The Hidden World of Visio Shapes

Chris Roth
September 9, 2006

So you've created beautiful graphics in Visio, and you know how to add phantasmal ShapeSheet smarts to your shapes, now its time for you to make it all disappear!

I often get asked how to toggle the visibility of various elements of Visio drawings, and I usually offer up one of many solutions. But I don't think that those "many solutions" have ever been catalogued in a single place. Until, of course, now…

The Traditional Way

Let's start with the traditional method of hiding things – the method I personally use most often. This method uses cells that specifically control the visibility of various parts of the shape.

The ShapeSheet has a number of cells that turn components on and off, all of which are TRUE/FALSE cells:

GeometryN.NoFill
GeometryN.NoLine
GeometryN.NoShow
Miscellaneous.HideText

While NoFill and NoLine allow you to fine-tune the look of a geometry section, NoShow makes the whole component invisible.

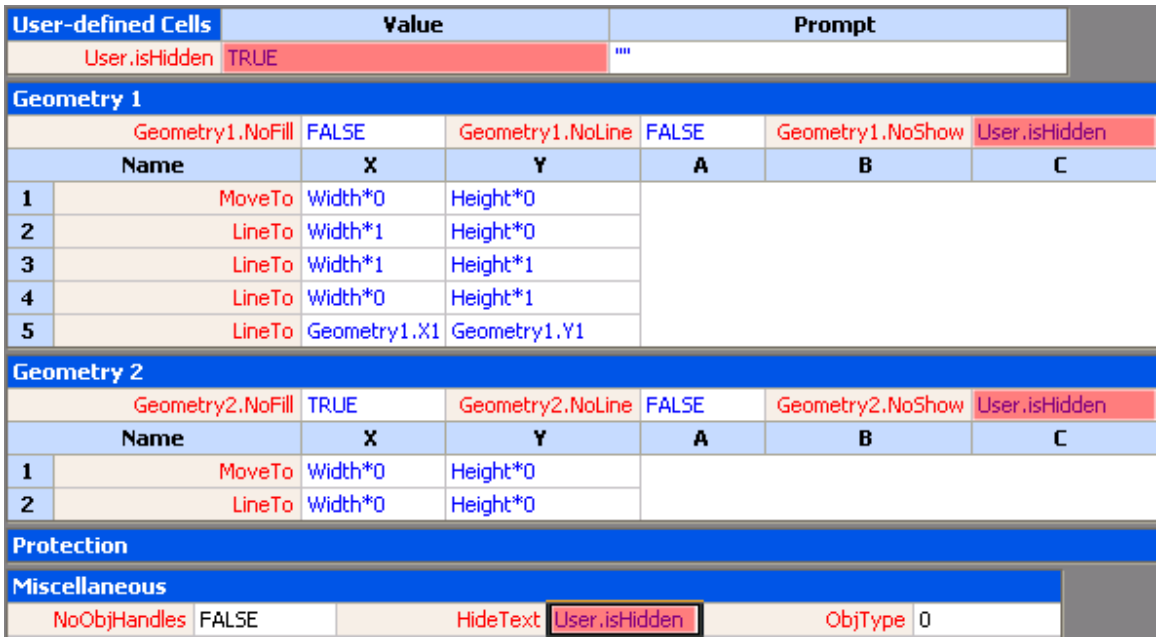
Of course, sometimes you may want to hide an entire shape – like when it is a sub-shape inside of a group. In this case, you have to set every single NoShow cell in each geometry section, because there is no cell that hides the entire shape.

If you need to hide the text of the shape, that is easily accomplished via the Miscellaneous section's HideText cell.

If you have lots of elements to show and hide, it's good to centralize the control in a user-cell. Below, we can see a shape configured to hide all geometry and text, according to the value of User.isHidden.

Since all of the Visio's hide-cells are true for the invisible state, it's good to match that grammar when naming of your parameter. Thus, when User.isHidden is true, things get hidden.

