SoftCraft Graphics Custom Control

The SCGraphic custom control can be used in Visual Basic versions 1 through 3 and Visual C++. The control provides basic shapes of rectangle, ellipse, polyline, polygon, arc, pie and a regular n-sided polygon (the Shape property determines what shape the control displays). Any closed shape can be filled with a variety of patterns, included graduated fills from one color to another. In Visual Basic versions 2 and 3, the colors can be pure 256-color palette colors.

Properties

AngleEnd, AngleStart, ArrowSize, ArrowType, DragIcon, DragMode, DrawInside, FillColor, FillColor2, FillPattern, Height, Index, InhibitEraseOnRedraw, Left, LineColor, LinePattern, LineWidth, MouseEvents, Name, NumPoints, PaletteSteps, RoundRadius, SelectByInk, ShadowColor, ShadowDepthX, ShadowDepthY, Shape, ShowOutlineOnly, Tag, Top, Use256Palette, Visible, Width

Polyline/Polygon Properties

Other Topics

Events

Printing

Sample Code

Runtime Distribution and License Information

Address

SoftCraft, Inc.
16 N. Carroll Street, Madison, WI 53703
Sales: 800-351-0500
Support: 608-257-3300

Runtime Distribution and License Information

To distribute your Visual Basic or Visual C++ application with a SoftCraft Graphic Custom Control, you need to also distribute the runtime file for the control, SCGrphic.vbx. You may distribute the SCGrphic.vbx file without charge as long as it is not modified in any way.

To use the SoftCraft Graphic Custom Control in design mode, you must have the license file, SCLic.dll, located in the same directory as the SCGraphic.vbx file. If you get a message about a missing license file, you must place the license file in the proper directory and then exit Windows and restart.

Note: You are *not* allowed to include SCLic.dll with any application that you develop and distribute; only SCGrphic.vbx may be distributed.

AngleEnd

Ending angle for a pie or arc. Measured in degrees counter-clockwise from horizontal.

The boundary of a pie or arc is a portion of an ellipse that fills the containing rectangle of the control.

Also see AngleStart and Shape.

AngleStart

Starting angle for a pie or arc. Also the location for the center of the first side in an Ngon. Measured in degrees counter-clockwise from horizontal.

The boundary of a pie or arc is a portion of an ellipse that fills the containing rectangle of the control.

Also see AngleEnd and Shape.

ArrowSize

Relative size of the arrowhead on a polyline shape. The normal value is 3. Larger values produce larger arrowheads and smaller values produce smaller arrowheads.

The ArrowType property must be set to indicate upon which end(s) the arrowhead should appear.

The pre-defined values are:

```
0
1 - Tiny
2 - Small
3 - Normal
4
5 - Large
6
7
8 - Huge
```

(but you can assign even larger values if you wish).

ArrowType

Specify whether arrowheads are to be placed on either end or both ends of a polyline. Use the ArrowSize property to adjust the size of the arrowhead.

The possible values are:

```
0 - None
1 - Forward
2 - Backward
3 - Both
```

DragIcon

Standard VB property (see Visual Basic Help).

DragMode

Standard VB property (see Visual Basic Help).

DrawInside

Normally, if a shape has a thick border (LineWidth), half of the border line is drawn outside of the shape and half is drawn inside. For example, in the case of a rectangle, half of the border line would fall outside the bounding area for the rectangle. This is usually desireable and works well for VB versions 2 and later. For VB version 1 and VC++, however, all controls must be drawn completely inside the containing rectangle for the control.

If this property is set to true, the shape is reduced in size so that the border line and the shadow (ShadowDepthXand ShadowDepthY) fit within the containing rectangle for the shape.

Note: Bezier curves with large curvature control handles may extend outside of the bounding area for the shape. For VB version 1 and VC++ you must position the polyline points within the bounding area far enough to accomodate the curvature (i.e., coordinates sufficiently greater than 0 and less than 1000). See the PolyLine/Polygon Properties and associated examples.

This property is FALSE by default in VB version 2 and later and is TRUE in VB version 1 and VC++.

FillColor

Color of a closed shape. If the FillPattern is a graduated fill, this is the top, left or center color depending on the pattern. If the FillPattern is a hatch pattern, this is the color of the hatch lines.

FillColor2

Second color of a closed shape that has a graduated or hatched FillPattern. If the FillPattern is a graduated fill, this is the bottom, right or outside color depending on the pattern. If the FillPattern is a hatch pattern, this is the color of the background behind the hatch lines.

FillPattern

See the property table for the list of possible solid, hatched and graduated fill patterns. This is ignored for open shapes like polylines and arcs.

Use the FillColor and FillColor2 properties to set the colors of a graduated fill and the hatch/background colors.

The PaletteSteps property controls the smoothness of graduated fills.

The Use256Palette property determines whether dithered colors or pure colors are used on 256-color paletteized devices.

The possible values are:

```
0 - Solid
1 - Clear
5 - Hatch Horizontal
6 - Hatch Vertical
7 - Hatch Diagonal Forward
8 - Hatch Diagonal Backward
9 - Hatch Cross
10 - Hatch Diagonal Cross
16 - Graduated Vertical
17 - Graduated Horizontal
18 - Graduated Elliptical
19 - Graduated Down Right
20 - Graduated Down Left
```

Height

Standard VB property (see Visual Basic Help).

