Rebuilding define.pdf by reading Word and XFDF files

David Garbutt

BIOP, Basel, Switzerland

Abstract

Although the industry is transitioning to the CDISC define.xml to document SAS transport files submitted to the FDA there are still many studies to be submitted that were prepared with the old methods and for which documentation has already been created. Continuing with the old method is not viable because FDA now expects the define.pdf content to include derivation information for variables. This information is typically stored in the Statistical Analysis Plan. This presentation shows how SAS XML features were used to add this vital functionality to an existing tool building define.pdf. We use SAS XML maps to read CRF page location information for each variable from an annotated CRF (saved as XFDF). We also use an XML map to extract variable/dataset metadata from MS Word tables contained in pre-existing .doc files. The advantages of using LaTeX to generate the define.pdf are also briefly discussed.

*Todo list

1. Introduction

In this paper I aim to illustrate how reading XML works in SAS by sharing a useful example. We have heard much about XML used for CDISC SDTM and other data but how many of us have actually written SAS programs to read XML? What would be your reaction to being told you needed to pull out from a set of legacy Word files the meta data tables to reuse? And how about reading a PDF file to extract for each variable a list of pages it occurs on? Both of these problems end with the need to read an XML file. There are at least three XML formats available for Word files (WordML, ODT and docx) and comment information in PDF files can also be exported to an XML format called XFDF. In this talk we look at new SAS 9.2

facilities that help with these tasks.

- Using the XML engine with an XMLmap file

Other topics to be covered are

- exporting readable XML from Word
- exporting PDF comments in XFDF format
- code using PROC TRANSPOSE to recreate the metadata table
- creating LaTeX from SAS.

This example is based around a system used for creating the define.pdf file and supplied with data submitted to the FDA in support of drug submissions and illustrates how this system was upgraded to meet the latest specifications from the FDA.

2. Define pdf contents and definition

2.1. The FDA example

The define.pdf contains Data Definition Tables (DDT) that document the XPT files being sent as part of a submission. These specifications have been revised to specify the content needed for CDISC submitted studies. The CDISC specifications include as a text field the derivation method used to create all derived variables used in Analysis datasets. There are also specifications for non-CDISC studies in Appendix 2 in the FDA documentation, FDA (2010). Figure 1 shows the example given in that document of a Demography dataset. Notice that the column 'codes' includes actual format values not just the format name and in the column 'comments' we find both page references to the blank CRF[1] and information labelling a variable as *derived* and giving the formula used for calculating it.

This derivation information was not included in older guidance and has spread from the CDISC version to the version of legacy non-CDISC studies. This implies that submissions made now should include this information where possible.

2.2. The new information needed

The original metadata requested could all be extracted from the XPT files being submitted. The new data definition table has information in it that will normally come from different sources. These new sources are shown in Figure 2 and marked with a cross.

There are four distinct sources of information that need to be linked. The unique key is Dataset + variable.

Variable, label, & type can be extracted from the XPT files themselves. Format name is also available here. This is the best source because this documentation then provides a check on the correctness of the XPT file preparation because it can be compared with the specification.

Code list can be found in the format XPT file and is uniquely indexed by

[1] This is a confusing name because what is meant is not a clean CRF but a CRF annotated to show what variable name is used for each field. The file as sent to the FDA must be annotated *and* called blankCRF.pdf.

[2] Unless there is more than one format dataset, in which case format names could be duplicated and then search order of the format datasets must also be known.

as the CRF field name if different from the variable name in the dataset. Providing a hypertext link from each raw data variable in the data definition table to the appropriate location of the blankcrf.pdf also helps the review process. An example of part of a data definition table for the demographics dataset for study 1234 is provided below.

Study 1234 – Demographics Dataset Variables

Variable	Label	Type	Codes	Comments[1]
USUBJID	Unique patient ID number	char		Demographics page 3
SEX	Sex of subject	char	f = female m = male	Demographics page 3
BDATE	Birth date	date		Demographics page 3
DUR	Duration of Treatment	num		Derived STOP DATE – START DATE
TRT	Assigned treatment group	num	0= placebo 5= 5mg/day	

