VML to SVG Migration Guide:

A Comparative Overview of Architecture and Implementation

By Seth McEvoy

March 16, 2010

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

About this Document

This migration guide provides a comparative overview of the architecture and implementation of the vector markup languages, VML (Vector Markup Language) and SVG (Scalable Vector Graphics). It includes working code examples, line-by-line code annotations, screen shots, and discussion material that demonstrates how SVG has been developed and expanded from its VML roots.

This material is presented in two parts. The first part compares the 12 key architectural concepts of both languages, showing how SVG has expanded and changed from VML, and which aspects are the same. The second part of this guide consists of an SVG conversion of a 50-line VML animation program. The line-by-line code annotations explain some of the differences between the implementations of the two languages.

Introduction to VML and SVG

The markup languages VML and SVG provide a fast light-weight technology for displaying vector graphics in a webpage. These languages make it possible to create graphic webpages that load quickly and take up very little space. This is because the graphics are defined by text descriptions instead of cumbersome bitmaps. VML and SVG both use XML (eXtended Markup Language) markup to define web vector graphic images that can be completely modified by scripting languages such as JavaScript, Python, or Perl.

Microsoft Office invented the VML technology in 1998 for use in Office products as a way to include vector-based art when saving and loading Office documents to webpages. Major corporations using VML include Google and Amazon. SVG is an expansion of VML that includes additional features and functionality, but the basic architectural principles are still very similar. Both use XML to define shapes, colors, line-weight, and position, but SVG has expanded many features. Numerous software companies have been involved in the development of SVG, most notably are IBM, Adobe, Macromedia, and Sun. These companies created a committee in 1998 to define a common vector language standard for the World Wide Web Consortium. SVG is currently supported in varying degrees by Apple Safari, Google Chrome, Firefox, Opera, and Internet Explorer.

VML is still available in IE9 but Microsoft expects web sites to transition to SVG in the future.

For the official VML documentation, visit http://msdn.microsoft.com/en-us/library/bb264280(VS.85).aspx.

For the W3C specification for SVG 1.1, visit http://www.w3.org/TR/SVG11/.

PART ONE: ARCHITECTURAL OVERVIEW OF VML AND SVG

This part presents a basic introduction to the 12 key architectural concepts and their implementations of both languages. Their differences and similarities are discussed and code examples and screen shots are used to explain the details. Visual samples of SVG output are provided to demonstrate the capabilities of the language.

Table of Contents

This material compares VML and SVG in the following 12 sections:

Section 1: Placement in a Webpage

Both VML and SVG can add vector graphics to a webpage, but each technology does so in a completely different way.

VML Procedure: Placement in a Webpage

To put VML in a webpage, you must add two definitions to a standard HTML page.

1. Define the VML namespace as an attribute of the HTML element.
2. Define a VML behavior as a style.

These two definitions enable MSIE to recognize and render VML tags. With these two highlighted lines entered, the core HTML code looks like the following.

```
<html xmlns:v="urn:schemas-microsoft-com:vml">

<head>

<style> v\:* { behavior: url(#default#VML); }</style >

</head>

<body>
```

```
</body>

</html>
```

To add VML elements, preface them with “v:” to indicate the VML namespace. For example, to create a rectangle, use this code.

```
<v:rect

       style="width:50;

        height:50"

        fillcolor="green"

        strokecolor="black"/>
```

This creates a 50-x-50–pixel green rectangle with a black stroke outline. The fill and stroke are attributes and the width and height are style definitions.

The complete webpage would look like this.

```
<html xmlns:v="urn:schemas-microsoft-com:vml">

<head>

<style> v\:* { behavior: url(#default#VML); }</style >

</head>

<body>

<v:rect

       style="width:50;

       height:50"

       fillcolor="green"

       strokecolor="black"/>

</body>

</html>
```

A snapshot of this webpage would look like this.

