Sdt Content Control conventions for repeats & conditionals  
OpenDoPE: Open Document Processing Ecosystem

# Introduction

Creating documents programmatically, or programmatically inserting data into a template, is a very common requirement.

A naive approach for doing this is via search/replace. This is problematic, for several reasons:

* Microsoft Word break your text string across several XML elements, so searches fail.
* It doesn't extend naturally to conditional inclusion/exclusion of existing paragraphs, nor to repeating table rows etc

Legacy approaches use Microsoft Word fields, and bookmarks.

In contrast, ISO/IEC 29500  (OpenXML) and Word 2007's content control data binding provides a natural, modern way to insert text, for example

Dear Click here to enter text.

That leaves the question of how to handle things like:

**conditional inclusion** of paragraphs or other units of content

**repeat** (eg of list items, table rows, or other units of content)

inclusion of other documents (**altChunk**)

# What is content control data binding?

A content control is a container for content. A rich text content control can contain paragraphs and tables, or just a run of text.

The OpenXML element for a content control is <w:sdt>.

A user can include an arbitrary XML document in their docx file - such an XML file is called a custom xml part.

A content control can be bound to an element in the custom xml file, via an XPath expression. In this case, the content control includes a <w:databinding>:

<w:sdt>

<w:sdtPr>

<w:dataBinding w:xpath="/invoice[1]/customer[1]/name[1]"/>

# The value of content control data binding

The beauty of this is that the binding is bi-directional:

when you open a docx in Word, the contents of each content control is updated with the value of the XPath

if the user types something else in the content control, Word automatically copies that value into the CustomXML document

This means that users can change the XML "answer file" simply by working in Word. No additional work is required by the programmer to enable this.

Similarly, updates to the answer file are automatically reflected in the docx. No additional work is required by the programmer.

This capability is a natural basis for a modern document processing system.

The other thing which is nice about custom xml, is that the XML can conform to whatever schema you wish to use in your business. You aren't forced to use the proprietary schema of any vendor.

# What are Repeats and Conditions?

A document processing solution typically has to manage conditional inclusion, and repeats.

A content control is said to be ***conditional*** if it (and its contents) are included/excluded from the document based on whether some condition is true or false.

Without a way to say that a content control is conditional, an XML file can't control whether paragraphs, tables etc appear in a document.

A content control is a ***repeat*** if it designates that its contents are to be included more than once.

For example, a row of a table for each invoice/order item, or person.

Without a way to say that a content control is to be repeated, an XML file can't contain variable amounts of repetitive content.

# Problem Statement

The problem is that the Open XML specification does not standardise how conditionals and repeats can be done, and nor does Microsoft give any guidance or convention.

This is a significant limitation on document generation, which each system typically has to address.

A standardised way of doing repeats and conditionals would prevent businesses from re-inventing the wheel, and provide for enhanced interoperability.

Further, the specification is silent on how to interact with the user to modify the XML data.

In summary, this document provides a standard solution to the following requirements:

1. store user data in an XML format of their choice
2. include that data in the Word docx, as per OpenXML spec
3. provide a way of expressing:
   1. conditional inclusion/exclusion
   2. repeats
   3. component document inclusion
4. facilitate both interactive and non-interactive processing of the above

The purpose of this document is to suggest a convention, which various tools could implement.

# Convention

The convention is to use the Content Control's **tag** to indicate that that control is a condition, repeat, or component inclusion.

What we actually put there is a condition ID or repeat ID.

The actual conditions and repeats are defined in their own XML parts.

These parts are separate from the user's custom XML part, which is not altered by this convention.

# Processing model

The tag is preprocessed via an appropriate tool, to produce a new docx document. This convention supports both:

* ***non-interactive*** processing (whereby the custom xml answer file is processed to determine the docx content), and
* ***interactive*** processing (whereby a user is asked a series of questions, to determine the docx content).

Any content controls with a condition ID tag evaluated to false will be missing from this new docx document.

Any content controls which has repeat ID tag will have their content appear n times, where n is the number of child nodes

Any content controls which has component ID tag will its content replaced by the docx specified. This docx component can itself contain OpenDoPE markup, in which case it also needs to be processed.

Interactive processing typically works by updating the user's custom XML part in response to the user's answers to questions; the user's custom XML part can then be processed in the same way it would be processed in non-interactive processing, to generate the document content proper.

