PharmaSUG 2011 - Paper CC22

Importing and Parsing Comments From a PDF Document With Help From Perl Regular Expressions

Joel Campbell, PPD, Inc., Wilmington, NC
Ryan Wilkins, PPD, Inc., Wilmington, NC

ABSTRACT

This paper presents a case study of importing and parsing annotations contained in a blank aCRF PDF document for use in creation of the CDISC required define document. The first section details how the PRXMATCH function was used to overcome a critical loss of data that occurred during the import phase which required the use of the SAS® XML Mapper and XML Libname Engine. The second section presents an introduction to the Perl regular expression functions that can be used to parse a standardized set of annotations that are ultimately used to provide the CDISC required hyperlinks from metadata definitions to the CRF page where the data originate. This paper assumes that the reader has basic knowledge of the Perl regular expression syntax used in SAS and does not cover the technical aspects of using the SAS XML Mapper or XML Libname Engine.

INTRODUCTION

Comments contained in a PDF file may be imported into SAS by first exporting the comments to an XML Forms Data Formatted (XFDF) file and then importing the XFDF file in a SAS data step via the XML LIBNAME Engine. An XML Map is required when using the XML LIBNAME Engine, which can be generated via the SAS XML Mapper utility[1]. Adobe Acrobat Professional® version 6.0.6 was used to export comments to XFDF file format in the example presented. Once imported, Perl regular expressions can be leveraged to parse standardized annotations in order to populate the links and content of a CDISC required metadata define document. A standard CDISC annotation construct has been described[2] and was used in the example described in this paper.

Functions and call routines using regular expressions are powerful tools for processing text that are available in SAS version 8 and later, and their utility evolved greatly with the release of the Perl regular expressions in SAS versions 9.1 and later. For example, beginning with version 9.1, the PRXMATCH function may be used interchangably with the tradtional INDEX function as a stand alone statement where more overhead was required in earlier versions. Also, the regular expression syntax used in the PRX functions is more closely related to the standard Perl syntax and may be considered more intuitive to programmers already familiar with the Perl regular expression language.

This paper aims to give teams a starting point for importing comments from a PDF into SAS and to make teams aware of a potential loss of data that can occur during the process. Also, basic examples of PRX functions that are useful for parsing such comments are provided.

OVERCOMING CRITICAL DATA LOSS WHEN IMPORTING BLANKCRF.XFDF

Using Adobe Acrobat Professional, annotations contained as comments in a PDF file may be exported into an XFDF file via Document > Export Comments. The exported XFDF may then be imported into sas via the XML LIBNAME Engine with a required XML Map. Creation of the XML Map is hardly a trivial process, but in our experience with the example presented here, regardless of the XML map used, the PDF-exported XFDF file was structured in such a way that data was inevitably lost during the import into SAS. Figure 1 contains a screenshot of a standard annotation which could not be reconstructed correctly when imported into SAS.

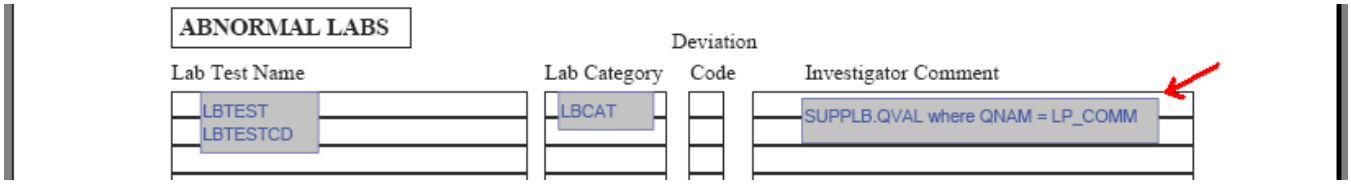

Figure 1. Standard annotation contained in blankcrf.pdf. Highlighted comment could not be imported into SAS correctly without restructuring the XFDF file created by Adobe Acrobat Professional version 6.0.6.