SVG Procedure: Placement in a Webpage

VML has only one way to place graphics in a webpage. SVG differs from VML in that it has several ways to place graphics in a webpage, none of which are the same as VML. SVG uses these methods to put graphics in a webpage:

- SVG stand-alone webpage
- SVG embedded in a webpage
- SVG merged with XHTML
- SVG inline with a standard HTML webpage.

SVG Stand-Alone Webpage

You can create a stand-alone SVG document with an.svg file extension. Use the following steps:

1. Create a text file with the .svg extension.
2. Add XML and DOCTYPE statements.
3. Add opening and closing SVG elements with namespace, width, and height.

The core SVG code would look like this.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!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”

width=”200” height=”200”>

</svg>
```

Add SVG elements between the opening and closing SVG element tags to create graphics. For example, to add a rectangle, insert this code.

```
<rect

   x="100"

   y="100"

   width="50"

   height="50"

   fill="red"

   stroke="blue" />
```

This adds a standard rectangle, 50 x 50 pixels, with red fill and blue stroke outline. The position of the rectangle will be at 100, 100 in the webpage coordinate system.

The complete code would look like this.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!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

width=200 height=200>

    <rect

       x="100"

       y="100"

       width="50"

       height="50"

       fill="red"

       stroke="blue" />

</svg>
```

A snapshot of the webpage would look like this.

SVG Embedded in a Webpage

After creating an SVG file with an .svg extension, you can embed the SVG file into a separate standard HTML webpage using **iframe**, **object**, **img**, or **embed** elements.

The .svg file is pulled into the webpage by reference. For example, the following code would display the previous SVG image on a webpage in four different ways.

```
<html>

<head>

</head>

<body>


<p>iframe</p>

<iframe src="rect.svg" height="200" width="200" frameborder="0">

</iframe>


<p>object</p>

<object data="rect.svg" type="image/svg+xml" height="200" width="200">

</object>


<p>embed</p>

<embed src="rect.svg" type="image/svg+xml" height="200" width="200">

</embed>


<p>image</p>

<image src="rect.svg" type="image/svg+xml" height="200" width="200">

</image>


</body>

</html>
```

SVG Merged with XHTML

Because SVG is written in XML, SVG elements can be added as an SVG document fragment to a standard XHTML page.

First, you must create an XHTML page with the .xhtml file extension. Next, you must add the SVG namespace. Use the following code to do this.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<head>
```

```
</head>
<body>

</body>
</html>
```

Then, as you add SVG elements in the XML code, prefix them with "svg:" to identify them as part of the SVG namespace. For example, to add a circle, insert this code in the body of your page.

```
<svg:svg width="200" height="200">

<svg:circle cx="100" cy="100" r="50px"

       fill="lime" stroke="maroon" stroke-width="5"/>

</svg:svg>
```

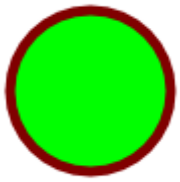
The complete XHTML page would contain this code.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<head>
</head>
<body>
       <svg:svg width="200" height="200">
       <svg:circle cx="100" cy="100" r="50px"

              fill="lime" stroke="maroon" stroke-width="5"/>

       </svg:svg>
</body>
</html>
```

A snapshot of the page in a browser would look like this.



SVG Inline with a Standard HTML Webpage

The previous three techniques for displaying SVG have been in use for several years. A newer technique has evolved in conjunction with HTML 5, allowing SVG elements to be treated as if they were HTML elements.

For example, create a minimum HTML webpage, and include the standards-based DOCTYPE element:

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
</body>
</html>
```

Then insert the SVG code in the body of the HTML document.

