Paper 036-29

XML in the DATA Step

Michael Palmer, Zurich Biostatistics, Inc., Morristown, NJ

ABSTRACT

SAS® Institute's (SI) XMLMAP imports XML via the LIBNAME XML engine and a mapping file, by-passing the DATA-step. For export of XML, SI offers custom tagsets and the Output Delivery System (ODS). Custom tagsets are built with PROC TEMPLATE, by-passing the DATA-step. By contrast, the DATA-step method discussed in this paper uses a single, uniform methodology to import, export, and transform user-defined XML vocabularies in the familiar DATA-step. With XMLMAP, all of the information about the original XML hierarchical structure is lost and export back to XML is not readily available except for several XML formats built into the XML engine. For XML export, one has to use arcane customized tagsets and ODS. Nevertheless, SI's toolbox does work and has reportedly scaled to large, real-life uses. The DATA-step method, on the other hand, indexes an XML hierarchy and uses these indexes to flatten the XML hierarchy and to preserve the information about the hierarchy in a DATA-step friendly format. The method has been in use successfully and supports complex XML vocabularies such as the pharmaceutical industry's CDISC XML standard for clinical data and proprietary XML schemas in the financial industry. It has scaled to XML files of over one million records.

INTRODUCTION

XML is a format for data that is hierarchical, text-based, and consists of named content. These three characteristics make XML difficult to work with in the row-and-column-oriented DATA-step and file structure in SAS. This paper contrasts the SAS Institute's (SI) XML tools with a DATA-step friendly way to work with XML vocabularies. Both methods support the import of XML into SAS, the export of XML from SAS, and the processing and transformation of XML in the DATA-step.

A toolkit for working with XML in SAS should have the properties listed below because they bring true XML capability to the SAS workplace while preserving established ways of working in SAS.

1. The toolkit works with a very broad class of user-defined XML vocabularies.
2. The toolkit uses one uniform methodology for the import, export, and transformation of any instance of any data-centric XML vocabulary.
3. XML work all takes place in the base SAS DATA-step. The programming techniques necessary to implement the method are familiar to even beginning SAS programmers.

SAS Institute offerings for working with XML include a LIBNAME engine for importing and exporting some fairly simple types of XML, including some common industry forms. The XMLMAP option of the LIBNAME Engine can import a broader class of XML directly to SAS datasets. A graphical user interface tool is also available for assisting in the creation of XML mapping files using a familiar drag-and-drop motif. The XML LIBNAME engine can automatically export a SAS-defined type of XML and some common industry forms. With customized tagset extension programming, ODS may export some user-defined types of XML. Version 9 also contains broader support for parsing XML in the DATA-step.

WHAT'S THE PROBLEM WITH XML AND SAS?

XML is hierarchical, unlike the typical SAS dataset. In the typical SAS setting, data exist in fields on records in datasets. In a given dataset, every record has precisely the same fields with precisely the same attributes. One relates a data item to another data item by the fact that they share, or do not share, key variables and key variable values. In XML, a data item relates to other data items by the ancestors, descendants, and siblings that they share. XML is hierarchical so a data item inherits identity from its ancestors, shares identity with its siblings, and passes on identity to its descendents. To identify a data item, one has to literally traverse all of its ancestors. This traversal creates a path through the XML file. By contrast, in the typical SAS dataset, one identifies a data item by looking at key fields on the same record.

XML is text. All XML is text, accessible in a simple text editor. In SAS datasets, by contrast, fields can be text or numeric and, despite the rich set of text-processing functions in SAS, numeric data is easier to process than text data. In addition, proprietary tools such as SAS Viewer or the base SAS product are necessary to access data and attribute information in a SAS dataset.

XML consists of named content. The name of each item in an XML file is written completely in text when that element's scope begins and written out again completely when that element's scope ends. The sequence of items in

that scope is predictable to some extent, but not fixed like it is for a typical SAS dataset. In addition, field attributes, such as field name, are not explicit in SAS but are stored separately from the data itself.

From the point of view of a SAS user, XML is trouble for these three reasons: it's *hierarchical*, it's *all text*, and *all content is explicitly named*.

Despite these complications, SAS users would like to be able to import XML, export XML, and work with XML in the DATA-step as easily as they work with other data originating in other formats.

A WISH LIST FOR A SUCCESSFUL METHODOLOGY

A successful methodology for working with XML in the DATA-step would include the following three features.

One, the methodology should flatten the hierarchical XML structure into the row-and-column structure of SAS datasets but without losing the information needed to recreate the hierarchy. That is, the ancestor-sibling-descendent structure in native XML should be indexed into a flat representation.

Two, the methodology should replace the unpredictably long and complex text strings that describe paths to data items in XML with numbers. Long and complex text strings are cumbersome to work with in the SAS DATA-step. Numbers are relatively easy to work with.

Three, the methodology should fit the numerically indexed content into a regular row-and-column SAS dataset.