[1]Use footnotes for longer comments

Version 1.5 1

Figure 1: FDA define.pdf example showing a data definition table

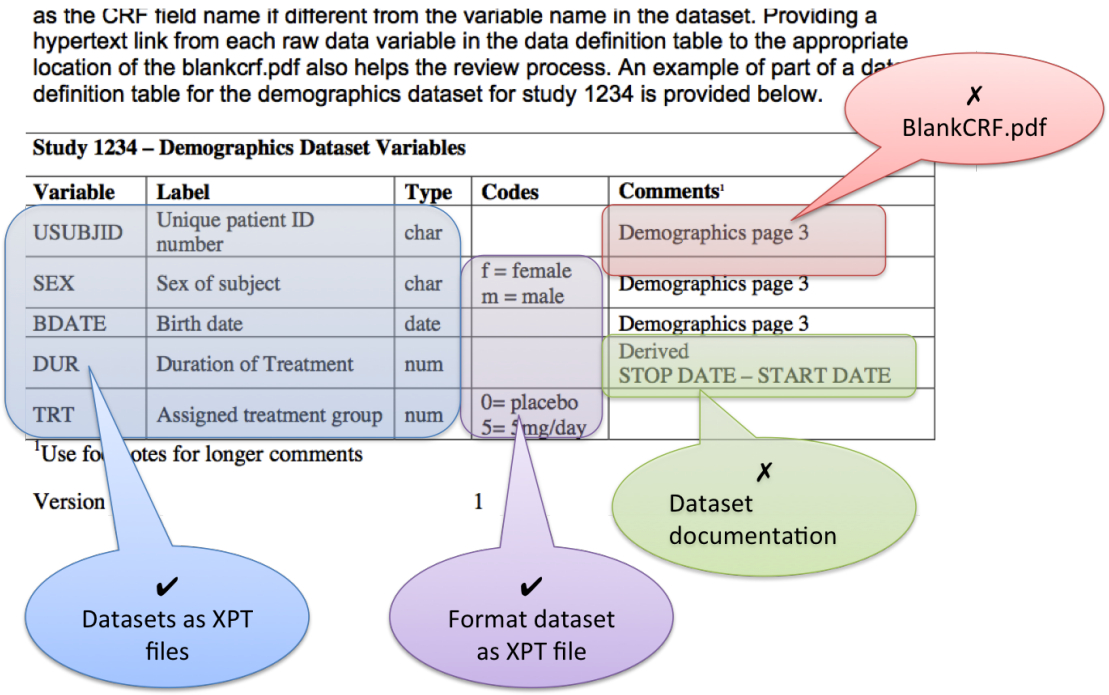


Figure 2: Where do the fields in DDT come from?

format name[2] only.

CRF Page references could, in principle[3] be extracted from an annotated CRF, database definition or metadata from a data entry and/or management system. They can also be entered manually.

Derived variable source expression could in principle be extracted from the Statistical Analysis Plan (where the derived datasets are defined), or from the programs themselves.

In an ideal situation these fields would come from CDISC metadata and be available to a program making a DDD. But in the real world and certainly for many legacy trials this will not be the case. The life of a trial can be quite long — two years from conception to final report is not uncommon — and this means that although CDISC will gradually become the native data format for internal work, submissions for the next few years (5?) will contain studies done with in-house data models as well. Consequently, a way to provide the new format of DDD is needed that can include (and merge in the data step sense) the page references and the derivation definitions. We will need:

1. Tabular information from the dataset documentation (our example will be WORD, but other formats will do). In this documentation a table exists with a row indexed by dataset and variable containing a column holding the free text defining the variable
   $\Longrightarrow$ we need to read WORD files containing the meta data!
2. Information on which variables from which dataset occur on each page of the blankCRF.pdf which we assume is annotated with Acrobat
   $\Longrightarrow$ we need to read the blankCRF!

We will see how to read the information these two types of information in the sections below.

3. Example create define pdf with LaTeX

The process used for creating define.xml is shown in Figure 3 and sample LaTeX code created for the FDA example table is in Figure 4. The output is shown in Figure 5.

We will use LaTeX for creating the PDF – for various reasons:

- It creates a good looking document[4]
- Cross platform
- Free
- Tried and tested[5]
- No performance issue handling even very large documents
- Flexible with many optional layout packages available
- The markup is all text based and easy to write with SAS data _null_ and it is less verbose than XML. Many text editors support it with colour coding.

[3] meaning 'the information is there' with no judgement implied about how difficult that might be.

[4] there are various technical reasons for this and one is that the line breaking algorithm optimises the whole paragraph and not single lines as in Word Processing program like MS Word. For further details see http://en.wikipedia.org/wiki/Word_wrap and Knuth & Plass (1981).

[5] the first release was in 1978, (http://en.wikipedia.org/wiki/TeX) SAS was on its third release.

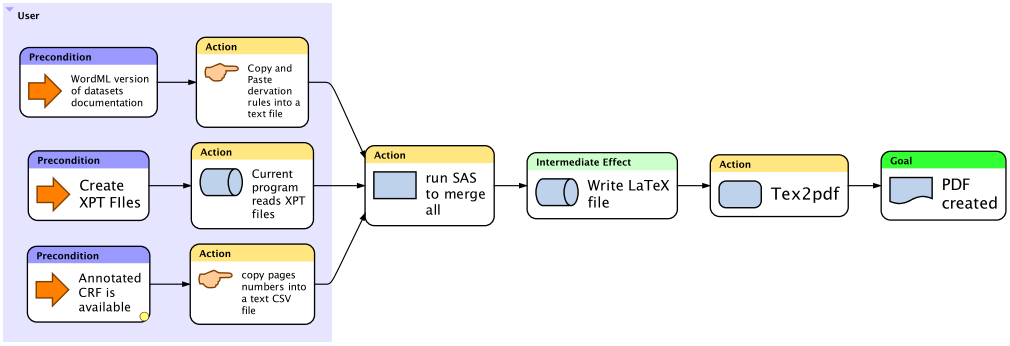


Figure 3: Original process to create define.pdf with suggestion for adding variable derivation

- Very easy to write table cells with multiple lines with line wrapping and hyphenation. This is ideal for a table like the one we wish to make because the text can be output from one SAS character variable and any extra line ends in the LaTeX file are removed. Paragraph markers can be inserted with markup (e.g. \linebreak[4]). The column separator (&) is an absolute one.

- PDF links to locations within the document and links to external documents are easily inserted and c an include page numbers for PDF documents. Indices can also be generated very simply as can tables of content, lists of tables, abbreviations etc.

- LaTeX is also supported from ODS markup tagsets so in other applications SAS output can be integrated. This would be easier if we were running on a Windows server but on Unix that functionality is not supported by SAS nor provided by Microsoft. A related reason is we are running the program using ClearCase® to make an audited build of the define.pdf so we run that as a batch process using clearmake. LaTeX is a command line program[6] so it integrates very easily with SAS.

- Because an external intermediate text file is created that file can be saved and debugged separately if PDF creation fails.

- Good technical help is available on the internet via various fora and documentation.

- This particular application had been running successfully since 2003 and the change needed was not one that justified a redesign.