```
<svg>
      <circle cx="100" cy="100" r="50" fill="gold"></svg>
</svg>
```

The SVG elements and attributes are treated the same as any other HTML elements and attributes. This makes it easy to author, script, test, and modify SVG because all of the normal HTML tools are available.

The complete code for this inline SVG example is as follows.

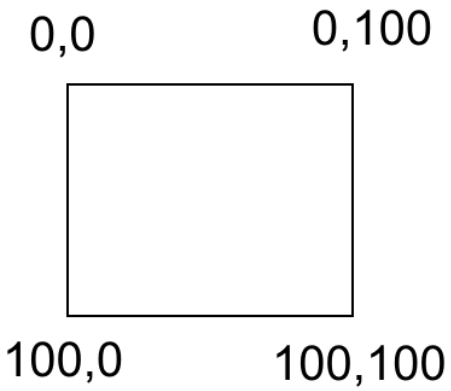
```
<!DOCTYPE HTML>
<html>

<head>
</head>
<body>
<svg>
      <circle cx="100" cy="100" r="50" fill="gold"></svg>
</svg>
</body>
</html>
```

The finished webpage looks like this.

Section 2: Coordinate Systems

VML and SVG both use the same coordinate systems, where 0, 0 is in the upper-left corner, the *x* values increase moving right across the page, and the *y* values increase moving down the page. The following diagram shows the *x*, *y* values of the four corners of a 100 x 100 rectangle.



Section 3: Building Blocks

VML and SVG each have the same concept of using building blocks to create simple visual components, but they implement these concepts in different ways.

The building blocks common to both are:

- Circle
- Square
- Polygon
- Line
- Curve
- Path
- Image

VML Procedure: Building Blocks

The basic building block for VML is the Shape element. All VML elements use Shape as a basic template. The Shape element has 75 attributes that define its characteristics, and 13 subelements that further define them.

In addition, VML has the following defined shapes that have all the attributes of the Shape element, but also have additional attributes.

- Rect (rectangle)
- RoundRect
- Line
- Polyline
- Oval
- Image
- Curve

One of the additional VML building blocks is the subelement **Path**, which defines a path using a quoted series of commands and numbers.

SVG Procedure: Building Blocks

SVG provides several building blocks that are the same as VML but with different names. SVG building blocks are independent of each other. Unlike VML, SVG has no master Shape element building block that all others inherit from.

The standard building blocks for SVG are:

- Rect (rectangle)
- Line
- Polyline
- Polygon
- Ellipse (includes Circle)
- Path
- Image

VML and SVG Building Block Comparison

The following chart compares the VML and SVG building blocks.

VML	**SVG**
Oval	Circle
Oval	Ellipse

Rect, RoundRect	Rect
Polyline	Polyline, Polygon
Line	Line
Curve	Path (Bezier)
Image	Image
Path	Path

Section 4: Grouping Objects

VML and SVG both allow a collection of shapes to be grouped so they can be modified as if they were one object. The only real difference is in the name of the grouping element.

VML Procedure: Grouping

VML uses the **group** element to enclose a group. For example:

```
<v:group>
…
</v:group>
```

SVG Procedure: Grouping

SVG has a similar name for group objects. It uses the **g** element to enclose a group. For example:

```
<g>
…
</g>
```

Section 5: Text

VML and SVG treat text very differently. VML only has text as an attribute of an object, but SVG has a **text** element which can be modified by attributes.

VML Procedure: Text

VML uses two elements to display text:

- **TextBox**
- **TextPath**

TextBox

The **TextBox** element uses text as an attribute of a rectangle. This makes for fast creation of flow charts and organizational charts. The text expands or contracts to fill the box. However, because this redrawing does not use standard Windows fonts, the result may not be readable in extreme cases.

TextPath

Similarly, the **TextPath** element uses text as an attribute of a path. This lets you draw the text along any path. The text, however, is expanded or contracted to fit the length of the path, again using redrawing that does not use standard Windows fonts.

SVG Procedure: Text

Unlike VML, where text is just an attribute of a box or path, SVG has an actual **text** element. This allows attributes to modify the text directly. For example, you can modify attributes like typeface, color, and fill.

SVG text works only with single lines of text and does not have a provision for line breaks or line wrapping.

The following shows a few examples of SVG text manipulation.

Stretched Text

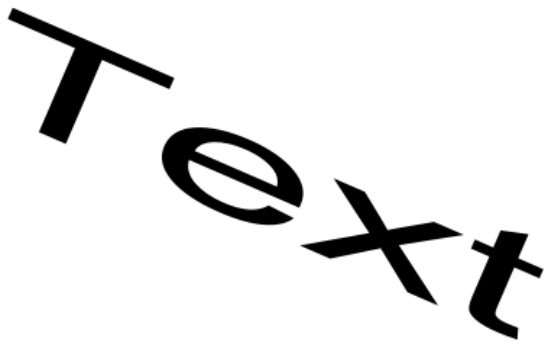
Because SVG is based on scalable graphics, you can stretch text without any loss of fidelity.

Normal Text

Normal Text Stretched

Rotated Text

SVG can transform text. One of the most popular transformations is rotation. An example of rotated text is as follows.



Section 6: Fills

VML and SVG both provide similar ways to fill a closed shape with color and patterns, but SVG also provides a way to fill a shape with repeated vector patterns.

VML Procedure: Fills

VML uses the **fill** element to define fills. There are three types of fill:

- Color
- Gradient (simple or polar)
- Bitmap (tile or stretched)

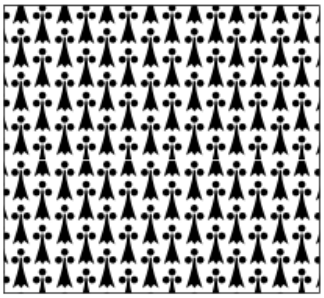
SVG Procedure: Fills

SVG uses the **fill** attribute to define fills. It uses the same three fills as VML, but adds a fourth fill.

The four SVG types of fill are:

- Color
- Gradient (simple or polar)
- Bitmap (tile or stretched)
- Vector pattern (tile or stretched)

The following shows a simple vector pattern fill:



Section 7: Strokes

VML and SVG both have similar ways to define the width, color, and other attributes of the strokes that outline a shape. However, the two languages differ in how they treat the ends of a stroked line, such as arrowheads or endcaps.

VML Procedure: Strokes

VML uses attributes to define the width, color, and other attributes of a stroke. It can also define arrowheads and endcaps.

SVG Procedure: Strokes

Like VML, SVG uses the same attributes to define width, color, and other attributes of a stroke. Unlike VML, it uses a separate **marker** element to define arrowheads.

Section 8: Clipping

VML and SVG both provide ways to clip shapes.

VML Procedure: Clipping

VML provides a **VMLFrame** element for clipping as well as a primitive form of clipping using objects.

SVG Procedure: Clipping

Unlike the primitive single clipping technique of VML, SVG supports clipping paths, masking, and simple alpha blending compositing.

Section 9: STYLES

Both VML and SVG support CSS and DOM styles in the same way.

Section 10: Transformations

VML provides limited support for transformations, but SVG provides ways to transform an object's position, sizing, and orientation in complex mathematical ways.

VML Transformations

VML has the following transformations:

- Shadow
- Skew
- Extrusion

SVG Transformations

VML has only three transformations, but SVG has six. SVG does not have an equivalent of VML's pseudo-3D **Shadow** and **Extrusion** transformations.

SVG provides a rich set of mathematical transformation using the **transform** attribute.

- Matrix (a,b,c,d,e,f)
- Translate (x,y)
- Scale(x,y)
- Rotate(angle,x,y)
- SkewX (angle)
- SkewY(angle)

SVG transformations can be combined or nested.

The code example in Part 2 shows an SVG transformation using the **rotate transform** attribute.

Section 11: Programming and Events

VML and SVG provide the same programming access through scripting and both use standard DOM and CSS events.

Section 12: Data Types

VML and SVG have different sets of data types.

VML Data Types

VML has the following basic data types:

- Double
- Fixed
- Integer
- String
- Length
- Measure
- Angle
- Color

VML also has data type objects that have attributes and methods using the standard data types. They are prefixed by "IVg". For example, **IVgGradientColor** array has **value** and **length** attributes that are String and Integer, and the methods of **AddColor** and **RemoveColor**.

VML uses standard CSS units such as **em** and **px**, but also uses the **emu**, which is the English Metrical Unit.

SVG Data Types

SVG uses many of the same data types as VML. Four are the same, two are the same but with a different name, and two are completely new.

SVG has the following data types:

- Integer
- Number
- String
- Length
- Angle
- Color
- Coordinate
- Paint
- List
- Percentage

Data Type Comparison

The following table compares the data types of VML and SVG.

VML	SVG
Integer, Double	Integer
Fixed	Number
String	String
Length, Measure	Length
Angle	Angle
Color	Color
n/a	Paint
n/a	List
n/a	Percentage

Part Two: VML to SVG Conversion Code Example

This part provides a working SVG code example that has been converted from a 50-line VML program that demonstrates the creation and rotation of a rectangle in a webpage. This code example shows how the SVG rotation concept has expanded from VML. Both the VML and SVG programs include a discussion of how the code works and line-by-line code annotations that detail the architectural concepts and their implementation for each language. DOM (Document Object Model) scripting is used in these examples to demonstrate advanced JavaScript techniques that manipulate vector graphic images in real time.

VML Rotate Rectangle Program

This program creates a rectangle in a webpage using VML. After the rectangle is created, it spins the rectangle until you stop it. Buttons are provided to create, spin, and stop the rectangle.

Introduction

The discussion and code for this program is divided into the following tasks:

- Setting up VML in an HTML webpage.
- Creating a VML rectangle.
- Rotating the rectangle.
- Stopping the rotation.

Task 1: Setting up the VML in an HTML webpage

First, you must set up the HTML and JavaScript framework for the program. VML requires specific code to define namespace and behavior. The scripting is in the head and three buttons are in the body. Also, a **div** element is set up in the body that is used to anchor the newly created rectangle.

When you load the completed code example into Internet Explorer, you may have to give permission to run the program. If you are using the Internet Explorer Platform Preview, you may have to set the Debug option to an earlier version of IE.

Once loaded, press the first button to create the rectangle, the second to spin it, and the third to stop it.

This type of programming uses the Document Object Model (DOM) and is commonly called DOM scripting. This advanced JavaScript technique allows for fast and flexible manipulation of vector images in a webpage. Unlike static webpages, DOM scripting allows for the creation of dynamic applications that can interact with the user.

The body of the HTML document doesn't contain any VML code; instead, VML objects are created in memory through programming and attached to a ** ** in the document. You can then manipulate the objects very easily through code.

