Chapter

9

Drawing with SVG

This chapter contains the following topics:

- Working with SVG
- Supporting SVG in e.Report Designer Professional
- Examining the SVG code in a drawing control
- Debugging an SVG program

Working with SVG

Scalable Vector Graphics, or SVG, is a language for describing two-dimensional graphics in XML. SVG is a W3C standard which builds on other standards such as CSS. With SVG you can combine complex gradients with filters and bitmap graphics to create both realistic and abstract images. The W3C specification for SVG can be found at http://www.w3.org/TR/SVG11/index.html. The CSS online specification is at http://www.w3.org/TR/REC-CSS2/cover.html. SVG has a rich variety of graphic abilities. e.Report Designer Professional provides several examples of SVG applications. There are also many resources available to help you learn this language. Figure 9-1, shows images that have been created with SVG.

Figure 9-1 Images created with SVG

SVG is an XML-based language. The following SVG example draws a rectangle.

```
<svg>
   <rect
      x='10%' y='10%'
      width='80%' height='80%'
      fill='red'
      stroke='black' stroke-width='4pt'
   />
</svg>
```

While SVG is used to describe graphics, it is also a language that is intended to work in conjunction with other languages, in that it exposes graphical objects so that other applications can use them. It is an open standard that has numerous tools emerging to support it.

In a report design, SVG images can be placed in a chart, drawing or browser scripting control. You send SVG to a browser scripting control when you want a browser that supports SVG to render the image. e.Report Designer Professional supports the following SVG primitives:

- Lines, rectangles, polygons, circles, ellipses, arcs, B-spline curves
- Text
- Fills, patterns, textures
- Gradients, filters, lighting effects

- Any combination of the above

For example, you can use SVG to create rotated text for column headers, corporate logos, diagrams, annotations on charts, specialized chart types and dashboard components.

SVG is supported in several web browsers. SVG viewing applications, which allow you to view SVG images, are available on the web.

Supporting SVG in e.Report Designer Professional

Support of SVG consists of several Actuate Basic functions and an AFC class, AcDrawingSVGPlane. These work in conjunction with the methods of a drawing or chart control. The Actuate Basic functions provide a string in the locale-neutral format that SVG requires, and are described in Table 9-1. You use these functions to create attributes or parameters in the proper format as required for SVG. If you are using a non-neutral locale, you must use these functions to make your SVG program function properly. In addition, some functions also simplify the creation of SVG code. For example, SVGFontStyle returns the correct string as required by SVG given the specified font parameters. The following code segments

```
    svg = SVGFontStyle( "Label", pointLabelFont,
+                       "text-rendering:geometricPrecision;" )
```

and

```
    svg = "<style type='text/css'>" &
+       "<![CDATA[.Label{font-family:Arial;" &
+       "font-size:10.0px;font-weight:normal;" &
+       "font-style:normal;" &
+       "text-decoration:none;fill:#000000;" &
+       "stroke:none;text-rendering:geometricPrecision;}]]>" &
+       "</style>"
```

are equivalent.

For a full description on the use of these functions, refer to *Programming with Actuate Basic*.

Table 9-1 Actuate Basic SVG functions

Function	Use
SVGAttr	Returns a string for setting attributes.
SVGColorAttr	Returns a string for setting color attributes.
SVGDbl	Returns a string for setting a numeric value.
SVGFontStyle	Returns a string for setting font values.
SVGStr	Returns a string for setting a string value.

Table 9-1 Actuate Basic SVG functions

Function	Use
SVGStyle	Returns a string for setting a CSS style.

SVG drawings are created dynamically when the report is generated. They are rendered as a bitmap during viewing, and scale to the appropriate zoom factor or resolution. Users do not need an SVG viewer to view SVG drawings in a report. If you want to send SVG to a browser that supports SVG it must be embedded within a browser scripting control.

On Windows systems with a large size dpi setting, when you view a report within e.Report Designer Professional, the results may appear diffrently from how it appears when viewing the output in DHTML.

e.Report Designer Professional does not include an SVG editing tool, however various tools and plug-ins are available on the web. SVG animation and interactivity are not supported. Consult the examples installed with e.Report Designer Professional in \Program Files\Actuate10\eRDPro\Examples\DesignAndLayout\Charts and \Program Files\Actuate10\eRDPro\Examples\DesignAndLayout\Drawings. The chart examples that use SVG have names that begin with the word Drawing.

Understanding measurements and scaling

You use the viewBox attribute of the svg tag to apply a scale to an SVG drawing plane. This allows you to specify the default units. When you use viewBox, all of the absolute units of measures, such as inches, are scaled in the drawing. Use viewBox to scale the default units to points. Do not use explicit units.

