Paper CC-09

Using SAS® to Report Data in XML Format

Qin Wang, PhD
District of Columbia Courts, Washington, D.C.

ABSTRACT

XML (Extensible Markup Language) has become increasingly important as a required format for data reporting because of its textual nature and Unicode support. However, converting data into XML format is quite challenging for many data analysts. We are familiar with data in rectangular row and column format created via programs such as Excel, SQL, or SAS®. Converting a big sum of data from the row and column structure to a hierarchical or tree structure which XML accepts can be difficult conceptually and technologically to us. In addition, XML reporting often comes with very strict format requirements for data elements.

This paper introduces methods to use SAS software (SAS®8.2 or SAS ®9.1 on Windows@ platform) to perform the following tasks.

1. Use put command to write an output data file in XML format;
2. Use WHERE statement and VERIFY, SUBSTR, RXPARSE, RXCHANGE functions to scan data elements and make necessary corrections.

INTRODUCTION

The targeted readers of this paper are those who have the mid-level SAS programming skills, but no IT background (do not know JavaScript). The data amount you are required to report should be large and complicated enough to make manual input unrealistic. The input data source can be various such as Oracle database, Excel or any format accepted by SAS via Import Procedure. The output file contains data in XML format and is ready to submit.

TRANSFORMING DATA FROM RECTANGULAR DATA STRUCTURE TO XML FORMAT

There are three major requirements associated with XML format

- Requirement 1. Output data in hierarchical structure with pre-determined tag names and tag structure.

- Requirement 2. Pre-determined blocking tags should be inserted into the output document. For example, every record should start with “Award” and ended with “Award”. The information about current year award such as phase, dollar amount will be nested between a tag call “Solicitation” at the beginning and the end of this group of information.
- Requirement 3. If one variable in one record has missing information, that variable name and value for that particular variable of that record should be excluded (in a relational database, we keep the variable name, but put a missing value or blank in that field).

Most data analysts are familiar with data in a rectangular structure like this:

Agency Tracking Number	Program	Phase Number	Agency	Year	Number
CM000028	SBIR	1	HHS	2007	PHS2007-1
CM000036	SBIR	1	HHS	2007	PHS2007-1

However, in a hierarchical data structure like MXL, these two records will be written like this:

```
- <Award>
  <AgencyTrackingNumber>CM000028</AgencyTrackingNumber>
  <Program>SBIR</Program>
- <Phase action="insert">
  <phaseNumber>1</phaseNumber>
  <Agency>HHS</Agency>
- <Solicitation>
  <Year>2007</Year>
  <Number>PHS2007-1</Number>
  </Solicitation>
   </Award>
- <Award>
  <AgencyTrackingNumber>CM000036</AgencyTrackingNumber>
  <Program>SBIR</Program>
- <Phase action="insert">
  <phaseNumber>1</phaseNumber>
                  <Agency>HHS</Agency>
- <Solicitation>
  <Year>2007</Year>
  <Number>PHS2007-1</Number>
      </Solicitation>
   </Award>
```

While many software programs are able to save files in XML format (e.g. Excel) and there are many software programs created just to handle XML conversion (e.g. XML editor), they are not able to meet reporting requirement easily. For example, if we save the above 2 records to XML format via Excel, we will get something like that showing below. Variable names are replaced by data types.

```
<Row>
  <Cell ss:StyleID="s34"><Data ss:Type="String">CM000028</Data></Cell>
  <Cell ss:StyleID="s34"><Data ss:Type="String">SBIR</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="Number">1</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="String">HHS</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="Number">2007</Data></Cell>
  <Cell ss:StyleID="s34"><Data ss:Type="String">PHS2007-1</Data></Cell>
 </Row>
 <Row>
  <Cell ss:StyleID="s34"><Data ss:Type="String">CM000036</Data></Cell>
  <Cell ss:StyleID="s34"><Data ss:Type="String">SBIR</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="Number">1</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="String">HHS</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="Number">2007</Data></Cell>
  <Cell ss:StyleID="s34"><Data ss:Type="String">PHS2007-1</Data></Cell>
 </Row>
```

After studying several alternatives (SQL, EXCEL and XML Editor) and weighing cost consideration, I decided to use SAS for conversion. The SAS “put” command has the power to produce a perfect XML output file. SAS also allows me to query data; combine data from different resources; and manipulate data all in one program. The entire procedure is therefore automated.