Index

Standard VB property used for setting values in a property array (see Visual Basic Help).

InhibitEraseOnRedraw

Normally, when you change a property of a shape, the entire containing rectangle of the shape is erased (to the background color of the containing control) and then the shape is redrawn.

If you set this property to TRUE, the background is not erased and the shape is drawn over whatever happens to be there. In many cases, this is what you want. For example, if you simply change the FillColor or LineColor of a shape, without moving or sizing it, you do not need to erase the shape before redrawing it because it will be redrawn in exactly the same position.

If your shapes appear to be flashing more than you expect, try setting this property to TRUE to minimize flashing.

Note: If your shapes are on top of another control, such as a panel or picture, those controls will be redrawn before the shape is redrawn regardless of the value of this property. When the parent control is redrawn it will erase any overlapping shapes. Therefore, to use this property effectively, the shapes should be placed directly on the form background or on a control that has a transparent fill mode.

Left

Standard VB property (see Visual Basic Help).

LineColor

The color of the line (or border) outlining the shape. This is ignored if the LinePattern is Clear.

LinePattern

See the property table for the list of solid and dashed line patterns.

Note: Some display and printer devices cannot display wide lines (more than one pixel) unless the line is solid. In this case the device may default to solid lines.

See the other Line properties: LineColor and LineWidth.

The possible values are:

```
0 - Solid
1 - Dashed
2 - Dotted
3 - Dash Dot
4 - Dash Dot Dot
5 - Transparent
```

LineWidth

Thickness of the line (or border) outlining the shape. Like Height and Width, this is a scalable distance number that is automatically adjusted when the form scale mode is changed. That is, it is NOT pixels unless pixels is chosen as the scale mode for the form.

Note: Some display and printer devices cannot display wide lines (more than one pixel) unless the line is solid. In this case the device may default to solid lines.

Also see the DrawInside property for information on how the border is actually drawn with respect to the size of the control.

MouseEvents

Set to true if you want the control to generate mouse events, such as Click, MouseUp, MouseDown, MouseMove, DragDrop, etc. If you set this to false, the control will not generate mouse events and therefore is not selectable and will never be the target of a DragDrop.

See Events.

Name

Standard VB property (see Visual Basic Help).

NumPoints

The number of points in an Ngon, Polyline or Polygon. Note that in a Polygon, the first and last points are joined with a line segment. That is, NumPoints also specifies the number of sides in a Polygon.

Note: NumPoints must be set **before** the locations of the individual points are specified (see the example in SampleCode and the Polyline/Polygon Properties).

PaletteSteps

Determines the number of bands in a graduated fill. The normal value is 20. Larger values produce smoother graduations, but require more processing time and use up more system colors on a 256-color palettized device.

When printing to a color printer, you should increase the PaletteSteps property for smooth graduations; a value of 80 steps per inch works very well for all color devices.

Low-resolution (300 d.p.i. or less) monochrome printers do not provide many gray levels, so a small value (perhaps 10 steps per inch) is adequate for these devices. High-resolution monochrome devices, such as typesetters, benefit from larger values (e.g., 80 steps per inch).

Also see the Use256Palette property.

RoundRadius

For rectangles, Ngons, polylines and polygons, this rounds the corners with the specified radius. Like Height and Width, this is a scalable distance number that is automatically adjusted when the form scale mode is changed.

SelectByInk

Set to true if you want the user to select the shape (generate a Click event) by the colored pixels in the shape. If false, a Click event is generated whenever the user clicks in the rectangular area of the shape. This is very useful if you have a lot of shapes in an area or have overlapping shapes. However, it can be difficult to select transparent shapes with this flag set because the user then has to click on the border of the shape.

The ACCMOVE.FRM demo form in the Sample Code shows how this property affects the users selection of a control.

ShadowColor

If ShadowDepthX or ShadowDepthY is set, a shadow of the object is drawn in this color.

Also see the DrawInside property for information on how the shadow is actually drawn with respect to the size of the control.

ShadowDepthX

Horizontal offset distance of the shadow from the shape. Units are in Twips.

If the shape has a large LineWidth and a small shadow, the shadow may be hidden under the shapes border.

Also see the DrawInside property for information on how the shadow is actually drawn with respect to the size of the control.

ShadowDepthY

Vertical offset distance of the shadow from the shape. Units are in Twips and positive Y is down.

If the shape has a large LineWidth and a small shadow, the shadow may be hidden under the shapes border.

Also see the DrawInside property for information on how the shadow is actually drawn with respect to the size of the control.

Shape

Specifies the shape. Possible values are rectangle, ellipse, polyline, polygon, arc, pie and a regular n-sided polygon.

An arc or pie is created by selecting a portion of an ellipse: the bounding rectangle of the shape specifies the size of the ellipse and the AngleStart and AngleEnd properties specify the portion of the ellipse that is used for the arc or pie.

Polygons and Polylines are defined by a series of points positioned within the bounding rectangle of the shape (see Polyline/Polygon Properties). For Polylines and Polygons, the NumPoints property must be set before setting the individual point locations.

