Paper TU10

XML for SAS® Programmers

Frederick Pratter, Computer Science/Multimedia Studies Program,
Eastern Oregon University, La Grande OR

ABSTRACT

XML (the e**X**tended **M**arkup **L**anguage) is an open standard for the definition, transmission, validation, and interpretation of data. The standard was developed by the Worldwide Web Consortium (W3C) in order to provide a simple and efficient way to manage self-documenting data files. SAS® Software includes a number of useful tools for creating and parsing XML data. These include:

- The *XML libname engine* is used to import and export documents in XML format into or from a SAS dataset, optionally using an XMLMap. The new XML92 engine now supports XML Maps on output as well as input.
- The *XML Mapper* application is a graphical interface to analyze the structure of an XML document or an XML schema and generate XML syntax for the XMLMap.
- The ODS MARKUP output destination is used to create XML from SAS output; ODS MARKUP creates but does not read XML documents.

This paper will introduce these concepts and present some brief examples of each, including some of the new features available in SAS 9.2.

INTRODUCTION

Increasingly, the Web browser has become the standard graphical user interface. Platform independent, universal, and free, the HTML browser is the default technology for collecting and displaying information. Unfortunately, this ubiquity has resulted in a number of unavoidable consequences for those of us who have to produce the software to support these activities. For one, everybody has had to learn HTML. The big advantage is that the playing field has been leveled—now we are all working with an equal handicap.

An important component of these new standards is XML, the eXtended Markup Language. This paper focuses on the SAS utilities that are available for parsing and generating XML. In the process, the following questions will be considered:

- what is XML?
- what is the difference between XML and HTML?
- what does XML do?
- what SAS tools are available for XML processing?
- what do they do, and why would you want to use them?

The main difference between XML and HTML is that they were designed with different goals in mind. HTML was intended as a way of displaying information. As such, it includes both the data and the presentation formatting. XML was designed to separate the data from the presentation; it contains only the data. It is important to recognize that XML is not a replacement for HTML. They are used for different purposes, and consequently it is likely that the two markup languages will continue to coexist peacefully.

XML BASICS

XML is a very simple idea really. Its primary purpose is the exchange of data between dissimilar systems. As such, it is an extension of earlier standards, such as comma delimited (CSV) files. A plain text file containing variable names on the first line and rows of data separated by commas has been a standard means for data transfer since the early pre-DOS days. The well known drawback of comma delimited files is that of course you cannot send text strings that contain commas. You can always surround the text with quotes, but then you cannot send strings containing quotes. XML makes it all so much easier, since you can send almost anything enclosed between opening and closing tags. (If you need to send the angle brackets “<” and “>”, the HTML *entities* < and > can be used.)

There is another, more important advantage of XML over CSV format. This is that the former can be used to structure hierarchical data, whereas CSV files must be flat tables. (This causes a problem for SAS, of which more later.) This is an immense improvement, since it allows data to be exchanged at any level of complexity. The receiving system

need know nothing whatever about the data, except that it is in XML format, since the document carries its own description with it.

The following example shows a portion of an XML document, based on one of the SAS sample datasets, SASHELP.CLASS:

```
<?xml version="1.0" encoding="UTF-8"?>
<class>
   <student id=”1”>
      <Name>Alfred</Name>
      <Sex>M</Sex>
      <Age>14</Age>
      <Height>69</Height>
      <Weight>112.5</Weight>
   </student>
   <student id=”2”>
      <Name>Alice</Name>
      <Sex>F</Sex>
      <Age>13</Age>
      <Height>56.5</Height>
      <Weight>84</Weight>
   </student>
      …
</class>
```

This little example illustrates most of the rules of XML. Note that XML document are simple text files, and can be e-mailed or displayed in a browser or even in `vi` or Notepad. The first statement is required, indicating that the document is in XML, that the version is 1.0 (actually, there is no other version), and that the character encoding is the Unicode 8-bit format, which is backwards compatible with ASCII character codes.

All XML documents must have a *root tag*; in this case it is `<class>`. Unlike HTML, every XML element must also have a closing tag. (An empty tag, that is one with no associated data, can be abbreviated, as for example `<br />` in HTML.) Thus the last line of the example is `</class>`, which closes the root tag. White space is ignored and elements can begin and end anywhere on the line.

