Creating SmartArt with docx4j

Warning: this feature is not fully implemented at present. If you wish to use it, you may need to be prepared to get your hands dirty. You will probably have to look at the SmartArt XML, and will probably need to edit/add to the XSLTs. In other words, it is recommended for docx4j developers only!

docx4j supports reading docx and pptx files which contain SmartArt.

From docx4j 2.7.0, you can also:

* work with glox files
* generate SmartArt (with somewhat less effort)

# Glox files

A .glox file is the definition for a piece of SmartArt.

See "Creating a Layout Definition File Template" at <http://msdn.microsoft.com/en-us/library/dd439448(v=office.12).aspx>

A glox file is an OpenXML package containing just 2 parts.

The code in src/glox4j/java allows you to open such a package.

You can extract a glox file for a piece of SmartArt from a pptx or docx containing that SmartArt (see ExtractGloxFromExemplar).

If the glox contains sample data, you can go the other way ie use the glox to generate a sample docx or pptx.

# Generating SmartArt

Microsoft Word and Powerpoint (2007 and later) provide a user friendly interface for creating SmartArt:



As the image above suggests, SmartArt is basically just a list of text (or nested lists) rendered as a graphic.

However, the Office products do not offer a developer friendly way of providing that text, and saying "make me some SmartArt from it"

The problem is that you need to provide a data part (eg /word/diagrams/data1.xml) - DiagramDataPart in docx4j - which is rather complex.

* those 6 list items translate into 376 lines of data1.xml
* it is hard to produce a dataModel which the Office applications will accept.

We need a simpler format to describe this data, so it can be used to generate corresponding SmartArt (ie Microsoft's dataModel).

Our simpler format:

* must represent the nested text list
* since the text can be formatted, provide a way to do that
* provide a way to represent images (since some SmartArt can have pictures in it)

The approach docx4j takes is to transform this simpler format into an appropriate diagram data part. The diagram data part (along with the other 3 required SmartArt parts) can then be injected into a docx or pptx. (In principle, xlsx would work as well, although there is no sample to demonstrate this yet).

# Hierarchical data format (SmartArtDataHierarchy.xsd)

Consider the following example:



In our format, this is represented as:

<?xml version="1.0" encoding="utf-8"?>

<odgm:SmartArtDataHierarchy loTypeId="urn:microsoft.com/office/officeart/2009/layout/CirclePictureHierarchy"

xmlns:odgm="http://opendope.org/SmartArt/DataHierarchy" >

<odgm:list>

<odgm:listItem id="0">

<odgm:list>

<odgm:listItem id="1">

<odgm:textBody>

<odgm:p>apes sub-tree</odgm:p>

<odgm:p>of primates</odgm:p>

</odgm:textBody>

<odgm:list>

<odgm:listItem id="2">

<odgm:textBody>

<odgm:p>gibbon</odgm:p>

</odgm:textBody>

<odgm:imageRef contentRef="{00000043-0000-0000-0000-000000000000}"/>

</odgm:listItem>

<odgm:listItem id="22">

<odgm:textBody/>

<odgm:list>

<odgm:listItem id="25">

<odgm:textBody/>

<odgm:list>

<odgm:listItem id="8">

<odgm:textBody>

<odgm:p>gorilla</odgm:p>

</odgm:textBody>

<odgm:imageRef contentRef="{00000067-0000-0000-0000-000000000000}"/>

</odgm:listItem>

<odgm:listItem id="28">

<odgm:textBody/>

<odgm:list>

<odgm:listItem id="11">

<odgm:textBody>

<odgm:p>human</odgm:p>

</odgm:textBody>

<odgm:imageRef contentRef="{00000079-0000-0000-0000-000000000000}"/>

</odgm:listItem>

<odgm:listItem id="31">

<odgm:textBody/>

<odgm:list>

<odgm:listItem id="14">

<odgm:textBody>

<odgm:p>chimp</odgm:p>

</odgm:textBody>

<odgm:imageRef custLinFactNeighborY="1712" custLinFactNeighborX="1230" contentRef="{00000091-0000-0000-0000-000000000000}"/>