The possible values are:

```
0 - Rectangle
1 - Ellipse
2 - Polyline
3 - Polygon
4 - Arc
5 - Pie
6 - Ngon
```

ShowOutlineOnly

If true, the shape is drawn very rapidly using a transparent fill pattern and a thin border. This is useful in draft modes or when moving a shape.

An example of the use of this property is shown in the ACCMOVE.FRM demo form in the Sample Code.

Tag

Standard VB property (see Visual Basic Help).

Top

Standard VB property (see Visual Basic Help).

Use256Palette

Flag indicating whether to use palettized 8-bit colors or dithered 4-bit colors in a 256-color device. This flag has no effect on a 4-bit (standard VGA) or 16/24-bit color device (deep color devices always use pure colors). You can adjust the smoothness of the color bands with the PaletteSteps property.

Note: Printers are never palettized so you do not need to adjust this property when printing; it will always be ignored.

Visible

Standard VB property (see Visual Basic Help).

Width

Standard VB property (see Visual Basic Help).

Polyline/Polygon Properties

The locations for the points in a polyline or polygon are specified using the following property arrays: Point locations are specified in a 1000x1000 coordinate space that is scaled to the size of the control rectangle. The 0,0 point is the top-left corner and the 1000,1000 point is the bottom-right corner.

To specify a Polyline or Polygon, you must first set the NumPoints property and then set the individual point locations and, optionally, the Bezier control handle offsets.

PointX(n): X location of a point.

PointY(n): Y location of a point.

PointXOffsetIn(n): Bezier curvature control handle for the segment coming into the point. This is specified as an X offset from the point using the same 1000x1000 coordinate system. The special value of 32760 is used to get an auto-curvature value that provides nice curvature based on the locations of adjacent points (the YOffset is ignored if the XOffset has the special value)..

PointYOffsetIn(n): Bezier curvature control handle for the segment coming into the point. This is specified as a Y offset from the point using the same 1000x1000 coordinate system.

PointXOffsetOut(n): Bezier curvature control handle for the segment coming out of the point. This is specified as an X offset from the point using the same 1000x1000 coordinate system. The special value of 32760 is used to get an auto-curvature value that provides nice curvature based on the locations of adjacent points (the YOffset is ignored if the XOffset has the special value).

PointYOffsetOut(n): Bezier curvature control handle for the segment coming out of the point. This is specified as a Y offset from the point using the same 1000x1000 coordinate system.

See the code in the Load event of SAMPLES.FRM below (in Sample Code) for examples of how these properties are set.

Events

The shapes generate the following standard events: **Click, DblClick, MouseUp, MouseDown, MouseMove, DragDrop, DragOver**. The MouseEvents property must be set to true for these events to fire.

The ACCMOVE.FRM demo form in the Sample Code shows how some of these events are used to allow the user to move a control.

Also see Visual Basic Help for the standard descriptions of these standard events.

Printing

The SoftCraft Graphic custom control provides a special exported function (much like a VB method) that allows you to print the custom controls with much higher quality than VB allows with its normal printing procedures. You can still use the normal VB printing procedures, but the SCGraphic print procedure provides better results.

The print procedure, PrintSCG, determines the capabilities of the printer with respect to Bezier curves and ClipToPath. For capable printers, such as PostScript printers, Bezier curves and graduated fills are printed with special Windows low-level printing methods.

Note: When you use the PrintSCG procedure to print high-quality output on typesetting devices (or even high-resolution laser printers) you can adjust the halftone frequency and angle using the Advanced Options button(s) in the printer driver. (Use the Control Panel, Printers applet Setup button to adjust these options.)

The syntax for the PrintSCG function is:

```
PrintSCG cntl, hDC, leftoffset, topoffset
```

where:

cntl is the control to be printed,
hDC is the printer device context (i.e., Printer.hDC),
leftoffset is the offset of the left edge of the form from the edge of the paper (i.e., Printer.ScaleLeft),
topoffset is the offset of the top edge of the form from the edge of the paper (i.e., Printer.ScaleTop).

The following code is from the demo program.

```
' Declaration of the exported function (like a method) for high-quality printing of SCGraphic
controls
Declare Sub PrintSCG Lib "scgrphic.vbx" (hCtl As Control, ByVal hDC As Integer, ByVal xOrg As
Integer, ByVal yOrg As Integer)
' Print a form outline in the center of the page and then
' print all of the SCGraphic controls on the form
Sub PrintFrm (frm As Form)
    Dim nCtl As Integer
    ' center the form on the page
    Printer.ScaleLeft = -(Printer.Width - frm.Width) / 2
    Printer.ScaleTop = -(Printer.Height - frm.Height) / 2
    Printer.Line (0, 0)-Step(frm.Width - 120, frm.Height - 420), , B  ' adjust Height and Width
for title bar and borders if desired
    ' At least one Printer method (such as Line above) must
    ' be used before calling PrintSCG to ensure a valid hDC.
    For nCtl = frm.Controls.Count - 1 To 0 Step -1
        ' Kludge:  VB provides no way to get the Zorder to
        ' sort overlapping controls, but this reverse control order works for the demo
        If TypeOf frm.Controls(nCtl) Is SCGraphic Then
```

```
            PrintSCG frm.Controls(nCtl), Printer.hDC, Printer.ScaleLeft, Printer.ScaleTop
        End If
    Next nCtl
    Printer.EndDoc
End Sub
```

Sample Code

The code for the demo program is shown below. The main form simply has buttons that show other forms. The other forms illustrate various capabilities of the shapes.