```
<html xmlns:v="urn:schemas-microsoft-com:vml">

<head>


<!-- VML requires VML namespace and behavior. -->

<style>

v\:* { behavior: url(#default#VML);}

</style>


<script type="text/javascript">
```

```
// Your JavaScript code will go here.
</script>

</head>

<body>

<br><br>

<!-- Button to create rectangle. -->
<input type="BUTTON" value="Make Rectangle" onclick="makeRect()">

<!-- Button to rotate rectangle. -->
<input type="BUTTON" value="Rotate Rectangle" onclick="rotateRect()">

<!-- Button to close webpage. -->
<input type="BUTTON" value="Stop!" onclick=”clearInterval(spin)”>

<br><br>

<!-- Node where new rectangle will be attached. -->
<div id="anchorDiv"></div>

</body>
</html>
```

Task 2: Creating the Rectangle

The Document Object Model allows you to create objects in memory using JavaScript and then attach them to the document so they can be displayed.

The following code creates a rectangle using the VML namespace template for a rectangle. The **createElement** method is used to create the rectangle, and CSS styles are used to define the height,

width, color, and ID of the rectangle. Finally, the **appendChild** method is used to attach the rectangle to the document at the "anchorDiv" **div** element.

Also a global variable, "spin" is created that will be used later to stop the spinning rectangle.

The **makeRect** function is called when the **Make Rectangle** button is pushed.

