

Paper 220-2012

Advanced XML Processing with SAS® 9.3

Thomas W. Cox, SAS Institute Inc., Cary, NC

ABSTRACT

XML has expanded far beyond the scope originally envisioned by its creators, and this has resulted in the addition of companion standards such as XML Namespaces and XML Schema. This paper describes how SAS has advanced our XML technology in SAS 9.3 to more fully support these standards. It also examines some of the challenges you might encounter when processing complex XML and describes some best practices to help overcome the challenges.

INTRODUCTION

SAS is deeply committed to broad support for enterprise data integration, and has a long history of providing bi-directional access to third-party data formats and storage systems via the SAS/ACCESS® software (SASWWW). XML is no exception. However, the hierarchical nature and enormous flexibility of XML result in some additional challenges for both SAS and its customers.

To better understand those challenges, this paper begins by reviewing the origins and evolution of XML. It then examines some solutions available in the SAS® XML LIBNAME engine and the SAS® XML Mapper, with an emphasis on features introduced in SAS 9.3. The paper further explores situations where there might not be a simple solution, and describes some heuristics and best practices derived from many years of experience with XML.

XML

What Is XML Not?

A tutorial on XML is beyond the scope of this paper, and it is assumed the reader has a basic working knowledge of XML. (If not, there are many good examples in the SAS Global Forum and SUGI proceedings (240-30), (099-2008), (173-28). However, it is useful to explore some of the things that XML is *not* in order to understand what can be expected of it.

XML is **not**:

- **a standard**, strictly speaking. Rather, it is a loose collection of *specifications* from multiple W3C working committees, which might be either recommendations or draft proposals (W3CXML). While there **is** a single XML core specification, reading an XML file often involves four or more of these specifications, for example, XML, XML Path, XML Namespaces, and XML Schema.
- **a product**. There is no single source to maintain and support XML. Conformance to the specifications, and to which revisions of those specifications, is not enforced. Parsers, editors, and other tools are available from a variety of sources spanning the continuum from open source to enterprise level commercial vendors, and adherence to the specifications varies widely.
- **a solution**. XML is a markup language for file formats, and a simple one at that. That flexibility has allowed XML to be used as markup for text documents, configuration files, protocols, object persistence mechanisms, programming languages, and data interchange. Not surprisingly, the properties of XML that provide the flexibility to perform all of these tasks prevent it from being optimal for any of them.

Collectively, these constraints have created a paradoxical growth pattern in which the freedom allowed by the loose original specifications has resulted in a nearly global adoption of a lowest-common-denominator XML, but has made it extremely difficult to promote upgrades and fixes to the XML specifications and implementation.

XML Advantages and Disadvantages

With these constraints of XML in mind, it is useful to examine the benefits of XML, both those that are claimed and those that are seen in practice. Given the domain of this paper, the benefits are evaluated from the perspective of using XML as a data interchange format. XML is:

- **self-describing**. It is a frequently seen claim that XML contains its own metadata, in that the element tags describe the content they contain. It quickly becomes clear that in practice this is often not the case – if it were, there would be no need for XML Schemas, DTDs, taxonomies, annotations, and so on. There is a clear trade-off between how generic an XML structure can be and the number of contextual clues it can contain.

Advanced XML Processing with SAS® 9.3, continued

- **human-readable.** While this is sometimes true, it is also one of the most common sources of errors in XML files. Many users fail to realize that the default encoding for XML is UTF-8, which uses variable length characters. Editing XML files in an editor that is not encoding aware frequently leads to the corruption of the XML file. It is also clearly the case that when non-textual data – images, audio, video, and so on - is introduced, the concept of human readability is subjugated to the ability to display that data in a text-oriented form.
- **internationalization.** This is one of the strongest arguments for the use of XML. While many software systems continue to lag in Unicode support, XML has enforced encoding support since its inception by requiring that any encoding error constitutes a fatal parsing error.
- **ubiquitous.** XML certainly has near universal support, at least for basic parsing.
- **Separates content from presentation.** While XML and XSLT can facilitate this, they by no means require it. XHTML, for example, obviously combines content with presentation.
- **extensible.** This is both an advantage and disadvantage. One of the original design goals of XML was simplicity, yet this has often been sacrificed for extensibility.
- **hierarchical versus tabular.** It is often stated that a benefit of XML is that it supports the representation of hierarchical data that “cannot” be stored or (more truthfully) are more difficult to store in a tabular or relational form. While this is probably the case when XML is used as a persistence mechanism or configuration file, this tends to be reversed when using XML as a data interchange format.