DEMO.BAS

```
Option Explicit
' Colors from CONSTANT.TXT
Global Const BLACK = &H0&
Global Const RED = &HFF&
Global Const GREEN = &HFF00&
Global Const YELLOW = &HFFFF&
Global Const BLUE = &HFF0000
Global Const MAGENTA = &HFF00FF
Global Const CYAN = &HFFFF00
Global Const WHITE = &HFFFFFF

' Bezier Constant for approximating conic sections
Global Const BEZCONIC = 551.92
Global Const BEZAUTO = 32760

Global Const PI = 3.14159265

' Declaration of the exported function (like a method) for high-quality printing of SCGraphic
controls
Declare Sub PrintSCG Lib "scgrphic.vbx" (hCtl As Control, ByVal hDC As Integer, ByVal xOrg As
Integer, ByVal yOrg As Integer)

' Compute a color that is an interpolation of two other
' colors.  The return value is a color that is percent
' of the way between col1 and col2.
Function BetweenColor (col1 As Long, col2 As Long, percent As Integer)
    Dim R1, G1, B1, R2, G2, B2
    R1 = col1 Mod 256
    G1 = col1 \ 256 Mod 256
    B1 = col1 \ 65536 Mod 256
    R2 = col2 Mod 256
    G2 = col2 \ 256 Mod 256
    B2 = col2 \ 65536 Mod 256
    R1 = R1 + (R2 - R1) * percent / 100
    G1 = G1 + (G2 - G1) * percent / 100
    B1 = B1 + (B2 - B1) * percent / 100
    BetweenColor = RGB(R1, G1, B1)
End Function

' Print a form outline in the center of the page and then
' print all of the SCGraphic controls on the form
Sub PrintFrm (frm As Form)
    Dim nCtl As Integer
    ' center the form on the page
    Printer.ScaleLeft = -(Printer.Width - frm.Width) / 2
    Printer.ScaleTop = -(Printer.Height - frm.Height) / 2
    Printer.Line (0, 0)-Step(frm.Width - 120, frm.Height - 420), , B  ' adjust Height and Width
for title bar and borders if desired
    ' At least one Printer method (such as Line above) must
    ' be used before calling PrintSCG to ensure a valid hDC.
    For nCtl = frm.Controls.Count - 1 To 0 Step -1
        ' Kludge:  Can't figure out how to get the Zorder to
        ' sort overlapping controls, but this reverse control order works for the demo
        If TypeOf frm.Controls(nCtl) Is SCGraphic Then
            PrintSCG frm.Controls(nCtl), Printer.hDC, Printer.ScaleLeft, Printer.ScaleTop
        End If
```

```
    Next nCtl
    Printer.EndDoc
End Sub
```

MAIN.FRM (frmMain)

```
Option Explicit
Dim frmCurrent As Form

Sub AccMove_Click ()
    frmAccMove.Show
    Set frmCurrent = frmAccMove
End Sub

Sub Composite_Click ()
    frmComp.Show
    Set frmCurrent = frmComp
End Sub

Sub Exit_Click ()
    End
End Sub

Sub Form_Unload (Cancel As Integer)
    End
End Sub

Sub Print_Click ()
    If Not (frmCurrent Is Nothing) Then
        Screen.MousePointer = 11  ' hourglass
        PrintFrm frmCurrent
        Screen.MousePointer = 0   ' default
    End If
End Sub

Sub Resize_Click ()
    frmResize.Show
    Set frmCurrent = frmResize
End Sub

Sub Samples_Click ()
    frmSamples.Show
    Set frmCurrent = frmSamples
End Sub

Sub SimpMove_Click ()
    frmSimpMove.Show
    Set frmCurrent = frmSimpMove
End Sub
```

