Freeform Shapes in the Office Drawing Format

DIaLOGIKa/makz/clam 10 February 2009

Contents

Introduction

This document explains how a free formed shape is stored in the binary Word and PowerPoint files.

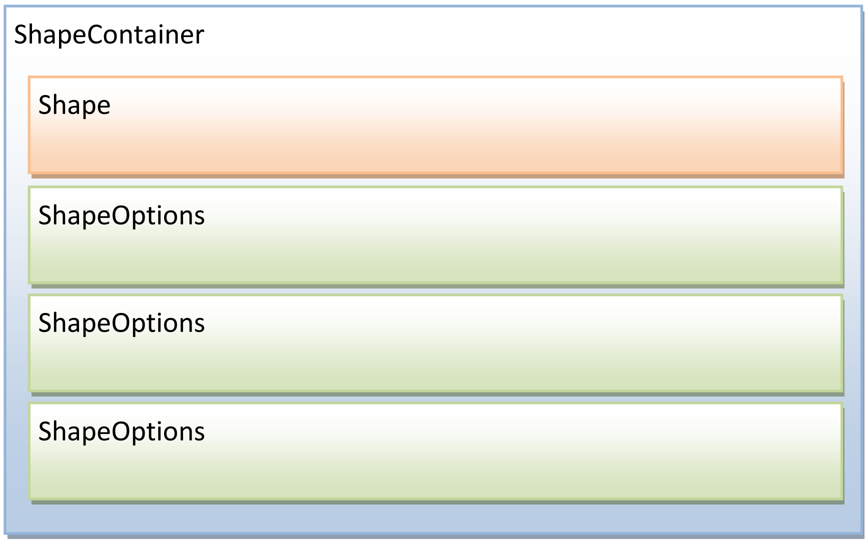
The following descriptions and algorithms are based on the *Office Drawing97- 2007 Binary Format Specification* and our own findings.

This guide implies a basic knowledge of the Office Drawing Layer (Escher) format and should be used as a supplement to the official specification in order to facilitate the understanding of some loose points in the specification.

Short Introduction to the Escher Architecture

The Office Drawing Format is based on a container hierarchy that is called "Escher". In Escher every object is called "record". A record consists of a header and a body. The header defines some information about the record such as the type of the record. If the type is a container type the record can hold other records in its body, if it is an atom record it can only hold data in its body.

Shapes are usually defined as *ShapeContainer* records. Such a *ShapeContainer* contains at least one *Shape* record and it may contain several *ShapeOptions* records:



The *Shape* record holds basic information about the shape itself: e.g. the type of the shape: is it a rectangle or a triangle?

The *ShapeOptions* records mainly hold formatting information like the color of the outline or what fill effects are applied to the shape.

A *ShapeOptions* record isn't a container record but it stores several *OptionEntry* structures inside the body. Each *OptionEntry* has an ID that identifies the property that is modified and a simple property value that is limited to a 32bit integer. It can optionally have a complex value that is a byte array of a much larger size.

The Representation of a Freeform Shape

When is My Shape a Freeform Shape?

A shape can be identified as a freeform shape when the *Instance* value of the *Shape* record's header is set to 0. The *Instance* of a shape declares the type of the shape. The value 0 means "msosptNotPrimitive".

Freeform shapes have no predefined type which means that the rendering application cannot know its polygon by default. A rectangle will always look like a rectangle but a freeform shape is drawn by the user.

For this reason a freeform shape must store its polygon in the file.

Defining the Polygon of a Freeform Shape

The Office Drawing 97-2007 Specification calls the shape's polygon "path". Such a path consists of rendering instructions that allows the rendering application to display the shape.

The path is stored in two *OptionEntry* structures:

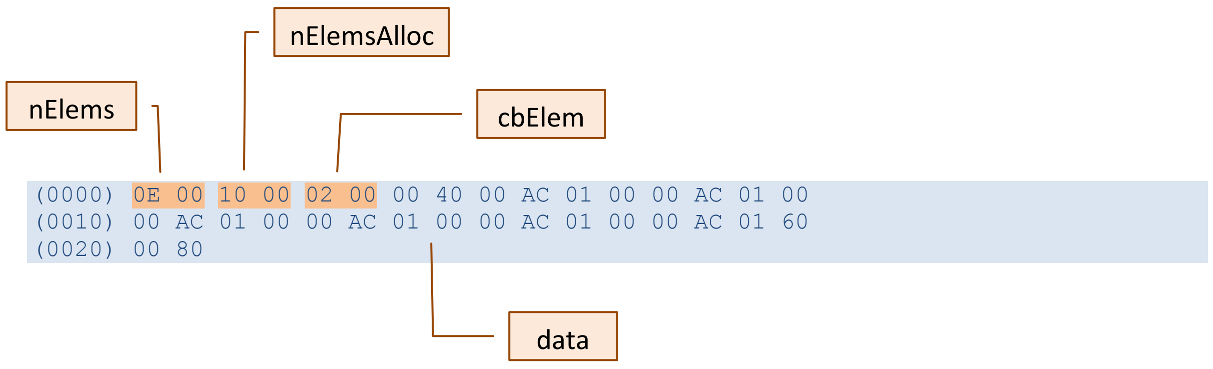
- ID 326 = pSegmentInfo
- ID 325 = pVertices

The pSegmentInfo entry is an array that contains rendering commands. The pVertices entry is an array that contains the parameters (points) of the commands stored in *pSegmentinfo*.

Both arrays are of the format *IMsoArray*.

The IMsoArray Structure

A *IMsoArray* consists of 3 16bit Integer values followed by the data in the array:



```
(0000) 0E 00 10 00 02 00 00 40 00 AC 01 00 00 AC 01 00
(0010) 00 AC 01 00 00 AC 01 00 00 AC 01 00 00 AC 01 60
(0020) 00 80
```

The first Integer *nElems* defines the length of the array, nElemsAlloc is a*n unsigned integer that specifies the maximum number of elements this record can contain. This value MUST be greater than or equal to nElems* and *cbElem* is the length of every single element in bytes. If *cbElem* has the value 0xfff0 then every element is a truncated 8 byte element (only the element's 4 low-order bytes are recorded, the 4 high-order bytes are 0x0).

The data block contains the array elements and has a size of *nElems* * *cbElem*.

The pSegmentInfo Structure

The stored elements are segments of the path. Each segment has a length of 2 bytes. The segment type is stored in the upper 3 bits; the segment count is stored in the lower 3 bits:

```
(0000) 0E 00 10 00 02 00 00 40 00 AC 01 00 00 AC 01 00
(0010) 00 AC 01 00 00 AC 01 00 00 AC 01 00 00 AC 01 60
(0020) 00 80
```

Segment 1 = 0x4000

The first segment can be split into:

```
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

SegmentTyp = 2   SegmentCount = 0

The segment type must be mapped to the enumeration MSOPATHTYPE:

```
msopathLineTo = 0,
msopathCurveTo,
msopathMoveTo,
msopathClose,
msopathEnd,
msopathEscape,
msopathClientEscape,
msopathInvalid
```

A value of 2 corresponds to the “Move To” command.

The segment count gives the count to repeat the command. Commands which cannot be repeated (like the “Move To” command) will have a count of 0.

The pVertices structure

The elements stored in the *pVertices* array are points in the coordinate system. Each point has a length of 4 bytes. The upper two bytes are a 16bit Integer standing for the x-coordinate, the lower 16bit stand for the y-coordinate:

```
(0000) 06 00 06 00 F0 FF 00 00 7C 13 00 00 BB 08 6F 06
(0010) 00 00 24 0D 8D 08 24 0D 93 13 00 00 7C 13
```

Point 1 = X 0x0000, Y 0x137C

Building the path

The *pSegmentInfo* array defines the sequence of the rendering commands, but some of the commands need arguments. The “Move To” command from the upper example needs a target point in the coordinate system where to move to.

Every command has a fix count of required arguments. “Line To” and “Move To” require one point as target, “Curve To” requires 3 points to build a Bezier curve. The other commands don’t need any argument.

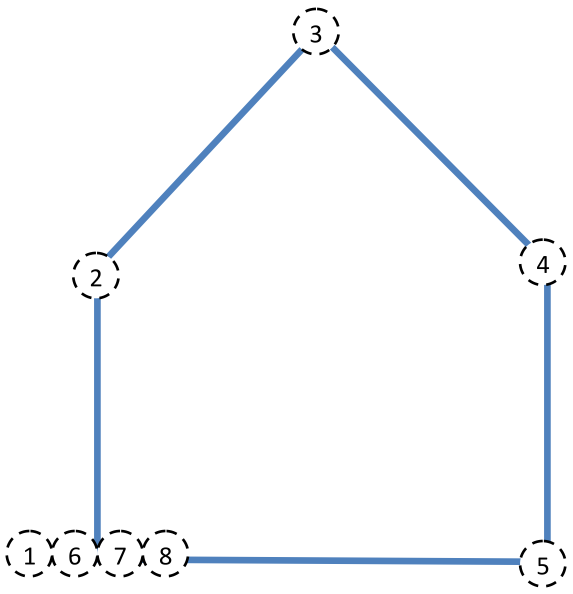
So we can build the path by following the commands in the pSegmentInfo array. Escape commands store editing information and can be ignored:

	pSegmentInfo	Count
1	msopathMoveTo	0
	msopathEscape	0
2	msopathLineTo	1
	msopathEscape	0
3	msopathLineTo	1
	msopathEscape	0
4	msopathLineTo	1
	msopathEscape	0
5	msopathLineTo	1
	msopathEscape	0
6	msopathLineTo	1
	msopathEscape	0
7	msopathClose	1
8	msopathEnd	0

	pSegmentInfo
1	X = 0 Y = 4988
2	X = 0 Y = 2235
3	X = 1647 Y = 0
4	X = 3364 Y = 2189
5	X = 3364 Y = 5011
6	X = 0 Y = 4988

This path draws the following shape

1) Move to: X = 0 Y = 4988
2) Line to: X = 0 Y = 2235
3) Line to: X = 1647 Y = 0
4) Line to: X = 3364 Y = 2189
5) Line to: X = 3364 Y = 5011
6) Line to: X = 0 Y = 4988
7) Close the path
8) End the path



Mapping Freeform Shapes to VectorML

Freeform Shapes are also represented by a path in VectorML. This path is a string that is used in an XML attribute. All commands and the points are serialized into specific string syntax:

```
<v:shape path="m0,4988l0,2235l1647,0l3364,2189l3364,5011l0,4988xe" />
```

The commands are mapped to single characters (shown in bold and red in the example above):

```
msopathLineTo = l
msopathCurveTo = c
```

```
msopathMoveTo = m
msopathClose x
msopathEnd = e
```

Mapping Freeform Shapes to DrawingML

Freeform shapes are represented as a custom geometry (the *custGeom* element) in DrawingML. It consists of the entries in the corresponding *pVertices* array (specified inside the *cxnLst* sub-element) and the entries in the corresponding *pSegmentInfo* array (specified inside a *path* element in the *pathLst* sub-element).

Each point in the *pVertices* array is mapped to a *cxn* element. The coordinates are specified in a *pos* sub-element.

Each entry in the *pSegmentInfo* array is mapped to a specific command element. E.g. `msopathLineTo` commands are mapped to *lnTo* elements and `msopathMoveTo` commands are mapped to *moveTo* elements. The parameters (points) of a command are specified in *pt* sub-elements.

```xml
<a:custGeom>
  <a:cxnLst>
    <a:cxn ang="0">
      <a:pos x="0" y="725" />
    </a:cxn>
    <a:cxn ang="0">
      <a:pos x="817" y="0" />
    </a:cxn>
    <a:cxn ang="0">
      <a:pos x="2495" y="680" />
    </a:cxn>
    <a:cxn ang="0">
      <a:pos x="1678" y="1497" />
    </a:cxn>
    <a:cxn ang="0">
      <a:pos x="0" y="725" />
    </a:cxn>
  </a:cxnLst>
  <a:rect l="0" t="0" r="r" b="b" />
  <a:pathLst>
    <a:path w="2495" h="1497">
      <a:moveTo>
        <a:pt x="0" y="725" />
      </a:moveTo>
      <a:lnTo>
        <a:pt x="817" y="0" />
      </a:lnTo>
      <a:lnTo>
        <a:pt x="2495" y="680" />
      </a:lnTo>
      <a:lnTo>
        <a:pt x="1678" y="1497" />
      </a:lnTo>
      <a:lnTo>
        <a:pt x="0" y="725" />
      </a:lnTo>
      <a:close />
    </a:path>
  </a:pathLst>
</a:custGeom>
```