Add-in to Convert Visio Shapes to PPTX Using Guides

1. Introduction

Despite many visual similarities, the shapes included in the built-in Microsoft Visio stencils and the built-in shapes included in Microsoft PowerPoint are very different programmatically. Because of these differences, drawings cannot be easily transferred from Visio to PowerPoint. Traditionally, transferring shapes from Visio to PowerPoint relied on copying and pasting the shapes as pictures or using OLE.

The two applications work with shapes in a very different way. To Visio, each "type" of shape in the **Shapes** window is actually content (a "master" shape) that is contained in another document (a "stencil" or .vss file). When you drag a shape from the **Shapes** window to the drawing page, Visio basically creates a link of the master shape. Most of the values that are used to specify the formatting for the Visio.Shape object are contained as formulas in the ShapeSheet, and must be accessed by referencing the appropriate cells. One benefit is that users can create and share custom stencils and master shapes to define new data fields, and to set their own business logic in a shape.

In PowerPoint, the difference between shapes is built into the object model. The [MsoAutoShapeType](http://msdn.microsoft.com/en-us/library/office/microsoft.office.core.msoautoshapetype(v=office.14).aspx) enumeration predefines the type of shapes that can be created in PowerPoint. (Note that Microsoft Word and Microsoft Excel also use the **MsoAutoShapeType** enumeration to define shapes.) The Shape object uses class properties to specify formatting. The shapes do not allow for new data fields or custom logic (without programmatic customization).

1.1 ShapeSheet

A Microsoft® Visio® object is stored internally as a set of formulas. For example, when you view a shape in a drawing window, you see it rendered graphically and see it behave according to its formulas. When you view the same shape in a ShapeSheet® window, you see the underlying formulas that determine how the shape looks and behaves on the drawing page. These two windows simply provide different views of the same shape.

In a drawing window, some of the changes you make to an object affect its formulas. For example, when you move a shape with the **Pointer**tool, Visio changes and then reevaluates the formulas that define the shape's center of rotation, or *pin*, on the drawing page, because those formulas determine the shape's location on the page. However, a ShapeSheet window gives you more precise control over the appearance and behavior of the object, because you can edit the object's formulas to change its behavior. Whether you change an object in a drawing window or a ShapeSheet window, the modifications are automatically saved when you save the Visio document that contains the object.

For more information on publishing content

<http://msdn.microsoft.com/en-us/library/office/aa200960(v=office.10).aspx>

1.2 Guides

Guides serve a similar purpose as Visio formulas. Guides are far less expressive. They have a name and a prefix formula expression attributes. Guides are defined either by <gdLst> or <avLst>. <gdLst> specifies guides that have formulas associated with them, and <avLst> specifies guides that have values associated with them. <avLst> are primarily used in conjunction with adjust handles.

Consider the case where the user would like to specify a triangle with its bottom edge defined not by static points but by using a varying parameter, namely a guide. Consider the diagrams and DrawingML shown below. This first triangle has been drawn with a bottom edge that is equal to the 2/3 the value of the shape height. Thus we see in the figure below that the triangle appears to occupy 2/3 of the vertical space within the shape bounding box.



<a:xfrm>

<a:off x="3200400" y="1600200"/>

<a:ext cx="1705233" cy="679622"/>

</a:xfrm>

<a:custGeom>

<a:avLst/>

<a:gdLst>

<a:gd name="myGuide" fmla="\*/ h 2 3"/>

</a:gdLst>

<a:ahLst/>

<a:cxnLst/>

<a:rect l="0" t="0" r="0" b="0"/>

<a:pathLst>

<a:path w="1705233" h="679622">

<a:moveTo>

<a:pt x="0" y="myGuide"/>

</a:moveTo>

<a:lnTo>

<a:pt x="1705233" y="myGuide"/>

</a:lnTo>

<a:lnTo>

<a:pt x="852616" y="0"/>

</a:lnTo>

<a:close/>

</a:path>

</a:pathLst>

</a:custGeom>

If however we change the guide to half that, namely 1/3. Then we see the entire bottom edge of the triangle move to now only occupy 1/3 of the total space within the shape bounding box. This is because both of the bottom points in this triangle depend on this guide for their coordinate positions. The triangle and corresponding DrawingML shown below illustrate this point.



<a:gdLst>

<a:gd name="myGuide" fmla="\*/ h 1 3"/>

</a:gdLst>

Specifies the formula that will be used to calculate the value for a guide. Each formula has a certain number of arguments and a specific set of operations to perform on these arguments in order to generate a value for a guide. There are a total of 17 different formulas available.

