Paper 179-27

<XML> at SAS® - A More Capable XML Libname Engine

Anthony Friebel, SAS Institute, Inc., Cary, NC

ABSTRACT

At SAS, we use XML in several different ways. XML finds application as a persistence method in Vertical Applications, a query protocol in Warehouse Administrator software, an output form in SAS ODS software, and a data source in base SAS software. This paper will focus primarily on XML as a data source and the SAS XML Libname Engine, which can be used to convert XML directly to SAS data sets.

INTRODUCTION

Last year about this time, my boss, Paul Kent, stood on this platform at SUGI and his message went something like this

"… XML might just be the 'ASCII text file' of the new millennium. A universal and accepted way to transfer information between computer applications…" (ack 1)

REALITY CHECK

Anyone that has been around sufficiently long in the industry knows the real function of an "ASCII text file" … inter-application glue. They're those gritty little files that carry the day-to-day information which direct the behemoth monthly/quarterly-based legacy systems in the care and feeding of the daily use systems in place throughout your business environment.

The spin-doctors now refer to that as business knowledge. Is this a new idea? Far from it. But the explosion of XML is due primarily to the vocabulary it dictates, and the meaningful IT conversations that take place within that context.

With XML as a transport vehicle, we no longer have to know the next 20 bytes are the customer ID. We specifically mark-up our data file with meaningful tags we decide upon ourselves, such as <Customer-ID> which encloses the valuable data with even more valuable metadata… now, that's progress.

But what does all this XML stuff mean to me as a SAS programmer, manacled to the ergonomic keyboard in my corporate cubicle? Well, XML is for real, it's here to stay, it's got a terrific press agent, and it's becoming more and more prevalent in the day-to-day activities of a corporate code monger. The business *terra compute* has shifted sufficiently under XML's weight, so it is increasingly more likely that some pointy-haired-boss figure is going to approach you about extracting sales projections from that whizzo XML file he just got from the guys in the ice castle next door. What are you going to do?

SXLE

SAS programming XML pioneers are already familiar with SXLE. The abbreviation stands for the "SAS XML Libname Engine", and is pronounced "6-L". But what is it ?

"… An import/export tool for leveraging XML Data into/from SAS Datasets…" (ack 2)

It's a libname data engine, a part of the BASE product, which digests "supported" forms of XML and projects their contents as SAS data sets and variables.

We began work on the SXLE project one year to the day after W3C submittal of the XML specification, and when SXLE was initially released in version 8.0, we presented a very plain (generic), open element XML form, with simple enclosure and data construction.

Coincidentally, Oracle 8i did nearly the same thing. That offering differed only in tag name selection, and indention level.

THE PROBLEM – DATA CHAOS

The scene broadened during the 8.1 release cycle with some additional XML form support having been added, and SXLE was on its way to becoming a transport tool to/from XML. But something else was happening. The XML world was evolving explosively. B2B was actually causing (*gasp*) conversation among IT people and many more XML forms were becoming available. Not all of which were readily consumed by the SXLE version available that week.

Our decision to support those primordial emerging standards was becoming trampled under a stampede of fittest of breed competition fallout. Which forms were we to support? The problem was clear. The answer needed to change. SXLE had to evolve.

Hey! What's the problem? It's all XML isn't it? Yes, and No.

Rectangularization is the nasty business of making clean, neat rows and columns of data from even the ugliest of hierarchically arranged XML representations. Almost none of the real world XML is so neatly arranged, is it? This is especially true if that XML behemoth isn't originating in the DBMS arenas. Yet the nature of most SAS procedure processing and libname engines is rectangular, and rectangular data is what SXLE must construct from its input, even if the original form isn't rectangular.

The process is further complicated by the fact that while XML defines a vocabulary for data conversation, it does not restrict the subject material. Data authors are free to script short stories, novels, trilogies, or encyclopedic epics. Through the 8.2 release point, SXLE was hamstrung with a narrow list of supported forms. If your XML form wasn't a form on the support list, it couldn't be imported. You then had to resort to other methodologies to pre-process or convert your input to something SXLE could digest.