The following code retrieves the size of the drawing, and then sets the default units in the SVG code to points. It then creates the SVG code to draw a rectangle, and sets the string containing the code into an AcDrawingSVGPlane.

```
' Get the size of the drawing in points
    Dim w As Double
    w = Size.Width / OnePoint
    Dim h As Double
    h = Size.Height / OnePoint

   Dim svg As String
    ' Scale the drawing to use points as the default units
svg = "<svg version='1.1'"
+       ' Standard SVG 1.1 namespaces
+       & " xmlns='http://www.w3.org/2000/svg'"
+       & " xmlns:xlink='http://www.w3.org/1999/xlink'"
+     ' Do not collapse whitespace in text
+     & " xml:space='preserve'"
```

```
+     ' Scale the SVG to use points as the default units
+     & " viewBox='0 0 " & SVGDbl( w ) & " "
+     & SVGDbl( h ) & "'>"
    svg = svg
+        & "<rect x='10%' y='10%'"
+        & " width='" & (w * 0.8) & "'"
+        & " height='" & (h * 0.8) & "'"
+        & " fill='green' stroke='black' stroke-width='4'/>"
+        & "</svg>"

    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
    svgPlane.SetSVG( svg )

    Super::Finish( )
End Sub
```

Using drawing planes

SVG code is placed within drawing planes. A drawing may contain multiple drawing planes which are drawn on top of each other. Each drawing plane has its own content.

You must explicitly insert or append drawing planes in your code. There is no default drawing plane in a drawing. Any drawing plane can be positioned, sized, or hidden. You can delete drawing planes, except for a drawing plane that contains a chart.

Drawing planes are typically used in the Start( ), OnRow( ), or Finish( ) methods of a drawing control, or DrawOnChart( ) for a chart control. To create an SVG image, you place all the SVG into one string variable, and then set that into a drawing plane as shown in the following code.

```
Sub Finish( )
    Dim svg As String

' create SVG code here
    svg = ...

    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
    svgPlane.SetSVG( svg )

    Super::Finish( )
End Sub
```

For more information on the AcDrawing and AcDrawingSVGPlane classes, see *Programming with Actuate Foundation Classes.*

How to use SVG in a report design

1. Determine what SVG elements and filters you will use.
2. Drag a drawing or chart control from Toolbox into a frame.

3 Copy code from one of the examples and modify it as necessary. Report designs that use SVG install with e.Report Designer Professional in \Program Files\Actuate10\eRDPro\Examples\DesignAndLayout\Charts and \Program Files\Actuate10\eRDPro\Examples\DesignAndLayout \Drawings. The chart examples that use SVG have names that begin with the word Drawing.

4 Build up your SVG in stages, making sure you have a valid drawing at each stage.

If you run into problems, add code to dump your SVG to a file and use a tool like XML-SPY to validate your SVG. You may also try to view your SVG using an SVG viewer. See "Debugging an SVG program," later in this chapter, for more information on how to dump your SVG code to various locations.

Drawing on charts

AcChart is a subclass of AcDrawing. By default, a chart creates a single special drawing plane that displays the chart. This plane does not use SVG. You can insert additional drawing planes behind and in front of the chart plane. Drawing on charts may not appear 100% accurate when a report is viewed within e.Report Designer Professional.

You can hide the chart plane, but not delete it. To draw on a chart, you override the AcChart::DrawOnChart( ) method. In addition, you can use chart method calls to get the position and size of the plot areas or pie in a chart.

In e.Report Designer Professional, open \Program Files\Actuate10\eRDPro \Examples\DesignAndLayout\Charts\DrawingRadialPieLabels \DrawingRadialPieLabels.rod.