Tags can be nested, but they have to be closed in the opposite order from the opening tags. Thus while

```
<h1><b>Hello World</h1></b>
```

may be valid HTML, it is illegal in XML.

Next, XML tags are case sensitive: `<message>` cannot be closed with `</MESSAGE>`.

XML elements can have attributes, just like HTML. However in XML the attribute value must always be quoted, for example `<student id=”1”>`. Attributes in XML are used to convey information that describes the elements; it is not always clear what should be an element and what an attribute.

VALIDATING XML DOCUMENTS

The rules for XML, if followed correctly, result in a document that is “well formed”. An XML document that is not well formed cannot be parsed—the application should return an error if, for example, a tag is not properly closed. But XML documents can also be *validated*. A valid XML document is one that corresponds to a specified *Document Type Definition*, or DTD. It is not necessary to have a DTD to use XML, but it adds a layer of data integrity checking that can be very useful. The purpose of a DTD is to specify the legal elements of an XML document. A DTD can be declared inline within the XML, or as an external reference.

The following example shows an external DTD for the XML shown above:

```
<!ELEMENT class (student)+>
<!ELEMENT student (Name,Sex,Age,Height,Weight)>
<!ATTLIST student id CDATA #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Sex (#PCDATA)>
<!ELEMENT Age (#PCDATA)>
```

```
<!ELEMENT Height (#PCDATA)>
<!ELEMENT Weight (#PCDATA)>
```

The Document Type Definition indicates which elements can appear inside a legal `<class>` element. The elements are further described as "parsed character" data, that is, as text that will be treated as markup. One of the elements, `<student>` has a required character attribute `id`. It would be possible to specify that in addition to being required, the `id` must also be one of the values from an enumerated list, but this was not done in the example. A DTD can be used to verify that the data you receive is valid, just as others can validate the data you send them.

In order to validate the sample XML document, it is necessary to save the DTD in some public location and then include in the XML document a reference to the DTD, as for example:

```
<!DOCTYPE note SYSTEM "class.dtd">
```

This indicates to the parser that the rules in the specified DTD must be observed. This is a `SYSTEM DTD`, indicating that it is located locally; the pathname above references a file in the same directory as the XML, which is likely to be the case. In addition, a number of standard DTDs have been developed by independent groups responsible for data exchange. These are public DTDs, as for example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

This Document Type Definition specifies legal HTML elements so that an HTML document can be validated as XML.

The problem with DTDs is that they are not themselves XML. *XML Schema* is an XML-based alternative to DTDs that describes the structure of an XML document in XML. The XML Schema language is also referred to as *XML Schema Definition* (XSD). Schemas are harder to write than DTDs, so they have been slow to catch on, but they offer a richer and more powerful syntax, so they are used for applications where validation is important, as in the CDISC standard.

A schema for the class document might be specified as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">
  <xs:element name="class">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="student"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="Sex"/>
        <xs:element ref="Age"/>
        <xs:element ref="Height"/>
        <xs:element ref="Weight"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Name" type="xs:NCName"/>
  <xs:element name="Sex" type="xs:NCName"/>
  <xs:element name="Age" type="xs:integer"/>
  <xs:element name="Height" type="xs:decimal"/>
  <xs:element name="Weight" type="xs:decimal"/>
</xs:schema>
```

Note that unlike the DTD, the schema (class.xsd) is an XML document. In addition, it more comprehensively describes the XML document. In particular, the schema requires the presence of an id attribute and specifies that the student element may occur more than once, but the name, sex, age, height and weight elements may appear only once per student. Clearly this is both more complex and more precise than the DTD.

SAS XML LIBNAME ENGINE

PARSING XML

There is a special set of standards for reading ("parsing") well-formed XML documents. These standards are embodied in a number of freely available programs for traversing XML documents to extract the information contained therein. Most of these are written in Java, although that is not at all necessary. It would certainly be possible to write your own XML parser in a SAS DATA step, but the general feeling is that given the availability of well tested ones, why bother? XML parsers are available for downloading from Sun, Microsoft and Oracle among other locations; a full list of sources is available at http://www.xml.com/pub/rg/XML_Parsers. In addition, SAS has provided a reasonably powerful set of tools for parsing and generating XML. The explanation of the XML libname engine below, explains how to parse documents in SAS.

The XML libname engine can be used both to import and export XML documents. That is, a SAS data set can be written out in XML, and, providing it is properly structured, an XML document can be read into a SAS dataset. The XML libname engine supports DTDs and schemas on output but not, unfortunately, on input; the engine just assumes that input documents are valid and well-formed.

Creating an XML document in SAS is a lead pipe cinch, as the following code illustrates:

```
LIBNAME xmlout XML "class.xml";
DATA xmlout.class;
     SET sashelp.class;
RUN;
```

This simple program creates an XML document similar to the one shown in the first example. The difference is that the libname engine does not create attributes; all of the variables from the SAS dataset are converted to elements in the XML. (There are some other issues as well, which are explored further in the section below on the XML Mapper applicaton.)

The XML libname engine also provides support for CDISC formats. The Clinical Data Interchange Standards Consortium (CDISC) develops XML standards for the pharmaceutical industry. To export a CDISC XML document using an XML libname you can use something like the following code.

```
LIBNAME output XML "ae.xml" XMLTYPE=CDISCODM FORMATACTIVE=YES;
DATA output.ae;
       SET odm.ae;
RUN;
```

You can also import a CDISC XML file using something very similar; for more information, see Palmer, 2002 at and the list of references at. In addition to the XML libname engine, support for CDISC is available as a separate procedure: PROC CDISC. In SAS 9.2 two formats are supported: the Operational Data Model (ODM) and the Study Data Tabulation Model (SDTM)*;* for more information on using PROC CDISC to read and write XML documents.

"Exporting an XML Document in CDISC ODM Markup", currently available in the *SAS® 9.2 XML LIBNAME Engine: User's Guide).*

"Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Format Type" in the *SAS® 9.2 XML LIBNAME Engine: User's Guide* at

USING XML MAPPER APPLICATION

Parsing XML in SAS is straightforward as long as the document follows the required structure (. The problem is that by nature, SAS datasets are tabular, that is the data are in rows and columns, but XML documents may be hierarchical. If your XML document does not meet these requirements, it is necessary to turn to transform it so that it does. This is accomplished by means of a separate document, called an *XML Map*. In order to simplify the process of constructing maps, SAS offers a utility called XML Mapper. This is not actually part of SAS, but rather a standalone Java program. A map is just a set of instructions for the libname engine that explain how to read or write data to a SAS dataset. In addition, it is the only way to write out a dataset containing attributes.

The XML Mapper application is used to pick and choose the variables and observations from the XML. SAS views an XML document stored in a single file as a library, so more than one map can be constructed from a single document, at different levels of aggregation. The resulting map is specified to the engine by the `xmlmap=` option on the libname statement.

The following example shows how it is possible to output the SASHELP.CLASS dataset as an XML document that includes an ID attribute, using the SAS XML92 libname engine (this is actually how the first example was generated):

```
FILENAME out 'student.xml';
FILENAME map 'class.map';

LIBNAME out XML92 XMLTYPE=XMLMAP XMLMAP=MAP XMLENCODING='UTF-8';
DATA out.students;
   SET SASHELP.class;
   id=_n_;
RUN;
```

The new XML92 libname engine provides a variety of useful options; here the `XMLTYPE=XMLMAP` indicates that a map is available to describe the output. The `XMLENCODING` option overrides the default value (the current session value, which is operating system dependent).

The map used for the above example was generated using XML Mapper; the syntax used is from the W3C XPATH standard.

```
<?xml version="1.0" encoding="windows-1252"?>
<SXLEMAP name="class" version="1.2">
    <TABLE name="student">
        <TABLE-PATH syntax="XPath">/class/student</TABLE-PATH>
        <COLUMN name="id">
            <PATH syntax="XPath">/class/student/@id</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>
        <COLUMN name="Name">
            <PATH syntax="XPath">/class/student/Name</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>8</LENGTH>
        </COLUMN>
        <COLUMN name="Sex">
            <PATH syntax="XPath">/class/student/Sex</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>1</LENGTH>
        </COLUMN>
        <COLUMN name="Age">
            <PATH syntax="XPath">/class/student/Age</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>
        <COLUMN name="Height">
            <PATH syntax="XPath">/class/student/Height</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>double</DATATYPE>
        </COLUMN>
        <COLUMN name="Weight">
            <PATH syntax="XPath">/class/student/Weight</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>double</DATATYPE>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