A toolkit that has these characteristics along with the capability to support general data-centric XML import, export, and transformation would go a long way towards meeting the need for a familiar DATA-step way to work with XML.

SAS INSTITUTE'S XML METHODOLOGY

Starting with SAS version 8.2, SAS has included production-level tools for working with user-specified XML. The tools have evolved and been refined with each new SAS version, but the basic approach has stayed the same. Import from XML is controlled by a mapping file that identifies paths in the XML and, for each path, defines a destination SAS dataset and destination field in that dataset. This is implemented with the XML engine of the LIBNAME statement.

XML IMPORT

Figure 1 shows a sample of XML for the rss XML vocabulary. This XML sample and the accompanying material were downloaded from the SAS web site.

The rss vocabulary, used to illustrate XML import at the SAS web site, has several features that make it a good choice to illustrate processing of data-centric XML. For instance, it has data in attributes, e.g., `version="0.91"`, and elements and it has several levels of hierarchy from the root element, `<rss>`, to the deepest level, `<title>`. The right side of Figure 1 shows the rss sample. The left side of Figure 1 shows the XMLMAP file that the SAS XML engine uses to move data from rss to dataset named channel.

Distinguishing features of a map for XMLMAP are at least two. It is itself XML and it relies on the XPATH XML standard. The various XPATH statements in the map use XPATH syntax to define a sequence of XML element type names and attribute names from the `<rss>` root element to specific data in the rss vocabulary. XPATH syntax is defined in the World Wide Web consortium's XPATH standard. For each path defined in rss, XMLMAP also defines a field in the channel dataset. The XML engine in SAS moves the data point at the end of each path to the designated field in channel. Figure 2 shows how to run XMLMAP in SAS and Figure 3 shows the result of running the rss map with the rss sample in Figure 1. A dataset called channel is created with two fields, title and version.

Figure 1. XMLMAP with a Sample of XML

```
<SXLEMap>
   <TABLE name="channel">
      <TABLE_XPATH> /rss/channel
</TABLE_XPATH>
      <COLUMN name="title">
       <XPATH> /rss/channel/title </XPATH>
       <TYPE> character </TYPE>
       <DATATYPE> string </DATATYPE>
       <LENGTH> 200 </LENGTH>
      </COLUMN>
      <COLUMN name="version">
       <XPATH> /rss@version </XPATH>
       <TYPE> character </TYPE>
       <DATATYPE> string </DATATYPE>
       <LENGTH> 8 </LENGTH>
      </COLUMN>
   </TABLE>
</SXLEMap>
```

```
<rss version="0.91">
     <channel>
            <title>WriteTheWeb</title>
     </channel>
</rss>
```

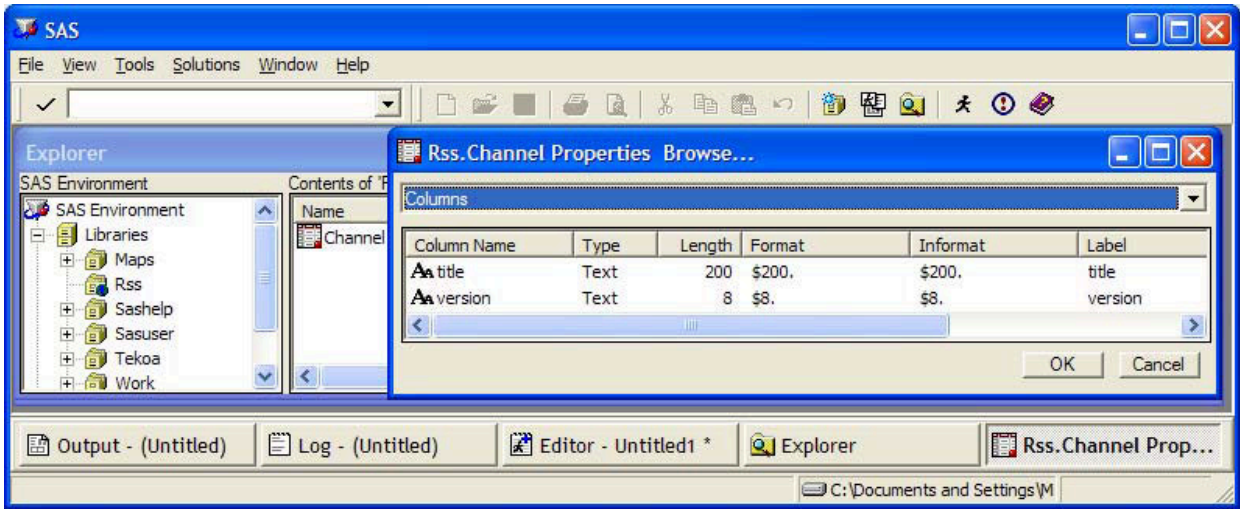
Figure 2. Running XMLMAP

```
filename map '\SUGI2004\ImEx\rss.map';
libname rss xml 'C:\Presentations\ SUGI2004\ImEx\rss112.xml' xmlmap=map;
```

Invokes XML engine

Designates XMLMAP

Figure 3. XMLMAP results



XML EXPORT

Creating XML via custom tagsets in PROC TEMPLATE and exporting it with the XML engine is a totally different process from importing XML with XMLMAP and the XML engine.