# Example documents

An example can be found in <http://dev.plutext.org/svn/docx4j/trunk/docx4j/sample-docs/databinding/>

invoice.docx contains examples of conditionals and repeats, using the proposed conventions.

The custom xml used in the example is:

<invoice>

<customer>

<name>Joe Bloggs</name>

</customer>

<items>

<item>

<name>apples</name>

<price>$20</price>

</item>

<item>

<name>bananas</name>

<price>$30</price>

</item>

<item>

<name>cherries</name>

<price>$40</price>

</item>

<total>$90</total>

</items>

<misc>

<includeBankDetails>true</includeBankDetails>

</misc>

</invoice>

invoice\_preprocessed\_OUT.xml is the result of processing invoice.docx, using the docx4j implementation of this convention.

invoice\_bound\_OUT.xml is the result of processing all the binding information (ie the equivalent of what Word does when opening invoice\_preprocessed\_OUT.xml).

Notice that Word 2007 can open all 3 documents, and behaves as one would expect.

# XPath definitions

There is a Custom XML Part which specifies all the XPath expressions used, in repeats, conditions, and standard bindings.

For example:

<od:xpaths xmlns:od="http://opendope.org/xpaths">

<od:xpath id="x1">

<od:dataBinding storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" xpath="/invoice[1]/customer[1]/name[1]"/>

</od:xpath>

<od:xpath id="x2">

<od:dataBinding storeItemID="{8b049945-9dfe-4726-9de9-cf5691e53858}" xpath="/invoice[1]/items"/>

</od:xpath>

<od:xpath id="x3">

<od:dataBinding storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" xpath="/invoice[1]/items/item[1]/name"/>

</od:xpath>

</od:xpaths>

Because a repeat tag refers to an ID in this part, the entry in the tag can be kept short (which is important since Word 2007 has a 74 character limit on the tag length).

If the document is setup for interactive processing, each entry in this part will also reference a questionID:

<od:xpath id="x1" questionID="q1">

For this reason, content controls which are just standard binds also reference the XPaths part via their tag.

# od:xpath tag

As explained above, content controls which are just standard binds reference the XPaths part via their tag. This is so that in interactive mode, their question can also be found by looking in the XPaths part.

So a plain vanilla bind looks like:

<w:sdtPr>

<w:dataBinding w:storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" w:xpath="/invoice[1]/customer[1]/name[1]"/>

<w:tag w:val="od:xpath=x1 "/>

</w:sdtPr>

with the XPaths part containing the same information:

<od:xpath id="x1">

<od:dataBinding storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" xpath="/invoice[1]/customer[1]/name[1]"/>

</od:xpath>

(It is only really useful if there is a @questionID )

# Questions Part

If the template is set up for interactive processing, an additional Custom XML Part specifies the questions which are used to elicit values from the user for insertion into the answer file Custom XML part.

<questionnaire xmlns="http://opendope.org/questions">

<questions>

<question id="q1">

<text>Customer name?</text>

<response>

<free>

<format>text</format>

</free>

</response>

</question>

<question id="q2">

<text>How many items?</text>

<response>

<free>

<format>integer</format>

</free>

</response>

</question>

</questions>

</questionnaire>

# od:condition tag

Content controls can surround various things, including table rows and table cells.

The content control is included if its condition evaluates to true, and excluded if it is false. In the case where:

* a content control surrounding a table row evaluates to false, the row is deleted.
* a content control surrounding a table cell evaluates to false, the cell is retained, but its contents are deleted.

If you look at invoice.docx, you'll see it contains an sdt with:

<w:sdtPr>

<w:tag w:val="od:condition=c5"/>

</w:sdtPr>

A condition tag is generally attached to a rich text content control.

This refers to the conditions CustomXML part:

<od:conditions xmlns:od="http://opendope.org/conditions" >

<od:condition id="c5">

<od:xpathref id="x5"/>

</od:condition>

<od:condition id="c6">

<od:xpathref id="x6"/>

</od:condition>

</od:conditions>

In this example, the condition simply uses the value of

<od:xpath id="x5"/>

but boolean expressions are also permitted.

# Document Setup

The free OpenDoPE Word Add-In is recommended for setting up data bindings as per these conventions. You can download it from http://dev.plutext.org/opendope/setup.exe

Please note you'll need to first install the full .NET 4.0 framework (the Client profile is not enough).

