

Adding Style to Reports

Dave Gannon and Nich Mann



Tired of creating boring reports? Dave Gannon and Nich Mann show you how to add some visual pizzazz.

For as we have, you've been working with Access for eight years, you get fed up with producing the same tired looking reports. You know the ones—they're very business-like and feature horizontal double-lines everywhere to separate data rows (yawn). You create reports like this because your users want to see reports like this—it's a very *professional* look. But, as developers, we know that other styles can be created that look just as effective. In this article we'll show you how to add an interesting visual to your reports. You'll also learn how to use one of Access's graphical functions (in depth!) and get a routine that will automate the whole process for you. Spicing up your reports can be as simple as adding this routine to your project and calling it.

Figure 1 shows the kind of graphics that we've been adding to our reports.

There are several ways in which we could have achieved our objective. One method is to use graphics embedded in the report. We decided against this, however, as graphics either rely on external files (dangerous if the file gets moved) or are pasted directly onto reports (which can bloat your database). We decided to create our graphics by drawing directly onto the report using the `Line()` and `Circle()` methods.

Drawing circles and arcs

Drawing a circle, arc, or pie slice is pretty straightforward, once you know how. The key is to understand how the `Circle` statement draws these objects. The statement is defined as:

```
object.Circle [Step](x, y), radius[, [color][,
[start][, [end][, aspect]]]]
```

The parameters of the `Circle` method are:

- *x* and *y* represent the center of the circle or arc.
- *radius* is the radius length.
- *color* is the color that will be used to draw the arc.
- *start* and *end* are the start angle and end angle (in radians) of the arc.
- *aspect* determines whether a circle or elliptical shape is drawn.

Microsoft Help also mentions that the `Circle`

statement expects radians to define start and end points, and that the limits are -2 or $+2$ radians.

That's all very interesting, but it's less than helpful (what exactly is a radian?). Here's how you really use the `Circle` function. We normally specify and think of angles in degrees, not radians—we know that a 90-degree angle is a right angle, for instance. So we need a method to convert degrees to radians. The formula in VBA is:

```
Const PI = 3.1415926
angle = degrees * (-2 * PI / 360)
```

A circle is defined as a maximum of $-2 * \pi$ radians, so to find 1 degree the code divides the maximum number of radians by 360. This number is then multiplied by the number of degrees we want for the angle of any arc that we draw. For instance, to define a semicircle we'd use this code to create an arc for half a circle:

```
Const PI = 3.1415926
Dim startAngle as single, endAngle as single

startAngle = 90 * (-2 * PI / 360)
endAngle = 270 * (-2 * PI / 360)
```

The only other thing you need to remember is that the resulting angles from this formula will be negative. According to Help, if the start and end angles are negative, you get a pie slice. If they're positive, you get an arc. As an example, the following code generates a negative angle (and the result shown in Figure 2):

Name	Date of Birth	Address	Salary
Information Systems			
William DeWitt	27 Aug 1955	The Lanes	417,800.00
David Smithfield	04 May 1965	18 Carson Court	419,000.00
Sally Fradley	10 Oct 1976	584 Raven Gardens	421,300.00
Arthur Sagely	01 Jan 1982	27 Highbrook Drive	448,000.00
			€102,800.00
Accounts			
Kate Drayton	08 May 1988	48 Palm Hill Mount	422,000.00
Toby Cornthorpe	05 Dec 1990	5 The Drawings	428,000.00
			€850,000.00
Field Services			
Lenny Blunden	08 May 1988	4 Parkfield Grove	418,800.00
			€18,800.00

Figure 1. Formatting style 1.

```

Const PI = 3.14159265359
Const RADS_TO_DEGREES = (-2 * PI / 360)

Dim sngLeftStart As Single, sngLeftEnd As Single

sngLeftStart = 90 * RADS_TO_DEGREES
sngLeftEnd = 270 * RADS_TO_DEGREES

Me.Circle (2000, 2000), 1000, vbBlack, _
    (sngLeftStart), (sngLeftEnd)

```

However, if we make the start and end angles positive, we simply get an arc (as shown in [Figure 3](#)):

```

Me.Circle (2000, 2000), 1000, vbBlack, _
    (sngLeftStart) * -1, (sngLeftEnd) * -1

```

Now that you know this, you can draw a circle or part of a circle that's filled or unfilled. The way that a slice is filled depends upon two other factors: the FillColor and the FillStyle of the report. We demonstrate filled circles in the code later in this article.