On import, SAS retains none of the hierarchical information in the XML file, and this has a significant implication, particularly when it's time to export XML from SAS. For user-specified XML schemas or DTDs, the export route from SAS to XML goes through customized ODS tagsets and PROC TEMPLATE rather than through the XML mapping file and LIBNAME engine used to import XML. This requires learning what amounts to a new language. That language is described in SAS documentation, particularly the Base SAS Community XML pages at www.sas.com. The URL for SI's XML support is in the "References" section at the end of this paper. This paper is concerned with a 100% DATA-step solution to XML export and so it will not cover in detail the esoteric subjects of ODS tagsets and PROC TEMPLATE.

Using the SI tools, there is nothing as simple as a "reverse map" that connects a field in a dataset to the user's own XML via a path definition. Instead, the export process, in essence, defines one, and only one, XML schema that it supports and allows users to put a SAS dataset name and SAS field names into that schema along SAS-defined paths. Getting the dataset and field names into the exported XML requires programmatically defining a tagset in PROC TEMPLATE.

Figure 4 shows the skeleton of a PROC TEMPLATE for exporting XML and Figure 5 fills in that skeleton for two of the events. Figure 6 shows the SAS code that creates the export XML.

One might think that through carefully defining events, using the special syntax for that in PROC TEMPLATE and the even more special syntax and reserved names supported by the XML engine, it would be possible, although tedious, to have the XML engine export user-defined XML. For example, the author attempted to export the rss XML from SAS after successfully importing it into SAS using the XMLMAP. This is not possible (the impossibility confirmed by a source at SI) for several reasons. First, the XML engine is not case aware. XML element and attribute names are case sensitive but the XML engine is not aware of case. For example, `<channel>` in Figure 1 is not the same as `<CHANNEL>` in Figure 7. Second, the XML engine for export puts all data into elements, one field in the source SAS dataset per element, and groups them by source dataset row. For example, "version" in Figure 1, is an attribute on the root element, `<rss>` and it is mapped to the same dataset as the element content for `<title>`. It is impossible with the XML engine to export "version" as an attribute on an element that contains other data. Third, the XML engine can export data from just one SAS dataset at a time so it is impossible to put together exported XML that is a composite of data from several source datasets. One can merge data from several source datasets before the export to XML, as long as such a merge makes sense given the key structure of the datasets, and then export the long records to the mirror-image XML that the XML engine can produce. But, XML offers much more flexibility in data structures than this and very often the exact structure is dictated by needs outside the SAS world. It is impossible for the XML engine to match the options that XML offers for data structures.

Figure 4. Creating the Tagset

```
proc template;
define tagset Tagsets.ZBIioXML;
   define event XMLversion;
  . . .
   end;
   define event table_body;
  . . .
   end;
   define event row;
  . . .
   end;
   define event data;
  . . .
   end;
   define event field;
  . . .
  end;
end;
```

Figure 5. Defining an Event

```
define event data;
      start:
      break / if !cmp( SECTION , "body" );
      ndent;
      trigger field;
      finish:
      break / if !cmp( SECTION , "body" );
      trigger field;
   end;
```

```
define event field;
      start:
      put "<" NAME ;
      put "> " ;
      put VALUE  ;
      put " "  ;
      break  ;
      finish:
      put "</" NAME ">";
      put nl;
      xdent;
      break;
   end;
```