</odgm:listItem>

<odgm:listItem id="15">

<odgm:textBody>

<odgm:p>bonobo</odgm:p>

</odgm:textBody>

<odgm:imageRef contentRef="{00000097-0000-0000-0000-000000000000}"/>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

<odgm:listItem id="5">

<odgm:textBody>

<odgm:p>orangutan</odgm:p>

</odgm:textBody>

<odgm:imageRef contentRef="{00000055-0000-0000-0000-000000000000}"/>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

</odgm:list>

</odgm:listItem>

</odgm:list>

<odgm:images>

<odgm:image contentType="image/jpeg" id="{00000043-0000-0000-0000-000000000000}">/9j/4AAQSkZJRgABA...NscHF36eVVOV//Z</odgm:image>

<odgm:image contentType="image/png" id="{00000067-0000-0000-0000-000000000000}">iVBORw0KGgoAAAANSU...AAAAElFTkSuQmCC</odgm:image>

<odgm:image contentType="image/jpeg" id="{00000079-0000-0000-0000-000000000000}">/9j/4AAQSkZJRgABA...Nl1BQaUgggg/9k=</odgm:image>

<odgm:image contentType="image/jpeg" id="{00000091-0000-0000-0000-000000000000}">/9j/4AAQSkZJRgABA...Q9UIeqEPVCH/9k=</odgm:image>

<odgm:image contentType="image/jpeg" id="{00000097-0000-0000-0000-000000000000}">/9j/4AAQSkZJRgABA.../8AbUdTST//2Q==</odgm:image>

<odgm:image contentType="image/jpeg" id="{00000055-0000-0000-0000-000000000000}">/9j/4AAQSkZJRgABA...lJJNN2Oyh1f/9k=</odgm:image>

</odgm:images>

<odgm:texts/>

</odgm:SmartArtDataHierarchy>

Notice the hierarchical/nested list structure.

A list is made up of one or more list items;

A list item:

* has an id
* may contain
  + text

<odgm:textBody>

<odgm:p>gibbon</odgm:p>

</odgm:textBody>

* + a reference to an image

<odgm:imageRef contentRef="{00000043-0000-0000-0000-000000000000}"/>

images are binary image data (eg GIF, JPEG, PNG, TIFF) base64 encoded

* + a nested list

<odgm:list>

Ultimately, this hierarchical/nested format will be transformed into an appropriate diagram data part. Microsoft's diagram data part format is "flattened" ie this explicit hierarchical structure is not represented in the natural XML way, but rather, as a set of connections (dgm:cxn) defined between nodes:

<dgm:cxnLst>

<dgm:cxn srcId="1" destId="2" />

<dgm:cxn srcId="2" destId="3" />

As a way to get started, org.docx4j.openpackaging.parts.DrawingML.**DiagramDataUnflatten** can be used to generate our hierarchical/nested format from a piece of SmartArt in a sample pptx or docx. (This approach was used to create the above example).

The XSD may be found in xsd/OpenDoPE/SmartArtDataHierarchy.xsd

The OpenDoPE namespace is used, because ultimately it is envisaged that OpenDoPE will allow the creation of SmartArt via this approach.

Note also the package org.opendope.SmartArt.dataHierarchy, which is a JAXB representation of SmartArtDataHierarchy.xsd

# Creating SmartArt

docx4j contains 2 classes:

* org.docx4j.openpackaging.parts.DrawingML.CreateDocxWithSmartArt
* org.docx4j.openpackaging.parts.DrawingML.CreatePptxWithSmartArt

which can be used as a starting point to create SmartArt using the above data format.

As you will see from their main method, you need:

* an XML file specifying the list of text items you want to render graphically, as described in the previous section
* the layout definition for the SmartArt, either in the docx already, or from a glox file
* an XSLT which can convert a transformed version of that XML file into a SmartArt data file (DiagramLayoutTree4AlgHier.xslt in the main method).

It is this XSLT (and quite possibly the one used in DiagramLayoutPart.generateLayoutTreeXSLT) which is/are likely to need work if you wish to use the stuff in docx4j 2.7.0 for your own SmartArt layouts.