SAMPLES.FRM (frmSamples)

```
Option Explicit
Const MAXSAMPLE = 7

Sub Form_Load ()
    Dim i, j As Single
    ' set the initial fill colors for the samples (this could
    ' have been done at design time instead)
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).FillPattern = 17 ' graduated horizontal
        SCGraphic1(i).FillColor = MAGENTA
        SCGraphic1(i).FillColor2 = CYAN
    Next i
```

```
    ' set different shadow colors for the open shapes for interest
    SCGraphic1(2).ShadowColor = YELLOW
    SCGraphic1(7).ShadowColor = YELLOW

    ' controls 2 through 4 are poly's; set some sample points
    ' just so we see something interesting
    For i = 2 To 4
        SCGraphic1(i).NumPoints = 5
        SCGraphic1(i).PointX(0) = 100
        SCGraphic1(i).PointY(0) = 900
        SCGraphic1(i).PointX(1) = 100
        SCGraphic1(i).PointY(1) = 100
        SCGraphic1(i).PointX(2) = 500
        SCGraphic1(i).PointY(2) = 700
        SCGraphic1(i).PointX(3) = 900
        SCGraphic1(i).PointY(3) = 100
        SCGraphic1(i).PointX(4) = 900
        SCGraphic1(i).PointY(4) = 900
    Next i

    ' make polyline 4 have some auto curvature points (only the X component needs to be set if
it's auto)
    SCGraphic1(4).PointXOffsetIn(1) = BEZAUTO
    SCGraphic1(4).PointXOffsetIn(2) = BEZAUTO
    SCGraphic1(4).PointXOffsetIn(3) = BEZAUTO
    SCGraphic1(4).PointXOffsetOut(1) = BEZAUTO
    SCGraphic1(4).PointXOffsetOut(2) = BEZAUTO
    SCGraphic1(4).PointXOffsetOut(3) = BEZAUTO
End Sub

Sub GradDiag_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).FillPattern = 19 'graduated down right
    Next i
End Sub

Sub GradHorz_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).FillPattern = 17 'graduated horizontal
    Next i
End Sub

Sub GradVert_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).FillPattern = 16 'graduated vertical
    Next i
End Sub

Sub ShadowOff_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).ShadowDepthX = 0
        SCGraphic1(i).ShadowDepthY = 0
    Next i
End Sub

Sub ShadowOn_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
        SCGraphic1(i).ShadowDepthX = 50  ' assuming units are still twips
        SCGraphic1(i).ShadowDepthY = 50
    Next i
End Sub

Sub Solid_Click ()
    Dim i, j As Single
    For i = 0 To MAXSAMPLE
```

```
        SCGraphic1(i).FillPattern = 0 'solid
    Next i
End Sub
```