Figure 6. Running ODS Tagsets

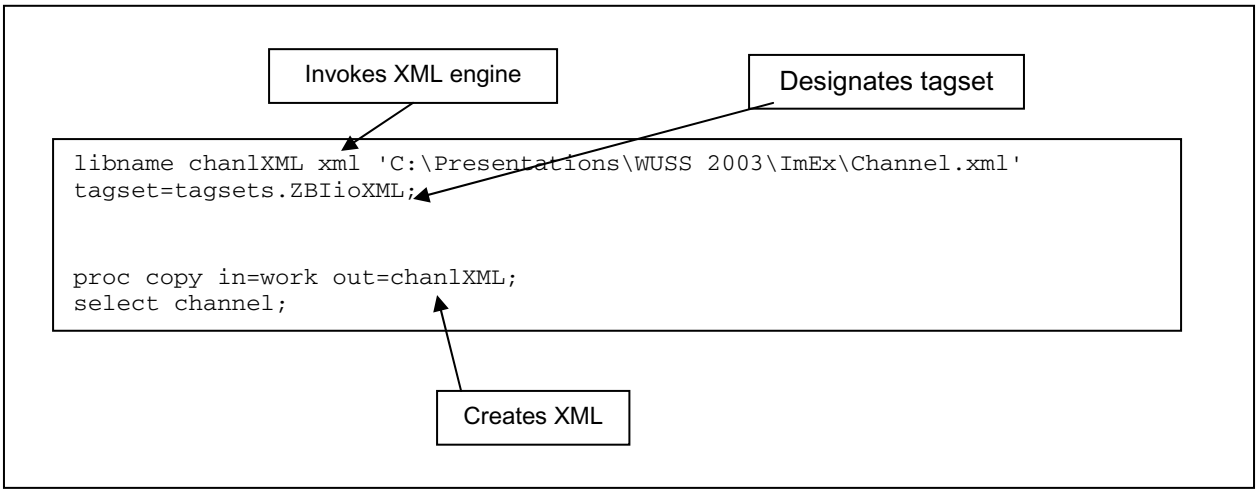
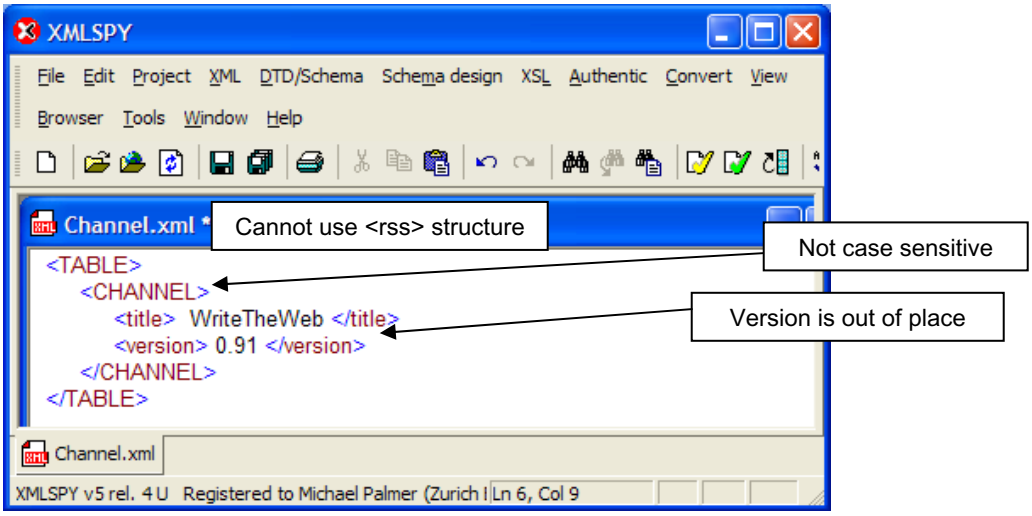


Figure 7. XML Engine-exported XML



SAS AND XML : OIL AND WATER?

Given these limitations of the XML engine in SAS, a natural question is whether or not SAS is the right place to work with XML at all.

SAS is a truly powerful tool because it combines superb data management capabilities with state-of-the-art statistical capabilities and ties it all together with a fully functional programming environment. In addition, SAS has tremendous cross-platform capabilities. This puts SAS-based systems in the middle of many sophisticated information systems that involve complex human-machine and machine-machine interactions.

XML has become ubiquitous in information technology because it is a vendor-neutral, platform-independent format for self-describing information. XML has become the preferred format for heterogeneous information systems. In other words, SAS and XML really have to work well together. The author's experience, primarily in the pharmaceutical and medical worlds but also with some exposure to financial systems, is that SAS and XML can do better than just work well together. XML can integrate SAS into heterogeneous systems for truly remarkable results with the use of a functional toolkit that feels comfortable to SAS users. The rest of this paper will describe that toolkit.

A DATA-STEP XML METHODOLOGY

A straightforward tool for working with XML in the DATA-step can be based on the indexing of levels of hierarchy in an XML instance.

An algorithm for numbering levels of depth in an XML hierarchy and for numbering siblings at a given level satisfies the three conditions given above for an XML solution in SAS:

1. retain information on the XML parent-child-sib relationships
2. in numerical indexes
3. stored in flat SAS records.

But the straightforward solution has a problem. The problem is that, unlike flat files, XML instances for a given XML vocabulary don't necessarily have identical, or even similar structures. They can, and often have, very heterogeneous structures. It's even possible to have the identical content, which should end up in identical data structures in SAS, in very different XML structures from the same XML vocabulary. This situation makes the straightforward algorithm unreliable because the indexes that it produces will not be invariant to the way an XML instance happens to be put together. To make the indexing invariant, it has to be tied to the structure of the XML vocabulary for the instance, not to the instance itself. The construction of such an invariant index is a central part of the methodology and software tools discussed below.

The methodology supports highly generic programming, and it has been implemented to take advantage of that capability. In the author's implementation, there are two macros, %importXML and %exportXML. These are available for free from the author.