For more information on publishing content: “**Office open Xml Part4 –Mark Up Language Reference.pdf Page No. 3663 “.**

1. Architecture Consideration

Methodology for converting Visio formula in shapesheet is shown in fig.



1. Formula for Custom Geometry and Connection points are retrieved from Visio shapesheet using shape.get\_CellsSRC.Formula.
2. Formula retrieved from Visio shapesheet is in infix format, it needs to be transferred into postfix formula so that guides can easily be written in their proper format.

Eg: **Infix Formula:** 1+SQRT(X)\*COS(SIN(HEIGHT))

**Postfix Formula:** 1 x sqrt height sin cos \* +

1. Guide is then written for each token accordingly.

Current Add-in converts the following cells from the **Visio.Shape** object to PPTX:

* Width and height
* X-axis and y-axis position, measured from the top-left corner of the Visio drawing page
* Shape text, font, font color, and font size
* Paragraph alignment (vertical and horizontal) and text orientation
* Text block size
* Fill color and transparency
* Line color, weight, pattern, and line end styles
* Beginning and end points for connector shapes

1. Methodology

3.1 Formula Retrieval

Formula is retrieved from Visio shapesheet using shape.get\_CellsSRC.Formula.

3.2 Infix Formula to Postfix Formula

Formula retrieved from Visio shapesheet is in Infix Order, it needs to be transferred into postfix formula so that guides can easily be written in their proper format.

* + 1. Tokens Retrieval

For converting Infix formula into Postfix, tokens are retrieved from the formula and then the algorithm is applied to convert Infix formula into Postfix formula.

Here is the List Of tokens:

|  |  |
| --- | --- |
| **Type** | **Example** |
| Function | Sin, Cos, Tan, Sinh, Tanh, Cosh, Asin,  Acos, Atan, Mod, Sqrt, Max, Min, Log10, And, Or, Not. |
| Operators | +, -, \*, /, %, ,^,= , : , - , + , < , >,>=,<= . |
| Parentheses | (, ), {, }. |
| Variable | Geometry.x1, Width, Height ,23 |

* + 1. Converting Infix into Postfix Order

Visio infix to postfix algorithm is used for converting Infix formula into postfix formula, summarizes priority order in decreasing order as follows.

|  |  |
| --- | --- |
| **S.No.** | **Values** |
| 1. | {, },( ,) |
| 2. | Sin, Cos, Tan, Sinh, Tanh, Cosh, Asin,  Acos, Atan, Mod, Sqrt, Max, Min, Log10, And, Or, Not. |
| 3. | Uniary + , - |
| 4. | ^ |
| 5. | \*/ |
| 6. | + , - |
| 7. | > , <, <= ,>= |
| 8. | = ,<> |

**Special Case:** Since Max, Min, AND, OR functions can take more than two argument so special attention is given to it (maintaining a list of number of arguments it takes in that formula).

* 1. Writing Guide for Postfix Formula

**Algorithm** for writing Postfix formula into guide is describes as follows.

1. While there are input tokens left
   1. Read next token from input.
   2. If token is a variable or a constant
      1. Push it onto stack.
   3. Otherwise, the token is an operator (operator here includes both operators and functions)
      1. It is known a priori that the operator takes n arguments but special attention is given to Max, Min, AND, OR, since these functions can take variable number of arguments (maintaining list of arguments it takes while retrieving tokens for these functions).
      2. If there are fewer than n values on the stack then error
      3. Else, Pop the top n values from the stack.
      4. Call the functions related to the operator and write guide for it.
      5. Push the last written guide or constant onto the stack.
2. If there is one value in the stack that value is the pointer to the guide for that formula.
3. If there are more values in the stack then formula is incorrect but it’s a rare case since Visio itself handle such cases.

The main problem for writing guides in pptx is that there are only 17 functions in guides. But there are so many formulas in Visio so we need to transfer each and every function of Visio into Guides. List of how functions in Visio are converted into guides :