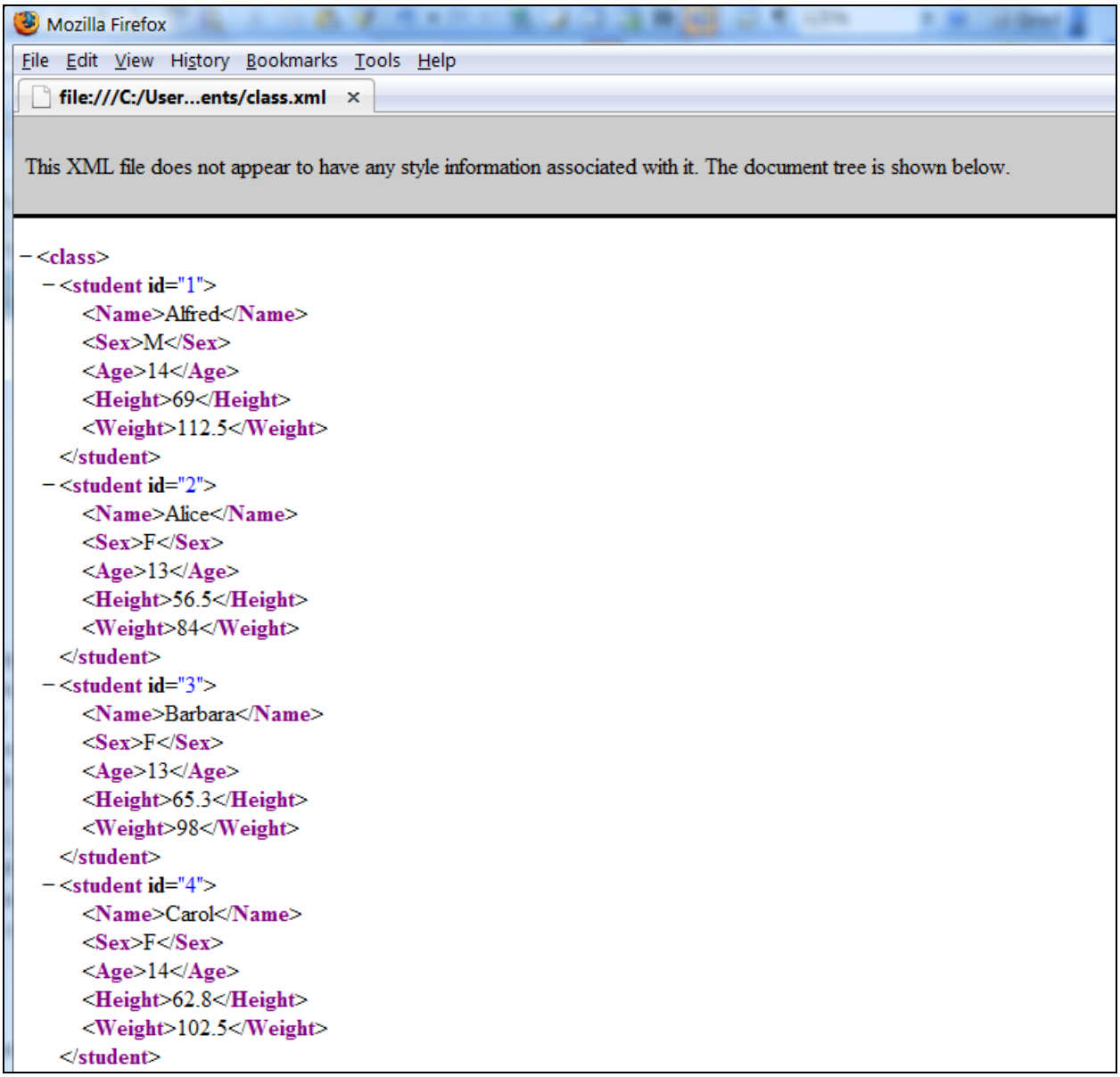
Note that the map is itself an XML document. It is also important to recognize that the SAS table name is specified as an attribute on the TABLE element. In this case it is “student” even though the name of the document (and the root element of the resulting XML) is “class. The data types and variable lengths are used for input, not output, but you will get a warning message if they are not correct.