```
//Flag to stop rectangle from spinning.
var spin;


// Make rectangle.
function makeRect() {


        // Create element in memory.
        var r = document.createElement("v:rect");


        // Define width, height, color, and unique ID.
        r.style.width = 100;
        r.style.height = 100;
        r.fillcolor = "purple";
        r.id = "myRect";


        // Attach rectangle to the document at the  the specified Div.
        anchorDiv.appendChild(r);
        }
```

Task 3: Spin the Rectangle

After the rectangle is created, two functions are used to spin the object.

The first function, **rotateRect**, is called by the **Rotate Rectangle** button. This function uses the **setInterval** command to call the function that actually rotates the rectangle every ten milliseconds. The global variable "spin" is used as a pointer to the timer making the call.

The second function, **spinRect**, is called by the **setInterval** method which was triggered in the **rotateRect** function. It uses the VML method **Rotate** to rotate the object 11 degrees every time it is called.

```
// Set up the rotation.
```

```
function rotateRect() {

        // Call spinRect function every 10 milliseconds.
        // The spin variable allows us to clear the call to setInterval.
        spin = setInterval("spinRect()", 10);
        }

// Spin the rectangle by specified increment every time function called.
function spinRect() {

        // Increment rectangle rotation by 11 degrees.
        myRect.rotation += 11;
        }
```

Task 4: Stop the Rectangle Spinning

The **Stop** button calls the **clearInterval** method using the spin variable that was set in the **rotateRect** function. This simply stops the specified **setInterval** method from calling **rotateRect** again.

Complete Program Listing

Copy and paste this into a text document with the .html extension.

```
<html xmlns:v="urn:schemas-microsoft-com:vml">
<head>

<!-- VML requires VML namespace and behavior. -->
<style>
v\:* { behavior: url(#default#VML);}
</style>

<script type="text/javascript">

//Flag to stop rectangle from spinning.
var spin;
```

```
// Make rectangle.
function makeRect() {

	// Create element in memory.
	var r = document.createElement("v:rect");

	// Define width, height, color, and unique ID.
	r.style.width = 100;
	r.style.height = 100;
	r.fillcolor = "purple";
	r.id = "myRect";

	// Attach rectangle to the document at the  the specified Div.
	anchorDiv.appendChild(r);
	}

// Set up the rotation.
function rotateRect() {

	// Call spinRect function every 10 milliseconds.
	// The spin variable allows us to clear the call to setInterval.
	spin = setInterval("spinRect()", 10);
	}

// Spin the rectangle by specified increment every time function called.
function spinRect() {