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **Functions** | **Guides Conversion** |
|  | Sin x | sin 1 x |
|  | Cos x | cos 1 x |
|  | Tan x | tan 1 x |
|  | ASIN | x + 1/6 x^3 + 3/40 x^5 + 5/112 x^7  Gd0=\*/x x 1  Gd1=\*/ gd0 x 1  Gd2=\*/gd1 gd0 1  Gd3=\*/gd2 gd0 1  Gd4=\*/gd1 1 6  Gd5=\*/gd2 3 40  Gd6=\*/gd3 5 112  Gd7=+-gd4 x 0  Gd8=+-gd5 gd7 0  Gd9=+-gd6 gd8 0 |
|  | ACOS | Description: Machine generated alternative text: I 13 35 57 ix—x— x —x —  Gd0= \*1 31415926535898 10^13  Gd1= \*gd0 1 2  Gd2= +- gd1 0 x  Gd3=\*/ x x 1  Gd4= \*/ gd3 x 1  Gd5=\*/ gd4 gd3 1  Gd6=\*/ gd5 gd3 1  Gd7=\*/gd4 1 6  Gd8=\*/gd5 3 40  Gd9=\*/gd6 5 112  Gd10=+ - gd7 gd8 0  Gd11 = + - gd10 gd9 0  Gd12 = + - gd2 0 gd11 |
|  | ATAN2 | at2 1 x |
|  | COSH | 1+x^2/2+x^4/24+x^6/720  Gd0=\*/x x 1  Gd1=\*/gd0 gd0 1  Gd2=\*/gd1 gd0 1  Gd3=\*/gd0 1 2  Gd4=\*/gd1 1 24  Gd5=\*/ gd2 1 720  Gd6=+-gd3 1 0  Gd7=+-gd6 gd4 0  Gd8=+-gd7 gd5 0 |
|  | SINH | x+x^3/6+x^5/60+x^7/5040  Gd0=\*/x x 1  Gd1=\*/ gd0 x 1  Gd2=\*/gd1 gd0 1  Gd3=\*/gd2 gd0 1  Gd4=\*/gd1 1 6  Gd5=\*/gd2 1 60  Gd6=\*/gd3 1 5040  Gd7=+-gd4 x 0  Gd8=+-gd5 gd7 0  Gd9=+-gd6 gd8 0 |
|  | TANH | x-x^3/3+2/15 x^5 – 17/315 x^7  Gd0=\*/x x 1  Gd1=\*/ gd0 x 1  Gd2=\*/gd1 gd0 1  Gd3=\*/gd2 gd0 1  Gd4=\*/gd1 1 3  Gd5=\*/gd2 2 15  Gd6=\*/gd3 17 315  Gd7=+-x 0 gd4  Gd8=+-gd7 gd5 gd6 |
|  | Sqrt x | sqrt x |
|  | Max(a,b,c) | Gd0 = max a b  Gd1 = max gd0 c |
|  | Min(a,b,c) | Gd0 = min a b  Gd1 = min gd0 c |
|  | Log10 (1+x) | x – x^2/2! + X^3/3! – X^4/4! + X^5/5! -- X^6/6!  Gd0 = +-y 0 1  Gd2 =\*/ gd0 gd0 2  Gd3 =\*/ gd2 gd0 6  Gd4=\*/ gd3 gd0 24  Gd5=\*/ gd4 gd0 120  Gd6=\*/ gd5 gd0 720  Gd7=+- gd0 gd3 gd2  Gd8=+- gd4 gd6 gd5  Gd9= +-gd7 0 gd8 |
|  | Add(a,b) | Gd0 = +-a b 0 |
|  | Subtract(a,b) | Gd0 = +-a 0 b |
|  | Multiply(a,b) | Gd0 = \*/a b 1 |
|  | Divide(a,b) | Gd0 = \*/ 1 a b |
|  | Pow (a,b) | If b is not integer then guide can’t be written  Else using **exponentiation by squaring method** (eg: POW(a,4) )  Gd0=\*/1 1 1  Gd1=\*/a a 1  Gd2=\*/gd1 gd1 1  Gd3=\*/gd0 gd2 1  Gd4=\*/gd2 gd2 1  Gd3 is the final value for Pow(a,4) |
|  | And (w, x, y, z, …) | (w ? (x ? (y ? (z? 1:0) :0) :0) :0)  Gd0= ?: z 1 0  Gd1=?: y gd0 0  Gd2=?: x gd1 0  Gd3=?:w gd2 0 |
|  | Or (w, x, y, z) | (w ? 1: (x ? 1: (y ? 1:(z? 1:0) ) ) )  Gd0= ?: z 1 0  Gd1=?: y 1 gd0  Gd2=?: x 1 gd1  Gd3=?: w 1 gd2 |
|  | Not(x) | x ? 0 : 1  Gd0 = ?: x 0 1 |
|  | Uminus (x) | + - 0 0 x |
|  | Bound(value,min,max) | Pin x y z |

Let’s take a simple formula and see how it’s written into guide.