The State of XML

XML has been in development for over 15 years now. The chart below provides some indication of how the complexity of the XML has grown since its inception. Of interest is that the initial release of the SAS XML LIBNAME engine is concurrent with the initial release of the XPath recommendation, an integral technology to the XMLMap technology (MAPPAT).

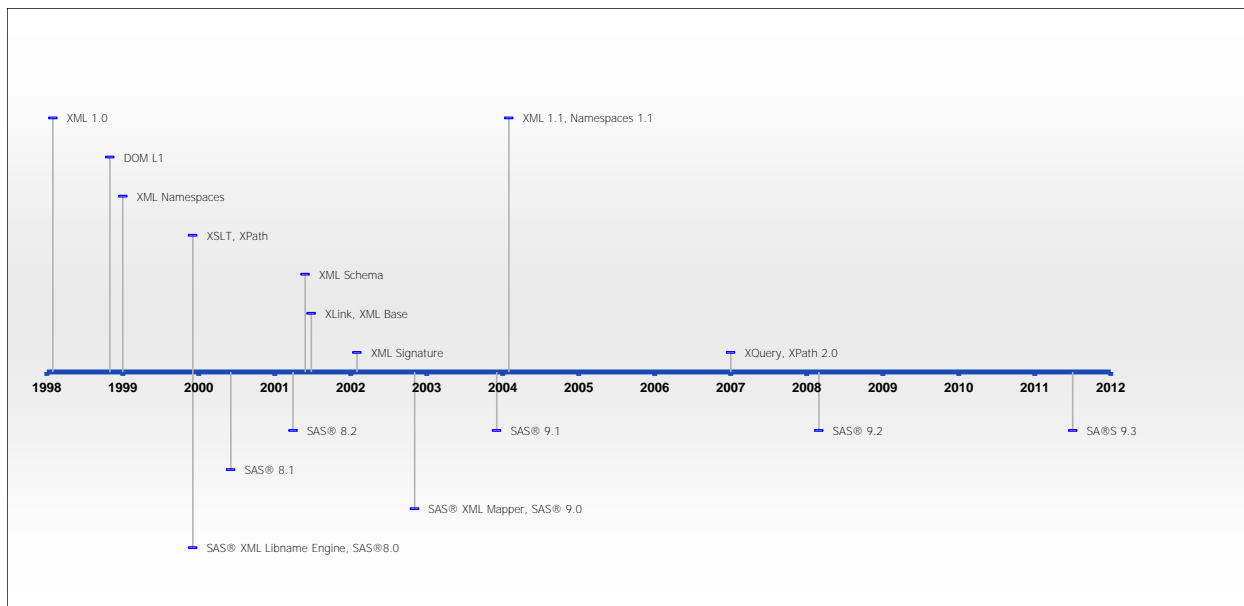


Figure 1: XML Timeline

The good news for XML users is that there are a wide variety of mature XML parsers and tools available. The bad news is the state of the XML family of specifications - the adoption rate of specifications from XML 1.1 onward is quite poor (NW BLOG). Microsoft, for example, has declined to implement XML 1.1 [WC F I O P]. Similarly, XHTML 1.0 and 2.0 never materialized and left HTML5 implementers still attempting to robustly recover (in unspecified ways) from the vast number of syntax errors found in HTML on the Web. That leaves the XML specifications in the unfortunate position of being an immature, stagnant, and yet widely used technology all at the same time. That being said, XML is no danger of disappearing. While some uses of XML such as object persistence, configuration files, and protocols

Advanced XML Processing with SAS® 9.3, continued

have been shifting to simpler formats like JSON (JSON) the investment in XML standards such as XBRL and ACORD demonstrate that XML will be with us for some time. (XBRL) (ACORD).

XML NAMESPACES

What Are XML Namespaces?

One of the major additions to the XML functionality in SAS 9.3 Base SAS is support for XML Namespaces. XML Namespaces are an optional extension to the XML core specification and related specifications to prevent name collisions between multiple schemas when they are used with the scope of an XML file or an outer XML Schema.

