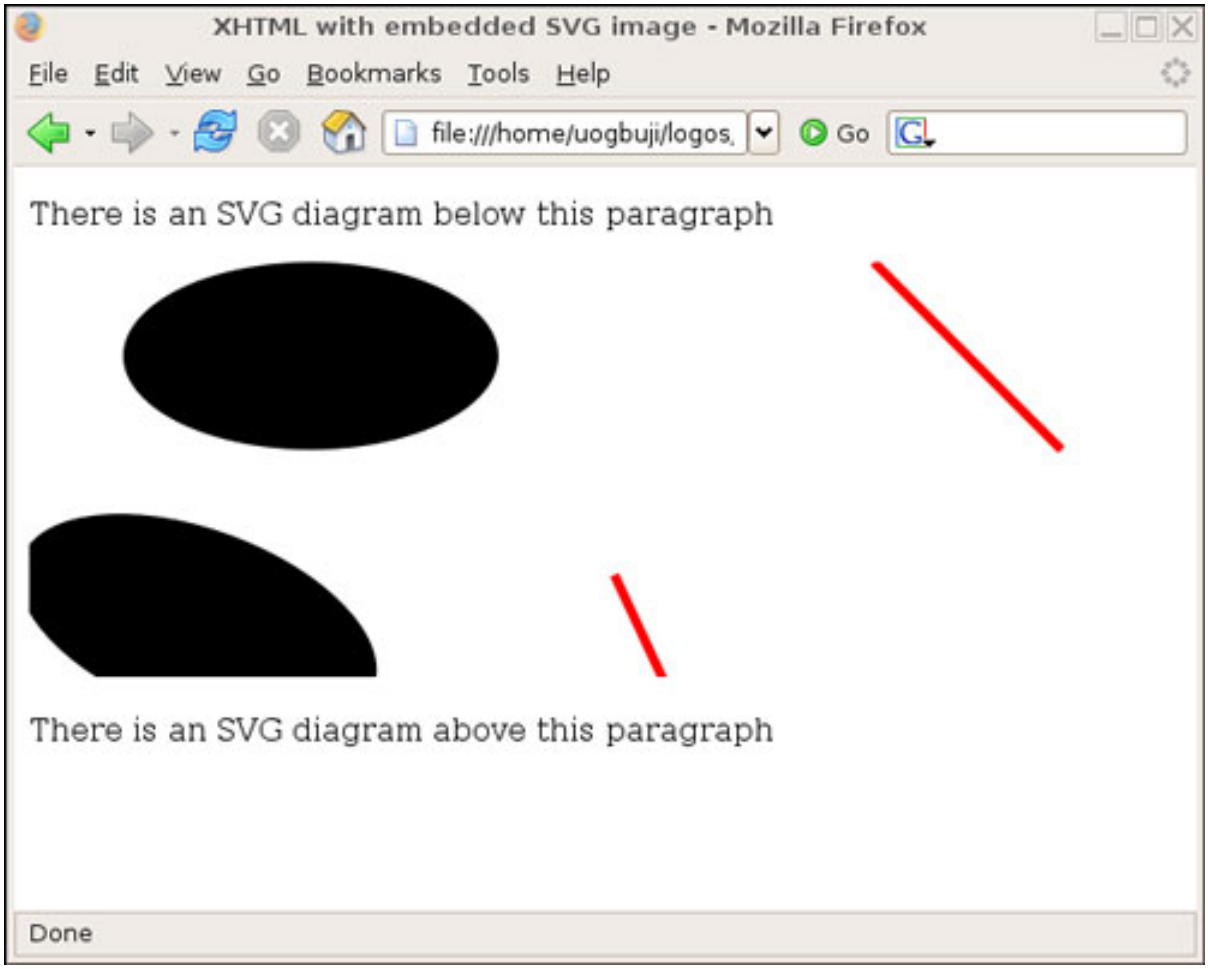
Listing 13. XHTML file with an embedded SVG image (redefined default namespace)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>XHTML with embedded SVG image (redefined default namespace)</title>
  </head>
```