As shown in Figure 2, and for reasons unknown to the author, the "=" in the problem comment appears at a different hierarchical level than the rest of the text. In this case, the "=" appears at the "span" level, while the rest of the text appears at the "p" level. This is the root of the data loss, as it appears that the imported data cannot be read from both sides of the "=" correctly.

```
><p
>SUPPLB.QVAL where QNAM <span style="font-weight:0;font-style:italic"
>= </span
>LP_COMM </p
></body
```

Figure 2. Resulting blankcrf.xfdf entry for the comment highlighted in Figure 1 shows that the "=" portion of the comment is contained at the tag level while the remainder of the comment is split into two parts at the <p> tag level. The portion of the comment at the <p> tag level that appears after the close of the tag level is not captured during import.

```
<p>
        SUPPLB.QVAL where QNAM
                <span style="font-weight:0;font-style:italic">
                        =
                </span>
        LP_COMM
</p>
```

Figure 2b. Contents of Figure 2 modified and formatted for clarity.

Using a number of XML Maps generated by the SAS XML Mapper for importing, the part of the comment at the <p> tag level that appears after the close of the tag level is not captured during import. The resulting imported comment is shown below.

```
SUPPLB.QVAL where QNAM =
```

RESTRUCTURING THE XFDF FILE

While we tried a number of variations of the XML map to overcome this data loss, none were able to import the problem comment correctly. Furthemore, getting an updated blankcrf.pdf with revised comment was not an option, so our approach was to pre-process the blankcrf.xfdf file to remove the level altogether. This required removing the and tags along with any text contained within. This could have been accomplished in SAS using tradtional functions like index() and substr(), but using the PRXMATCH function allowed us to find both tagsets with a single statement.

The null data step below reads in the blankcrf.xfdf file, removes the and tags, and creates blankcrf_parsed_.xfdf. The resulting entry in the parsed XFDF for the problem comment is shown in Figure 3.

```
data _null_;
    length line _line $32767.;
    infile "blankcrf.xfdf" lrecl = 32767 length = lg;
    input line $varying32767. lg;
        retain trigger _line keeppos;

        if trigger then do;
            line=substrn(_line,1,keeppos)||substr(line,2);
            trigger=0;
        end;
        START=PRXMATCH('/<\/?span[^>]*$/i',line);
        if START then do;
            _line=substrn(line,1,start-1);
            keeppos=start-1; ***keeppos logic guarantees that spacing is maintained;
            trigger=1;
        end;
        else if compress(line) ne '' then do;
            FILE "blankcrf_parsed_.xfdf" lrecl=32767;
            put line;
        end;
run;
```

```
><p
>SUPPLB.QVAL where QNAM = LP_COMM </p
></body
```

Figure 3. Resulting entry in blankcrf_parsed_.xfdf for the same comment shown in Figure 2.

Since the level of tag has been removed from all comments as shown in Figure 3, a modified XML Map must be used and is included in the appendix for reference. The code used to import the parsed XFDF file is shown below. Note that the dataset name, annots, in the _xfdf library is defined in the XML map.

```
libname _xfdf xml 'blankcrf_parsed_.xfdf'
        xmlmap='XFDF.MAP'
        access=readonly;

data _annots (drop = rawPage);
    set _xfdf.annots;
    page = rawpage + 1;
run;
```

In this way, the critial loss of data is alleviated and we can move on to parsing the imported comments.

PARSING IMPORTED COMMENTS WITH PERL REGULAR EXPRESSIONS

SAS programmers who have ever tried to parse dynamic data to match patterns using traditional functions like INDEX and SUBSTR know that creating these statements can become confusing and labourious even when trying to match patterns with low relative complexity. While complex Perl regular expressions can also be confusing even for those well experienced with them, the flexibility and power they provide easily make up for the effort required to implement them.

In the application presented in this paper, the ultimate goal is to provide a dataset that can link metadata definitions to the CRF page(s) where the data originates. In the case of the highlighted comment in Figure 1, which happens to be found on page 25 of the aCRF, the goal may be to populate the following values as shown:

```
Original comment: SUPPLB.QVAL where QNAM = LP_COMM
aCRF Page: 25
Domain: SUPPLB
Variable: QNAM
Value Level Annotation (VarVL): LP_COMM
```

Since the page number and original comment are already provided by the imported data, the task at hand is to parse the comment to identify the domain, variable, and where applicable, the VarVL referenced by the annotation. A standaradized annotation construct has been used so we can categorize all comments of this form as "*DOMAIN*.X where *VAR* = *VARVL*" using the PRXPARSE, PRXMATCH, and PRXPOSN functions in the datastep below.

```
data parsed(drop=RE:);
 set _annots;
    length DOMAIN VAR VARVL $8.;
    if _n_=1 then do;
        RE1=
        prxparse('/^ *([a-z_]+) *. *[a-z_]+ +where +([a-z_]+) *= *([a-z_]+) *$/i');
        retain RE:;
    end;
    if prxmatch(RE1,strip(textfragment)) then do;
        DOMAIN=prxposn(RE1,1,strip(textfragment));
        VAR   =prxposn(RE1,2,strip(textfragment));
        VARVL =prxposn(RE1,3,strip(textfragment));
        put textfragment "---> " domain= var= varvl=;
    end;
run;
```