- By pointing to first and last record (N position), the required text can be easily inserted at the beginning and end of the file. For example, at the beginning of the file (N=1), I added text such as <?xml version="1.0"?> and '<SBA_TECHNet_Transfer version="1.0" >. At the end of the file (N=LST), I also added text '</SBA_TECHNet_Transfer>' to close the tag.
- The “BY” statement after “SET” brings in records from the input file one by one and writes one by one to output file (phase2notfound).
- In order to exclude variables with missing value, I used an “if then” statement before each variable.

```
Filename outxml "I:\Annual_ Reports\SBIR\phase2notfound.xml";

data _null_;
 file outxml; /*XML Output File */
 set  one NOBS=Lst; /* SAS File with input data   */
 by AgencyTrackingNumber;
   %let tab="    ";
 if _n_=1 then do;
   put '<?xml version="1.0"?>';
         put '<SBA_TECHNet_Transfer version="1.0" >';
         end;
   put &tab'  <Award action="update" awardNumber="'awardNumber'">' ;
   if AgencyTrackingNumber ne ' ' then
   put &tab' <AgencyTrackingNumber>'AgencyTrackingNumber'</AgencyTrackingNumber>';
   if program ne ' '     then put &tab'     <Program>'program'</Program>';
   ... . . .

                  if _n_=lst then do;
          put '</SBA_TECHNet_Transfer>';
          end;
run;
```

CORRECTING DATA TO COMPLY WITH DATA FIELD FORMAT REQUIREMENTS

The data in XML format needs to be uploaded via recipient's website. This process enables the party who requested data to enforce very strict rules on data quality and automatically prevent any invalid data submission. Mainly, there are two data issues which we need to solve in order to pass validation.

- Issue 1. Data is not standard. For example, in the email field, some entries has no "@" sign. Some put two email addresses, therefore with two "@" sign. Some people put extension on their telephone number and, therefore telephone field has more than 10 digits.

- Issue 2. Some mandatory data fields contain embedded characters like carriage returns, line feeds, tabs, page breaks. Those embedded characters need to be removed. For example, the abstract of a grant application was saved as a word document and was scanned and stored into an Oracle database. Those characters are accepted by Oracle as CLOB data type. However, those special characters cannot be accepted by XML because XML, by definition, accepts textual contents only. We have to remove them in order to submit data.

    a) In order to solve the first issue, I used "Where" statement to identify and set missing value for the wrong fields. I also used "length" and "substr" statement to treat irregular telephone numbers.

```
Data Wrongemail;
   Set a;
  Where email not contains ('@') ;
Run;
Data Rightemail;
   Set a;
  Where email contains ('@') ;
Run;
Data B;
  /* records in wrongemail will missing value at email field  */
   Set wrongemail(drop=email)  Rightemail;
Run;
```

b) It is much more difficult to solve issue 2. Firstly, I used Rxparse function to find invalid characters and replace them with acceptable values. Rxparse scans a field, finds restricted signs and replaces them with words and give the new field a new variable name. In the following example, we replaced "&" with 'and' and saved it as a new variable name (abstract2); replaced "<" with 'lt' and saved as abstract3. We also replaced ">" with 'gt' and saved it as abstract.

```
length abstract2 $12000.;
rx=rxparse("'&' to 'and'");
call rxchange(rx,10000, abstract_text, abstract2);

length abstract3 $12000.;
rx=rxparse("'<' to 'lt'");
call rxchange(rx,10000, abstract2, abstract3);

length abstract $12000.;
rx=rxparse("'>' to 'gt'");
call rxchange(rx,10000, abstract3, abstract);

Drop abstract_test abstract2 abstract3;
```

c) Using the VERIFY function to exclude embedded characters. Verify (character-value, verify string) will check if a string contains any unwanted values.

VERIFY function is opposite to COMPRESS function. When the COMPRESS function compresses the characters we want to exclude, VERIFY function verifies and passes the characters we want to keep (" ABCDEFGHIJKLMNOPQRSTUVWXYZ,.-/+%;:1234567890()[]?*&'~><_=#").

Test2 returns a numerical value to indicate the position of unwanted characters (embedded characters) when it is greater than 0.

SUBSTR(left of =) function finishes up the job by finding the position indicated by Test2 and replacing the embedded characters with a blank.

```
data abstract_b;
 set abstract(keep=appl_id abstract_text);
         do until (test2=0);
         test2=verify(abstract_text,
           " ABCDEFGHIJKLMNOPQRSTUVWXYZ,.-
/+%;:1234567890()[]?*&'~><_=#");

    if test2>0 then do;
      substr(abstract_text,test2,1)=' ';
      end;
    end
run;
```

CONCLUSION

Most papers about using SAS to do XML conversion are quite complicated to non-IT people. They also do not address the data issues. Content and format, although sounding like different issues, have to be solved together to complete the task. After much experimentation of other methods, I concluded that PUT command is a simple way to perform XML conversion. Furthermore, the functions introduced in this paper are effective ways to clean up the data. My final SAS program combined these two steps together and is automated to the degree that junior level analysts can understand and use it for routine data submission.

ACKNOWLEDGEMENT

Thanks to NIH/OD, my former employer, for challenging me with this XML conversion task. Thanks also to DC Courts, my current employer, for the time and budget to prepare this presentation. My special thanks and appreciation go to Terry McCormack for, among all other things, editing my final draft as a non-SAS user.

DISCLAIMER

The views expressed in this paper are solely those of the author and do not represent the views of the District of Columbia Courts and National Institute of Health.

AUTHOR CONTACT INFORMATION

Qin Wang, PhD
Statistical Associate
District of Columbia Courts
500 Indiana Avenue, Northwest
Washington, DC 20001
(202) 879-2862

Qin.Wang@DCSC.GOV

TRADEMARKS