There other approaches to creating PDF in particular by using the ODS PDF des-

[6] there are various front-ends available free and otherwise. This document has been created with one of the free tools called LyX. http://www.lyx.org 'a WYSIWYM' LaTeX editor.

```
% preamble omitted for clarity
\begin{document}
\section{Demog dataset}
This set holds the demography details. The content is copied directly
from the FDA example document in Appendix 2.
\href{...long... URL}{UCM199759}
{\sffamily  \begin{center}
\begin {longtable}{|>{\textsc}l %  col 1
|>{\small \RaggedRight}l%          col 2
|l%                                col 3
|>{\small \RaggedRight} p{2.3cm}%  col 4
|>{\small \RaggedRight}p{3.6cm}|}% col 5
 \hline
\multicolumn{5}{|l|}
{Study 1234 ---  Demographics Dataset Variables}
\\ \hline
Variable&Label&Type&Codes&Comments [1] \\ \hline
\endhead
\hline
\multicolumn{5}{l}{[1]Use footnotes for longer comments}
\endfoot
usubjid&Unique subject ID number&char&
&Demographics \href{1234blankCRF.pdf#page.3}{Page 3}\\
\hline
sex&Sex of subject&char&f~=~female m~=~male&
Demographics page 3\\ \hline
bdate&Birth date&date&&Demographics page 3\\
\hline
\textsc{dur}&Duration of Treatment&num&
&Derived STOP~DATE --
START~DATE\\ \hline
trt&Assigned treatment group&num&0~=~placebo 5~=~5mg/day
& \\ \hline

\end{longtable}
\end{center}
}
Back to normal text.
\section{LAB dataset}
\end{document}
```

Figure 4: LaTeX source code for the DDT in Figure 5.

1 Demog dataset

This set holds the demography details. The content is copied directly from the FDA example document in Appendix 2.
UCM199759

Study 1234 — Demographics Dataset Variables				
Variable	Label	Type	Codes	Comments [1]
usubjid	Unique subject ID number	char		Demographics Page 3
sex	Sex of subject	char	f = female m = male	Demographics page 3
bdate	Birth date	date		Demographics page 3
DUR	Duration of Treatment	num		Derived STOP DATE – START DATE
trt	Assigned treatment group	num	0 = placebo 5 = 5mg/day	

[1]Use footnotes for longer comments

Back to normal text.

2 LAB dataset

Figure 5: PDF page created by LATEX code in Figure 4

tination Jansen (2006). This same paper notes that external PDF links generated with SAS 9.1 only point to page 0 and 'this is not expected to be fixed in V9.2'. There is still a reason to keep using the LaTeX / SAS system from 2003.

4. XML Mapper and XML maps

Using a SAS XMLmap effectively allows the XML libname engine to be under program control as it creates dataset(s) from the XML file. This sounds trivial but is actually very powerful because these programs automatically extract the content inside the tag or attributes and silently *throw the tags away*. To write a program with a data step to read almost any data structure is possible, but it is not necessarily easy, or simple. An XML map is a simple way to get information out of what is essentially a text file with markup — it does the job that other XML technology (XSLT) can do but in a more concise way and with the advantage that the SAS XML engine does the transformation so no third party tools are needed[7].

An XMLmap file is a set of instructions that act like triggers. In a SAS data step program an equivalent example is the use of `first.by-variable` to make certain actions at the beginning of the by group. There you write code that is just executed when that condition is fulfilled. In the XML map case the actions are limited to copying content into a column. A big advantage of this method is that it is easy to extract types of objects out of a document — tables for example. No code is needed to handle parts of the file that do not match the path given.

XMLmaps can be created with the interactive SAS XML Mapper application or directly with a text editor. They are then saved and named in a SAS libname statement and used to read the XML file. An XML map can create more than one dataset from a single XML file, and these datasets can share variables.

The SAS XML Mapper is a stand-alone Java program that allows you to read in an XML document and then interactively drag structure elements to a table and build a mapping of the documents's elements into SAS datasets. It does not handle large documents well and you will need to make a shorter version to work with.

4.1. Working with large XML files

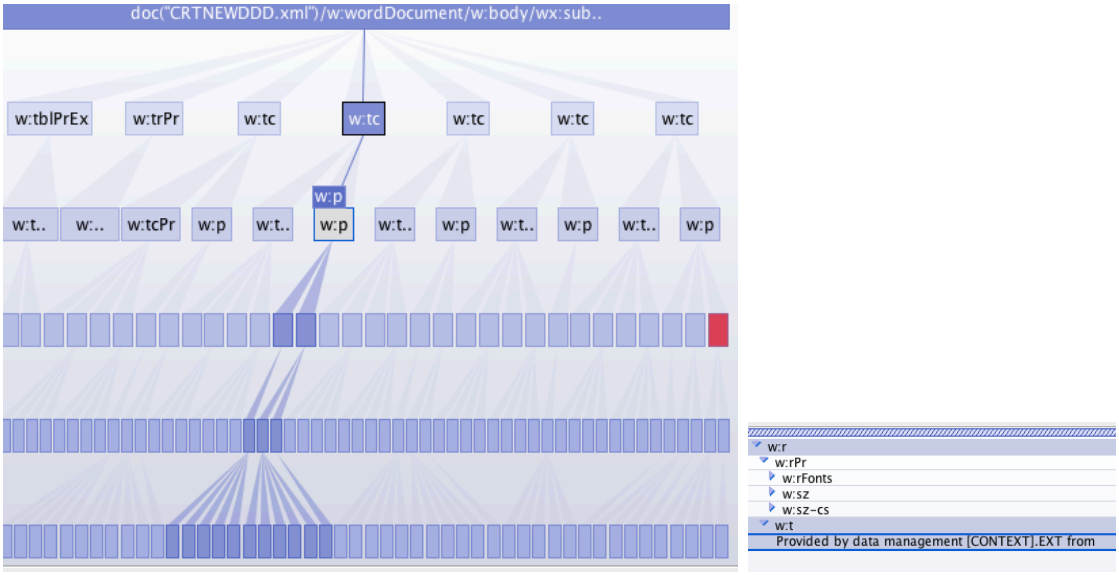
An XML version of a Word document although it is a text file may be 2-3 MB for even a short document (20 pages) this means that tools are needed that can read the XML file and its structure for viewing and possibly editing[8].