Determining position

The next step is to determine where to draw the circle/arc. We wanted to add a rounded feel to a rectangular control, so we first had to figure out where on the page the control was. This is easily achieved using the Top, Width, and Left properties of the control, as this example shows:

```

Dim sngLeftX As Single
Dim sngRightX As Single
Dim sngTopY As Single
Dim sngBottomY As Single

sngLeftX = ctl.Left
sngRightX = ctl.Left + ctl.Width
sngTopY = ctl.Top
sngBottomY = ctl.Top + ctl.Height

sngTopPos = ctl.Top + (ctl.Height / 2)

```

In the last line we took the location of the top of our control and added to it half of the control's height. This gives us the mid-point vertically inside the control.

You also need to know the fill color of the control, so that the arc ends will seamlessly match the control's appearance. We retrieved that color with this code:

```

Dim lFillColor As Long
lFillColor = ctl.BackColor

```

Now that we know the position and the color, we can

Figure 2.
Pie slice example.



draw the arc using the angles that we've defined. First, we did all the necessary calculations and gathered the color information:

```

' variables for angles of arcs
Dim sngLeftStart As Single
Dim sngLeftEnd As Single
Dim sngRightStart As Single
Dim sngRightEnd As Single
Dim sngTopPos As Single

sngLeftStart = 90 * RADS_TO_DEGREES
sngLeftEnd = 270 * RADS_TO_DEGREES
sngRightStart = 270 * RADS_TO_DEGREES
sngRightEnd = 90 * RADS_TO_DEGREES

rpt.FillColor = lFillColor
rpt.FillStyle = 0

```

We then used that information to draw our arcs:

```

rpt.Circle (ctl.Left, sngTopPos), _
    ((ctl.Height / 2) - 1), lFillColor
rpt.Circle ((ctl.Left + ctl.Width), sngTopPos), _
    ((ctl.Height / 2) - 1)

```

That's really all there is to it. We made the report look even snazzier by drawing drop-shadows and an outline for the control. However, these are just more arcs in a different color, so the principle is the same.

Automating the process

It soon became obvious that writing this amount of code for each control would be more work than we had time for. So we created a simple procedure that can be called from the Format event of a section of the report. The routine is defined as follows:

```

Public Sub DrawElliptoid (Rpt As Report, _
    ctl As Control, bOutline As Boolean, _
    bShadow As Boolean)

```

You can call the routine from the Format event of a section like this:

```

Private Sub GroupHeader1_Format(Cancel As Integer, _
    FormatCount As Integer)
Dim ctl As Control

Set ctl = Me.rectNameDetail
DrawElliptoid Reports!rptstylisedformatting, _
    ctl, True, True

End Sub

```

The procedure then draws the arc ends on the

Figure 3.
Arc example.



specified control, with a drop-shadow and an outline. Have a look at the sample database (available in the Download file at www.vb123.com/kb) and you'll see how the code draws the various components to create a very attractive rounded control.

There are a few problems with this approach, though. For instance, when previewing on screen, the ends look ragged and seem to be out of line with the control. This is just a problem with Print Preview, and the lines will print correctly when you send the report to paper. More seriously, you'll notice that when you've defined an outline for a control, the top line for a control won't be visible when printing. For reports where this occurs, you'll have to abandon the outline.