THE SOLUTION – XMLMAP

What is an XMLMap? An XMLMap is an external file, itself XML-formatted, that contains data directives. The directives contain the definition of tables, row repetition boundaries, and column contents existing in your original XML input file. The XMLMap is supplied via libname or data set option to SXLE.

XMLMaps use the XPath methodology - a formal specification exists at www.w3c.org - which puts a UNIX-like path description on each element of the XML input structure. The XPath subset supported by SXLE via the XMLMap (a three specific form subset) allows elements and attributes to be individually selected for inclusion in the generated (rectangular) data set. XMLMaps are the vehicle for rectangularizing wildcat XML input files.

XMLMAP IN ACTION

Here is an example of some XML formatted data. Although fairly simply constructed, and relatively easy for a human to read, it is not a form that SXLE directly supports.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
  <CONFERENCE> Eastern
    <DIVISION> Southeast
      <TEAM name="Thrashers"  abbrev="ATL" />
      <TEAM name="Hurricanes" abbrev="CAR" />
      <TEAM name="Panthers"   abbrev="FLA" />
      <TEAM name="Lightning"  abbrev="TB" />
      <TEAM name="Capitals"   abbrev="WSH" />
   </DIVISION>
 </CONFERENCE>

 <CONFERENCE> Western
   <DIVISION> Pacific
     <TEAM name="Stars"   abbrev="DAL" />
     <TEAM name="Kings"   abbrev="LA" />
     <TEAM name="Ducks"   abbrev="ANA" />
     <TEAM name="Coyotes" abbrev="PHX" />
     <TEAM name="Sharks"  abbrev="SJ" />
   </DIVISION>
  </CONFERENCE>
</NHL>
```

Figure 1. A non-supported XML formatted data fragment

Editor note: Readers should review the contents of tutorial at http://www.sas.com/rnd/base/topics/sxle82/prod82/index.html *for an explanation on the specific constructions SXLE requires*

PROCESS

In order to import this data via SXLE, we need to create an XMLMap. There are a few basic steps related to data content investigation that we should perform before we jump right into the XMLMap syntax construction.

1. Locate and identify distinct tables of information

We want a final data set from this input that contains some of the teams of the National Hockey League. We have only that information contained here, so we can construct a single table called TEAMS in the XMLMap. Other XML files you might encounter may contain more than one table or group of related information. Multiple tables are supported by the XMLMap syntax.

2. Identify the repeating pattern of data elements

In Figure 1, information about individual teams occurs in a <TEAM> tag located within <CONFERENCE> and <DIVISION> enclosures. We want to generate a new observation each time we process a <TEAM> element tag.

3. Collect column definitions for each table

The data content form is mixed in this example. Some data occurs as XML *pcdata* fields (e.g., CONFERENCE) and other data is contained in attribute value pairs (e.g., team NAME). We note the data types are all string values.

Our constructed observation will also include the team NAME, and ABBREV. A length of 30 characters is sufficient for the NAME, and 3 characters is enough for the ABBREV field contents.

4. Add foreign keys or external context required

It might be a good idea to also carry along information about the league orientation for our teams in the final data set. We need to also extract CONFERENCE and DIVISION data values from the input stream.

5. Construct Xpath locators for each column definition

The Xpath identifies a position in the input XML file from which to extract data. This is the only 'tricky' part of the map generation. Element *pcdata* is treated differently than is promotion of attribute values. We have no conditional selection criteria involved.

MAP SYNTAX

With the investigative steps accomplished, we can generate the syntax for the XMLMap. The results of the above analysis are annotated in the figure by step.