DiagramLayoutPart.generateLayoutTreeXSLT uses an XSLT to interpret the layout part, creating as output an XSLT which is capable of converting our hierarchical/nested format into a tree of dgm:layoutNode. For example:

<?xml version="1.0" encoding="UTF-8"?>

<dgm:layoutNode xmlns:dgm="http://schemas.openxmlformats.org/drawingml/2006/diagram"modelId="N10002" name="hierChild1" presAssocID="0">

<dgm:layoutNode modelId="N10020" name="hierRoot1" presAssocID="1">

<dgm:layoutNode modelId="N10023" name="composite" presAssocID="1">

<dgm:layoutNode modelId="N10026" styleLbl="node0" name="image" depth="1" presStyleIdx="1" presStyleCnt="52" presAssocID="1"/>

<dgm:layoutNode modelId="N1002D" styleLbl="revTx" name="text" depth="1" presStyleIdx="0" presStyleCnt="1" presAssocID="1"/>

</dgm:layoutNode>

<dgm:layoutNode modelId="N10034" name="hierChild2" presAssocID="0">

<dgm:layoutNode modelId="N10037" name="Name10" presAssocID="2"/>

<dgm:layoutNode modelId="N1003A" name="hierRoot2" presAssocID="2">

<dgm:layoutNode modelId="N1003D" name="composite2" presAssocID="2">

<dgm:layoutNode modelId="N10040" name="image2" depth="2" presStyleIdx="2" presStyleCnt="52" presAssocID="2"/>

<dgm:layoutNode modelId="N10046" styleLbl="revTx" name="text2" depth="2" presStyleIdx="0" presStyleCnt="2" presAssocID="2"/>

</dgm:layoutNode>

<dgm:layoutNode modelId="N1004D" name="hierChild3" presAssocID="0"/>

</dgm:layoutNode>

<dgm:layoutNode modelId="N10050" name="Name10" presAssocID="22"/>

<dgm:layoutNode modelId="N10053" name="hierRoot2" presAssocID="22">

<dgm:layoutNode modelId="N10056" name="composite2" presAssocID="22">

<dgm:layoutNode modelId="N10059" name="image2" depth="2" presStyleIdx="3" presStyleCnt="52" presAssocID="22"/>

<dgm:layoutNode modelId="N1005F" styleLbl="revTx" name="text2" depth="2" presStyleIdx="1" presStyleCnt="2" presAssocID="22"/>

</dgm:layoutNode>

<dgm:layoutNode modelId="N10066" name="hierChild3" presAssocID="0">

<dgm:layoutNode modelId="N10069" name="Name17" presAssocID="25"/>

<dgm:layoutNode modelId="N1006C" name="hierRoot3" presAssocID="25">

<dgm:layoutNode modelId="N1006F" name="composite3" presAssocID="25">

<dgm:layoutNode modelId="N10072" name="image3" depth="3" presStyleIdx="4" presStyleCnt="52" presAssocID="25"/>

<dgm:layoutNode modelId="N10078" styleLbl="revTx" name="text3" depth="3" presStyleIdx="0" presStyleCnt="2" presAssocID="25"/>

</dgm:layoutNode>

:

:

<dgm:layoutNode modelId="N10118" name="Name17" presAssocID="5"/>

<dgm:layoutNode modelId="N1011B" name="hierRoot3" presAssocID="5">

<dgm:layoutNode modelId="N1011E" name="composite3" presAssocID="5">

<dgm:layoutNode modelId="N10121" name="image3" depth="3" presStyleIdx="11" presStyleCnt="52" presAssocID="5"/>

<dgm:layoutNode modelId="N10127" styleLbl="revTx" name="text3" depth="3" presStyleIdx="1" presStyleCnt="2" presAssocID="5"/>

</dgm:layoutNode>

<dgm:layoutNode modelId="N1012E" name="hierChild4" presAssocID="0"/>

</dgm:layoutNode>

</dgm:layoutNode>

</dgm:layoutNode>

</dgm:layoutNode>

</dgm:layoutNode>

</dgm:layoutNode>

The final XSLT (for example DiagramLayoutTree4AlgHier.xslt) is used to convert this dgm:layoutNode tree (in conjunction with our initial xml, which it reads again) to a diagram data part.