**Eg: 1+SQRT(WIDTH)\*COS(SIN(HEIGHT))**

**OUTPUT:** <gd fmla="**sqrt WIDTH**" name="**gd0**"/>

<gd fmla="**sin 1 HEIGHT**" name="**gd1**"/>

<gd fmla="**cos 1 gd1**" name="**gd2**"/>

<gd fmla="**\*/gd0 gd2 1**" name="**gd3**"/>

<gd fmla="**+-1 gd3 0**" name="**gd4**"/>

* 1. Mapping of Visio’s Drawing primitives to Oart

Visio contains 10 drawing primitives. They are mapped to Oart’s primitives but currently handling 3 only, as mention below:

* + 1. LineTo/MoveTo

All the formula from X and Y cell are directly converted into guide as described above.

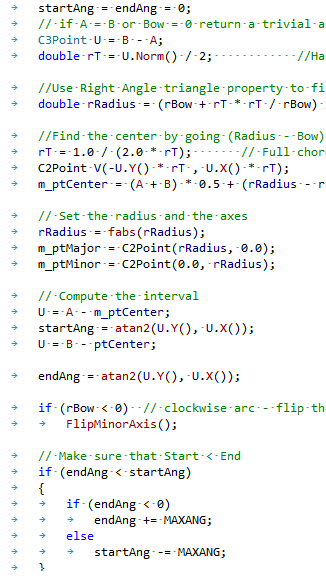
* + 1. ArcTo

Convert the Visio’s representation to the GVML Art::Path2DArcTo.

Visio has circular arc for arcTo and contains ending vertex of arc and the distance from the arc's midpoint to the midpoint of its chord: Bulge. Internally every arcTo curve in Visio is generated as an EllipticalArcTo and user can go and edit to convert geometry to ArcTo. So, arcTo is a special form of EllipticalArcTo. GVML supports just one arcTo.

ArcTo has the following set of parameters: EndVertexX, EndVertexY and Bulge. Also Starting point is known: StartVertexX, startVertexY.

GVML expects the parameters: Major Axis Length, Minor Axis Length, StartAngle and SweepAngle. Algorithm for converting Visio ArcTo to GVML is shown below:



**Implementation:** Following guides are written to convert Visio ArcTo to GVML as described below.

|  |
| --- |
| Gd0 = StartVertexX  Gd1 = EndVertexY //after converting it into GVML Coordinates  Gd2 = EndVertexX  Gd3 = EndVertexY //after converting it into GVML Coordinates  Gd4 = Bulge  Gd5 = + - 0 gd2 gd0  Gd6 = + - 0 gd3 gd1  Gd7 = mod gd5 gd6 0 //length of the chord  Gd8 = \*/ 1 gd7 2  Gd9 = \*/ gd8 gd8 gd4  Gd10 = + - gd4 gd9 0  Gd11 = \*/ 1 gd10 2 //rRadius = (rBow + rT \* rT / RBow) /2  Gd12 = \*/ 1 1gd7  Gd13 = + - 0 0 gd6  Gd14 = \*/ gd13 gd12 1  Gd15 = \*/ gd5 gd12 1  Gd16 = + - gd2 gd0 0  Gd17 = \*/ 1 gd16 2  Gd18 = + - gd3 gd1 0  Gd19 = \*/ 1 gd18 2  //now go distance of Radius-Bulge in direction perpendicular to the chord from midpoint of chord to find the center  Gd20 = + - 0 gd11 gd4  Gd21 = \*/ gd20 gd14 1  Gd22 = \*/ gd20 gd15 1  Gd23 = + - 0 gd17 gd21  Gd24 = + - 0 gd19 gd22  Gd25 = abs gd11  //Calculate the angle based on the center and points  Gd26 = + - 0 gd23 gd0  Gd27 = + - 0 gd24 gd1  Gd28 = at2 gd26 gd27  Gd29 = + - 0 0 gd28  Gd30 = + - 0 gd2 gd23  Gd31 = + - 0 gd3 gd24  Gd32 = at2 gd30 gd31  Gd33 = + - 0 0 gd32  //Adjust angle based on +- 2PI if required  Gd34 = ? : gd4 gd29 gd28  Gd35 = ? : gd4 gd33 gd32  Gd36 = + - 0 gd34 gd35  Gd37 = + - gd35 21600000 0  Gd38 = + - gd34 0 21600000  Gd39 = ? : gd35 gd35 gd37  Gd40 = ? : gd35 gd38 gd34  Gd41 = ? : gd36 gd39 gd35  Gd42 = ? : gd36 gd40 gd34  Gd43 = + - 0 gd41 gd42  Gd44 = + - gd42 gd41  Gd45 = ? : gd4 gd43 gd44  Gd46 = + - 0 0 gd42 //Start Angle  Gd47 = + - 0 0 gd45 //Sweep Angle |

* 1. Writing Connectors

Connectors in PPTX are implemented through 2 ways.