```
<?xml version="1.0" ?>
<SXLEMAP version="1.0">
❶ <TABLE name="TEAMS">
      ❷ <TABLE_XPATH>
          /NHL/CONFERENCE/DIVISION/TEAM
        </TABLE_XPATH>

      ❸ <COLUMN name="name">
        ❺ <XPATH>
            /NHL/CONFERENCE/DIVISION/TEAM/@name
            </XPATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>30</LENGTH>
        </COLUMN>

      ❸ <COLUMN name="abbrev">
        ❺ <XPATH>
            /NHL/CONFERENCE/DIVISION/TEAM/@abbrev
            </XPATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>3</LENGTH>
        </COLUMN>


      ❹ <COLUMN name="CONFERENCE" retain="YES">
        ❺ <XPATH>/NHL/CONFERENCE</XPATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

      ❹ <COLUMN name="DIVISION" retain="YES">
        ❺  <XPATH>
             /NHL/CONFERENCE/DIVISION
           </XPATH>
           <TYPE>character</TYPE>
           <DATATYPE>STRING</DATATYPE>
           <LENGTH>10</LENGTH>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

Figure 2. The XMLMap syntax corresponding to Figure 1.

SAS SYNTAX

The SAS syntax required to use the XMLMap is very straight-forward.

```
Filename NHL 'C:\Public\XML\NHL_Teams\NHL.xml';
Filename MAP 'C:\Public\XML\NHL_Teams\NHL.map';
Libname  NHL xml xmlmap=MAP;

Proc print data=NHL.TEAMS noobs; run;
```

Figure 3. The SAS syntax corresponding to Figures 1 and 2.

Executing the code fragment contained in Figure 3 produces output as follows

```
                The SAS System

CONFERENCE     DIVISION      name           abbrev

Eastern        Southeast     Thrashers      ATL
Eastern        Southeast     Hurricanes     CAR
Eastern        Southeast     Panthers       FLA
Eastern        Southeast     Lightning      TB
Eastern        Southeast     Capitals       WSH
Western        Pacific       Stars          DAL
Western        Pacific       Kings          LA
Western        Pacific       Ducks          ANA
Western        Pacific       Coyotes        PHX
Western        Pacific       Sharks         SJ
```

Figure 4. The SAS PROC PRINT output of Figure 3.

Savvy XML coders caught the addition of a RETAIN= attribute value pair on the column definition for our CONFERENCE and DIVISION columns. This programming element is much like the SAS data step programming feature. It forces retention of processed data values after an observation is written to the output data set. Because our *foreign key* fields occur outside the observation boundary (i.e., they are more sparsely populated in the hierarchical XML data than is our observation data) we need to retain their values for additional rows as they are encountered.

ARBITRARY REPETITION EXTRACTION

Here is another example of XML formatted data. This example contains an obvious repetition element, <Publication>, but contains the added twist that within each <Publication> an arbitrary number of <Topic> elements may occur.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Library>
  <Publication>
    <Title>Developer&apos;s Almanac</Title>
<Acquired>12-11-2000</Acquired>
    <Topic Major="Y">JAVA</Topic>
  </Publication>
  <Publication>
    <Title>Inside Visual C++</Title>
<Acquired>06-19-1998</Acquired>
    <Topic Major="Y">C</Topic>
    <Topic>Reference</Topic>
  </Publication>
  <Publication>
    <Title>Core Servlets</Title>
    <Acquired>05-30-2001</Acquired>
    <Topic Major="Y">JAVA</Topic>
    <Topic>Servlets</Topic>
    <Topic>Reference</Topic>
  </Publication>
</Library
```

Figure 5. An XML fragment containing arbitrary repetitions

Our first inclination might be to extract the data with a row boundary on <Publication> since this is the natural recurring pattern in the file. Doing so, however, will produce an unwanted concatenation of data (results not shown) since the <Topic> element can occur more than once within the single publication.

The arbitrary number of topics dictates that <Topic> is the better selection of row boundary for this extraction. Figure 6 shows the XMLMap constructed

```
<?xml version="1.0" ?>
<SXLEMAP version="1.0">
  <TABLE name="Publication">
    <TABLE_XPATH>
    ❶ /Library/Publication/Topic
    </TABLE_XPATH>

    <COLUMN name="Title" retain="YES">
      <XPATH>
      /Library/Publication/Title
      </XPATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>19</LENGTH>
    </COLUMN>