XML is defined such that whitespace is ignored and so an XML file may or may not contain Line-ends. This creates problems using a normal text editor unless it has code tidying implemented. The size of XML documents can also be an issue in many text editors and cause slow-downs or even crashes. XML is also often defined using the UTF-8 encoding. If this is the case your installation must be set-up to use UTF[9]. It is also pretty well vital to

[7] Only a subset of the XSLT functionality is supported though so for some tasks you will need XSLT

[8] Files this big give problems in the SAS XML Mapper tool. Reduce the file sizes for testing and map building

[9] set options -DBCS and -encoding UTF-8




```
<w:r xmlns:aml='http://schemas.microsoft.com/aml/2001/core'
   xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
   xmlns:ve='http://schemas.openxmlformats.org/markup-compatibility/2006'
   xmlns:o='urn:schemas-microsoft-com:office:office'
   xmlns:v='urn:schemas-microsoft-com:vml'
   xmlns:w10='urn:schemas-microsoft-com:office:word'
   xmlns:w='http://schemas.microsoft.com/office/word/2003/wordml'
   xmlns:wx='http://schemas.microsoft.com/office/word/2003/auxHint'
   xmlns:wsp='http://schemas.microsoft.com/office/word/2003/wordml/sp2'
   xmlns:sl='http://schemas.microsoft.com/schemaLibrary/2003/core'>
  <w:rPr>
    <w:rFonts w:fareast='Times New Roman'/>
    <w:sz w:val='16'/>
    <w:sz-cs w:val='16'/>
  </w:rPr>
  <w:t>Provided by data management [CONTEXT].EXT from</w:t>
</w:r>
```

Path in full: doc("CRTNEWDDD.xml")/w:wordDocument/w:body/wx:sub-section/wx:sub-section/wx:sub-section/w:tbl/w:tr/w:tc/w:p/w:r

Figure 6: A WordML file with part of a text paragraph highlighted with all its nested components including the XML selected

have an editor that can detect or at least show different character encodings. Even better if can convert files beween encodings.

XML tools

Some useful tools are well known such as XML Spy[10] and there are many others. The ones I would recommend are Serna and BaseX[11] which both have free versions capable enough for this task. Serna is more aligned towards document editing but its optional tree-structure display and searching make it easy to find particular text and understand the document structure. BaseX is also good with structure but takes a different approach by treating the XML file as a database. It provides structure display using the more compact tree-map. It also supports queries using the XPath syntax[12]. The XPath syntax[13] applies the idea and terminology of a file path to the tag hierarchy in the XML document. For example the relative path `//w:tr/w:tc` is matched by all elements within a table-cell within a table-row.

It is used by SAS for defining the triggers used when reading the data and debugging programs doing this means exploring the documents structure. Using BaseX you can find the text of interest and then directly obtain the XPath you need to find that element. Figure 6 shows an example of a document paragraph with the document structure as displayed in BaseX. There is style and other formatting information stored and the actual text is in a `w:t` element, and itself is contained within `w:p` (paragraph) and `w:r` (revision) elements.

4.2. Word XML formats

MS Word has supported XML formats since the 2003 version and there are at least three XML formats it can use. Since Word 2010 docx has become the default format. The 2003 format (standardised and often called WordML) is supported on most versions and platforms and since it is slightly simpler than docx and odt[14] we elected to use it for this project. It is created simply by using save-as from the file menu and selecting '2003 xml format'. An excellent discussion of the Word XML is in Hoyle (2006).

4.2.1. Word example document

Let us suppose we have the data from Study 1234 shown in Figure 1 saved in a word table. Figure 6 gives an idea how it might look saved as a WordML document. In this case we need the Variable column, the XPT file name in the first table row to be the key for merging with the XPT data.

4.2.2. Getting the table back in SAS

We can read the XML file using a libname to define the XML file's location and assign a XMLmap[15] .

```
filename   content
```

[10] http://www.altova.com/xml-editor/

[11] http://www.syntext.com/products/serna-free/ and http://basex.org/

[12] http://www.w3schools.com/xpath/xpath_syntax.asp

[13] Strictly speaking a subset of it

[14] These formats are actually zipped files. If you rename the file to .zip you can see the content.xml containing the document. Using these formats add steps to the program with no benefit

[15] For ODT, with more verbose tags, the XPath expression for a paragraph within a table cell is `//table:table/table:table-header-rows/table:table-row/table:table-cell/text:p`

Col1	Col2	Col3	Col4	Col5
Study 1234 — Demographics Dataset Variables				
Variable	Label	Type	Code	Comments[1]
sex	Sex of subject	char	f = Female m = male	Demographics Page 3
bdate	Birth date	date		Demographics page 3
DUR	Duration of treatment	num		Derived STOP DATE - START DATE
Trt	Assigned treatment group	Num	0 = placebo 5 = 5mg/day	

Table 1: FDA Sample data as table

```
    'Y:\CRT-phuse\demo\content.xml'
     encoding=UTF8 ;

filename  SXLEMAP
    'Y:\CRT-phuse\demo\DDD-ODT.map';

libname content xml92
        xmlmap=SXLEMAP
        access=READONLY ;

data DDDTables ;
   set content.DDDTables ;
run;
```

```
<COLUMN name='Text'>
  <PATH syntax='XPath'>
   w:tc/w:p/w:r/w:t
  </PATH>
  <TYPE>character</TYPE>
  <DATATYPE>string</DATATYPE>
  <LENGTH>32000</LENGTH>
</COLUMN>
```

This gives us a SAS dataset looking like Table 2a on the following page. Which has extracted only the table content and lost the table structure. There is no easy way to get it back because there will be more than one table in the document and possibly some tables that are not data definition tables. We could only recover the sructure by making strong assumptions about the document. What has gone wrong?