1. Custom Geometry
2. Preset Geometry

But Custom Geometry is not supported by PPTX uptil now. So connectors are implemented through Preset Geometry. In Preset Geometry, three type of connectors are defined-

1. Straight connector
2. Bent Connector (bent connector 1, bent Connector 2, .., bent connector 5)
3. Curved Connector (curved connector 1, curved connector 2, , curved connector 5)

**Implementation:** Connectors are decide upon the count of the total number of ArcTo (includes EllipticalArcTo, Ellipse, ArcTo) and LineTo Tag type. If ArcTo count is not zero then curved connector is used and curved connector type depends upon the count of ArcTo with maximum curved connector 5 is possible. Otherwise bent connector is use and type of bent connector depends upon the count of LineTo Tag Type.

* + 1. Id and Idx of the shape

To connect Connector to the shape, both id (shape to which connector is connected) and idx (connection point of the shape to which connector is connected) need to be known.

For id and idx, BeginX and EndX of the connector is use. Connectors “BeginX” and “EndX” in Visio are defined with two formulas:

1. **PAR (PNT (Square! Connections.X2, Square! Connections.Y2)):** It is a static type of connection point. It is static because even when the shape moves, connection point that the connector is glued to, doesn’t change.

1

2

**Before shifting**



2

1

**Connection point remain same when shape2 moves**

Both id and idx is easily calculate from the formula. Square gives the id and Connections.X2 in the formula of BeginX/EndX gives the idx value.

1. **WALKGLUE (EndTrigger, BegTrigger, WalkPreference):** It is a dynamic type of connection point. It is dynamic because Connection point change as shape moves.

1

2

1

**Before shifting**

1

2

**Connection point change when shape2 moves**

**Implementation:** It’s not possible to give such dynamic behavior in PPTX. So static glue is used while transferring connector from Visio to PPTX. For determining the id and idx values, BegTrigger and EndTrigger of the connector shape is used. BegTrigger / EndTrigger in Visio are defined as: XFTRIGGER (Square! EventXFMod).

* Square gives the id value.
* For idx value, both the connection list of the shape and MoveTo/ LastLineTo (depending upon whether BeginX/ EndX) are mapped to page coordinate, matching between these two values is done and the best match among these gives the idx point.

4. Limitations while Converting Formula

into Guides

4.1 Formula Limitation

1. Pow (a,b) can’t be converted into guide if b is not integer.
2. Problem of precision while using Taylor Series for functions are :
   1. ACos(x) when x reaches from 0.95 to 0.99 then error increases

0.95 - 0.317560 (expected)

0.38869 (calculated)

0.99 - 0.14153 (expected)

0.30614 (calculated)

2.2 Problem with log10

Log (1+x) = x – x^2/2! + X^3/3! – X^4/4! + X^5/5! - X^6/6!

So where to stop it.

1. For BITAND (a, b), too many guides are required to write.
   1. For retrieving each bit of a, need to divide it by 2 for 32 times, so 32 guides for **“a”** and 32 guides for **“**b**”**.
   2. AND each bit of **“a”** and **“b”,** hence 32 guides for 32 bits.
   3. Combine all the bits to get the final value, hence 32 guides are needed for 32 bits.

Total of 128 guides are needed for BITAND (a, b) and if considered it as 64bit, imagine how many guides are needed for it.

4.2 Connectors Limitation

1. Dynamic Behavior of Connection points in PPTX is not possible.
2. Custom Geometry of the connectors are not possible. Connectors with more than 5 bents are listed with only 5 bents since uptil now BentConnector5 is defined in Preset Geometry and same is the case with the curved connector.
3. Some of the shapes in Visio do not specify the connector angle i.e. the angle at which the connector is glue to the shape. DirX/A and DirY/B specifies the angle of the connector but in some shapes either both the values are “0 “e.g. Decagon or it contains “No **formula** “e.g. Flow chart Shapes.

5. Code Components

5.1 Code Overview

There are Mainly 5 classes for writing shapes in Visio to PPTX which are described as follows.

1. **Save To PPTX:** It interacts with Visio and retrieve shapesheet data in form of formula.
2. **Shape Writer:** It writes shape and fill color into PPTX. It interacts with infix2Postfix class to get the formula in Postfix order and pass this postfix formula to FunctionGuide class that write guides for each cell in shapesheet in PPTX.
3. **Infix2Postfix:** It converts Prefix Formula into Postfix formula after retrieving tokens from infix Formula.
4. **Function Guide:** It writes guides for each Postfix Formula into PPTX using algorithm described in section 3.3.

**5. PPTXWriter:** It writes the PPTX and save it.

5.2Sequence Diagram

Sequence diagram is shown here to describe how classes interact with Visio and another classes