FORMATTING XML DOCUMENTS

There is one more topic that is important for an understanding of how XML can be used, one which allows the introduction of two more acronyms-- CSS and XSL. *Cascading stylesheets* (CSS) were introduced in HTML 4.0 to specify document formatting. Style tags define how HTML elements are displayed. In this they are an extension of the `<font>` tag and `color=` attribute in earlier HTML versions.

While styles can be included in the document, it is more usual to save the styles in external files. In this way, a common "look and feel" can be maintained across a Web site. If all of the developers use a common set of style sheets, it is possible to change the appearance and format of all the pages in the site just by editing a single CSS documents. (They are called 'cascading" because if more than one stylesheet is used, each successive style definition will add new values and over-ride values previously defined.)

Microsoft Internet Explorer and Firefox both come with a built-in style sheet that can be used to display XML as a tree structure. In order to be able to view XML as anything other than the browser default, a link statement needs to be added to the XML, pointing to a custom stylesheet for that site. (Note that Firefox displays a warning message that it is using the default style for displaying the page):



Although it is certainly possible to use CSS for displaying XML it usually does not work very well. If you want more control over how the document is displayed, it is possible to use XSL, the eXtensible Stylesheet Language. This is a stylesheet language specifically developed for formatting XML for display.

XSL is generally conceded to be a lot of bother for very little marginal improvement over what you can do with CSS. The XSL transformation language (XSLT) on the other hand, has been widely adopted and continues to be a focus of considerable developer interest. XSLT is used, just like it sounds, to transform a XML document into something else, such as a different XML document, HTML, or even CSV. What is more, XSLT can filter and sort XML, address parts of an XML document, and output XML to different devices. SAS XML Maps are based on the XSL stylesheet language.

USING THE OUTPUT DELIVERY SYSTEM TO CREATE XML

For most users, the XML engine will provide sufficient functionality as is, at least for transferring data files. In addition, it is currently the only way (other than writing a custom DATA step) to read in XML documents. For more complex applications, and in particular for formatting procedure output, the Output Delivery System has been enhanced to support a wide range of output options.

XSLT represents the most sophisticated tool available for creating and manipulating markup documents. Most users will simply want to parse and or generate XML, and for that SAS has supplied simple and effective mechanisms in the XML libname engine and the ODS MARKUP output destination.

The Output Delivery System supplies a powerful and easy-to-use set of XML transformations that are far simpler to use than XSLT but which provide much of the same functionality.

SAS Release 8 introduced the `ODS XML` driver as an experimental driver, with no guarantee that the output would be valid XML. There was only one DTD available, which produced a document in a single standard format. With Release 9.1, SAS provided the new `ODS MARKUP` statement, allowing the user to export a variety of markup languages, including HTML, XML, CSV, DTD, CSS and XSL. The ODS MARKUP statement uses essentially the same syntax as the deprecated `ODS HTML` statement, except for the addition of `TAGSET=` option. The value of this option determines the type of output file to be created. (Again, it is important to note that ODS cannot be used to parse XML, only generate it.)

In addition to the list of tagsets available from SAS, it is also possible to create new ones, as well as customizing the SAS-supplied tagsets. The new TEMPLATE procedure is used to review, create, and customize tagsets. Creating an XML document using ODS is as simple as the following illustration:

```
ods listing close;
ods markup body='C:\My Documents\xml\class.xml';
proc print data=sashelp.class;run;
ods markup close;
```

In order for the recipient to validate the resulting XML, it is also possible to create the DTD at the same time as the XML, as the following example illustrates:

```
libname myfiles 'C:\My Documents\myfiles';
ods listing close;
ods markup body='C:\My Documents\xml\statepop.xml'
    frame='C:\My Documents\xml\statepop.dtd' tagset=default;
proc univariate data=myfiles.statepop;
    var citypop_90 citypop_80;
   title 'US Census of Population and Housing';
run;
ods markup close;
```

With XML it is necessary to have some kind of style sheet to be able to view this output as anything other than plain text. The obvious question arises, why go to all this trouble when ODS HTML will produce a satisfactory Web page?