This example report shows how to modify a chart to make room for SVG drawn labels on a radial pie chart. When you open the report, it contains a single frame with a chart control in it. The control's DrawOnChart( ) method is overridden as follows.

```
Sub DrawOnChart( baseLayer As AcChartLayer, overlayLayer As
   AcChartLayer, studyLayers() As AcChartLayer )
     ' Shrink and recenter the pie to make room for the labels
     Dim pieScale As Double
     pieScale = 0.95
     Dim chartPlane As AcDrawingChartPlane
     Set chartPlane = GetChartDrawingPlane( )
     chartPlane.SetSize( Size.Width * pieScale,
+                      Size.Height * pieScale)
     chartPlane.SetPosition( (Size.Width * (1 - pieScale)) / 2,
+                           (Size.Height * (1 - pieScale)) / 2)

     ' Get the center and radius of the pie (in twips)
     DescribeLayout( )
     Dim pieCenter As AcPoint
     pieCenter = baseLayer.GetPieCenter( )
     Dim pieRadius As AcTwips
```

```
    pieRadius = baseLayer.GetPieRadius( )

    ' Get the width, height, and pie center in points
    Dim w As Double
    w = Size.Width / OnePoint
    Dim h As Double
    h = Size.Height / OnePoint
    Dim cX As Double
    cX = pieCenter.X / OnePoint
    Dim cY As Double
    cY = pieCenter.Y / OnePoint

    ' The labels will be spaced slightly outside the pie
    Dim labelRadius As Double
    labelRadius = (pieRadius / OnePoint) + 8

    Dim svg As String
    svg = "<svg version='1.1'"
+       ' Standard SVG 1.1 namespaces
+       & " xmlns='http://www.w3.org/2000/svg'"
+       & " xmlns:xlink='http://www.w3.org/1999/xlink'"
+       ' Do not collapse whitespace in text
+       & " xml:space='preserve'"
+       ' Scale the SVG to use points as the default units
+       & " viewBox='0 0 " & SVGDbl( w ) & " "
+      & SVGDbl( h ) & "'>"

    ' Define the point label font style; use geometric
    ' precision text rendering to get consistent appearance
    ' regardless of angle
    Dim pointLabelFont As AcFont
    pointLabelFont = baseLayer.GetSeriesStyle
+                  ( 1 ).GetPointLabelStyle( ).Font
    svg = svg
+       & "<defs>"
+       &   SVGFontStyle( "Label", pointLabelFont,
+                         "text-rendering:geometricPrecision;" )
+       & "</defs>"

    ' Get the height of the label font
    Dim labelFontHeight As Double
    labelFontHeight = GetFontDisplayHeight( pointLabelFont )
+                               / OnePoint

    Dim series As AcChartSeries
    Set series = baseLayer.GetSeries( 1 )

    Dim numberOfCategories As Integer
    numberOfCategories = baseLayer.GetNumberOfCategories( )
```

```
    Dim totalValue As Double
    totalValue = series.GetSumOfSliceValues( )

    Dim cumulativeValue As Double

    ' Set the coordinate origin to the pie center
    svg = svg
+       & "<g transform='"
+       & " translate(" & SVGDbl( cX ) & "," & SVGDbl( cY )
+      & ")" & "'>"

    Dim i As Integer
    For i = 1 To numberOfCategories
        Dim category As AcChartCategory
        Set category = baseLayer.GetCategory( i )
        Dim labelText As String
        labelText = category.GetLabelText( )

        Dim point As AcChartPoint
        set point = series.GetPoint( i )
        Dim pointValue As Double
        pointValue = point.GetYValue( )

        Dim sliceAngle As Double
        sliceAngle = (pointValue / totalValue) * 360

        Dim labelAngle As Double
        labelAngle = ((cumulativeValue / totalValue) * 360) +
+                    (sliceAngle / 2) - 90

        Dim r As Double
        Dim textAnchor As String
        If (labelAngle <= 90) Then
            r = labelRadius
            ' Text on the right is left-aligned
            textAnchor = " text-anchor='start'"
        Else
            ' Flip the labels on the left side
            labelAngle = labelAngle + 180
            r = -labelRadius
            ' Text on the left is right-aligned
            textAnchor = " text-anchor='end'"
        End If

        ' Draw the label rotated and offset to center
       ' it on the slice
        svg = svg
+           & "<text class='Label'"
+           & " transform='"
+           &  " rotate(" & SVGDbl( labelAngle ) & ")"
```

```
+           &  " translate(" & SVGDbl( r ) & ","
+          & SVGDbl( labelFontHeight * 0.31 ) & ")"
+           &  "'"
+           &   textAnchor
+           & ">"
+           &   SVGStr( labelText )
+           & "</text>"
        cumulativeValue = cumulativeValue + pointValue
    Next i

    svg = svg
+       & "</g>"
+       & "</svg>"

    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
    svgPlane.SetSVG( svg )
End Sub
```

Utilizing chart information

To be able to draw on charts effectively, you must know the size and shape of the chart you are drawing on, or be able to modify the chart so that your drawings are appropriate for the chart. In the example code, the chart needs to be made smaller. The code to do that follows.

```
   Dim pieScale As Double
   pieScale = 0.95
   Dim chartPlane As AcDrawingChartPlane
   Set chartPlane = GetChartDrawingPlane( )
   chartPlane.SetSize( Size.Width * pieScale, Size.Height *
+                  pieScale)
   chartPlane.SetPosition( (Size.Width * (1 - pieScale)) / 2,
+                   (Size.Height * (1 - pieScale)) / 2)
```

This section of code retrieves the chart drawing plane. It then resets the size properties of the chart to be 95% of its original size by calling SetSize. It also then modifies the position of the chart so that it is again recentered.