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de



User-defined Cells	Value	Prompt
User.isHidden	TRUE	""

Geometry 1

Geometry1.NoFill	FALSE	Geometry1.NoLine	FALSE	Geometry1.NoShow	User.isHidden

	Name	X	Y	A	B	C
1	MoveTo	Width*0	Height*0			
2	LineTo	Width*1	Height*0			
3	LineTo	Width*1	Height*1			
4	LineTo	Width*0	Height*1			
5	LineTo	Geometry1.X1	Geometry1.Y1			

Geometry 2

Geometry2.NoFill	TRUE	Geometry2.NoLine	FALSE	Geometry2.NoShow	User.isHidden

	Name	X	Y	A	B	C
1	MoveTo	Width*0	Height*0			
2	LineTo	Width*0	Height*0			

Protection

Miscellaneous

NoObjHandles	FALSE	HideText	User.isHidden	ObjType	0

Formatting

Of course, if you need to hide an entire shape, it seems like it makes sense to just use formatting. Your natural inclination might be to use the color cells to hide our shape:

Fill.Format.FillForegnd
Fill Format.FillBkgnd
Fill Format.ShdwForegndTrans
Fill Format.ShdwBackgndTrans
Line Format.LineColor
Character.Char.Color

However, there isn't really a clear color, so we might choose white (color 1) to match the paper. This might work for most scenarios, but feels like a bit of a hack. You'd get into trouble if you had a shape hidden in this way laying on top of a visible shape, because the white would cover up the other shape.

In thinking about a clear color, we realize that it's not the color that's clear, but the pattern that's none! So let's consider the pattern cells:

Fill Format.FillPattern = IF(User.isHidden, 1, 0)
Fill Format.ShdwPattern
Line Format.LinePattern

While 0 represents the no-pattern for line and fill, there's no equivalent for text. So we'll still need to use Miscellaneous.HideText to make our characters invisible.

A shape that is 100% transparent certainly has a clear color, so this might be another strategy: make the shape completely see-through! These cells control transparency in a shape:

Fill.Format.FillForegndTrans IF(User.isHidden, 100%, 0)

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de

Fill.Format.FillBkgrndTrans
Fill.Format.ShdwForegndTrans
Fill.Format.ShdwBackgndTrans
Line Format.LineColorTrans
Character.Char.ColorTrans

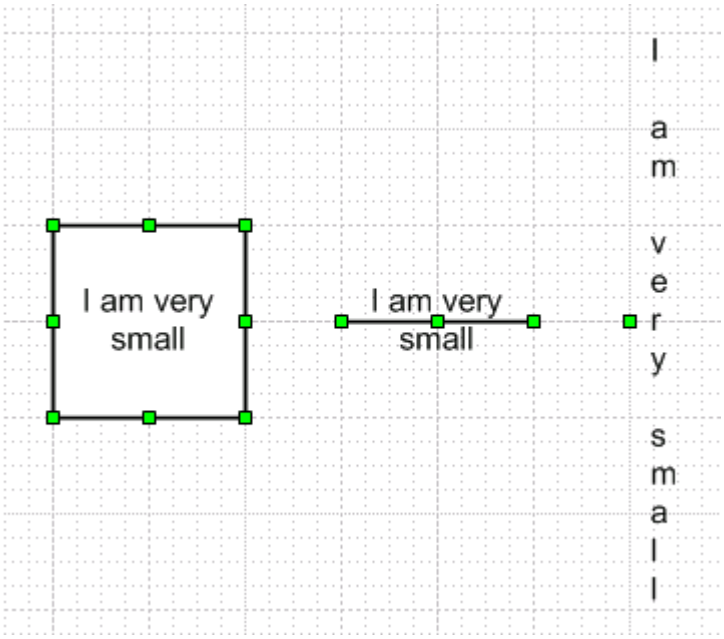
This is nice, in that it doesn't change the color of the shape in any way. But it can affect just as many cells as The Traditional Way, unless you have a very large number of geometry sections. Plus, it requires that messy IF statement that is considerably more typing than "= User.isHidden."

Another danger with using formatting cells has to do with styles. Even if you protect your formulas with the GUARD() function, the application of styles will blow the formula away. If a user chooses individual formatting options, like color, or pattern from a toolbar or dialog, then GUARD() survives. But if the user applies a style, then GUARD() gets blasted.

Size

See-through was a nice idea, but how about: too small to see? We can hide entire shapes by simply making them 0 x 0! Yes, you can make them infinitesimally small!