The answer requires a digression into static versus dynamic HTML content. Using ODS HTML to output the results of a procedure produces a static HTML document. If you want to change the formatting it is possible to do so, but re-running the procedure results in a new document with the original format. A certain amount of customizing is possible, but the ODS HTML statement produces a default format each time it is invoked. The obvious workaround is to use a stylesheet. Each time the procedure is run, a new HTML document will be generated, but the formatting will come from the stylesheet and not the document. The same process is much more efficient in XML, however. Rather than generating a default format and then overriding it, an XML document by its nature contains no inherent display format. Depending on the complexity of the report, a CSS or XSL stylesheet could be used to produce the output result. The programmer need only rerun the SAS code to generate a new XML document every time. This turns out to be pretty straightforward, using some of the new SAS/IntrNet and AppDev studio tools, but that is a topic for another paper.

One more ODS topic that needs to be mentioned is the new ExcelXP tagset. SAS has provided a demo at http://support.sas.com/rnd/base/ods/odsmarkup/excelxp_demo.html; Vince DelGobbo will be presenting at this conference on "Creating AND Importing Multi-Sheet Excel Workbooks the Easy Way with SAS®." Base SAS software in Release 9.1 and later can be used to easily transfer data from SAS files and procedures to MS Office using ODS and the ExcelXP tagset. The SAS documentation includes a handy introduction "TEMPLATE Procedure: Creating Markup Language Tagsets" that explains how to customize and modify your own tagsets.

CONCLUSION

XML is the new industry standard for data transfer between dissimilar platforms. In addition, it is an alternative to HTML for data display, when used with custom stylesheets to create a formatted document by the client's browser. SAS has provided the XML libname engine for the first function, and has modified the Output Delivery System in order to accommodate the second. These features are new in the most recent SAS releases, and are the subject of ongoing development. This paper has reviewed the current level of functionality of these tools. As always, SAS has solicited user input in determining the best mix of features and usability. It is important for the user community to try to begin to integrate these tools into practical applications, in order to drive the next generation of SAS XML utilities.

REFERENCES

- SAS Institute Inc. 2009. SAS® 9.2 XML LIBNAME Engine: User's Guide. Cary, NC: SAS Institute Inc. http://support.sas.com/documentation/cdl/en/engxml/61740/PDF/default/engxml.pdf.
- Vincent DelGobbo. "Creating AND Importing Multi-Sheet Excel Workbooks the Easy Way with SAS®.". SAS Institute Inc. 2006. Proceedings of the Thirty-first Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc. http://www2.sas.com/proceedings/sugi31/115-31.pdf
- Cisternas, Miriam and Ricardo Cisternas. "Reading and Writing XML files from SAS®," SUGI 29 (2003), http://www2.sas.com/proceedings/sugi29/119-29.pdf.
- Friebel, Anthony. "XML? We do that!" SUGI 28 (2002), http://www2.sas.com/proceedings/sugi28/173-28.pdf.
- Gebhart, Eric S. "ODS MARKUP: The SAS® Reports You've Always Dreamed Of," SUGI 30 (2004), http://www2.sas.com/proceedings/sugi30/085-30.pdf.
- Hoyle, Larry. "Reading Microsoft Word XML files with SAS®," SUGI 31 (2005), http://www2.sas.com/proceedings/sugi31/019-31.pdf.
- Palmer, Michael. "XML in the DATA Step," SUGI 29 (2003), http://www2.sas.com/proceedings/sugi28/025-28.pdf.
- Pratter, Frederick. "Beyond HTML: Using the SAS System® Version 8.2 with XML," SUGI 27 (2001), http://www2.sas.com/proceedings/sugi27/p002-27.pdf.
- Pratter, Frederick. "Using the SAS® Output Delivery System and PROC TEMPLATE to Create XHTML Files," SAS Global Forum 2007, http://www2.sas.com/proceedings/forum2007/118-2007.pdf.
- Shoemaker, Jack N. "XML Primer for SAS® Programmers," SUGI 30 (2004), http://www2.sas.com/proceedings/sugi30/240-30.pdf.
- *CDISC Procedure for SAS® Software: Release 8.2 and Later (http://support.sas.com/rnd/base/xmlengine/proccdisc/TW8774.pdf)*
- *CDISC Procedure for the CDISC SDTM 3.1 Format* (http://support.sas.com/rnd/base/xmlengine/proccdisc/cdiscsdtm.html)
- http://www.cdisc.org/pdf/TU03.pdf
- http://www.cdisc.org/publications/articles_archive.html.
- http://support.sas.com/documentation/cdl/en/engxml/61740/HTML/default/a002592089.htm)