As usual in computing nothing has gone wrong. Our XMLmap has done exactly what we asked — but the expression in the XMLmap: is true for *every* table cell with text in it, so all are extracted and are on the same level in one column. What we need here is awareness of order — in the tables we wish to extract there are always five columns and so we could write an expression for each column filling it with the firtst, second, third... table cell. XSLT does support such a syntax but it is not supported by XML maps or the SAS XML engine. If it were a data step we would add sorting variables and use `first.tablerow` and `last.tablerow` to count and add output statements to output just one observation per table row. It turns out there is an equivalent in a SAS XMLmap.

We can create ordinal variables that count events (i.e. reading tags), we can retain the values and initialise them when we wish. In this case to count table cells we increment with each cell start (`w:tc`), retain and initialise at the end of each row (`w:tr`).

1. Declare cell num as a counter – integer

_N_	Text
1	Study 1234 — Demographics Dataset Variables
2	Variable
3	Label
4	Type
5	Code
6	Comments[1]
7	sex
8	Sex of subject
9	char
10	f = Female m = male
11	Demographics Page 3
12	bdate
13	date
14	
15	Demographics page 3

(a) Result of reading the data in Table 1

_N_	cell#	Text
1	1	Study 1234 — Demographics Dataset Variables
2	1	Variable
3	2	Label
4	3	Type
5	4	Code
6	5	Comments[1]
7	1	sex
8	2	Sex of subject
9	3	char
10	4	f = Female m = male
11	5	Demographics Page 3
12	1	bdate
13	2	date
14	3	
15	4	Demographics page 3

(b) Result after adding cell counter

Table 2: Results of reading the WordML of FDA sample

2. Add one every time you hit a table cell start tag `w:tc`

3. Reset the counter at the beginning of the next table row `w:tr`

This gives a XMLmap code like this:

```
<COLUMN name='CellNum' ordinal='YES'
    retain='YES'>
  <INCREMENT-PATH beginend='Begin'
      syntax='XPath'>   w:tc
    </INCREMENT-PATH>
  <RESET-PATH beginend='END' syntax='
      XPath'>
       w:tr
    </RESET-PATH>
  <TYPE>numeric</TYPE>
  <DATATYPE>integer</DATATYPE>
</COLUMN>
```

Adding this to the XMLmap and rereading the data gives us a new table 2b on the current page

Which is a success.

For the final version we also added counters for tables, rows, and paragraphs within cells which allows the original table structure can be recaptured using several PROC TRANSPOSE steps. We preserved paragraphs within cells by concatenating paragraphs and adding para-separator characters that were later converted to LaTeX markup (`\linebreak[4]`).

The full SAS XMLmap is in the Appendix on page 17.

Extensions

Other types of MS Office documents are also available in XML form and so these

techniques can also be applied to these other files also.

4.3. Reading the blankcrf.pdf

The annotated CRF has comments added (sometimes by hand) as in Figure 7a that define which variable each field on the CRF will become. This information is needed to be able to create the links to CRF pages needed in the FDA sample document. There are many different conventions concerning how the comments should be added and although these conventions were the ones we worked with I suspect many companies use similar rules to ensure consistency in interpretation (see Wilkins & Campbell (2011) for ways to regularise these annotations).

1. Dataset names were marked by the keyword `panel:` and followed by the panel name, normally in the left margin.

2. Variables were below the Panel comments and placed as near as possible to the field in question and to the right of the Panel comment.

3. If a second panel was needed on a page then it would be located below the first and above its first field.

The panel could also be included in each variable field but this makes a lot more work for the annotator.

Adobe provides tools for extracting and inserting comments the file format is called FDF and more recently an XML version (XFDF) was added. With Acrobat and Reader you can export the comments as an XFDF file.

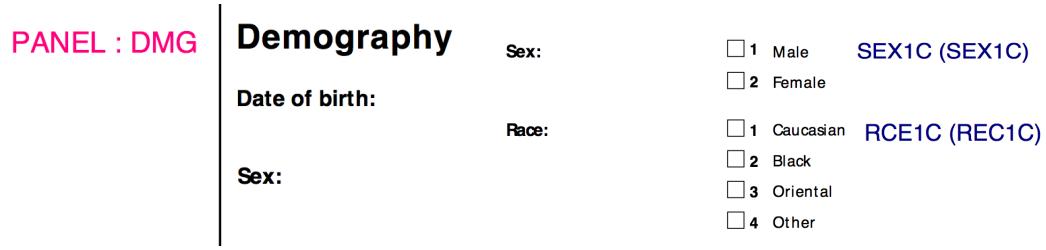
4.3.1. What is XFDF?

XFDF is an XML format and the comment for DMG is in file like this:

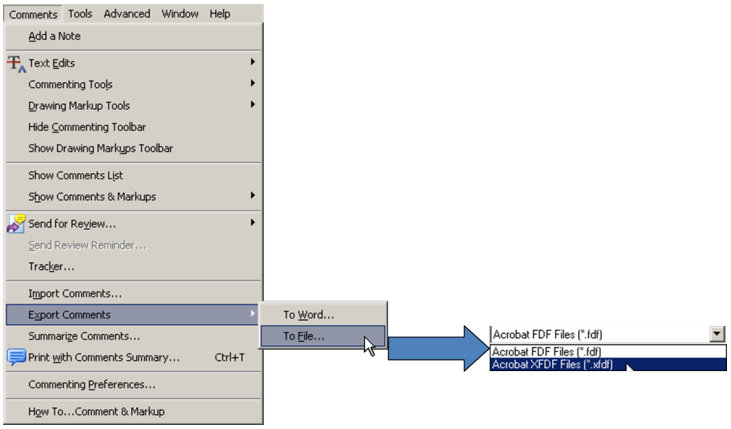
```
<freetext
   rect= '9.259186, 612.989014,
      93.826401, 630.27301'
   name='fdf2286f-111e-4cgh-b656-2
      fb7dd14dae1'
   width='0'
   flags='print'
   date="D:20110711111721+01'00'"
   title='BlankCRF'
   open='no'
   page='3'>
    <contents>PANEL : DMG</contents>
    <defaultappearance>[1 0 0.502] r
       /Helv 12 Tf
    </defaultappearance>
 </freetext>
```