Once the chart has been resized and recentered, we retrieve the layout of the chart as follows.

```
DescribeLayout( )
Dim pieCenter As AcPoint
pieCenter = baseLayer.GetPieCenter( )
Dim pieRadius As AcTwips
pieRadius = baseLayer.GetPieRadius( )
```

DescribeLayout is a method in AcChart that computes the layout of a chart without rendering it. You can get information about the chart's layout by calling

DescribeLayout( ) then calling methods such as GetPieCenter to retrieve information about the chart being drawn. These methods can only be called from DrawOnChart( ). The available chart information methods are listed in Table 9-2.

Table 9-2 Chart layout methods

Method	Use
GetPieCenter	Returns the position of the center of a pie chart relative to the top left corner of its parent chart's chart drawing plane. You can use this method only for two-dimensional pie charts.
GetPieRadius	Returns the radius of a pie chart. You can use this method only for two-dimensional pie charts.
GetPlotAreaPosition	Returns the position of a chart layer's plot area relative to the top left corner of its parent chart's chart drawing plane. You can use this method only for two-dimensional charts that are not pie charts.
GetPlotAreaSize	Returns the size of a chart layer's plot area. You can use this method only for two-dimensional charts that are not pie charts.

A section of code that retrieves and uses chart information follows.

```
Dim series As AcChartSeries
Set series = baseLayer.GetSeries( 1 )
Dim numberOfCategories As Integer
numberOfCategories = baseLayer.GetNumberOfCategories( )
Dim totalValue As Double
totalValue = series.GetSumOfSliceValues( )
```

GetSeries retrieves information about a particular series within a layer. In this section of code, we retrieve the number of categories and the sum of the slice values for use later in the method for text positioning.

Finally, we retrieve the text and its position on the chart, as shown in the following code.

```
Dim category As AcChartCategory
Set category = baseLayer.GetCategory( i )
Dim labelText As String
labelText = category.GetLabelText( )

Dim point As AcChartPoint
set point = series.GetPoint( i )
```

For more information on the AcChart class, see *Programming with Actuate Foundation Classes.*

Using text metrics

In the previous example, GetFontDisplayHeight is used to properly size and place the text onto the chart. Text metric methods in Actuate Basic are available to get text sizing information which simplifies the task of placing text on drawings or charts. These functions return their values in twips, and should be converted to points for use in SVG. The available text metric methods are described in Table 9-3.

Table 9-3 Text metric methods

Method	Use
GetDisplayHeight	Returns the height required to display a given text string without truncating it.
GetFontAverageCharWidth	Returns the average width of a character in a specified font.
GetFontDisplayHeight	Returns the height needed to display a character in a specified font.
GetTextWidth	Returns the width of a text string.

For more information about text metric functions, and examples of text metric function use, see *Programming with Actuate Basic.*

Examining the SVG code in a drawing control

In e.Report Designer Professional, open \Program Files\Actuate10\eRDPro \Examples\DesignAndLayout\Drawings\3DEffect\3DEffect.rod.