With these two macros, XML-formatted data from a very broad class of user-defined XML vocabularies can be imported, exported, and processed in SAS without having to leave the base SAS environment. A significant advantage of %importXML and %exportXML over SI's XML tools is that they can always import and export user-defined XML without unfamiliar programming techniques. They work very much like XML analogs to INPUT and OUTPUT statements in a DATA-step. This is a significant simplification of XML processing in SAS.

XML IN THE DATA-STEP

The XML fragment below will be used to illustrate the indexing algorithm. XML statements begin with a start tag of the form `<TagName>` and end with a tag of the form `</TagName>`. The scope of a tag may completely include other tags and content: `<Tag00><Tag01>Information</Tag01></Tag00>`. In addition, a tag may have one or more attributes in its scope: `<Tag00 Attribute00="Stuff">`. Tags that do not contain other tags or content may be written as empty tags: `<Tag02 Attribute01="More stuff"/>`.

The XML fragment below has three tag names. `<SubjectData>` is in the scope of `<ODM>` and `<ItemData>` is in the scope of `<SubjectData>`. `<ItemData>` has two attributes with data: `ItemOID` and `Value`.

```
<ODM >
  <SubjectData>
    <ItemData ItemOID="PT" Value="P002"/>
  </SubjectData>
</ODM>
```

SAS INSTITUTE'S XML METHODOLOGY

This XML fragment could be readily imported into SAS with the XML engine of the LIBNAME statement using the map file below. The XML engine uses the map file to read the XML, create SAS datasets with map-specified attributes, and import data into the datasets, all as described above.

```
<SXLEMap>
   <TABLE name="ItemData">
      <TABLE_XPATH> /ODM/SubjectData/ItemData </TABLE_XPATH>
      <COLUMN name="ItemOID">
         <XPATH> /ODM/SubjectData/ItemData@ItemOID </XPATH>
         <TYPE> character </TYPE>
         <DATATYPE> string </DATATYPE>
         <LENGTH> 20 </LENGTH>
      </COLUMN>
      <COLUMN name="Value">
         <XPATH> /ODM/SubjectData/ItemData@Value </XPATH>
         <TYPE> character </TYPE>
         <DATATYPE> string </DATATYPE>
         <LENGTH> 200 </LENGTH>
      </COLUMN>
    </TABLE>
</SXLEMap>
```

This map file (written in XML itself from an SXLEMap schema) specifies the creation of a table, that is, a SAS dataset named ItemData with two fields, or columns. One field is ItemOID and the other is Value and each has the attributes given in the map file.

The non-SAS thing in the map is the specification of paths. This specification comes from the XML world. The Xpath specification of the World Wide Web Consortium (W3C) is the source. Xpath is the W3C recommendation on how to identify a specific place in an XML instance. Paths are used because, as pointed out above, XML files are hierarchies with parents, sibs, and descendants, and one locates a specific point in a hierarchy by specifying the path to it. As implemented in the XML map, the paths are text strings that can get very long and complicated.

For non-XML data, such as a coma-delimited file or some other text file being imported into SAS, data import would typically take place in the DATA-step and might be done with an INPUT statement, data transformations, validation operations, and merging with other data, instead of with a mapping file. The XML engine skips that initial DATA-step

and its familiar syntax and powerful programming capabilities and moves data immediately into a destination SAS dataset. These SAS datasets can, of course, go into the traditional DATA-step processing.

The XMLMAP processing with the XML engine is only for importing XML formatted data into SAS. It has no role in exporting XML and, as mentioned above, the information needed to export XML from the DATA-step is lost by the XML engine with the XML map.

A DATA-STEP XML METHODOLOGY

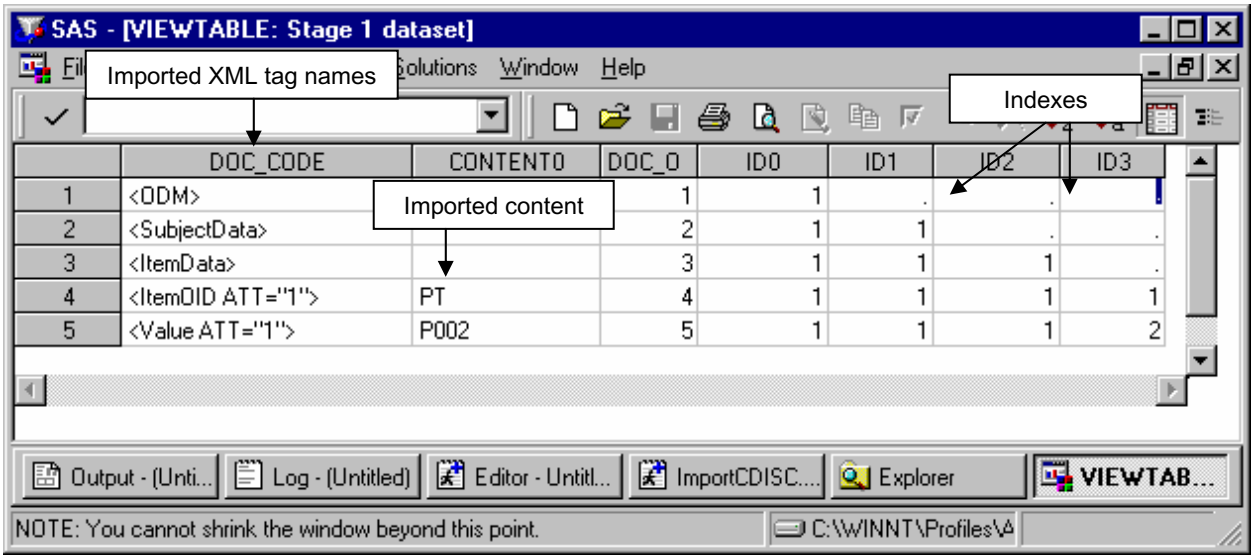
As discussed above, XML can be brought into and written from a DATA-step with the aid of an indexing algorithm for the XML hierarchy. For the XML fragment being used to illustrate XML processing, that indexing is straightforward, either manually or in a machine implementation.