For example, here is a simple XML file that uses namespaces to distinguish between three types of addresses.

```
<?xml version="1.0" encoding="UTF-8"?>
<PEOPLE xmlns:HOME="http://sample.url.org/home"
  xmlns:IP="http://sample.url.org/ip"
  xmlns:WORK="http://sample.url.org/work">
  <PERSON>
    <NAME>Joe Smith</NAME>
    <HOME:ADDRESS>1234 Elm Street</HOME:ADDRESS>
    <HOME:PHONE>999-555-0011</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 101</WORK:ADDRESS>
    <WORK:PHONE>999-555-0101</WORK:PHONE>
    <IP:ADDRESS>192.168.1.1</IP:ADDRESS>
  </PERSON>
  ...
  <PERSON>
    <NAME>Pat Perkinson</NAME>
    <HOME:ADDRESS>1395 Half Way</HOME:ADDRESS>
    <HOME:PHONE>999-555-0033</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 103</WORK:ADDRESS>
    <WORK:PHONE>999-555-0103</WORK:PHONE>
    <IP:ADDRESS>10.0.1.1</IP:ADDRESS>
  </PERSON>
</PEOPLE>
```

File 1: People.XML

Obviously, namespaces are not required in a simple scenario like this, but they become a necessity when combining schemas from two or more standards bodies.

Namespace Confusion

XML Namespaces are similar to many object-oriented programming languages in concept, but in execution, there are some aspects of XML Namespaces that tend to be confusing. This was likely the result of the XML Namespace specification being written after the XML core and XML Schema specifications as an optional enhancement.

One common source of misunderstanding is that namespaces are identified by a Uniform Resource Identifier, or URI. URIs are a superset of Uniform Resource *Locators*, or URLs, that are in common use as Internet Web page addresses. This similarity leads many users to attempt to navigate to the URI, or associate some physical meaning to the string, while there is no requirement that the address exists or that there be any content associated with it.

For example, XBRL uses the following schemas so frequently that they define a convention for naming the prefixes associated with their namespaces, as shown in Table 1. Note that the three URIs based on xbrl.org are strictly identifiers, as they have no corresponding content, while those based on w3.org are URLs in addition to being URIs, since they are associated with placeholder pages on the Web(XBRL).

link	http://www.xbrl.org/2003/linkbase
xbrli	http://www.xbrl.org/2003/instance
xl	http://www.xbrl.org/2003/XLink
xlink	http://www.w3.org/1999/xlink
xml	http://www.w3.org/XML/1998/namespace
xsi	http://www.w3.org/2001/XMLSchema-instance
xsd	http://www.w3.org/2001/XMLSchema

Table 1: XBRL Namespace Prefix Conventions

Advanced XML Processing with SAS® 9.3, continued

Another source of confusion is the prefix notation used in the declaration of namespaces. Many users assume that the prefix uniquely identifies a namespace URI throughout an entire XML file, when in fact the scope of a prefix is determined by its position within the hierarchical structure of the XML. This can lead to a single prefix referring to multiple namespace URIs, or multiple prefixes referring to a single namespace URI. While this capability is required to support nested XML Schemas, the added layer of indirection adds a great deal of complexity.

Further, it is possible to specify a namespace without any prefix, which is known as a default prefix. While this can make an XML file more concise, the use of default namespaces is not fully compatible with the XPath specification (EDANK). A full discussion of all the ramifications of the XML Namespace specification is beyond the scope of this paper.

Namespaces in SAS XML Mapper

To support XML Namespaces, many changes were required in the XMLV2 LIBNAME engine, XMLMap syntax, and SAS XML Mapper. Some of the major changes are shown in Figure 2. In highlight 1, note that the URI of the namespace is displayed next to the element name using the following notation: {URI}Element . This was selected because of the possibility of ambiguity in prefix names noted in the previous section. Highlight 2 shows the new interface for editing namespaces in the SAS XML Mapper. It is recommended that direct editing of namespaces be treated as a feature for advanced users only. The third highlight shows the representation of XML Namespaces within the XMLMap tree, and the fourth shows how they are rendered in the XMLMap.