This sample application shows how you can use SVG filters within a drawing to create a 3D Effect. When you open the report, it contains a single frame with a drawing control in it. The control's Finish( ) method is overridden as follows.

```
Sub Finish( )
    ' Get the size of the drawing in points
    Dim w As Double
    w = Size.Width / OnePoint
    Dim h As Double
    h = Size.Height / OnePoint

    Dim svg As String

' Begin header
    svg = "<svg version='1.1'"
+       ' Standard SVG 1.1 namespaces
+       & " xmlns='http://www.w3.org/2000/svg'"
+       & " xmlns:xlink='http://www.w3.org/1999/xlink'"
+       ' Do not collapse whitespace in text
```

```
+       & " xml:space='preserve'"
+       ' Scale the SVG to use points as the default units
+       & " viewBox='0 0 " & SVGDbl( w ) & " " & SVGDbl( h )
+      & "'>"
' End header

' Begin defined reference section
    ' Define a combined drop shadow and specular highlight
    ' filter
    svg = svg
+       & "<defs>"
+      &   "<filter id='3DFilter' filterUnits='userSpaceOnUse'"
+      &   "width='100%' height='100%'>"
+       &     "<feGaussianBlur in='SourceAlpha' "
+      &     "stdDeviation='3' result='blur'/>"
+       &     "<feOffset in='blur' dx='5' dy='5' "
+      &    "result='offsetBlur'/>"
+       &     "<feSpecularLighting in='blur' surfaceScale='5' "
+      &    "specularConstant='0.75' "
+       &    "specularExponent='20' lighting-color='#F8D8D0' "
+       &     "result='specOut'>"
+       &     "<fePointLight x='-2000' y='-4000' z='8000'/>"
+       &     "</feSpecularLighting>"
+       &     "<feComposite in='specOut' in2='SourceAlpha' "
+      &     "operator='in' result='specOut'/>"
+       &     "<feComposite in='SourceGraphic' in2='specOut' "
+      &    "operator='arithmetic' k1='0' k2='1' k3='1' "
+       &     "k4='0' result='litPaint'/>"
+       &     "<feMerge>"
+       &       "<feMergeNode in='offsetBlur'/>"
+       &       "<feMergeNode in='litPaint'/>"
+       &     "</feMerge>"
+       &   "</filter>"
+       & "</defs>"
' End defined reference section

' Begin drawing functions
    ' Background rectangle
    svg = svg
+       & "<rect x='0.5' y='0.5'"
+       &   SVGAttr( "width", w - 1 )
+       &   SVGAttr( "height", h - 1 )
+       & " fill='#808080' stroke='#404040' stroke-width='1'/>"

    ' Offset and apply the filter
    svg = svg
+       & "<g transform='translate(20,20)'>"
+       &   "<g filter='url(#3DFilter)'>"
```

```
    ' Draw the outline
    svg = svg
+       &     "<path fill='none' stroke='#B00000' "
+      &    "stroke-width='10' "
+       &     "d='M50,100 a50,50 0 1,1 0,-100 l130,"
+      &    "0 a50,50 0 1,1 0,100 Z'/>"

    ' Draw the button
    svg = svg
+       &     "<path fill='#B00000' stroke='none' "
+       &     "d='M50,80 a30,30 0 1,1 0,-60 l130,0 a30,"
+      &    "30 0 1,1 0,60 Z'/>"
+       &   "</g>"

    ' Draw the text unfiltered
    svg = svg
+       &   "<g font-size='36' font-family='Verdana' "
+       &   "fill='#602020' stroke='black' "
+      &  "stroke-width='0.67'>"
+       &   "<text x='40' y='60'>Drawing</text>"
+       &   "</g>"
+       & "</g>"

    svg = svg
+       & "</svg>"
' End drawing functions

    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
    svgPlane.SetSVG( svg )

    Super::Finish( )
End Sub
```

The overridden Finish( ) method generates a string containing SVG commands and sets it into an SVG drawing plane. It then calls Super::Finish() to render the image onto the display.

The SVG string is in several sections, all concatenated together. This example has a header, defined reference, and drawing function sections.

Examining the SVG header information

The SVG header contains information about name spaces and the coordinate system to be used.

```
svg = "<svg version='1.1'"
+       ' Standard SVG 1.1 namespaces
+       & " xmlns='http://www.w3.org/2000/svg'"
+       & " xmlns:xlink='http://www.w3.org/1999/xlink'"
+       ' Do not collapse whitespace in text
+       & " xml:space='preserve'"
+       ' Scale the SVG to use points as the default units
+       & " viewBox='0 0 " & SVGDbl( w ) & " " & SVGDbl( h )
+      & "'>"
```

This example uses two name spaces. The default namespace that identifies SVG, www.w3.org/2000/svg, and a linking namespace for creating references to existing SVG code, www.w3.org/1999/xlink.

Xml:space is an attribute that helps applications determine whether they should pay attention to white space. In this case, we want to preserve white space, and not collapse it.

The viewBox defines the internal coordinate system used by anything within the width and height of the SVG element. The values represent the coordinates of the left top corner, followed by the width and height of the bounding box. SVGDbl is called to convert the values of the size of the control to those appropriate to SVG.

Examining defined referenced elements

The defs element is a container element for referenced elements. These elements are not displayed when they are defined, but are used in references in later sections.