# od:repeat tag

invoice.docx contains the following example:

<w:sdt>

<w:sdtPr>

<w:alias w:val="Repeat"/>

<w:tag w:val="od:repeat=x2"/>

</w:sdtPr>

<w:sdtContent>

<w:tr>

<w:sdt>

<w:sdtPr>

<w:dataBinding w:storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" w:xpath="/invoice[1]/items/item[1]/name"/>

<w:alias w:val="Description"/>

<w:tag w:val="Description=Description&amp;od:xpath=x3"/>

<w:text/>

</w:sdtPr>

<w:sdtContent>

<w:tc>

<w:p>

<w:r>

<w:t>apples</w:t>

</w:r>

</w:p>

</w:tc>

</w:sdtContent>

</w:sdt>

<w:sdt>

<w:sdtPr>

<w:dataBinding w:storeItemID="{8B049945-9DFE-4726-9DE9-CF5691E53858}" w:xpath="/invoice[1]/items/item[1]/price"/>

<w:alias w:val="Price"/>

<w:tag w:val="price=price&amp;od:xpath=x4"/>

</w:sdtPr>

<w:sdtContent>

<w:tc>

<w:p>

<w:r>

<w:t>$20</w:t>

</w:r>

</w:p>

</w:tc>

</w:sdtContent>

</w:sdt>

</w:tr>

</w:sdtContent>

</w:sdt>

A repeat tag is generally attached to a rich text content control.

od:repeat refers to XPath x2, which is:

<od:xpath id="x2">

<od:dataBinding storeItemID="{8b049945-9dfe-4726-9de9-cf5691e53858}" xpath="/invoice[1]/items/**item**"/>

</od:xpath>

For non-interactive processing, the table row will be duplicated, once for each /invoice[1]/items/item.

For interactive processing, the number of /invoice[1]/items/item elements is determined interactively by the user. Processing then proceeds as per non-interactive processing. A document should only have one od:xpath with a given od:dataBinding/@xpath for repeat purposes.

When the repeat is being processed, any w:dataBinding on any child sdt will need to be altered to point at the nth item.

If two content controls have the same repeat tag, each should affect the document as if the other isn't there.

Various things can be repeated, including table rows and cells.

# od:component tag

An sdt which is to be replaced by the insertion of another docx will have:

<w:sdtPr>

<w:tag w:val="od:component=comp1"/>

</w:sdtPr>

The URL of the docx to be inserted is specified in the components Custom XML Part:

<components xmlns="http://opendope.org/components">

<component id="comp1" iri="http://www.foo.com/component-subdoc.docx"/>

</components>

When a component docx is included in the host docx, it is likely that Word will render a page break before and after. The reason for this is that the default type of section break is "next page". There are 2 ways to overcome this:

1. Use an explicit section break type before the component and/or in the last sectPr in the component (although you can't do the latter in the Word interface!)
2. Use od:continuousBefore and/or od:continuousAfter in the w:tag, for example  
     
    <w:tag w:val="od:component=comp1&amp;od:continuousBefore=true&amp;od:continuousAfter=true"/>

When a component docx is included in the host docx, its XPath, Condition, Question and Component parts (if any) are merged into the host's parts. Any entry with a duplicate ID is discarded.

# Implementation

This v2.2 of this convention is implemented in docx4j v2.6.0.

Source code can be found at <http://dev.plutext.org/svn/docx4j/trunk/docx4j/src/main/java/org/docx4j/model/datastorage/OpenDoPEHandler.java>

# Namespace

There is an XSD for each of the following:

the conditions part,

the xpaths part,

the components part,

the questions part.

The namespaces can be seen in the example XML above.

Jason Harrop  
Plutext

document version v2.2  
16 November 2010

This version supersedes the previous version 1 (which was implemented in docx4j v2.5.0). Compared to version 1, there are three main improvements:

1. content in the sdt tag is minimised, which is necessary since Word restricts the tag content to 64 characters

2. supports conditions made up of boolean expressions

3. supports interactive processing (ie user gets asked questions); v1 only supported non-interactive processing

4. supports component inclusion

version 2.1 stipulates that a repeat xpath points to the element to be repeated, not to its parent container