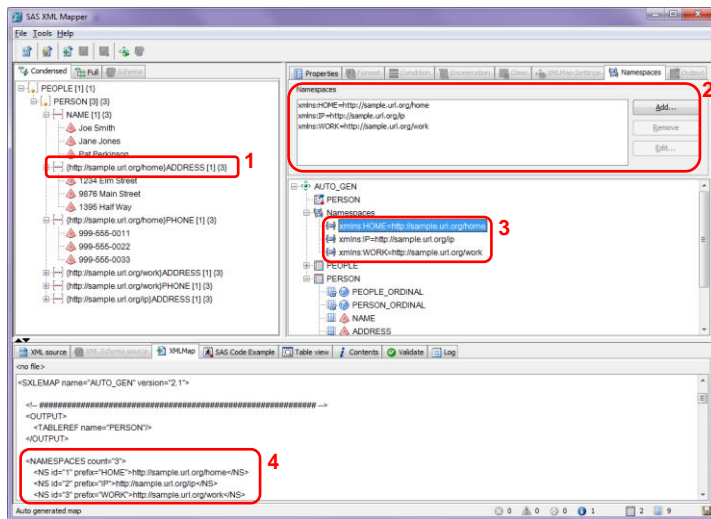


Figure 2: SAS XML Mapper Namespace Enhancements

Here are excerpts from the XMLMap that is created by automapping that XML file:

```
<SXLEMAP name="AUTO_GEN" version="2.1">
...
  <NAMESPACES count="3">
    <NS id="1" prefix="HOME">http://sample.url.org/home</NS>
    <NS id="2" prefix="IP">http://sample.url.org/ip</NS>
    <NS id="3" prefix="WORK">http://sample.url.org/work</NS>
  </NAMESPACES>
...
  <TABLE description="PERSON" name="PERSON">
    <TABLE-PATH syntax="XPath">/PEOPLE/PERSON</TABLE-PATH>
    ...
    <COLUMN name="NAME">
      <PATH syntax="XPath">/PEOPLE/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>13</LENGTH>
    </COLUMN>

    <COLUMN name="ADDRESS">
      <PATH syntax="XPathENR">/PEOPLE/PERSON/{1}ADDRESS</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

Advanced XML Processing with SAS® 9.3, continued

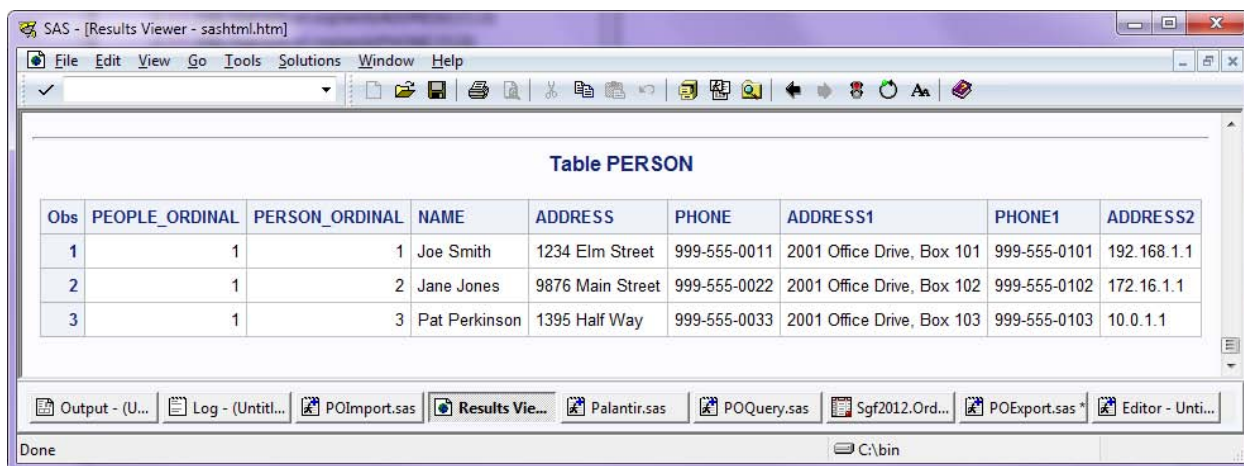
```
<LENGTH>16</LENGTH>
</COLUMN>
...
```

File 2: People.map

First, notice that there is a new NAMESPACEs section in the XMLMap, which is used for tracking the namespaces that are used in the XML file. Each is assigned a unique identification in the id attribute, since there is no guarantee that the prefix or URI is unique.

Second, a new type of path syntax is introduced that uses the id of the namespace embedded in the path (for example, "...{1}ADDRESS..." in the above example) rather than use either the prefix or URI. The syntax value "XPathENR" is short for "XPath with Embedded Namespace Reference." There are three reasons for this: first, it makes the reference unique, as noted above; second, it makes the XMLMap considerably more concise; and finally, it resolves the problem of referring to default namespaces.

Figure 3 shows the results of PROC PRINT on the main table generated by the XMLMap:



Obs	PEOPLE_ORDINAL	PERSON_ORDINAL	NAME	ADDRESS	PHONE	ADDRESS1	PHONE1	ADDRESS2
1	1	1	Joe Smith	1234 Elm Street	999-555-0011	2001 Office Drive, Box 101	999-555-0101	192.168.1.1
2	1	2	Jane Jones	9876 Main Street	999-555-0022	2001 Office Drive, Box 102	999-555-0102	172.16.1.1
3	1	3	Pat Perkinson	1395 Half Way	999-555-0033	2001 Office Drive, Box 103	999-555-0103	10.0.1.1

Figure 3: PROC PRINT Result of XMLMap with Namespaces

AUTOMAPPING

The automapping functionality in SAS XML Mapper was introduced in SAS 9.1.3 SP3. It was never intended to fully replace the drag-and-drop creation of XMLMaps, but it does have certain aspects that make it more useful than might be immediately obvious. Several of those aspects are discussed in this section, using the FAQ (Frequently Asked Questions) approach.

Why does the automapper create so many tables?

Two reasons: first, the automapping algorithm creates a representation of the XML that is as relational as possible from the available context. It does this, along with the creation of surrogate keys, so that the tables can be rejoined using PROC SQL. An example of this is demonstrated below.

Secondly, the XML file instances might not fully represent the cardinality of all analogous XML files. That is, if the XML file that is used to generate an XMLMap contains only one instance of a given element, a human user might choose to map that element as a column in a higher level table. In contrast, automapping creates another table for that element, so that if another XML file comes along that contains multiple recurrences of that element, it is properly captured, rather than overwriting or appending data.

How can I get SAS XML Mapper to automap my schema?

While XML Mapper is capable of automapping many XML Schemas, there are some limitations. There are two solutions available if you run into one of them. The simplest is to use an XML file for automapping instead of the XML schema. The other approach is to improve the design of the schema. In order to do so, it helps to understand why some of the limitations occur.

Fundamentally, XML Schemas are more expressive than XML files. An XML file can have only one structure, whereas a single XML Schema can literally represent an infinite number of XML file structures. In terms of graph

Advanced XML Processing with SAS® 9.3, continued

theory, an XML file is always a tree, which is a directed, connected acyclic graph with a unique root. However, an XML Schema definition can be a disjoint cyclic graph, which allows recursion, multiple roots, and disconnected subsets. This flexibility was built into the XML Schema specification to support the original purpose of XML – semi-structured documents. While it functions well in that capacity, it tends to be a hindrance to the data exchange usage of XML.