	// Increment rectangle rotation by 11 degrees.
	myRect.rotation += 11;
```

```
        }


</script>


</head>


<body>


<br><br>


<!-- Button to create rectangle. -->

<input type="BUTTON" value="Make Rectangle" onclick="makeRect()">


<!-- Button to rotate rectangle. -->

<input type="BUTTON" value="Rotate Rectangle" onclick="rotateRect()">


<!-- Button to close webpage. -->

<input type="BUTTON" value="Stop!" onclick="clearInterval(spin)">


<br><br>


<!-- Node where new rectangle will be attached. -->

<div id="anchorDiv"></div>


</body>

</html>
```

SVG Rotate Rectangle Program

This program creates a rectangle in a webpage using SVG. After the rectangle is created, the program spins the rectangle until you stop it. Buttons are provided to create, spin, and stop the rectangle.

Introduction

The discussion and code for this program is divided into the following tasks:

- Setting up SVG in an HTML webpage.
- Creating an SVG rectangle.
- Rotating the rectangle.
- Stopping the rotation.

Task 1: Setting up SVG in an HTML webpage

First, you must set up the HTML and JavaScript framework for the program. The scripting is in the head and three buttons are in the body. Also, a **div** element is set up in the body that is used to anchor the newly created rectangle. Be sure to use the DOCTYPE instructions as the first line to conform to web standards.

If you are using the IE9 Platform Preview, you may have to set the Debug option to IE9 and you must load the program from the **Open** menu.

After the program loads, press the first button to create the rectangle, the second to spin it, and the third to stop it.

This type of programming uses the Document Object Model (DOM) and is commonly called DOM scripting. Like the VML program, the SVG code example uses this advanced JavaScript technique to allow fast and flexible manipulation of vector images in a webpage. Unlike static webpages, DOM scripting allows for the creation of dynamic applications that can interact with the user.

The body of the HTML document doesn't contain any SVG code; instead, SVG objects are created in memory through programming and attached to a ** ** in the document. You can then manipulate the objects very easily through code.

```
<!DOCTYPE HTML>

<html>

<head>


<script type="text/ecmascript">


// Scripts go here.


</script>


</head>
```

```
<body>


<br><br>


<!-- Button to create rectangle. -->

<input type="BUTTON" value="Make Rectangle" onclick="makeRect()">


<!-- Button to rotate rectangle. -->

<input type="BUTTON" value="Rotate Rectangle" onclick="rotateRect()">


<!-- Button to close webpage with spin variable. -->

<input type="BUTTON" value="Stop!" onclick=clearInterval(spin)>


<br><br>


<!-- The rectangle will be attached to the document here. -->

<div id="myAnchor"></div>

</body>

</html>
```

Task 2: Creating the Rectangle

The Document Object Model allows you to create objects in memory using JavaScript and then attach them to the document so they can be displayed.

Every SVG document must have a parent SVG element that all SVG elements will be added to. The following code will create the parent SVG element and then create a rectangle using the SVG namespace template.

The **createElementNS** method creates the parent SVG element mySVG and defines the width and height. Then the **createElementNS** method is used again to create the rectangle by calling the SVG namespace. The **setAttributeNS** method is used to define the height, width, x, y, color, fill, and stroke of the rectangle.

Once the rectangle is created, it is attached to the SVG parent with with **appendChild**. Next the SVG parent is attached to the "anchorDiv" **div** element, using **getElementById and appendChild.**

Global variables are also added:

- “svgNS” is used to define the path to the SVG namespace.
- “mySVG” is used to define the parent SVG element.
- “myRect” is used as a place holder for the rectangle.
- “spin” is created that will be used later to stop the spinning rectangle.
- “myAngle” is the initial value of the angle to rotate.
- “myX” and “myY” will be used to define the center of rotation.

The **makeRect** function is called when the **Make Rectangle** button is pushed.

```
// Global variables.


// Define SVG namespace.

var svgNS = "http://www.w3.org/2000/svg";
```

```
// Placeholder for parent SVG element to be created.
var mySvg;

// Placeholder for rectangle object to be created.
var myRect;

// Flag to stop rectangle spinning.
var spin;

// Initial angle to start rotation from.
var myAngle = 0;

// Values of center of rotation.
var myX = 150;
var myY = 150;

// Make Rectangle.
function makeRect() {

	// Create parent SVG element with width and height.
	mySvg = document.createElementNS(svgNS,"svg");
	mySvg.setAttributeNS(null,"width",600);
	mySvg.setAttributeNS(null,"height",600);

	// Create rectangle element from SVG namespace.
	myRect = document.createElementNS(svgNS,"rect");

	// Set rectangle's attributes.
	myRect.setAttributeNS(null,"width",100);
	myRect.setAttributeNS(null,"height",100);
```

```
myRect.setAttributeNS(null,"x",100);

myRect.setAttributeNS(null,"y",100);

myRect.setAttributeNS(null,"fill","lightcoral");

myRect.setAttributeNS(null,"stroke","deepskyblue");

myRect.setAttributeNS(null,"stroke-width","5");


// Append rectangle to the parent SVG element.