This part of the example uses the defs section to create a chained filter that creates a drop shadow and lighting effects.

```
' Define a combined drop shadow and specular highlight filter
    svg = svg
+       & "<defs>"
+       &   "<filter id='3DFilter' filterUnits='userSpaceOnUse'
   width='100%' height='100%'>"
+       &     "<feGaussianBlur in='SourceAlpha' "
+      &     "stdDeviation='3' result='blur'/>"
+       &     "<feOffset in='blur' dx='5' dy='5' "
+      &    "result='offsetBlur'/>"
+       &     "<feSpecularLighting in='blur' surfaceScale='5' "
+      &    "specularConstant='0.75' "
+       &    "specularExponent='20' lighting-color='#F8D8D0' "
+       &     "result='specOut'>"
+       &     "<fePointLight x='-2000' y='-4000' z='8000'/>"
+       &     "</feSpecularLighting>"
+       &     "<feComposite in='specOut' in2='SourceAlpha' "
+      &     "operator='in' result='specOut'/>"
+       &     "<feComposite in='SourceGraphic' in2='specOut' "
+      &    "operator='arithmetic' k1='0' k2='1' k3='1' "
+       &     "k4='0' result='litPaint'/>"
+       &     "<feMerge>"
+       &       "<feMergeNode in='offsetBlur'/>"
+       &       "<feMergeNode in='litPaint'/>"
+       &     "</feMerge>"
+       &   "</filter>"
+       & "</defs>"
```

A filter is a transformation that changes an image prior to it being rendered. A filter is prefixed with the characters fe which stands for filter effect. SVG has several different primitive filters that modify the image in various ways. They can be chained together, causing their effects to be cumulative. By applying these filters in different combinations and chains you can create a much larger set of usable filters. The filters used in this example are feGaussianBlur, feOffset, fePointLight, feSpecularLighting, feComposite, and feMerge.

The filter element itself appears as follows.

```
<filter id='3DFilter' filterUnits='userSpaceOnUse'
   width='100%' height='100%'>
```

The id attribute specifies the name of the filter, 3DFilter. This name is used to reference the filter in other sections of the SVG code. userSpaceOnUse specifies that the filter uses the coordinate system that is in effect when the filter is being called. Width and height specify how large an area the filter covers.

Following the filter tag, a set of filter primitives are used to modify the image, starting with feGaussianBlur. The key attributes for chaining filters are in and result. In is assigned

the name of the image to modify, and result is where the transformed image is placed. The resulting image can then be used as input into another transformation. For example, feGaussianBlur takes SourceAlpha as its in value, and has blur as its result value. The filter feOffset then takes the blur result from feGaussianBlur as its in value, and produces its result in offsetBlur. The chain can continue on in this way until the final desired effect is achieved.

Two special values are used for the original source image for in values. They are SourceGraphic and SourceAlpha. SourceGraphic represents the original image that is being transformed, and SourceAlpha represents the alpha channel, or mask, of that image. A mask is a grayscale bitmap image. The pure white areas in the image represent the portions of your original image that will be 100% protected when combined with a second image. The pure black portions of the image represent the areas of your original image that are completely masked out, or erased. Masks support the creation of soft fades, decorative edges, and translucent effects.

Examining drawing functions

After setting values for the header, and defining the filter, the elements are placed into the image. The first image to be drawn is a grey rectangle.

```
' Background rectangle
    svg = svg
+       & "<rect x='0.5' y='0.5'"
+       &   SVGAttr( "width", w - 1 )
+       &   SVGAttr( "height", h - 1 )
+       & " fill='#808080' stroke='#404040' stroke-width='1'/>"
```

There are several shape primitives in SVG, including rect, line, circle, ellipse, path, and polygon. A path represents the outline of a shape which can be filled, stroked, used as a clipping path, or any combination of the three. Once the background rectangle is drawn, two groups of elements are created with the g tag.

```
svg = svg
+       & "<g transform='translate(20,20)'>"
+       &   "<g filter='url(#3DFilter)'>"
```

The first grouping translates all the elements within the group by 20 in both the x and y directions. The second grouping utilizes the 3DFilter created in the defs section. All geometry created within this grouping will have the chained filter 3DFilter applied to it.