One obvious case where an XML Schema cannot be automapped is when the schema is recursive. This is because the XMLMap technology is based on listing XPathS. Since a recursive schema contains an infinite number of XPathS, it is not possible to list them all.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="node">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="node"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

File 3: Recursive.xsd

Another problem with schemas is ambiguity. It is a frequent practice to create XML Schemas with many or all elements exposed globally. This practice is sometimes called the “Salami Slice” approach to schema design. Unfortunately, in contrast to a fully-nested or “Russian Doll” Schema, this makes it impossible to programmatically determine the root element of the schema. Prior to SAS 9.3, SAS XML Mapper arbitrarily used the first element definition appearing in such a schema as the root, often with poor results. Now SAS XML Mapper opens a dialog box and allows the user to choose which of the global elements to use as the root.

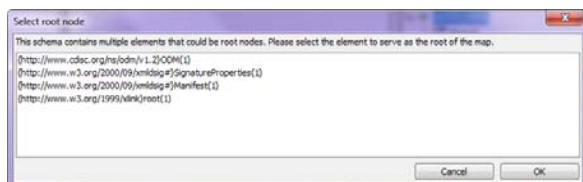


Figure 4: Selecting Root Node in an Ambiguous XML Schema

The varied complexities of XML Schema design are beyond the scope of this paper. A valuable resource is the article “Introducing Design Patterns in XML Schemas” (KHAN). It discusses the “Salami Slice,” “Russian Doll,” and other XML Schema models and their tradeoffs.

ROUND-TRIPPING XML

What Is Round-Tripping XML?

The ability to “round-trip” XML is frequently requested by customers. In researching these requests, we have found that the desired functionality is usually one of the following:

- the ability to read an arbitrary XML file into SAS, do *something* with it, and then rewrite it exactly as it came in, with the exception of those changes
- the ability to take SAS data sets, which might or might not have come from an XML source, and write them to an arbitrary XML file
- the ability to update XML files in place

In practice, the first two of those cases can be solved by using the same techniques, while the last is simply not recommended due to the inefficiencies it entails.

The fundamental problem of round-tripping complex XML is probably best described by analogy. Imagine a bowl of strawberries and a carton of ice cream. It is a trivial process to throw them in a blender and create a milkshake. Reversing the process is not so trivial – at the least it requires that you tag every strawberry molecule with an identifier and track how it is related to other molecules from the same strawberry...and that you kept any scraps or

Advanced XML Processing with SAS® 9.3, continued

bad spots you might have cut out. And what if the arrangement of the strawberries in the original bowl was important?

It is clear that the amount of metadata that must be collected to perform full round-tripping is extensive. Not only must the explicit relationships be captured, but the implicit ones as well, such as disjoint element sequencing. (Imagine the sequence A1, B1, A2, B2, C1 – the question arises whether A1, A2, B1, B2, C1 is an equivalent ordering. It is in some cases and not in others.)

Round-Tripping Example

With a few caveats, it is not that hard to generate arbitrary XML output from SAS. Consider the following XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<order ordnum="0120983" date="2011-11-03">
  <customer id="C1837">
    <shipTo>
      <name>Joe Smith</name>
      <address>
        <street>100 Alfred Lerner Way</street>
        <city>Cleveland</city>
        <state>OH</state>
        <zip>44114</zip>
      </address>
    </shipTo>
    <billTo>
      <name>Joe Smith</name>
      <address>
        <street>George Hallas Dr</street>
        <city>Canton</city>
        <state>OH</state>
        <zip>44708</zip>
      </address>
    </billTo>
  </customer>
  <payment type="account" terms="NET30">
    PO#1231231
  </payment>
  <items>
    <item sku="FB123" qty="12" price="59.99">
      Football, leather, official
    </item>
    <item sku="JRSYDC1279" qty="1" price="229.99">
      Football jersey, Classic, Dallas Cowboys, Roger Staubach
    </item>
  </items>
</order>
```