The only caveat here is that the font-size won't automatically shrink with the shape. If you don't use the HideText cell, then you'll end up with an ugly, very tall, single-character wide column of text floating above an invisible shape!



Note that this strategy will require formulas in the shape's Width and Height cells. If you GUARD these cells to protect your formulas, then the user won't be able to change the

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de

size of the shape. However, if you're using Visio 2003, though, then this won't be a problem. A new function, SETATREFEXPR, will let you have the best of both worlds.

This is how you would set the width and height formulas:

```
Shape Transform.Width = SETATREFEXPR( 3in ) * NOT( User.isHidden )
Shape Transform.Height = SETATREFEXPR( 3in )*NOT(User.isHidden)
```

If User.isHidden is TRUE, or 1, then the expressions evaluate to zero, and the shape disappears. If FALSE, or 0, then the formula evaluates to the last size to which the shape was sized. This last-size-value is stored in the parentheses and is shown as 3in in this example. I know it looks a bit weird, but it works!

Crop

If you have a foreign object in a shape, such as a bitmap or a metafile, you can do an interesting technique that's related to cropping. You can simply shift the image out of the viewing panel!

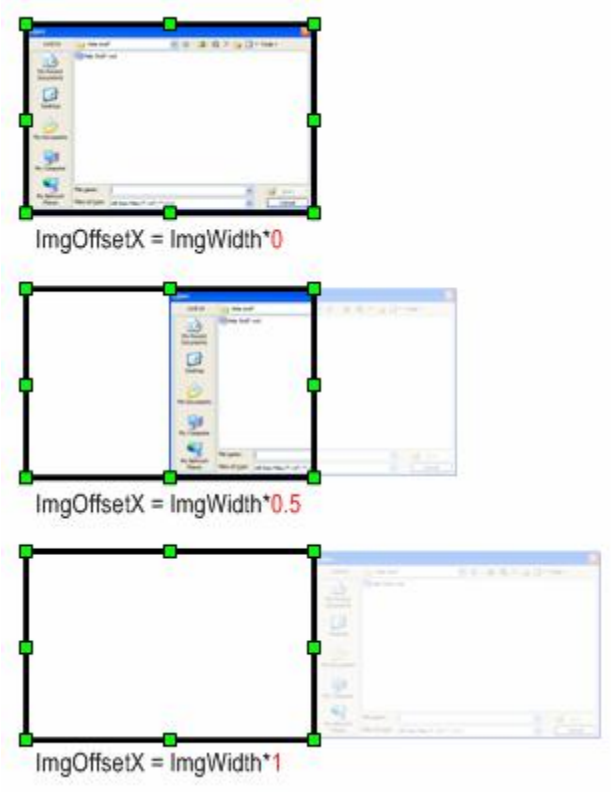
When you have a foreign object in Visio, the ShapeSheet will contain the section: Foreign Image Info. This controls the zoom of the image relative to its shape-container, and the degree to which it is panned. By default, an image exactly matches its containing shape, and we have four very boring formulas, as shown:

Foreign Image Info			
ImgWidth	Width*1	ImgOffsetX	ImgWidth*0
ImgHeight	Height*1	ImgOffsetY	ImgHeight*0

You can simply change either the X or Y offset cells to be "times one", and that will completely shift the image out of view, giving you nothing to look at!

```
Foreign Image Info.ImageOffsetX = ImgWidth * User.isHidden
Foreign Image Info.ImgOffsetY = ImgHeight*0
```

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de



Notice the black line on the shape. That's actually geometry! By default a foreign image shape has one geometry section that forms a rectangle around the shape. You'll want to delete that section or hide it as well

Layers

You can turn off a whole class of shapes by simply assigning them to a layer, then turning off the visibility of that layer. You add a shape to a layer via the *Format > Layer dialog*, and you change layer properties via the *View > Layer Properties…* dialog.