The attribute `rect` holds the coordinates of the comment box and the `<contents>` tag encloses the text. The attribute `page` has the page number and is created by Acrobat no user input is needed to obtain this. Here is the piece that reads the page number. The XPath has an `@page` to extract the attribute `page` from the tags. Notice we can also get the content into the same SAS observation even though it is nested in a different tag. The full listing is in the Appendix on page 19.

```
<COLUMN name='page'>
   <PATH syntax='XPath'>
      /xfdf/annots/freetext/@page
   </PATH>
   <TYPE>numeric</TYPE>
   <DATATYPE>integer</DATATYPE>
</COLUMN>
<COLUMN name='contents'>
   <PATH syntax='XPath'>
      /xfdf/annots/freetext/contents
   </PATH>
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>200</LENGTH>
</COLUMN>
```



(a) Comments on a field (PANEL: DMG) in pink



(b) How to export comments on a PDF as an XFDF file

4.3.2. Getting the locations and pages in SAS

Having read the XFDF file we need to associate each comment with a Panel and we can do this using the rules mentioned above. The program is listed in full in Appendix subsection A.2 on page 19 and separates the comments with PANEL from the others and puts the panel list in a separate dataset.[16] This dataset suffices if you only need a page map by dataset. If you need also a page map per variable[17] you need to merge the panel data with the variable information. This is possible because panel and variables have page number to use as a key. So the program sorts by the page coordinates and then merges back the panel information carrying forward the panel name to each variable. This page information can then be used when writing the LaTeX file to create a link to the correct CRF pages.

An example output CSV file using vertical bar (|) as a delimiter is on the next page

Pooled data

If standards were followed for naming all the blankCRF.pdf files it would not need much extension to adapt this method further for reading multiple XFDF files if present and adding the study name as a key for comprehensive pooling of studies.

[16] This has to be done because the XFDF file always has the marginal panel comments out put first before the other texts

[17] In the FDA example there is actually a case where the page for one variable in a dataset differs from the overall page list for that dataset.

```
panel|Page1|Page2|Page3|Page4|Page5|Page6|Page7|Page8|Page9
AEV|37|||||||
ANQ|36|||||||
BALAN|80|||||||
BCLP|39|40|41|90|91|92|93|94|
CONCMED|6|38||||||
CONMP|31|32|33|34|||||
CNDRUG|4|||||||
COMMENT|30|64|65|66|90|91|92|93|94
CRFMETA|1|||||||
CRISIS|5|||||||
RECVER|18|19|73||||||
DARIG|35|||||||
DJG|10|11|12|13|||||
```

Figure 7: Page map output as CSV file

5. Example define pdf with data from XML

Once we have the page lists and the formats and comments we can add these to the DDD as shown in Figure 5 on page 7. Of course there is still a lot of work to do merging, matching and error checking, but we have the information needed without needing any re-entry or copy and paste. The new process is shown in Figure 8 on the next page. For a much improved output the user only has to save two files both created from pre-existing documentation.

6. Overview and looking forward

The paper from Larry Hoyle about reading Word XML dates from 2006 which is five years ago. This technique is not new but deserves to be better known. It has major benefits to the programmer and therefore to users. As we transition to CDISC submissions we still have a large number of old studies for the foreseeable future to be documented and submitted and this paper shows a way that we could proceed to bring these older studies up to the standard of the new ones. It has not been stressed but the DDD document we extracted the comments from was actually prepared as a Word document by hand and contained all the tables created by hand and mouse. The possibility we have now to read Word tables and create beautiful PDF documents means we could actually redesign the whole information flow and automate much of the DDD document production too by basing it on tables extracted from the XPT files. Another possibility is to take tables of metadata from the Statistical Analysis Plan and use them to build the SAS datasets (Rittmann (2010)) .

LaTeX perhaps still has a role here because it also likes to run on a server in batch like SAS this makes it an easier tool to integrate with than Word. It makes various workflows possible and could

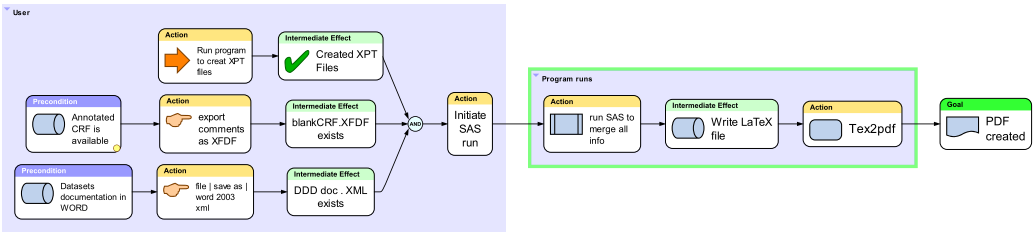


Figure 8: The revised process only uses two extra files to be extracted and no extra manual input

minimise the human, and error prone, input of large amounts of descriptive information.

Further Reading

The paper by Larry Hoyle below has three examples of using XML with SAS and is a must-read. His other work with XML is also worth following up.

