VBA Drawing with Microsoft Excel 2007

Geoff Whale. School of Computer Science & Engineering, The University of New South Wales

Abstract: The Excel 2007 Macro Recorder does not capture drawing actions, making it difficult for a casual developer to see how to create shapes programmatically. However, the VBA engine is capable of building and displaying shapes, and will generally replay shape code from earlier versions. This paper describes the common shapes and how to manipulate them using VBA.

1 Introduction

system

Microsoft developers omitted drawing actions from the Macro Recorder implementation of Excel 2007, thus removing the primary learning tool for programmers wanting to understand shapes. The Word 2007 recorder does capture drawing actions, but the code is not compatible with Excel for several reasons, including variations in the object model and, more importantly, differences in implementation of some key methods such as AddLine and AddConnector, resulting in discrepancies in both the appearance and position of some elements.

Accordingly, the only ways to learn about VBA drawing code involve access to an earlier version such as Excel 2003, tedious trawling through the Help system or reading tutorials such as the present document.

2 VBA Shapes and Coordinates

Drawings consist of a collection of Shape objects associated with a worksheet. Each object has a position, along with many other properties. Positions and sizes are expressed in points (a point is 1/72 of an inch) and are represented as singleprecision reals (Single type). For a typical display resolution of 96dpi, that means that each unit is 4/3 pixels. For 120dpi (larger fonts), each unit is 5/3 pixels. To be safe, drawings should generally be limited to about 600 x 400 units.

A screen position has an X and a Y coordinate, with X increasing to the right and Y increasing downward. The origin (0,0) is the top left corner of the worksheet.

Figure 1. Е A В С D Drawing 1 coordinate 267 Pixels 2 3 4 5 6 Shape: Rectangle 7 Top left position: X = 50, Y = 758 Width = 200pt (= 267px @ 96dpi) 9 Height = 10010 11 12

3 Creating Shapes

Shapes are created by applying a particular method to the **ActiveSheet.Shapes** collection, depending on the kind of shape:

Shape	Method	Arguments
Lines	AddLine	Start and end coords
AutoShape	AddShape	Shape type, bounding box coords and dimensions
Freeform object (polygon)	BuildfreeForm	Node type and initial point; other points created with AddNodes and finally ConvertToShape
Text box	AddTextBox	Text orientation, box coords and dimensions

All objects are given default attributes, which can later be changed using property assignments. This includes the text in a textbox.

Autoshapes include rectangles, ellipses and circles, block arrows, callouts, flowchart symbols and many other predefined symbols. Their *bounding box* is the smallest rectangle that encloses the shape, and the positional coordinates are the coords of the top left corner of the bounding box. For example, a circle is defined not using its centre and radius as you may expect but as the square that just encloses it.

Polygons are created using three methods, as described in 3.4 below.

3.1 The Shape type and method calls

When the Macro Recorder is able to capture drawing code it will make use of the **Selection** object to reference the last created shape to apply property changes. For example, to create a dashed line along the major diagonal of the rectangle in Figure 1, the following code would be captured:

```
ActiveSheet.Shapes.AddLine(50, 75, 250, 175).Select
Selection.ShapeRange.Line.DashStyle = msoLineDash
```

AddLine creates a new Shape object and adds it to the collection. Its name is the kind of shape followed by a counter representing the number of such objects created since the interpreter started. For example, if this was the fifth line of any kind, the entry in the collection would be given the name "Line 5".

If the user later selects the line and makes changes to it, the generated code begins:

```
ActiveSheet.Shapes("Line 5").Select
```

Important note: this kind of shape code cannot be replayed reliably. Shapes retain their names indefinitely, even when deleted. So if you were to delete the dashed line and replay the code, you would get a new one with a new name. Any reference to Line 5 generates an error because the collection element no longer exists.

As an alternative to using **Select**, each of the methods returns a **Shape** object reference. Use **Set** to assign to a variable. For example,

```
Dim shpDiagonal As Shape
```

```
Set shpDiagonal = ActiveSheet.Shapes.AddLine(50, 75, 250, 175)
shpDiagonal.ShapeRange.Line.DashStyle = msoLineDash
```

Keeping track of many shapes using variables is tedious: as far as possible, apply attributes as soon as the object is created.

3.2 Autoshapes

AddShape has five arguments: the kind of shape, which is a value from the msoAutoShapeType enumeration, and the top-left coordinates and width and height. This creates a dotted rectangle and overlays it with an ellipse with the same bounding box:

```
Set shpRect = ActiveSheet.Shapes.AddShape(msoShapeRectangle, _
50, 75, 250, 175)
shpRect.Line.DashStyle = msoLineSquareDot
shpRect.Line.Weight = 1.75
shpRect.Fill.Visible = msoFalse
Set shpEllipse = ActiveSheet.Shapes.AddShape(msoShapeOval, _
50, 75, 250, 175)
```

The Line and Fill properties are explained in section 4.



The full list of shapes types is available by typing msoAutoShapeType into the VB Editor's help search. An excerpt is shown in Appendix A.

3.3 Text boxes

The AddText method requires a text orientation and box coordinates. This just gives you a rectangle (with default outline and fill) to which you can add text using the default font. Usually you'll want to change the characteristics, see section 4.4.