And you can even manipulate layer visibility via the ShapeSheet. Suppose we have three layers on a page: Bob, Steve and Connector. In the page's ShapeSheet, the Layers section would look something like this:

Layers		Visible	Print	Active	Lock
1	"Bob"	1	1	0	0
2	"Steve"	1	1	0	0
3	"Connector"	1	1	0	0

We could add a right-mouse action to a shape, or to the page that would allow users to quickly toggle the visibility of this layer. In our case, layer Steve is in the second row, so it's (his?) visibility cell is called Layers.Visible[2]

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de

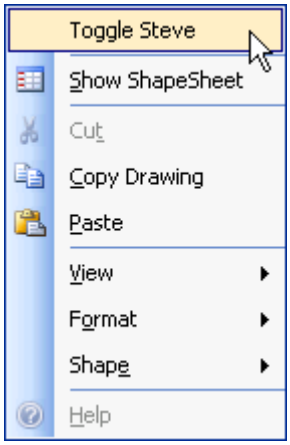
```
Actions.ToggleSteve.Action  =
SETF(GetRef(ThePage!Layers.Visible[2]),NOT(ThePage!Layers.Visible[2]))
```

An action in the page's ShapeSheet doesn't need the ThePage! Prefix, so the formula would be simpler:

```
Actions.ToggleSteve.Action = SETF(GetRef(Layers.Visible[2]),NOT(Layers.Visible[2]))
```

It's worth pointing out that it might not be such a good idea to put this action on a shape that belongs to the "Steve" layer itself, because you won't have anything to right-click on the first time you hide the layer!

But right-mouse actions on the page are cool. You can quickly right-click the page, and toggle the "Steve" layer, and our life's goals instantly become fulfilled!



Handles

You can also show and hide elements of the user interface, i.e.: handles and highlight-boxes. The miscellaneous section has five cells that deal with UI:

```
Miscellaneous.NoObjHandles
Miscellaneous.NoCtlHandles
Miscellaneous.NoAlignBox

Miscellaneous.DynFeedback
Miscellaneous.NoLiveDynamics
```

Normally, when you resize a shape, you see what it looks like, as you are dragging. If your shape is especially complex, this can bog down, and be jerky and distracting. It's a good idea to set NoLiveDynamics to TRUE for complex shapes. Then the user will just see a "rubber-band" outline of the shape's width and height as a sizing handle is dragged.

DynFeedback seems to only apply the Dynamic Connector shape, and controls the complexity of the ghosting displayed by the connector as you drag an endpoint. This cell

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de

didn't seem to really affect anything. I believe it might be a leftover from older versions of the Dynamic Connector, and isn't particularly interesting anymore.

You can have shapes that are visible, but don't print by setting the cell:

```
Miscellaneous.NonPrinting
```

Individual Control Handles

NoCtlHandles will turn off *all* of your control handles, but if you want to hide individual control handles, there is another technique: just add 5 to either the .XCon or .YCon cells. These cells are seen in the ShapeSheet under the column names: "X Behavior" and "Y Behavior", but for some reason, the cells have these funny "Con" names. They specify how the control responds when the shape is resized (e.g.: "stay relative to the left/bottom" or "reposition proportionally")

A common application of this technique is used for control handles that control the location of the text block. If the shape has no text, then we don't want to have a control handle dangling in space. So we check for no-text, and turn off the control handle if this is true.

The system of formulas looks something like this:

```
Text Transform.TxtPinX = Controls.Row_1
Text Transform.TxtPinY = Controls.Row_1.Y

Controls.Row_1 = 0 + 5 *  StrSame( ShapeText (TheText ), "" )
```

So if StrSame() is TRUE, that means we have no text, and we also have 5 x 1, which will hide the control handle. You only have to do this in either the X or Y cell, not both. I put 0 + at the beginning to illustrate that the 5-part can be separate from whatever control handle behavior you choose.

Automation methods:

Of course, there are many objects that you can hide via automation as well. These are fully documented in the Developer Reference help that ships with Visio, or you can get even more information from the Visio SDK. I'll at least point you in the right direction here by giving you a rundown of the properties that you can set, and the objects they affect.

Visible property:

- Application object
- InvisibleApp object
- Menu object
- MenuItem object

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de

- MenuSet object
- Toolbar object
- ToolbarItem object
- Window object

Hidden property:

- Master object
- Style object

You can also create an invisible instance of the Visio application with code such as this:

```
Visio.InvisibleAppvisAppInvis = new Visio.InvisibleApp();
```

Well, that’s about it for now. Time for me to disappear!

The World of Hidden Shapes.pdf

PDF wurde mit pdfFactory Pro-Prüfversion erstellt. www.context-gmbh.de