More details about XML formats and Word can be found with more details on the format at http://www.forensicswiki.org/wiki/Word_Document_%28DOCX%29/ and http://www.xmlw.ie/aboutxml/wordml.htm. There have been quite a lot of work with XFDF exemplified by Dirk Spruck and Monica Kawohl's paper http://www.lexjansen.com/pharmasug/2004/coderscorner/cc02.pdf on using FDF to create the annotations on an annotated blankCRF.pdf There is also work creating define.pdf from Define.xml Jansen (2006) , and creating SAS programs from define.xml but I have found no work on managing legacy and CDISC data together and little on acquiring data from Word documents.

References

Hoyle, L, 2006, *Reading Microsoft Word XML files with SAS®*, SUGI proceedings http://www2.sas.com/proceedings/sugi31/019-31.pdf 10

Jansen, L, 2008, *Using the SAS XML Mapper and ODS to create a PDF representration of the define.xml*, http://www.lexjansen.com/phuse/2008/cd/cd04.pdf 8, 16

FDA, 2010, *Study Data Specifications*, http://www.fda.gov/downloads/Drugs/DevelopmentApprovalProcess/FormsSubmissionRequirements/ElectronicSubmissions/UCM199759.pdf At version 1.5.1 when this work was done. Currently at version 1.6 (2011-06-22), Appendix 2 is unchamged. 2

Knuth, D. E. and Plass, M. F. (1981), *Breaking paragraphs into lines.* Software: Practice and Experience, 11: 1119–1184. doi: 10.1002/spe.4380111102 4

Rittmann, M, 2010, *Automating the Link between Metadata and Analysis Datasets* , http://www.lexjansen.com/pharmasug/2010/ad/ad16.pdf 15

Wilkins, R and Campbell, J, *A Regular Language: The Annotated Case Report Form*, http://www.lexjansen.com/pharmasug/2011/cd/pharmasug-2011-cd18.pdf 13

Contact information

I would value your comments and questions on this paper.
Please contact me at:

David J. Garbutt, Principal Consultant, BIOP AG,
Centralbahnstrasse 9,
CH – 4051 Basel
Switzerland
Work Phone: +41 61 227451
Fax: +41 44 390 3722
david.garbutt@biop.ch
LinkedIn profile

Acknowledgements

I would like to thank my family for their tolerance, support and ability to play. I would also like to thank my colleague Jean-Yves Robo for problem solving and being there as we worked on this challenging project. My colleague at BIOP Rohit Banga deserves a special mention for vital information on the SAS unicode set-up at a vital moment.

A. Appendix

A.1. A complete XML map for ODT format XML

```
<?xml version='1.0' encoding='UTF-8'?>
<SXLEMAP name='Tables' version='1.9'>

<!-- ####### ODT version ################ -->
<TABLE description='Tables extracted from DDD doc in ODT format'
```

```
   name='DDDTables'>
<TABLE-PATH syntax='XPath'>//table:table-cell/text:p
</TABLE-PATH>
<TABLE-END-PATH beginend='END'
   syntax='XPath'>//table:table/table:table-row/
</TABLE-END-PATH>

<COLUMN class='ORDINAL' name='TableNumber' retain='YES'>
 <DESCRIPTION>table:table</DESCRIPTION>
   <INCREMENT-PATH beginend='BEGIN'  syntax='XPath'>
      //table:table
   </INCREMENT-PATH>
   <TYPE>numeric</TYPE>
   <DATATYPE>integer</DATATYPE>
</COLUMN>

<COLUMN class='ORDINAL' name='RowNumber' retain='YES'>
   <DESCRIPTION>
      table:table-row
   </DESCRIPTION>
   <INCREMENT-PATH beginend='BEGIN'
      syntax='XPath'>//table:table-row
   </INCREMENT-PATH>
   <RESET-PATH beginend='END'
      syntax='XPath'>//table:table
   </RESET-PATH>
   <TYPE>numeric</TYPE>
   <DATATYPE>integer</DATATYPE>
</COLUMN>

<COLUMN class='ORDINAL' name='CellNum' retain='YES'>
   <DESCRIPTION>Count cells within each table row
   </DESCRIPTION>
   <INCREMENT-PATH beginend='BEGIN' syntax='XPath'>
      //table:table-cell
    </INCREMENT-PATH>
    <RESET-PATH beginend='END' syntax='XPath'>
      //table:table-row
    </RESET-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
</COLUMN>

<COLUMN class='ORDINAL' name='ParaNum' retain='YES'>
   <DESCRIPTION>Counts Multiple paragraphs within a Table cell
   </DESCRIPTION>
   <INCREMENT-PATH beginend='BEGIN' syntax='XPath'>
      //table:table-cell/text:p
   </INCREMENT-PATH>
   <RESET-PATH beginend='END' syntax='XPath'>
      //table:table-cell
   </RESET-PATH>
   <TYPE>numeric</TYPE>
   <DATATYPE>integer</DATATYPE>
```

```
</COLUMN>

<COLUMN name='Text'>
    <PATH syntax='XPath'>//table:table-cell/text:p</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>32000</LENGTH>
</COLUMN>
</TABLE>
```

A.2. The SAS XMLmap for XFDF

```
<?xml version='1.0' encoding='windows-1252'?>

<!-- ############## XFDF version ################ -->
<SXLEMAP name='AdobeXFDFmap' version='1.2'>
<TABLE name='freetext'>
<TABLE-PATH syntax='XPath'>
   /xfdf/annots/freetext
</TABLE-PATH>
<COLUMN name='rect'>
  <PATH syntax='XPath'>
      /xfdf/annots/freetext/@rect
   </PATH>
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>60</LENGTH>
</COLUMN>

<COLUMN name='title'>
  <PATH syntax='XPath'>
      /xfdf/annots/freetext/@title
   </PATH>
   <TYPE>character</TYPE>
   <DATATYPE>string</DATATYPE>
   <LENGTH>13</LENGTH>
</COLUMN>

<COLUMN name='contents'>
    <PATH syntax='XPath'>
      /xfdf/annots/freetext/contents
    </PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>60</LENGTH>
</COLUMN>

<COLUMN name='date'>
    <PATH syntax='XPath'>
      /xfdf/annots/freetext/@date
    </PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>23</LENGTH>
```

```
    <FORMAT width='10'>IS8601DA</FORMAT>
    <INFORMAT width='10'>IS8601DA</INFORMAT>
</COLUMN>

<COLUMN name='page'>
    <PATH syntax='XPath'>
      /xfdf/annots/freetext/@page
    </PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
</COLUMN>
</TABLE>
</SXLEMAP>
```