After the two path elements are created which draw the outline and the button, the section closes the grouping that has the filter applied to it.

```
' Draw the outline
    svg = svg
+       &     "<path fill='none' stroke='#B00000' "
+       &    "stroke-width='10' "
+       &     "d='M50,100 a50,50 0 1,1 0,-100 l130,"
+       &    "0 a50,50 0 1,1 0,100 Z'/>"

    ' Draw the button
    svg = svg
+       &     "<path fill='#B00000' stroke='none' "
+       &     "d='M50,80 a30,30 0 1,1 0,-60 l130,"
+       &    "0 a30,30 0 1,1 0,60 Z'/>"
+       &   "</g>"
```

Since the filter grouping is closed, the text that is drawn next is unfiltered, but still translated. However, before the text is drawn, a new group is created that specifies a font. The text is drawn in that font, and then the font group is closed. The translation font is next closed. Finally, the SVG tag is closed.

```
' Draw the text unfiltered
    svg = svg
+       &   "<g font-size='36' font-family='Verdana' "
+       &   "fill='#602020' stroke='black' "
+       &  "stroke-width='0.67'>"
+       &     "<text x='40' y='60'>Drawing</text>"
+       &   "</g>"
+       & "</g>"

    svg = svg
+       & "</svg>"
```

Once the SVG string has been completed, an AcDrawingSVGPlane class is created. It is added to the drawing plane, and the SVG text is set into it. Super::Finish( ) is called to render the image.

```
Dim svgPlane As AcDrawingSVGPlane
Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
svgPlane.SetSVG( svg )

Super::Finish( )
```

Debugging an SVG program

DebugSVG.rol is included in the DebuggingSVG.rod, located at \Program Files \Actuate10\eRDPro\Examples\DesignAndLayout\Drawings\DebuggingSVG. DebugSVG.rol contains a control named DebugSVG. This control is a subclass of

AcTextControl which has been customized to extract the SVG text from a drawing plane and display it. In addition, the control can send the SVG text to e.Report Designer Professional's Output window or dump it out to a text file. The behavior of the control can be selected based on certain properties, as shown in Table 9-4.

Table 9-4 DebugSVG properties

Property	Use
DrawingClassName	Set this property to the name of the drawing control that you want to debug.
DrawingPlaneIndex	Set this property to the index of the drawing plane that you want to debug.
DumpFileName	Set this property to the name of the file to which you want to dump the SVG text. If this property is blank, the SVG text will not be dumped to a file. A sequence number and the .svg extension will be added automatically to the file name you specify.
DumpFileFixSize	Set this property to true to add absolute size information to the SVG text which is dumped. If this property is false, the SVG in the dump file will scale automatically when you view it using an external SVG viewer.
ShowInOutputPane	Set this property to true to display the SVG text in e.Report Designer Professional's output window.

Examining the DebugSVG control

To view the DebugSVG control, open \Program Files\Actuate10\eRDPro \Examples\DesignAndLayout\Drawings\DebuggingSVG\DebuggingSVG.rod. The Finish( ) method for the DebugSVG text control is overridden as follows.

```
Sub Finish( )
    Static s As Integer

    ' Find the drawing
    Dim drawing As AcDrawing
    Set drawing = GetContainer( ).FindContentByClass(
+                    DrawingClassName )

    ' Get the SVG drawing plane
    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = drawing.GetDrawingPlane( DrawingPlaneIndex )

    ' Extract the SVG
    Dim svg As String
    svg = svgPlane.GetSVG( )

    ' Display the SVG in this control
    DataValue = svg

    ' Dump the SVG to a file
    If (DumpFileName > "") Then
        Dim f As Integer
        f = FreeFile( )
        s = s + 1
        Open DumpFileName & Format( s, "_000" ) & ".svg"
+          For Output As f

       If DumpFileFixSize Then
           ' Adjust the SVG to specify absolute size so that
           ' the SVG debug file will display without scaling
            Dim svgElementEnd As Integer
            svgElementEnd = InStr( svg, ">" )
            svg = Left( svg, svgElementEnd - 1 )
+               & " width='" & SVGDbl( svgPlane.Size.Width /
+                         OnePoint ) & "pt'"
+               & " height='" & SVGDbl( svgPlane.Size.Height /
+                         OnePoint ) & "pt'"
+               & Mid( svg, svgElementEnd )
        End If
        Print #f, svg
        Close f
    End If

    ' Display the SVG in eRD Pro's Output pane
    If ShowInOutputPane And (GetAppContext( ) = DWBContext)
```

```
    Then
         ShowFactoryStatus( svg )
      End If

      Super::Finish( )
End Sub
```

The function retrieves the SVG code from the control, and displays it in one of several ways, based on property settings of the control. The first section of code retrieves the drawing control as specified by the DrawingClassName property.

```
    Dim drawing As AcDrawing
    Set drawing = GetContainer( ).FindContentByClass(
+                   DrawingClassName )
```

The next section retrieves the SVG drawing plane based on the DrawingPlaneIndex property.

```
Dim svgPlane As AcDrawingSVGPlane
Set svgPlane = drawing.GetDrawingPlane( DrawingPlaneIndex )
```

Once the SVG drawing plane is retrieved, GetSVG is called.

```
Dim svg As String
svg = svgPlane.GetSVG( )
```

After the SVG code is retrieved, the property settings of the DebugSVG control determine where to place the output.