3.4 Polygons (Freeform objects)

Unlike the other shapes, a freeform element shape is defined by an arbitrary number of points or nodes. The first point is supplied when the object is created, then each other node is added in turn with another line of VBA. When all points have been added one more call converts the object into a shape so that the appearance can be customised. If the last point is very close to the first, the shape is closed and can be filled.

The following example creates an equilateral triangle with specified bottom-left vertex and given side, by calculating the location of the required nodes.

x = 50: y = 300: side = 120 With ActiveSheet.Shapes.BuildFreeform(msoEditingAuto, x, y) ' first point .AddNodes msoSegmentLine, msoEditingAuto, x + side, y ' bottom right .AddNodes msoSegmentLine, msoEditingAuto, x + side / 2, ______ y - side * Sqr(3) / 2 ' top vertex .AddNodes msoSegmentLine, msoEditingAuto, x, y ' first point again Set shpTri = .ConvertToShape End With



The above example uses only straight lines, the other possibility is msoSegmentCurve,

which uses a quadratic Bézier curve instead. If the second **msoSegmentLine** was changed to **msoSegmentCurve** the figure would look like this:

Try replacing **msoEditingAuto** by **msoEditingCorner** or **msoEditingSmooth** or **msoEditingSymmetric** to see how this affects the shape of the outline at the node. **msoEditingAuto** just says use the most appropriate type given the adjacent segments.



4 Shape properties

The most important properties of a shape, besides its position on the screen, are the those that describe its stroke (outline) and fill. Each of these includes a colour, so we'll start by discussing that characteristic.

4.1 Colour

Versions prior to 2007 used a palette of 56 fixed colours, each of which was assigned a colour index, an integer between 1 and 56. An arbitrary colour using red, green and blue component values could be used as well, though it would be converted to the nearest standard colour. 2007 introduced the idea of themes, with a colour being based on a standard one but with a saturation factor applied to get a related tint.

The best approach from a portable programming point of view is to assign an RGB value and let Excel find the best way to store that internally, especially if the document is to be saved in the compatibility format (Excel 97-2003).

There are two ways to create a composite RGB (red, green, blue) value:

Define a hexadecimal (base-16) constant with components between 00 and FF (255), expressed from the left as blue, green and red (not the other way). For example, bright yellow has fully saturated blue and green components and no red, so

```
Const COLOUR_YELLOW = &HFFFF00
```

Note that **&H** prefixes a hexadecimal number.

▶ Use the built-in function **RGB**, which accepts three integers between 0 and 255 representing the red, green and blue components and returns the composite colour. Unfortunately VBA doesn't allow function calls in constant definitions, even when the arguments are constants, so this way isn't as readable.

... = RGB(255, 255, 0) 'Yellow

4.2 Outline properties

A shape's outline has several main characteristics,

- ▶ Its colour
- Its weight (thickness) in points
- ▶ Its dash style (one of a set of values from the msoDashStyle enumeration)
- ▶ (For open shapes such as lines) line ending symbols such as arrows

The code overleaf creates this horizontal 2.5-point, dashed red line with a triangular arrowhead at the end (the second point):

```
Set shp = ActiveSheet.Shapes.AddLine(50, 100, 250, 100)
With shp.Line
    .DashStyle = msoLineDash
    .ForeColor.RGB = RGB(255, 0, 0)
    .EndArrowheadStyle = msoArrowheadTriangle
    .Weight = 1.5
End With
```

The default dash style is **msoLineSolid**, giving a continuous line. Defaults were accepted for the properties **EndArrowHeadLength** and **EndArrowHeadWidth** but you can set them too (see Excel VBA Help for options).

The same properties can be set for any shape object, except the arrowhead styles if the object is closed.

To remove the outline entirely, say from a textbox where you want only the text to appear, use

```
shp.Line.Visible = msoFalse
```

4.3 Fill properties

Only closed shapes have fill, which describes the interior of the figure. It can have a solid or patterned appearance, or none, and it can be partly transparent. The following generates a 50% green autoshape arrow over a textured rectangle:



To remove fill altogether, set the Visible property of the shape to msoFalse.

4.4 Text properties

Text boxes have additional properties relating to the text itself. You can change various font characteristics, including typeface (family), style, size and colour.

```
Dim shpLabel As Shape
Set shpLabel =
    ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal,
        120, 155, BOX_WIDTH, BOX_HT)
With shpLabel.TextFrame.Characters ' applies to whole of text
    .Text = "Figure 1. Location of SS Titanic"
    .Font.Name = "Arial Narrow"
    .Font.Size = 12
    .Font.Bold = True
    .Font.Color = RGB(128, 0, 0) ' Dark red
End With
    ' change attributes of part of the text only
With shpLabel.TextFrame.Characters(Start:=23, Length:=10)
    .Font.Italic = True
End With
shpLabel.Fill.Visible = False
shpLabel.Line.Visible = False
```

Produces

Figure 1. Location of SS Titanic

The box width and height doesn't matter much if you remove the outline and fill. In general, you should write a function to generate a label with given text at a specified location with the same, consistent style. It would return the shape so that specific variations (such as the partial italic range above) could be applied.