```
  <body>
    <p>There is an SVG diagram below this paragraph</p>
    <svg version="1.1" width="600" height="220"
         xmlns="http://www.w3.org/2000/svg">
      <title>A pair of lines and a pair of ellipses</title>
      <defs>
        <style type="text/css"><![CDATA[
          line {
            stroke: red;
            stroke-width: 5;
          }
        ]]></style>
      </defs>
      <g>
        <title>Plain shapes</title>
        <ellipse cx="150" cy="50" rx="100" ry="50"/>
        <line x1="450" y1="0" x2="550" y2="100"/>
      </g>
      <g>
        <title>Rotated shapes</title>
        <ellipse cx="150" cy="150" rx="100" ry="50"
            transform="rotate(20)"/>
        <line x1="350" y1="50" x2="450" y2="150"
            transform="rotate(20)"/>
      </g>
    </svg>
    <p>There is an SVG diagram above this paragraph</p>
  </body>
</html>
```

If you view Listing 12 or 13 in the browser, you should see a display similar to Figure 12.

Figure 12. Browser display of Listing 13



Notice that in this case, if the browser doesn't have space to render the entire image, it's clipped rather than fitted with scrollbars (or scaled).

Viewing the source

One nice thing about embedding SVG in XHTML is that if users view the source of your Web pages, whether for education or to automate service exchange, they'll see the source of your images as well as the enclosing document, thus taking advantage of SVG's transparency. Of course, this is not such a great thing if you are trying to keep all your SVG tricks secret, but if that were the case, you can convert SVG to some less transparent format on the Web server for deployment. Most of the Web, however, flourishes through transparency and open data exchange, so SVG's friendliness to those who use browsers' view-source features is a big plus. If you view Listing 12 in the browser and then select the view-source feature (in Firefox, select the View menu and then Page Source), you should see a display similar to Figure 13.

Figure 13. Browser display of Listing 12

CSS in XHTML

The key benefit from close integration of SVG into XHTML is that you can use many of the browser's features for XHTML processing on the SVG transparently. To give you a taste of this, I'll demonstrate how you can merge SVG and XHTML style. Listing 14 (eg_4_4.svg in the download file) matches Listing 12, except that I've moved the stylesheet from the embedded SVG to the XHTML's header, and I've added a rule for paragraphs.

Listing 14. SVG in XHTML with SVG style declared in the XHTML header

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xml:lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
  <head>
    <title>SVG in XHTML with SVG style declared in the XHTML header</title>
    <style type="text/css"><![CDATA[
          line {
            stroke: red;
            stroke-width: 5;
          }
```

```
          p {text-decoration: underline;}
    ]]></style>
  </head>
  <body>
    <p>There is an SVG diagram below this paragraph</p>
    <svg:svg version="1.1" width="600" height="220">
      <svg:title>A pair of lines and a pair of ellipses</svg:title>
      <svg:g>
        <svg:title>Plain shapes</svg:title>
        <svg:ellipse cx="150" cy="50" rx="100" ry="50"/>
        <svg:line x1="450" y1="0" x2="550" y2="100"/>
      </svg:g>
      <svg:g>
        <svg:title>Rotated shapes</svg:title>
        <svg:ellipse cx="150" cy="150" rx="100" ry="50"
            transform="rotate(20)"/>
        <svg:line x1="350" y1="50" x2="450" y2="150"
            transform="rotate(20)"/>
      </svg:g>
    </svg:svg>
    <p>There is an SVG diagram above this paragraph</p>
  </body>
</html>
```

The style rules in the CSS work regardless of the prefix because CSS Level 3 makes it clear that element selectors match the local name of an element by default. The rule for p matches any element with that name regardless of namespace (or prefix). If you view Listing 14 in the browser, you should see a display similar to Figure 12, except that the text would be underlined.

Section 5. Wrap up

Summary

In this tutorial, you learned how to use SVG in Web development. In particular, you learned how to:

- Create circles
- Control the look of objects (styles)
- Work with groups
- Make use of drawing order
- Use transforms to change objects
- Manipulate opacity
- Create squares and polygons
- Create lines and ellipses

- Create text
- Use drawing tools to generate SVG
- Reference external SVG files from XHTML
- Embed SVG images in XHTML
- Use CSS in the XHTML header to style SVG as well as XHTML elements

With this basic understanding of SVG and how it works into Web development, you know enough to start experimenting. In future tutorials, you can learn more about combining advanced and dynamic Web technologies with SVG features.

Downloads

Description	Name	Size	Download method
Sample SVG files for this tutorial	x-svggraphics-tutorial-files.zip	9KB	HTTP

Information about download methods

Resources

Learn

- Not familiar with XML? Start with the many helpful resources in the developerWorks New to XML page, especially the tutorial Introduction to XML by Doug Tidwell (August 2002).
- Learn about XML namespaces in the article Plan to use XML namespaces, Part 1 by David Marston (April 2004), and learn more about how to use namespaces effectively in the articles Plan to use XML namespaces, Part 2 by David Marston (April 2004) and Principles of XML design: Use XML namespaces with care by Uche Ogbuji (April 2004).
- Understand XHTML and what sets it apart from HTML in the tutorial, XHTML, step-by-step, by Uche Ogbuji (September 2005).
- Bookmark the W3C's SVG home page, which includes links to all the SVG specifications, as well as more general information on the technology.
- Peruse the SVG Wiki, a rich source of resources including a cross-reference of SVG elements and listings of SVG tools.
- Learn some important tips and tricks for publishing SVG for maximum compatibility with available tools. Read SVG Authoring Guidelines by Jonathan Watt.
- Explore additional Scalable Vector Graphics articles and tutorials on developerworks.
- Peruse the developerWorks XML and Web architecture zones for more information on the technologies covered in this tutorial.

Get products and technologies

- Follow along by running the tutorial examples yourself. I tested the browser examples in this tutorial with Firefox, a popular and free Web browser based on Mozilla and available on Windows, Mac OS X, Linux, and other platforms. I tested stand-alone XSLT processing on the command line using 4Suite.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Uche Ogbuji
Uche Ogbuji is a consultant and cofounder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management. Fourthought develops 4Suite, an open source platform for XML, RDF, and knowledge-management applications. Mr. Ogbuji is also a lead developer of the

Versa RDF query language. He is a computer engineer and writer born in Nigeria, living and working in Boulder, CO. You can find more about Mr. Ogbuji at his Weblog Copia, or contact him at uche@ogbuji.net.