Here's all the code for our routine. We start by declaring the variables that we'll use and setting up the variables that we'll need later in the code:

```
Const PI = 3.14159265359
Const RADS_TO_DEGREES = (-2 * PI / 360)
Const OFFSET = 100

Dim sngLeftStart As Single
Dim sngLeftEnd As Single
Dim sngRightStart As Single
Dim sngRightEnd As Single
Dim sngTopPos As Single

Dim sngLeftX As Single
Dim sngRightX As Single
Dim sngTopY As Single
Dim sngBottomY As Single
Dim lFillColor As Long
```

The next step is to load the variables with the start positions for our arcs:

```
sngLeftStart = 90 * RADS_TO_DEGREES
sngLeftEnd = 270 * RADS_TO_DEGREES
sngRightStart = 270 * RADS_TO_DEGREES
sngRightEnd = 90 * RADS_TO_DEGREES

sngLeftX = ctl.Left
sngRightX = ctl.Left + ctl.Width
sngTopY = ctl.Top
sngBottomY = ctl.Top + ctl.Height

sngTopPos = ctl.Top + (ctl.Height / 2)
```

The next line sets the fill color of our arc to the same color as the control's back color. After that, we check to see whether the user wants shading. If the shading parameter is set, we draw the shadows first. We first draw the straight lines and then draw the arcs. Each shadow line is drawn in the shadow color (vbBlack), slightly offset from the line what we're going to draw next:

```
lFillColor = ctl.BackColor

If bShadow = True Then
    Rpt.Line (sngLeftX + OFFSET, sngTopY + OFFSET) _
        -(sngRightX + OFFSET, sngBottomY + OFFSET), _
        vbBlack, BF
    Rpt.FillColor = vbBlack
    Rpt.FillStyle = 0

    Rpt.Circle (ctl.Left + OFFSET, _
        sngTopPos + OFFSET), (ctl.Height / 2), _
        vbBlack, sngLeftStart, sngLeftEnd
```

```
Rpt.Circle ((ctl.Left + ctl.Width) + OFFSET, _
    sngTopPos + OFFSET), (ctl.Height / 2), _
    vbBlack, sngRightStart, sngRightEnd
End If
```

We're now ready to draw our first set of arcs. There are actually two sets of code here—one set when we're drawing an outline and one when we're not. If the bOutline parameter is true, we draw the arc in black to give a strong border. If the bFill parameter is false, the arc lines are drawn using black and then filled with the color of the control:

```
If bOutline = True Then
    Rpt.FillColor = lFillColor
    Rpt.FillStyle = 0
    Rpt.Circle (ctl.Left, sngTopPos), _
        ((ctl.Height / 2) - 1), lFillColor, _
        sngLeftStart, sngLeftEnd
    Rpt.Circle ((ctl.Left + ctl.Width), _
        sngTopPos), ((ctl.Height / 2) - 1), _
        lFillColor, sngRightStart, sngRightEnd
```

The next step is to draw the top and bottom lines in black. We add an arbitrary offset when drawing the top, because if the line is drawn along the exact top of the control it doesn't draw cleanly:

```
Rpt.Line (sngLeftX, sngTopY) _
    -(sngRightX, sngTopY), vbBlack
Rpt.Line (sngLeftX, sngBottomY) _
    -(sngRightX, sngBottomY), vbBlack
```

The next set of lines draw the arc outlines in black. We multiply the start and end angles by -1 to suppress the fill and draw only the arc:

```
Rpt.Circle (ctl.Left, sngTopPos), _
    (ctl.Height / 2), vbBlack, _
    (sngLeftStart) * -1, (sngLeftEnd) * -1
Rpt.Circle ((ctl.Left + ctl.Width), _
    sngTopPos), (ctl.Height / 2), _
    vbBlack, (sngRightStart) * -1, _
    (sngRightEnd) * -1
```

Here's the code when there is no outline:

```
Else
    Rpt.FillColor = lFillColor
    Rpt.FillStyle = 0
    Rpt.Circle (ctl.Left, sngTopPos), _
        ((ctl.Height / 2) - 1), lFillColor, _
        sngLeftStart, sngLeftEnd
    Rpt.Circle ((ctl.Left + ctl.Width), _
        sngTopPos), ((ctl.Height / 2) - 1), _
        lFillColor, sngRightStart, sngRightEnd
End If

End Sub
```

Armed with this routine, you're ready to make your reports more interesting. ▲

 [CIRCLE.ZIP at www.vb123.com/kb](http://www.vb123.com/kb)

Dave Gannon and Nich Mann have more than 15 years of Access development experience between them. They currently work in Harrogate, England, producing bespoke sharescheme administration solutions for Howells Associates Ltd.