// Append parent SVG element to the div node.

 mySvg.appendChild(myRect);

 document.getElementById("myAnchor").appendChild(mySvg);

}
```

Task 3: Spin the Rectangle

After the rectangle is created, two functions are used to spin the object.

The first function, **rotateRect**, is called by the **Rotate Rectangle** button. This function uses the **setInterval** command to call the function that actually rotates the rectangle every ten milliseconds. The global variable “spin” is used as a pointer to the timer making the call.

The second function, **spinRect**, is called by the **setInterval** method which was triggered in the **rotateRect** function. It uses the SVG translate attribute with the definition to rotate the object 11 degrees every time it is called. Note that the **rotate** definition has three parameters: the rotation angle, the x center of rotation, and the y center of rotation. Because a translation has to use SVG, the DOM, and JavaScript, the parameters must use quotes and plus signs to convert the floating-point number of JavaScript to SVG degrees.

```
// Do the rotation every 10 milliseconds until cancelled.

function rotateRect() {


    spin = setInterval("spinRect()", 10);

    }


// Spin rectangle by 11 degrees.

function spinRect() {


       // Rotation is a subset of the transform attribute.

       // Note the use of quotes and plus signs with variables in SVG attribute call.

       myRect.setAttributeNS(null,"transform","rotate(" + myAngle + "," + myX + "," +
myY + ")");

       myAngle = myAngle + 11;

       }
```

Task 4: Stop the Rectangle Spinning

The **Stop** button calls the **clearInterval** method using the spin variable that was set in the **rotateRect** function. This stops the specified **setInterval** method from calling **rotateRect** again.

Complete Program Listing

Copy and paste this into a text document with the .html extension.

```
       <!DOCTYPE HTML>

       <html>

       <head>


       <script type="text/ecmascript">

       var svgNS = "http://www.w3.org/2000/svg";

       var mySvg;

       var myRect;

       var spin;

       var myAngle = 0;

       var myX = 150;
```

```
var myY = 150;


// Create a rectangle.
function makeRect() {

	// Create SVG parent element.
	mySvg = document.createElementNS(svgNS,"svg");
	mySvg.setAttributeNS(null,"width",600);
	mySvg.setAttributeNS(null,"height",600);

	// Create rectangle.
	myRect = document.createElementNS(svgNS,"rect");
	myRect.setAttributeNS(null,"width",100);
	myRect.setAttributeNS(null,"height",100);
	myRect.setAttributeNS(null,"x",100);
	myRect.setAttributeNS(null,"y",100);
	myRect.setAttributeNS(null,"fill","lightcoral");
	myRect.setAttributeNS(null,"stroke","deepskyblue");
	myRect.setAttributeNS(null,"stroke-width","5");

	// Append rectangle to SVG parent element.
	mySvg.appendChild(myRect);
	// Append SVG parent to document.
	document.getElementById("myAnchor").appendChild(mySvg);
	}


// Spin the rectangle 11 degrees when called.
function spinRect() {
	myRect.setAttributeNS(null,"transform","rotate(" + myAngle + "," + myX +
"," + myY + ")");
```

```
        myAngle = myAngle + 11;
        }

// Call spinRect every 10 milliseconds.
function rotateRect() {
        spin = setInterval("spinRect()", 10);
        }

</script>

</head>

<body>

<br><br>

<!-- Button to create rectangle. -->
<input type="BUTTON" value="Make Rectangle" onclick="makeRect()">

<!-- Button to rotate rectangle. -->
<input type="BUTTON" value="Rotate Rectangle" onclick="rotateRect()">



<!-- Button to close web page. -->
<input type="BUTTON" value="Stop!" onclick="clearInterval(spin)">

<br><br>

<div id="myAnchor"></div>
</body>
```

```
</html>
```