File 3: PO.xml

Notice that PO.xml is nested several layers deep, and contains repeating groups of elements, such as <item>. The following process will bring this XML into SAS and then recreate the XML file from the SAS data sets.

First, Generate an XMLMap by using the SAS XML Mapper automapping feature, with the Generate surrogate keys option turned on, which is the default setting. Then use the SAS code generated by XML Mapper to use the XMLV2 Libname Engine and PROC COPY to move the datasets to the SAS library PO. If you then run PROC CONTENTS on each of the tables, you will see they have the following structure:

order			
#	Variable	Type	Len
1	order_ORDINAL	Num	8

2	ordnum	Char	32
3	date	Num	8
4	order	Char	32

Advanced XML Processing with SAS® 9.3, continued

customer			
#	Variable	Type	Len
1	order_ORDINAL	Num	8
2	customer_ORDINAL	Num	8
3	id	Char	32
4	customer	Char	32

shipTo			
#	Variable	Type	Len
1	customer_ORDINAL	Num	8
2	shipTo_ORDINAL	Num	8
3	name	Char	32
4	shipTo	Char	32

billTo			
#	Variable	Type	Len
1	customer_ORDINAL	Num	8
2	billTo_ORDINAL	Num	8
3	name	Char	32
4	billTo	Char	32

payment			
#	Variable	Type	Len
1	order_ORDINAL	Num	8
2	payment_ORDINAL	Num	8
3	type	Char	32
4	terms	Char	32
5	payment	Char	32

items			
#	Variable	Type	Len
1	order_ORDINAL	Num	8
2	items_ORDINAL	Num	8
3	items	Char	32

item			
#	Variable	Type	Len
1	items_ORDINAL	Num	8
2	item_ORDINAL	Num	8
3	sku	Char	32
4	qty	Num	8
5	price	Num	8
6	item	Char	80

Note that most tables contain two columns with the name *_ORDINAL. Think of each row in the table as an XML element. The first of the two _ORDINAL columns functions as a foreign key to the parent element that contained this row and the second is the key of this row. By using this approach it is easy to write joins for queries on the data.