Using the DebugSVG control

DebuggingSVG.rod consists of one frame containing a drawing control and a DebugSVG control, a subclass of the DebugSVG library component. It appears in Figure 9-2. The SVG generating code written in Actuate Basic is located within the Finish( ) method of the drawing control.

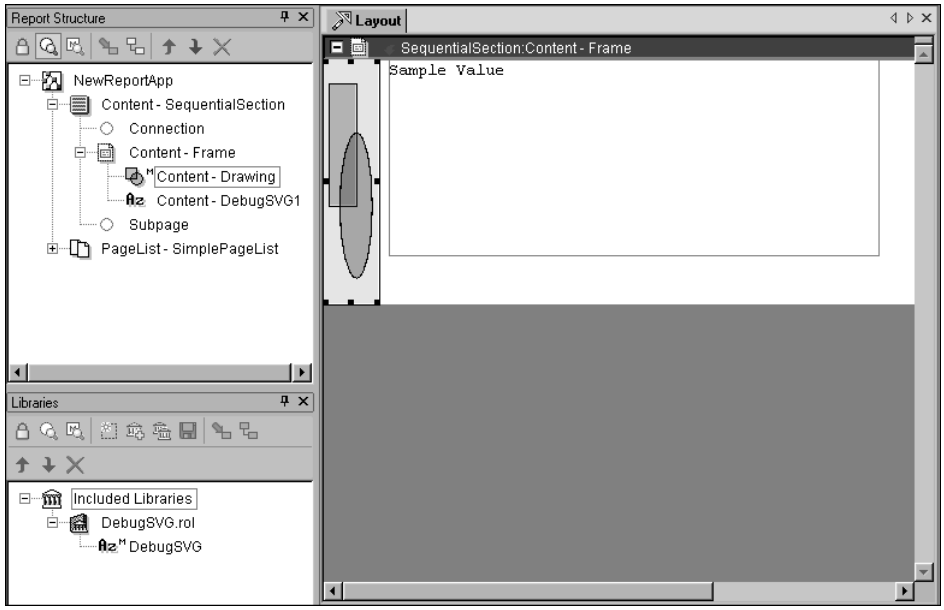

Figure 9-2 DebugSVG report design

How to examine the drawing control's SVG generating code

1. Select the drawing control in Layout.
2. Go to Properties—Methods. Open Sub Finish( ). Drawing::Finish appears, as shown below.

```
Sub Finish( )
    ' Get the size of the drawing in points
    Dim w As Double
    w = Size.Width / OnePoint
    Dim h As Double
    h = Size.Height / OnePoint

    Dim svg As String
    svg = "<svg version='1.1'"
+       ' Standard SVG 1.1 namespaces
+       & " xmlns='http://www.w3.org/2000/svg'"
+       & " xmlns:xlink='http://www.w3.org/1999/xlink'"
+       ' Do not collapse whitespace in text
+       & " xml:space='preserve'"
+       ' Scale the SVG to use points as the default units
+       & " viewBox='0 0 " & SVGDbl( w ) & " " & SVGDbl( h )
+      & "'>"

    ' Rotate and shift the coordinate system
    svg = svg
```

```
+       & "<g"
+       & " transform='"
+       &  " translate(" & SVGDbl( w - 20 ) & "," &
+      &  SVGDbl(h - 6 ) & ")"
+       &  " rotate(90)"
+       & "'>"

    ' Place some text
    svg = svg
+       & "<text font-family='Arial' font-size='20' "
+      & "text-anchor='end'>Hello World!</text>"

    svg = svg
+       & "</g>"
+       & "</svg>"

    Dim svgPlane As AcDrawingSVGPlane
    Set svgPlane = AddDrawingPlane( DrawingPlaneTypeSVG )
    svgPlane.SetSVG( svg )

    Super::Finish( )
End Sub
```

The SVG generated by this code creates the phrase Hello World! rotated 90 degrees. When you run the report, it appears as in Figure 9-3.

Figure 9-3 Report showing SVG code

The SVG code in the Finish( ) method generated the image. The SVG code that generated the image is displayed to the right of the image in the DebugSVG control.

At this point, you have the ability to observe the code and the resulting image created from it. Depending on the values you have set for the DebugSVG properties, you may also export the SVG code to other software tools to do further analysis.