FRMCOMP.FRM (frmComp)

```
Option Explicit
' Statically record the bottom and top positions of the
' composite shapes.  They are tied to the location of the
' scroll bar in the Load event.
Dim iCylBottom As Integer, iCylMaxLoc As Integer

' Draw the Cast Shadow composite shape.  scgCastFont is the
' front (rectangular) shape.  scgCastShad is the shadow
' shape, which is a polyline.  iValue is a number between
' 0 and 100 indicating how high to draw the shape.
' The two shapes need to be positioned at design-time so
' their lower-left corners are congruent.
Sub DrawCastShad (scgCastFront As SCGraphic, scgCastShad As SCGraphic, ByVal iValue As Integer)
    Const ANGLE = 50 * PI / 180  ' angle of the cast shadow (in radians)
    Dim iTop As Integer, iHeight As Integer, iWidth As Integer, iLeft As Integer
    Dim fWidthRatio As Single
    ' make the shapes invisible while we change various
    ' properties to avoid flashing
    scgCastFront.Visible = False
    scgCastShad.Visible = False
    ' stretch the front rect into its new position
    ' where iValue is the percentage of its maximum height
    iTop = iCylBottom - iValue / 100# * (iCylBottom - iCylMaxLoc)
    iWidth = scgCastFront.Width
    iLeft = scgCastFront.Left
    iHeight = iCylBottom - iTop
    ' we don't really need iLeft and iWidth, but using Move
    ' is better than setting Top and Height properties individually
    scgCastFront.Move iLeft, iTop, iWidth, iHeight
    ' compute the containing rectangle for the cast shadow
    iWidth = scgCastFront.Width + iHeight * Cos(ANGLE)
    iHeight = iHeight * Sin(ANGLE)
    scgCastShad.Move iLeft, iCylBottom - iHeight, iWidth, iHeight
    ' calculate the ratio of the width of the rectangle
    ' to the shadow to position the polygon points
    fWidthRatio = scgCastFront.Width / iWidth
    scgCastShad.PointX(0) = 0
    scgCastShad.PointY(0) = 1000
    scgCastShad.PointX(1) = 1000 * fWidthRatio
    scgCastShad.PointY(1) = 1000
    scgCastShad.PointX(2) = 1000
    scgCastShad.PointY(2) = 0
    scgCastShad.PointX(3) = 1000 * (1 - fWidthRatio)
    scgCastShad.PointY(3) = 0
    ' make the shapes visible agaon
    scgCastFront.Visible = True
    scgCastShad.Visible = True
End Sub

' Draw the Cylinder composite shape.  scgCylTop is the ellipse
' at the top of the cylinder.  scgCylLeft/Right are the two
' polylines that make up the two shaded halves of the cylinder.
' iValue is a number between 0 and 100 indicating how high
' to draw the cylinder.
' The three shapes must be positioned at design-time as
' shown in the sample form.
Sub DrawCylinder (scgCylTop As SCGraphic, scgCylLeft As SCGraphic, scgCylRight As SCGraphic,
ByVal iValue As Integer)
Dim iTop As Integer, iDepth As Integer, iHeight As Integer, fHeightPercent As Single
Dim lColor As Long
    ' Make the cylinder invisible while we change various
```

```
    ' properties to avoid flashing.  See the VB manual on
    ' p. 329 regarding the Move method and jerky motion.
    scgCylTop.Visible = False
    scgCylLeft.Visible = False
    scgCylRight.Visible = False
    ' move the ellipse at the top of the cylinder into its new position
    ' where iValue is the percentage of its maximum height
    iTop = iCylBottom - iValue / 100# * (iCylBottom - iCylMaxLoc)
    iDepth = scgCylTop.Height
    lColor = scgCylTop.FillColor
    ' because of the perspective, we lose a little of the value range, so adjust
    If iTop > iCylBottom - iDepth * 1.1 Then iTop = iCylBottom - iDepth * 1.1
    scgCylTop.Top = iTop
    ' adjust the top and height of the sides of the cylinder to match
    ' the new position of the ellipse at the top (attach at the center)
    iTop = iTop + iDepth / 2
    iHeight = iCylBottom - iTop  ' iCylBottom is a global, fixed position
    ' using Move is better than setting Top and Height properties individually
    scgCylLeft.Move scgCylLeft.Left, iTop, scgCylLeft.Width, iHeight
    scgCylRight.Move scgCylRight.Left, iTop, scgCylRight.Width, iHeight
    ' find the percentage of the height of the ellipse to the side
    fHeightPercent = iDepth / iHeight / 2#
    ' position the left side with correct Bezier handles
    scgCylLeft.PointX(0) = 0
    scgCylLeft.PointY(0) = 0
    scgCylLeft.PointX(1) = 1000
    scgCylLeft.PointY(1) = 0
    scgCylLeft.PointX(2) = 1000
    scgCylLeft.PointY(2) = 1000
    scgCylLeft.PointXOffsetOut(2) = -BEZCONIC
    scgCylLeft.PointX(3) = 0
    scgCylLeft.PointY(3) = 1000 * (1 - fHeightPercent)
    scgCylLeft.PointYOffsetIn(3) = BEZCONIC * fHeightPercent
    scgCylLeft.FillColor2 = BetweenColor(lColor, BLACK, 10)
    scgCylLeft.FillColor = BetweenColor(lColor, BLACK, 50)
    ' now do the right side
    scgCylRight.PointX(0) = 1000
    scgCylRight.PointY(0) = 0
    scgCylRight.PointX(1) = 0
    scgCylRight.PointY(1) = 0
    scgCylRight.PointX(2) = 0
    scgCylRight.PointY(2) = 1000
    scgCylRight.PointXOffsetOut(2) = BEZCONIC
    scgCylRight.PointX(3) = 1000
    scgCylRight.PointY(3) = 1000 * (1 - fHeightPercent)
    scgCylRight.PointYOffsetIn(3) = BEZCONIC * fHeightPercent
    scgCylRight.FillColor = BetweenColor(lColor, BLACK, 10)
    scgCylRight.FillColor2 = BetweenColor(lColor, BLACK, 50)
    ' make the cylinder visible again
    scgCylTop.Visible = True
    scgCylLeft.Visible = True
    scgCylRight.Visible = True
End Sub

' Draw the analog gauge.  scgGaugeBack is the background
' circle of the gauge.  scgGaugeArrow is the arrow pointer
' indicating the current value.  iValue is a number between
' 0 and 100 indicating the location of the arrow pointer.
' The two shapes must be positioned at design time.  The
' arrow shape should be the identical location and size of
' the background circle.
Sub DrawGauge (scgGaugeBack As SCGraphic, scgGaugeArrow As SCGraphic, ByVal iValue As Integer)
    Const MINANGLE = 225 * PI / 180  ' arrow angle corresponding to the 0 value
    Const MAXANGLE = -45 * PI / 180  ' arrow angle corresponding to the 100 value
    Const SPREAD = MAXANGLE - MINANGLE
    ' make the shapes invisible while we change various
    ' properties to avoid flashing
    scgGaugeBack.Visible = False
    scgGaugeArrow.Visible = False
    ' set the arrow angle according to the value
    scgGaugeArrow.PointX(0) = 500    ' the base of the arrow is at the center
```

```
    scgGaugeArrow.PointY(0) = 500
    scgGaugeArrow.PointX(1) = 500 + 450 * Cos(MINANGLE + SPREAD * (iValue / 100#))
    scgGaugeArrow.PointY(1) = 500 - 450 * Sin(MINANGLE + SPREAD * (iValue / 100#))
    ' make the shapes visible agaon
    scgGaugeBack.Visible = True
    scgGaugeArrow.Visible = True
End Sub

Sub Form_Load ()
    ' keep the bottom of the cylinder fixed at the bottom of the scroll bar
    iCylBottom = vsbValue.Top + vsbValue.Height
    ' let the cylinder grow to the height of the scroll bar
    iCylMaxLoc = vsbValue.Top
    ' simulate a scroll bar change to draw the initial screen
    vsbValue_Change
End Sub

Sub vsbValue_Change ()
    DrawCylinder scgCylTop, scgCylLeft, scgCylRight, vsbValue.Value
    DrawCastShad scgCastFront, scgCastShad, vsbValue.Value
    DrawGauge scgGaugeBack, scgGaugeArrow, vsbValue.Value
End Sub
```

ACCMOVE.FRM (frmAccMove)