To index this fragment, `ODM` is at the root, or ultimate, level so it has a value of 1 for the root level indexing variable. `SubjectData` is in the scope of `ODM` so it inherits a root level indexing variable of 1, and, in addition, since `SubjectData` is the first tag one level inside the root level, it has a second index variable with a value of 1. `ItemData`, inside the scope of `SubjectData`, inherits its index variables and gets one new one for itself. `ItemOID` and `Value` are attributes but, for purposes of indexing, they are treated just like tags inside the scope of `ItemData`. The XML fragment below shows the complete indexing.

```
1       <ODM >
1 1      <SubjectData >
1 1 1      <ItemData
1 1 1 1      ItemOID="PT"
1 1 1 2      Value="P002"/>
           </SubjectData >
         </ODM>
```

Looking at the indexed XML, it's straightforward to see how it could be stored in a SAS dataset. Figure 8 shows this XML instance, including the indexing, in a SAS dataset.

Figure 8. XML-image SAS dataset with index variables.



SIMPLE PROGRAMMING TECHNIQUES WORK

The SAS DATA-step code below shows how this XML imported into a SAS dataset can be processed using familiar DATA-step techniques and put directly into two fields, `NAME` and `VALUE`, in a SAS dataset. The SAS variable `content0` holds the imported data. For the ODM example, `content0` equals `PT` in the first line and `content0` equals `P002` in the second line.

```
/*<ItemData ItemOID=""/>*/
if ID01=1 and ID02=1 and ID03=1 and ID04=1 then NAME=trim(left(content0));

/*<ItemData Value=""/>*/
if ID01=1 and ID02=1 and ID03=1 and ID04=2 then VALUE=trim(left(content0));

output;
run;
```

In the next XML fragment below, a second `<ItemData>` element is added and indexed.

```
1       <ODM >
1 1      <SubjectData >
1 1 1      <ItemData
1 1 1 1      ItemOID="PT"
1 1 1 2      Value="P002"/>
1 1 2      <ItemData
1 1 2 1      ItemOID="SEX"
1 1 2 2      Value="m"/>
          </SubjectData >
        </ODM>
```

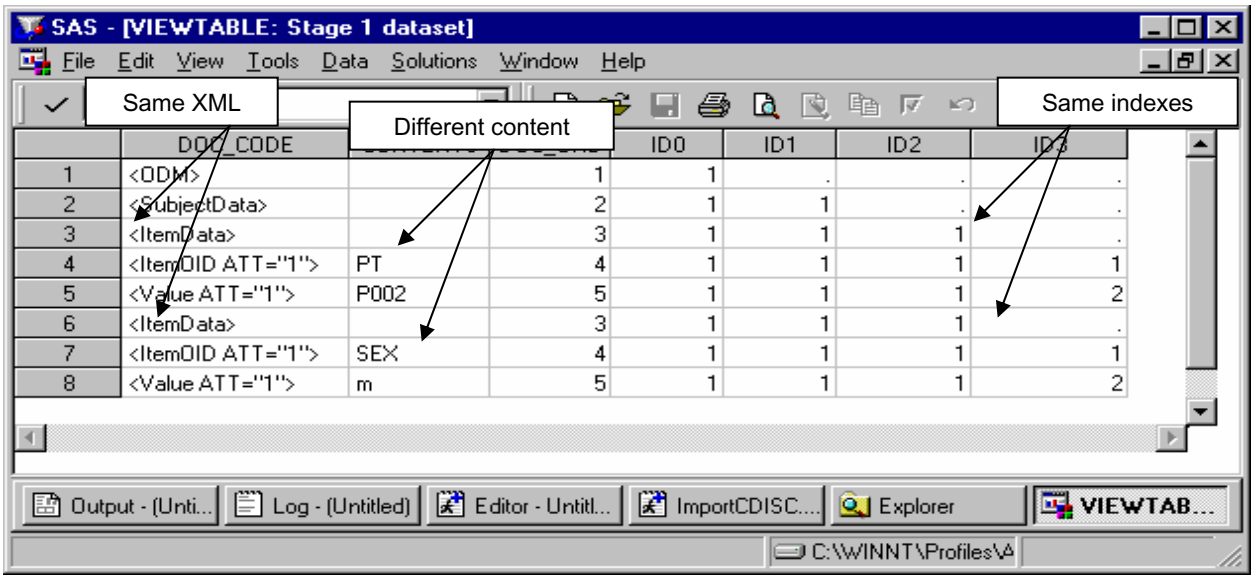
In order to process this new `ItemData`, the SAS code would also have to be modified to pickup the new 1 1 2 element. Obviously, a method of handling XML in SAS that required custom code to match the XML instance would not be useful.