For example,

```
PROC SQL DQUOTE=ANSI;
  SELECT SUM(item.qty * item.price) as Total
  FROM SGF2012.item, SGF2012.items, SGF2012."order"
  WHERE ((item.items_ORDINAL = items.items_ORDINAL) &&
         (items.order_ORDINAL = "order".order_ORDINAL) &&
         "order".ordnum='0120983');
run;
```

produces

Total
949.87

Similarly, to re-create the original XML file, continue chaining the keys together using where clauses and a fairly

Advanced XML Processing with SAS® 9.3, continued

simple DATA step program can be created. Note that the code is simplified in this case to take advantage of the lack of repeating elements, but the general principles can be extended to almost all cases.

```

data _null_;
set SGF2012.Order;
  format date IS8601DA.;
  file "C:\Users\example\Desktop\SGFData\POOut.xml";
  tag=catt(' ', '<order ordnum="' || ordnum || '" date="' || vvalue(date) || '">');
  %let orderkey=order_ORDINAL;
  put tag;

set SGF2012.Customer (where=(order_ORDINAL=&orderkey));
  %let custkey=customer_ORDINAL;
  tag=catt(' ', '<customer id="' || id || '">');
  put tag;

set SGF2012.shipTo (where=(customer_ORDINAL=&custkey));
  %let shtokey=shipTo_ORDINAL;
  tag=catt(' ', '<shipTo>');
  put tag;
  tag=catt(' ', '<name>', name, '</name>');
  put tag;

set SGF2012.shipAddr (where=(shipTo_ORDINAL=&shtokey));
  tag=catt(' ', '<address>');
  put tag;
  tag=catt(' ', '<street>', street, '</street>');
  put tag;
  tag=catt(' ', '<city>', city, '</city>');
  put tag;
  tag=catt(' ', '<zip>', zip, '</zip>');
  put tag;
  tag=catt(' ', '</address>');
  put tag;

  tag=catt(' ', '</shipTo>');
  put tag;

set SGF2012.billTo (where=(customer_ORDINAL=&custkey));
  %let btokey=billTo_ORDINAL;
  tag=catt(' ', '<billTo>');
  put tag;
  tag=catt(' ', '<name>', name, '</name>');
  put tag;

set SGF2012.billAddr (where=(billTo_ORDINAL=&btokey));
  tag=catt(' ', '<address>');
  put tag;
  tag=catt(' ', '<street>', street, '</street>');
  put tag;
  tag=catt(' ', '<city>', city, '</city>');
  put tag;
  tag=catt(' ', '<zip>', zip, '</zip>');
  put tag;
  tag=catt(' ', '</address>');
  put tag;

  tag=catt(' ', '</billTo>');
  put tag;

  tag=catt(' ', '</customer>');
  put tag;

set SGF2012.Payment (where=(order_ORDINAL=&orderkey));
  tag=catt(' ', '<payment type="' || type || '" terms="' || terms || '">', payment, '</payment>');
  put tag;

set SGF2012.Items (where=(order_ORDINAL=&orderkey));
  %let itemkey=items_ORDINAL;
  tag=catt(' ', '<items>');

```

Advanced XML Processing with SAS® 9.3, continued

```

    put tag;
run;

data _null_;
set SGF2012.Item (where=(items_ORDINAL=&itemkey));
file "C:\Users\sasthc\Desktop\SGFData\POOut.xml" mod;
tag=catt(' ', '<item sku="' , sku, '" qty="' , qty, '" price="' , price, '">', item, '</item>');
put tag;

run;

data _null_;
file "C:\Users\sasthc\Desktop\SGFData\POOut.xml" mod;
tag=catt(' ', '</items>');
put tag;

tag=catt(' ', '</order>');
put tag;
run;

```

File 4: POExport.sas

As mentioned earlier, there are some drawbacks to this method. The first is that it is prone to syntax errors. Each PUT statement is a chance to forget a quotation mark, slash, or similar character. Another is that it puts the burden of making sure the correct encoding is used on the user – a program that works on a UNIX host can fail on Windows, or one that works with one data set can fail on another if it contains Unicode characters. However, it is extremely flexible, and combined with automapping can be quite effective.

CONCLUSION

XML is a complex network of interrelated specifications that have developed in an organic fashion to create an environment that is rich in tools and partnerships but sometimes weak in consistency. The flexible nature of XML frequently results in it being a “Jack of all trades but master of none,” but in many applications, it is entrenched and will be part of enterprise data integration for the foreseeable future.

SAS is committed to support XML so that our customers can leverage the power of SAS analytics and solutions for their XML computing needs. In the SAS 9.3 timeframe, XML support has been delivered in the form of XML Namespace support, improved automapping, and numerous less visible performance and quality enhancements. Like any business, we must prioritize our efforts to meet those needs efficiently and effectively. Your continuing input will help us to do so.