Accurate shape movement is described on p. 283 of the VB3 Programmers Guide. This form demonstrates that the shapes produce the expected events and respond quickly to mouse movements. The ShowOutlineOnly property is used while the shape is being moved to optimize redraw speed.

```
Option Explicit
Dim WereMoving As Integer     ' record MouseDown/Up events
Dim StartX, StartY As Single  ' mouse location at the start of a move

Sub ByInk_Click ()
    pentagon.SelectByInk = True
End Sub

Sub ByRect_Click ()
    pentagon.SelectByInk = False
End Sub

Sub Empty_Click ()
    pentagon.FillPattern = 1   ' Clear fill pattern
End Sub

Sub Filled_Click ()
    pentagon.FillPattern = 16  ' graduated vertical
End Sub

Sub Form_Load ()
    WereMoving = False   ' the mouse is up to begin with
End Sub

Sub pentagon_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' record the MouseDown so MouseMove updates the shape
    WereMoving = True
    ' record the starting mouse position so we can move relative to that spot
    ' this is described in the VB3 manual on p. 283
    StartX = X
    StartY = Y
    ' use transparent shapes for faster redraw during mouse move
    ' we'll turn gradfills back on in MouseUp
    pentagon.ShowOutlineOnly = True
```

```
End Sub

Sub pentagon_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' a MouseDown event sets the WereMoving flag
    If WereMoving Then
        ' redraw the shape at the current mouse position
        pentagon.Move pentagon.Left + X - StartX, pentagon.Top + Y - StartY
    End If
End Sub

Sub pentagon_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' we finished a move so turn fills back on
    pentagon.ShowOutlineOnly = False
    ' we aren't moving until we get another MouseDown
    WereMoving = False
End Sub
```