Resulting log entry:

```
SUPPLB.QVAL where QNAM = LP_COMM ---> DOMAIN=SUPPLB VAR=QNAM VARVL=LP_COMM
```

Here's a brief explanation of the regular expression used in the example above… the "^" and "$" characters match the beginning and end of a string, respectively, so RE1 may be considered to match the entire string. In other words, if a string contains the target text plus any other text, the regular expression will not be matched. Contrast this to the INDEX function that simply looks for the occurrence *anywhere* in a string. The "i" appearing after the closing "/" of the requlat expression means that the expression will be insensitive to case. The "[a-z_]" defines a list of allowed characters; here we're looking for any letter or an underscore. The "+" will match 1 or more occurrences of the preceeding character, and the "*" allows for 0 or more occurrences; here we'll match the string whether or not the reviewer entered spaces around the equal sign. Finally, some matched text will be "grouped" for use with PRXPOSN with the "(" and ")" characters – reading from left to right, these are numbered 1, 2, and 3. Note that matched groups can be nested, which may be useful in some situations.

ADDITIONAL NOTES

In the example provided, note that the strip function is used whenever the TextFragment is supplied to a PRX function. This is important because it has a length of 32767 characters and parsing all of this length with a Perl regular expression can result in an extreme amount of processing time as opposed to the processing time when the strip function is used.

When parsing manually entered comments, it may be helpful to output a comprehensive review file as misinterpretations caused by typos or omissions are virtually inevitable.

While some of the PRX functions require that the regular expression be defined earlier in a PRXPARSE function call, many others allow the user to supply the regular expression in the function itself (see the PRXMATCH function call in the data _null_ example on page 2). This is a change from the RXPARSE family of functions available in version 8.

CONCLUSION

With the examples provided here, a programmer should be able to import annotations contained as comments in a PDF document, be mindful of and able to correct a potential loss of data, and parse the imported annotations for further use using selected PRX functions available in SAS version 9.1 and later.

REFERENCES

[1] Download location for the latest version of the XML Mapper, http://support.sas.com/kb/33/584.html

[2] Wilkins RB, Campbell JK, A Regular Language: The Annotated Case Report Form, PharmaSUG 2011

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Joel Campbell
Ryan Wilkins
PPD, Inc
929 North Front St.
Wilmington, NC 28401
joel.campbell@ppdi.com
ryan.wilkins@ppdi.com

APPENDIX – XFDF.MAP

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

<!-- ############################################################ -->
<!-- 2010-07-12T05:36:23 -->
<!-- SAS XML Libname Engine Map -->
<!-- Initially Generated by XML Mapper, 9.1.0300.20040709.2028 -->
<!-- ############################################################ -->
```

```
<!-- ###  Validation report                                    ### -->
<!-- ############################################################ -->
<!-- Map validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="xFdf" version="1.2"
xsi:noNamespaceSchemaLocation="http://www.sas.com/xml/schema/sxle12.xsd">

    <!-- ############################################################ -->
    <TABLE name="Annots">
        <TABLE-DESCRIPTION>Annotation location and text fragments.</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/xfdf/annots/freetext/contents-richtext/body/p</TABLE-PATH>

        <COLUMN name="rawPage" retain="YES">
            <PATH syntax="XPath">/xfdf/annots/freetext/@page</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
            <LABEL>Raw page number beginning with zero.</LABEL>
        </COLUMN>

        <COLUMN name="rect" retain="YES">
            <PATH syntax="XPath">/xfdf/annots/freetext/@rect</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>64</LENGTH>
            <LABEL>Left, bottom, right, top coords. Origin at lower-left corner of pg.</LABEL>
        </COLUMN>

        <COLUMN name="textFragment">
            <PATH syntax="XPath">/xfdf/annots/freetext/contents-richtext/body/p</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>32767</LENGTH>
            <LABEL>Text fragments in the annotation.</LABEL>
        </COLUMN>

    </TABLE>

</SXLEMAP>
```