REFERENCES

- 099-2008. Martell, Carol. 2008. “SAS® XML Mapper to the Rescue,” Proceedings of the SAS® Global Forum 2008 Conference. Cary, NC. SAS Institute Inc. Available at <http://www2.sas.com/proceedings/forum2008/099-2008.pdf>.
- 173-28. Friebel, Anthony. 2007. “XML? We do that!” Proceedings of the SAS® Global Forum 2007 Conference. Cary, NC. SAS Institute Inc. Available at <http://www2.sas.com/proceedings/sugi28/173-28.pdf>.
- 240-30. Shoemaker, Jack. 2005. “XML Primer for SAS Programmers.” Proceedings of the Thirtieth Annual SAS® Users Group International Conference. Cary, NC. SAS Institute Inc. Available at <http://www2.sas.com/proceedings/sugi30/240-30.pdf>.
- ACORD. “Standards Implementation.” WWW. 1/24/2012. Available at <http://www.acord.org/standards/implementation/Pages/default.aspx>.
- EDANK. Dankert, Edwin. “XPath and Default Namespace Handling.” WWW. 10/28/2010. Available at <http://www.edankert.com/defaultnamespaces.html>.
- JSON. “Introducing JSON”. WWW. 1/24/2012. Available at <http://www.json.org/>.
- KHAN. Khan, Ayub and Sum, Marina. “Introducing Design Patterns in XML Schemas”. 11/9/2006. Available at http://developers.sun.com/jenterprise/archive/nb_enterprise_pack/reference/techart/design_patterns.html.
- MAPPAT. Friebel et. al. “USPTO Patent Full-Text and Image Database”, 4/5/2011, Available at <http://patft.uspto.gov/netacgi/nph-Parser?Sect2=PTO1&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&i=50&d=PALL&RefSrch=yes&Query=PN/7921359>.

Advanced XML Processing with SAS® 9.3, continued

NWBLOG.Walsh, Norman. "XML 1.1: Dead on Arrival". 9/30/2004. Available at <http://norman.walsh.name/2004/09/30/xml11>.

SASWWW. "Enterprise Data Access". WWW. 1/1/2012, Available at <http://www.sas.com/software/data-management/enterprise-data-access/index.html>.

W3CXML. "Extensible Markup Language (XML)". WWW. 1/24/2012. Available at <http://www.w3.org/XML/>.

WCFIOP. "Web Services Protocols Interoperability Guide". 1/1/2012. Available at <http://msdn.microsoft.com/en-us/library/ms734776.aspx>.

WPXMLI. "List of XML markup languages". WWW. 2/3/2012. Available at http://en.wikipedia.org/wiki/List_of_XML_markup_languages.

XBRL. "eXtensible Business Reporting Language". WWW. 1/22/2012. Available at <http://www.xbrl.org/>.

ACKNOWLEDGMENTS

Lisa Brown, SAS
Anthony Friebel, Consultant
Karen Hoffman, SAS
Titus Lee, SAS
David Phillips, SAS
Mitchell Soltys, SAS
Deanna Warner, SAS

RECOMMENDED READING

- *SAS Institute Inc. 2011 Base SAS® 9.3 Procedures Guide. Cary, NC: SAS Institute Inc.*
- *SAS Institute Inc. 2011 SAS® 9.3 XML LIBNAME Engine: User's Guide. Cary, NC: SAS Institute Inc.*
- Extensible Markup Language (XML) 1.0 (Fifth Edition)
Available at <http://www.w3.org/TR/2008/REC-xml-20081126/>
- Namespaces in XML 1.0 (Second Edition)
Available at <http://www.w3.org/TR/2006/REC-xml-names-20060816/>
- XML Schema Part 0: Primer (Second Edition)
Available at <http://www.w3.org/TR/xmlschema-0/>
- XML Schema Part 1: Structures (Second Edition)
Available at <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/structures.html>
- XML Schema Part 2: Datatypes (Second Edition)
Available at <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html>
- XML Path Language (XPath)
Available at <http://www.w3.org/TR/xpath/>

CONTACT INFORMATION

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

Thomas W. Cox

SAS Campus Drive
SAS Institute Inc.
Cary, NC 27513
Work Phone: 919-531-4875
E-mail: Thomas.Cox@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.