The indexing algorithm has to recognize repeating XML structures and give them identical indexes, as below.

```
1       <ODM >
1 1      <SubjectData >
1 1 1      <ItemData
1 1 1 1      ItemOID="PT"
1 1 1 2      Value="P002"/>
1 1 1      <ItemData
1 1 1 1      ItemOID="SEX"
1 1 1 2      Value="m"/>
          </SubjectData >
        </ODM>
```

With both `ItemData` elements indexed identically, the programs becomes generic and more powerful. Figure 9 shows this XML imported into a SAS dataset.

Figure 9. Invariant indexing in the XML-image SAS dataset



A GENERAL SOLUTION

A general solution to the need for an indexing algorithm that is invariant to the XML instance, but depends on the underlying structure of the XML vocabulary, is called a canonical document (a candoc). The term candoc, the candoc approach, candoc technology, and implementation tools are called, in Zurich Biostatistics' terminology, Tekoa Technology.

The candoc for the XML fragment under discussion is, in indexed form,

```
1       <ODM >
1 1      <SubjectData >
1 1 1      <ItemData
1 1 1 1      ItemOID=""
1 1 1 2      Value=""/>
           </SubjectData >
         </ODM>
```

The candoc contains no content but defines the permitted tag names, attribute names, and relationships between all tags and attributes. It's a template to guide the import and export of an XML vocabulary. Using the candoc, every `ItemOID` with a path through the XML of `<ODM><SubjectData><ItemData>` will have exactly the same ID: 1 1 1 1, and will be processed correctly in the SAS code sample above. In other words, the code sample will work very much like a traditional INPUT statement in a DATA-step.

The indexed candoc enumerates all of the paths through the XML schema or DTD that it supports. The `DOC_ORD` numbers in Figure 10 show the path enumeration. To streamline the DATA-step programming illustrated above, `DOC_ORD` numbers can be used instead of the individual indexes,

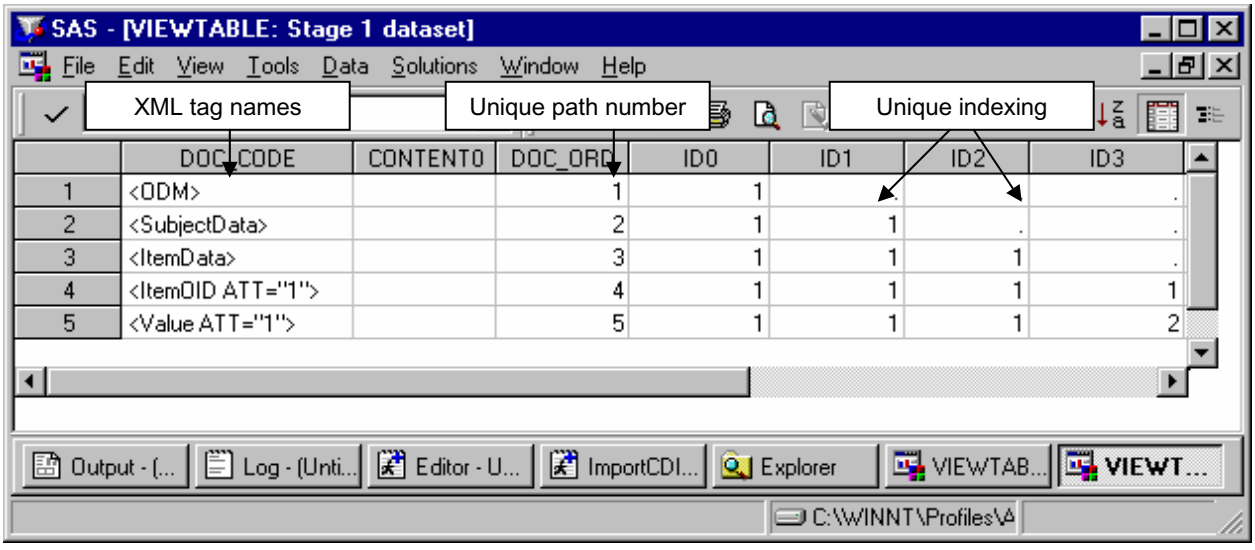
```
/*<ItemData ItemOID=""/>*/
if DOC_ORD=4 then NAME=trim(left(content0));

/*<ItemData Value=""/>*/
if DOC_ORD=5 then VALUE=trim(left(content0));

output;
run;
```

This brings the SAS programmer control of XML processing in the DATA-step but does not require knowledge of XPath. Since the %importXML tool automatically indexes any canonical document and automatically indexes any XML instance, the SAS programmer can use familiar DATA-step programming techniques, as shown above, to work with XML.