    <COLUMN name="Acquired" retain="YES">
      <XPATH>
      /Library/Publication/Acquired
      </XPATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>FLOAT</DATATYPE>
      <LENGTH>10</LENGTH>
    ❷ <FORMAT width="10" >mmddyy</FORMAT>
      <INFORMAT width="10" >mmddyy</INFORMAT>
    </COLUMN>

    <COLUMN name="Topic">
      <XPATH>
      /Library/Publication/Topic</XPATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>9</LENGTH>
    </COLUMN>

    <COLUMN name="Major">
      <XPATH>
      /Library/Publication/Topic/@Major
      </XPATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>1</LENGTH>
    ❸ <ENUM>
        <VALUE>Y</VALUE>
        <VALUE>N</VALUE>
      </ENUM>
    ❹ <DEFAULT>N</DEFAULT>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

Figure 6. The XMLMap syntax corresponding to Figure 5.

The SAS syntax required is

```
Filename REP 'C:\Public\XML\Repeat\Fig5.xml';
Filename REP 'C:\Public\XML\Repeat\Fig5.map';
Libname  REP xml xmlmap=MAP;

Proc print data=REP.Publication noobs; run;
```

Figure 7. The SAS syntax corresponding to Figures 1 and 2.

Executing the code fragment contained in Figure 7 produces output as follows

```
                The SAS System

Title                Acquired    Topic      Major

Developer's Almanac  12/11/2000  JAVA         Y
Inside Visual C++    06/19/1998  C            Y
Inside Visual C++    06/19/1998  Reference    N
Core Servlets        05/30/2001  JAVA         Y
Core Servlets        05/30/2001  Servlets     N
Core Servlets        05/30/2001  Reference    N
```

Figure 8. The SAS PROC PRINT output of Figure 7.

Figure 8 displays the expected results of our extraction. A few extra items of note are in this map, as annotated in Figure 6.

1. The extraction path for the table is <Topic>. We will generate a new observation each time we encounter a <Topic> element in the input file.

2. The DATE field is constructed using FORMAT/INFORMAT controls of the XMLMap. These controls are useful for situations where data must be converted for use by the system. User written formats and informats are supported, and they may be used independently of each other.

3. Enumerations are also supported by the XMLMap syntax. Here, the values expected of the Major= attribute must be either “Y” or “N”. Incoming data values not contained within the ENUM list is set to MISSING.

4. A regular SAS MISSING value is the default for data which does not occur in the input file processing unless specifically overridden by a non-MISSING default value specified by the <DEFAULT> element of the XMLMap syntax.

CONCLUSION

The promise of XML uniting all IT departments under a common umbrella is still a pipedream. In this presentation we've taken a look at new technology to help you leverage the power of the SAS system and the new XML-based environments, specifically the new enhancements to the XML libname engine which extend functionality beyond the specific supported forms.

We’ve also seen that via SXLE we can now treat almost any XML input file like any other SAS data source. The addition of the XMLMap functionality makes SXLE a more powerful tool by giving us direct access to data contained in many more forms of XML formatted documents.

Examples have shown basic and more advanced XMLMap applications. We’ve drawn up a basic “plan of attack” for generating XMLMaps. And we’ve shown that data repetition and row boundaries must be carefully considered when extracting data with an XMLMap.

ACKNOWLEDGEMENTS

1 Paul Kent, Director, Base SAS Software Research & Development

2 Jim Metcalf, Marketing Manager, SAS Technology Strategy, Worldwide Marketing

REFERENCES

Production:
www.sas.com/rnd/base/topics/sxle82/prod82/libnamexml.html
Tutorial: www.sas.com/rnd/base/topics/sxle82/prod82
XMLMap: www.sas.com/rnd/base/topics/sxle82/exp82/index.html
Resources: www.sas.com/rnd/base/index-xml-resources.html

CONTACT INFORMATION

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

Anthony Friebel
Special Projects Coordinator, SXLE architect
SAS Institute, Inc.
One SAS Circle
Cary, NC 27513
Email: XMLEngine@sas.com
Web: http://www.sas.com

COPYRIGHT INFORMATION