SIZE.FRM (frmResize)

```
Option Explicit
Dim nOperation As Integer     ' record move/size operation type
Dim bMouseDown As Integer     ' record mouse state
Dim StartX, StartY As Single  ' mouse location at the start of a move
Dim bImSelected As Integer    ' record whether the object is selected or not; deselect in
Form_Click
                              ' keep an array of Booleans (or use an unused shape property) if
you have multiple shapes

Const nHandleSize = 90        ' selection handle size (twips)
Const nMoveThreshold = 200    ' mouse move threshold for auto move mode (twips)

' Operation/handle constants
Const TL = 1  ' top-left
Const TC = 2  ' top-center
Const TR = 3  ' top-right
Const ML = 4  ' middle-left
Const MR = 5  ' middle-right
Const BL = 6  ' bottom-left
Const BC = 7  ' bottom-center
Const BR = 8  ' bottom-right
Const MV = 9  ' move operation

' Display sizing handles on a control (or clear the handles)
Sub ShowHandles (obj As Control, bOn As Integer)
    Dim nh As Integer
    Dim c As Single, r As Single, m As Single, b As Single

    nh = nHandleSize  ' just to reduce typing

    c = obj.Left + (obj.Width - nh) / 2  ' left/right center
    r = obj.Left + obj.Width - nh        ' right
    m = obj.Top + (obj.Height - nh) / 2  ' top/bottom middle
    b = obj.Top + obj.Height - nh        ' bottom

    If bOn Then
       DrawMode = 1  ' choose Black Pen or XOR (6) depending on the type of shapes and background
you have
       Line (obj.Left, obj.Top)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (c, obj.Top)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (r, obj.Top)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (obj.Left, m)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (r, m)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (obj.Left, b)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (c, b)-Step(nh, nh), RGB(0, 0, 0), BF
       Line (r, b)-Step(nh, nh), RGB(0, 0, 0), BF
       DrawMode = 1
    Else
       ' if you choose DrawMode = 6 above, you may be able to clean the handles
```

```
        ' by redrawing them with XOR (DrawMode = 6) again and eliminate the repaint of the shape
        obj.Visible = True ' repaint the object to eliminate handles
    End If
End Sub

' Check the given x,y coordinates to see if the position is
' within one of the sizing handles.  A number between 0 and 9
' is returned.  0 means the position is not in the control at
' all (shouldn't happen if this was called from MouseDown).
' 9 means it is not on a sizing handle, but is in the control.
' 1 thru 8 indicate sizing handles, numbered 1,2,3 on the top;
' 4,5 in the middle and 6,7,8 along the bottom (left to right).
' Use the constants TL, TC, etc. for these values
Function WhichHandle (obj As Control, X As Single, Y As Single) As Integer
    Dim nh As Integer, nRet As Integer
    Dim iL As Integer, iC As Integer, iR As Integer
    Dim iT As Integer, iM As Integer, iB As Integer
    Dim c As Single, r As Single, m As Single, b As Single

    nh = nHandleSize  ' just to reduce typing

    c = (obj.Width - nh) / 2  ' left/right center
    r = obj.Width - nh        ' right
    m = (obj.Height - nh) / 2  ' top/bottom middle
    b = obj.Height - nh        ' bottom

    ' we could do this more elegantly with rectangles and
    ' PtInRect, but this works and is probably fast even tho it's ugly
    ' iL, etc. record whether the position is in one dimension of a handle
    iL = False
    iC = False
    iR = False
    iT = False
    iM = False
    iB = False
    If (X > 0 And X < nh) Then iL = True  ' possibly in one of the left handles
    If (X > c And X < c + nh) Then iC = True
    If (X > r And X < r + nh) Then iR = True
    If (Y > 0 And Y < nh) Then iT = True
    If (Y > m And Y < m + nh) Then iM = True
    If (Y > b And Y < b + nh) Then iB = True

    nRet = 0
    If (iL And iT) Then nRet = TL
    If (iC And iT) Then nRet = TC
    If (iR And iT) Then nRet = TR
    If (iL And iM) Then nRet = ML
    If (iR And iM) Then nRet = MR
    If (iL And iB) Then nRet = BL
    If (iC And iB) Then nRet = BC
    If (iR And iB) Then nRet = BR
    ' if in none of the handles, double-check to make sure its in the object
    If (nRet = 0 And X > 0 And X < obj.Width And Y > 0 And Y < obj.Height) Then nRet = MV

    WhichHandle = nRet
End Function

Sub Form_Click ()
    ' Deselect the selected shape if the user clicks on the form
    ' Alternatively, you could deselect if the user clicks on the shape again
    If bImSelected Then
        bImSelected = False
        ShowHandles Rectangle, False
    End If
End Sub

Sub Form_Load ()
    bMouseDown = False   ' the mouse is up to begin with
    nOperation = 0       ' no move/size operation yet
    bImSelected = False  ' not selected
End Sub
```

```
Sub Rectangle_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' record MouseDown for subsequent MouseMove's
    bMouseDown = True
    ' record the starting mouse position so we can move relative to that spot
    ' this is described in the VB3 manual on p. 283
    StartX = X
    StartY = Y
    If bImSelected Then
        nOperation = WhichHandle(Rectangle, X, Y)
        ' use transparent shapes for faster redraw during mouse move
        ' we'll turn gradfills back on in MouseUp
        Rectangle.ShowOutlineOnly = True
        ' change the mouse cursor to indicate the operation
        Select Case nOperation
            Case TL, BR
                MousePointer = 8
            Case TR, BL
                MousePointer = 6
            Case TC, BC
                MousePointer = 7
            Case ML, MR
                MousePointer = 9
            Case MV
                MousePointer = 5
        End Select
    End If
End Sub

Sub Rectangle_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' nOperation records whether we are moving or sizing
    Select Case nOperation
        Case 0  ' no operation yet, but check for movement to enter one-click select and move mode
            If (bMouseDown And Abs(StartX - X) + Abs(StartY - Y) > nMoveThreshold) Then
                ' the mouse is down, the object isn't selected, but the mouse has moved a ways
                ' so select the object and begin moving without requiring a mouse up
                bImSelected = True
                nOperation = MV  ' movement
                Rectangle.ShowOutlineOnly = True
                MousePointer = 5
            End If
        ' use Abs on height and width to avoid negative widths
        Case TL  ' from top-left
            Rectangle.Move Rectangle.Left + X - StartX, Rectangle.Top + Y - StartY,
Abs(Rectangle.Width + StartX - X), Abs(Rectangle.Height + StartY - Y)
        Case TC  ' from top-center
            Rectangle.Move Rectangle.Left, Rectangle.Top + Y - StartY, Rectangle.Width,
Abs(Rectangle.Height + StartY - Y)
        Case TR  ' from top-right
            Rectangle.Move Rectangle.Left, Rectangle.Top + Y - StartY, Abs(X),
Abs(Rectangle.Height + StartY - Y)
        Case ML  ' from middle-left
            Rectangle.Move Rectangle.Left + X - StartX, Rectangle.Top, Abs(Rectangle.Width +
StartX - X)
        Case MR  ' from middle-right
            Rectangle.Move Rectangle.Left, Rectangle.Top, Abs(X)
        Case BL  ' from bottom-left
            Rectangle.Move Rectangle.Left + X - StartX, Rectangle.Top, Abs(Rectangle.Width +
StartX - X), Abs(Y)
        Case BC  ' from bottom-center
            Rectangle.Move Rectangle.Left, Rectangle.Top, Rectangle.Width, Abs(Y)
        Case BR  ' from bottom-right
            Rectangle.Move Rectangle.Left, Rectangle.Top, Abs(X), Abs(Y)
        Case MV  ' move
            Rectangle.Move Rectangle.Left + X - StartX, Rectangle.Top + Y - StartY
    End Select
End Sub

Sub Rectangle_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    If nOperation = 0 Then
        ' if we aren't moving or sizing yet just select
```

```
        If bMouseDown Then
            bImSelected = True  ' check MouseDown just in case we get an up without a down
            ShowHandles Rectangle, True  ' turn on the handles
        End If
    Else
        ' we finished a move so turn fills back on
        Rectangle.ShowOutlineOnly = False
        Rectangle.Refresh
        ShowHandles Rectangle, True  ' restore the handles after repainting the shape
    End If
    MousePointer = 0   ' reset back to the default mouse pointer
    bMouseDown = False
    nOperation = 0
End Sub
```

SIMPMOVE.FRM (frmSimpMove)

The VB DragMode=Automatic capability is used on this form to show shape movement using a single line of code.

```
Option Explicit

Sub Form_DragOver (Source As Control, X As Single, Y As Single, State As Integer)
    Source.Move X, Y
End Sub
```