A.2.1. SAS program calling the XFDF SAS XMLmap

This is a test program that reads the XFDF and creates CSV files for checking. Format information is also extracted from the comments fields. CSV file page maps are created per variable and per dataset.

```
/*************************************************
 *  Dave Garbutt, david.garbutt@biop.ch
*   Program to generate file with variable listing from
*   XML map and xfdf data file
*   created by Acrobat 7
*   first draft 20 aug 2010
 *************************************************/

/*
 *  Environment - subst standard variables to get path here

 */
filename  blcrf
    'C:\Data\My Dropbox\Projects\CRT-Tools\107-CRTblankcrf.xfdf';
filename  SXLEMAP 'C:\Data\My Dropbox\Projects\CRT-Tools\107-map.map';
libname   blcrf xml xmlmap=SXLEMAP access=READONLY;


data cmtv(Label='Variables and panel items - unsorted')
    cmtp(drop=varname contents
    (Label='Panel list with one record per page occurrence')
     ;
    length panel $20 varname $ 20 format $ 10 ;

    set blcrf.freetext;
    *--- get info out of the XML data file and parse variables
    to what we want ---;

    tmpstr = compress(upcase(contents), '()');
    *--- remove brackets around format name ---;
    page = page + 1 ;
```

```
    *--- First page in PDF file is numbered zero --;
    *-- get coords on page of the current comment box
    origin is the conventional one (bot left) and coords increase
    from L to R and bot to top
     vars named after the boxes coords for H(eight) and W(idth)---;

    Wstart=input(scanq(rect,1,','), 10.6) ;
    Hstart= input(scanq(rect,2,','), 10.6) ;
    Wend=input(scanq(rect,3,','), 10.6) ;
    Hend=input(scanq(rect,4,','), 10.6) ;
    height = max(Hstart,Hend);
    across = max(Wstart,Wend);

 *--- split panel and variable data to separate output files --;
if index(contents, 'PANEL') then do;
   panel = scan(tmpstr, 2, ':') ;  *  second word ;
   *-- carrying over the value of panel till it changes does not
        work when more than one panel per page
      because all panel comments are taken in xfdf before
      the variable comments
      this is done next after the sort step--;
   output cmtp ;
   end;
else if contents not in( 'REVISED CRF', 'NOT IN DATABASE')
        and index(contents, 'PANEL') = 0
     then do;
     varname = scanq(tmpstr, 1, ' ') ;
     format =  scanq(tmpstr, 2, ' ');
     if format = '=' then
        format = '' ;
     else
        contents = ' ';
     *- fi ;

        output cmtv;
   end;
drop title tmpstr rect date ;
run;

proc sort data=cmtv(drop= Hstart Hend Wstart Wend)
    out=cmtvs (Label='variables sorted to page order') ;
   by page descending Height  across   ;
   run;

     *--- now the rows are in order matching the CRF page
            from top to bottom & left to right.
            However the rows are sometimes also displaced
            up and down so the match may not be exact.
            Rounding might help.
```

```
      ;
data cmtvs2 (Label='variables sorted to page order') ;
   length panel2 $ 20 ;
   set cmtvs ;
   by page notsorted panel ;

   retain panel2 ;
   if panel ne '' then panel2 = panel ;
   if varname ne ' ' then output ;
   rename panel2=panel ;
   drop panel ;
run;

proc export data=cmtvs2 replace
   outfile='C:\Data\My Dropbox\Projects\CRT-Tools\blcrfv_all.csv'
   dbms=dlm;

     delimiter='|';
run;
*--- we have a record per comment in the CRF and second ds with
     one record per panel they come sorted by PAGE out of the xfdf file
     Now we want FIRST a list of variables with pages numbers
     per variable across ---;
proc sort data=cmtv NODUPKEY ;
   by varname format page ;
   run;
proc transpose
    data = cmtvs (drop= height across panel)
    out=rawdatav(drop=_name_ _label_
        Label='transposed variable to page map')
    prefix=Page;
   by varname  format ;
   var page ;
run;
*-- add code to get the data set from the variable name we
    already have and maybe check vs panel?
   AND then collapse so page list is per Panel/dataset not per variable
   ---;
proc export data=rawdatav replace
   outfile='C:\Data\My Dropbox\Projects\CRT-Tools\blcrfvariable.csv'
   dbms=dlm;
   delimiter='|';
run;
/*--- we have a record per comment in the CRF
     Now we want SECOND a list of pages numbers where each
    panel occurs PANEL (== dataset) across ---;*/

proc sort data=cmtp      nodupkey ;
   by  panel page ;
```

```
   run;
proc transpose
    data = cmtp
    out=rawdatap(drop=_name_ _label_
     Label='List of pages where each panel appears. One rec per panel')
    prefix=Page
           ;
   by panel  ;
   var page ;
   run;
*-- add code to get the data set from the variable name we already have --;
   *-- maybe check vs panel?
   AND then collapse so page list is per Panel/dataset not per variable
   ---;
proc export data=rawdatap replace
   outfile='C:\Data\My Dropbox\Projects\CRT-Tools\blcrfpanel.csv'
   dbms=dlm;
   delimiter='|';
run;
```