Figure 10. Canonical Document that guides invariant indexing for import and export.



A 100% BASE SAS SOLUTION FOR ALL XML

Candocs guide the import of XML into SAS and they guide the export of XML from SAS. ZBI's %exportXML tool uses a candoc to write XML from the kind of XML SAS dataset discussed in this paper. The XML export does not require ODS or Java or perl. Export, like import, is done completely in the base SAS DATA-step using familiar techniques.

Candocs, since they are completely under the control of the user, allow anyone who is comfortable with DATA-step programming to import and export any data-centric XML vocabulary.

The method has been in use successfully for several years and does support real life, complex, consortium-developed XML vocabularies such as the pharmaceutical industry's CDISC XML standard for clinical study data. The method has scaled successfully to XML files of over one million records.

XSLT-TYPE TRANSFORMATIONS

XSLT-type transformations can be done in the DATA-step using this candoc indexing approach. To transform, the index ID values are changed with DATA-step programming statements. Since the index ID values identify paths in XML, one instance can be transformed to another by changing the ID values and exporting the resulting XML with the appropriate candoc.

SUMMARY

XML's hierarchical, text-based, named content nature makes it cumbersome to import, export, and transform in the DATA-step.

SAS Institute's solution handles XML import and export as two separate cases.

XML import uses the LIBNAME XML engine with an auxillary mapping file. The mapping file, itself in XML, associates paths through the XML with destination SAS files. This by-passes the DATA-step and also bypasses the powerful and familiar DATA-step processing capabilities. Once in a destination SAS file, the imported data can be brought into the DATA-step.

For export to XML, SI's solution uses customized tagsets, PROC TEMPLATE, and ODS. These procedures are unfamiliar to many SAS programmers. Also, they can export only XML that itself has the row-column structure of SAS datasets. This is not really a general-purpose solution for XML export. Generic DATA-step writing of XML isn't part of the PROC TEMPLATE solution, either.

The various versions of SAS do support direct LIBNAME import and export of a SAS-specific XML schema and some standard non-SAS schemas. This support has varied from version to version of SAS.

An alternative to the LIBNAME XML engine for XML import and PROC TEMPLATE for export is to flatten the hierarchical XML into a numerically indexed representation that presents nothing new to the SAS programmer, unlike native XML. This method is the basis for the import and export of XML that takes place 100% in the familiar DATA-step. This DATA-step method has been implemented in a highly generic way, used with real life XML, and scaled up to production-size XML files of more than one million records.

CONCLUSION

XML and SAS can coexist nicely, for import of XML into SAS, export of XML from SAS, and transformation of one XML instance into another. This is true for any data-centric XML that a user may encounter.

SAS is a truly powerful tool because it combines superb data management capabilities with state-of-the-art statistical capabilities and ties it all together with a fully functional programming environment. In addition, SAS has tremendous cross-platform capabilities. This puts SAS-based systems in the middle of many heterogeneous information systems that involve complex human-machine and machine-machine interactions. XML has become ubiquitous in information technology because it is a vendor-neutral, platform-independent format for self-describing information. XML has become the preferred format for heterogeneous information systems. In other words, SAS and XML really have to work well together.

The author's experience, primarily in the pharmaceutical and medical worlds but also with some exposure to financial systems, is that SAS and XML can do better than just work well together. XML can integrate SAS into heterogeneous systems for truly remarkable results with the use of a functional toolkit that feels comfortable to SAS users.

REFERENCES

SAS Institute "Base SAS Community XML" http://support.sas.com/rnd/base/index-xml-resources.html (January 30, 2004)

James Clark and Steve DeRose "XML Path Language (XPath) Version 1.0" World Wide Web Consortium (W3C) http://www.w3.org/TR/xpath (January 30, 2004)

Zurich Biostatistics, Inc. "Presentations" http://www.zbi.net/NewFiles/leadership.html (January 30, 2004)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The DATA-step toolkit for working with XML in SAS (Tekoa toolkit) is available for free from the author. Contact the author at:

Michael Palmer
Zurich Biostatistics, Inc.
45 Park Place, So., PMB 178
Morristown, New Jersey 07960
Work Phone: 973-277-9034
Email: mcpalmer@zbi.net
Web: www.zbi.net