Paper 2-27

# Beyond HTML: Using the SAS® System Version 8.2 with XML

Frederick Pratter, Computer Science Department, University of Montana, Missoula MT

## INTRODUCTION

It is self-evident that the Web browser has become the standard graphical user interface for the 00s. 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. Furthermore, all of the clever interactive visual development environments we rely on are now obsolete, at least when developing for the Web. The IDE tools available for HTML development range from the primitive at best to the simply awful. The big advantage is that the playing field has been leveled—we are all working at an equal disadvantage.

Fortunately, SAS has risen to the challenge, introducing a number of new tools, including the Output Delivery System, SAS/IntrNet®, and the AppDev studio, that can make the programmer's life easier. A full review of ODS, webAF, htmSQL, and the Application Dispatcher would be beyond the scope of what can be covered in a one hour presentation. In this paper, I want to address one specific aspect of Web development using SAS, focusing on the SAS utilities that are available for parsing and generating XML. In the process, I want to address the following questions:

- 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?

There are a number of excellent papers available on using SAS with XML, from past user group meetings (Barnes, 2000; Kent, 2001) and on the sas.com® website (SAS, 2001b,c,d). There are at least four papers on using SAS with XML at this conference, including a Hands-on Workshop presented by Jack Showmaker and Greg Nelson. SAS also also offers a one-day Level IV course in "Using the Output Delivery System to Create XML Files."

In this paper, the focus will be less on the *how-to* of using XML and more on an overview of what XML does and why you, the over burdened SAS programmer, needs to learn yet one more set of TLAs.

## BASICS OF XML

XML (E**x**tensible **M**arkup **L**anguage) is a very simple idea really. Figure 1 shows an example of self-documenting XML, loosely borrowed from Paul Kent's SUGI 26 presentation.

```
<?xml version="1.0" ?>
<enclosingTag>
   <Talk>
       <Name>Paul</Name>
       <Title session="Advanced Tutorials">
          XML and SAS</Title>
       <When>Tuesday. 3pm.</When>
       <Where>hope to find out </Where>
     </Talk>
     <Talk>
       <Name>Paul</Name>
       <Title session="Forums">
         Futures Forum</Title>
       <When>Wednesday. 5 pm.</When>
       <Where>Room 220</Where>
   </Talk>
</enclosingTag>
```

**Figure 1. Sample XML**

This example illustrates most of the rules of XML. First, unlike HTML, every XML element must have a closing tag. (An empty tag, that is one with no associated data, can be abbreviated as `<Name/>.`) Tags, can be nested, but they have to be closed in the opposite order from the opening tags. Thus while

```
<h1><b>Title</h1></b>
```

may be valid HTML, it is illegal in XML.

Next, XML tags are case sensitive: <Message> cannot be closed with </message>. Furthermore, all XML documents must have a root tag; in this case it is `<enclosingTag>`. XML elements can have attributes, just like HTML. However in XML the attribute value must always be quoted, for example `<Name id="01234"> Paul</Name>`. 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.

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.

The primary purpose of XML is to exchange data between different systems. As such, it is only a modest 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 anything (except of course the symbols '<' and '>') enclosed between opening and closing tags. (If you need to send angle brackets, the HTML entities &lt; and &gt; 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 metadata with it. XML can even be used to store data as well, although data retrieval and transformations are somewhat more difficult than for database tables.

## Validating XML Documents

The rules for XML, if followed correctly, result in a document that is described as "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 in Figure 1.

```
<?xml version="1.0" ?>
<!DOCTYPE talk [
        <!ELEMENT talk (Name, Title, When,
        Where)>
        <!ELEMENT Name (#PCDATA)>
        <!ELEMENT Title (#PCDATA)>
        <!ELEMENT When (#PCDATA)>
        <!ELEMENT Where (#PCDATA)>
        <!ATTLIST Title session CDATA #REQUIRED>
]>
```

**Figure 2. Sample DTD**

This Document Type Definition indicates which elements can appear in a legal "Talk" document. The elements are further described as "parsed character" data, that is, as text that will be treated as markup. One of the elements, "Title" has a required character attribute, "session". It would be possible to specify that in addition to being required, session must also be one of the values from an enumerated list, but this was not done in the example. Clearly, a DTD can be used to verify that the data you receive is valid, just as others can validate the data you send them. A number of standard DTDs have been developed by independent groups responsible for data exchange. For example, in the insurance industry, the XMLife standard has been created to allow insurance companies and other interested parties to exchange information about policies and beneficiaries.

## Parsing XML

There is a special set of standards for reading ("parsing") and generating well-formed XML documents. These standards are embodied in a number of freely available parsers, that is, programs for traversing XML documents to extract the information contained therein. Most of these are written in Java, although that is not strictly necessary. It would certainly be possible to write your own XML parser, but the general feeling is that given the availability of well tested ones, it would be a bad idea. Parsers come in two flavors (DOM, the **D**ocument **O**bject **M**odel, and SAX, the **S**imple **A**PI for **X**ML). The difference between these is well beyond the scope of this paper, but a useful introduction to XML parsers can be found at www.ibiblio.org/xml/slides/gsdc/fundamentals. XML parsers are available for downloading from Microsoft, Oracle, as well as from the Apache Software Foundation. Currently, the best Java parser is probably JDOM, which combines the most useful features of SAX and DOM, and is available from www.jdom.org/downloads. You could write a DATA step program to read or write XML (see Barnes, 2000), but SAS has provided a reasonably powerful set of tools for parsing and generating XML, so why bother?

## 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 TLAs: CSS and XSL. **C**ascading **S**tyle**S**heets are used to display HTML elements. Stylesheets were introduced in HTML 4.0 to specify formatting for documents. 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.)

Note that Microsoft Internet Explorer 5.5 comes with a built-in style sheet that can be used to display XML as a tree structure. Netscape 6 also can display XML, although it uses a different style sheet. In order to be able to view XML as anything other than the browser default, an HTML LINK statement needs to be added to the XML, pointing to a custom stylesheet for that site.

Although it is certainly possible to use CSS for displaying XML, there is one disadvantage, which is that CSS requires the limited set of tags that are defined in HTML. Because XML does not use predefined tags, it is necessary to have a way to indicate how the document is to be displayed. If you want the document illustrated in Figure 1 to show up as a table it is necessary to use XSL, the e**X**tensible **S**tylesheet **L**anguage. This is a stylesheet language specifically developed for formatting XML for display. It 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. In the Output Delivery System for 8.2, SAS has supplied 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.

XSLT and ODS represent the most sophisticated tools available for manipulating XML. Most users will simply want to parse and or generate XML, and for that SAS has supplied a simple and effective mechanism in the XML engine.

## SAS XML LIBNAME ENGINE

Starting with SAS Release 8.1, the XML libname engine can be used to import and export XML documents. That is, a SAS data set can be easily written out in XML, and, providing it is properly structured, an XML document can be read into a SAS dataset. Creating an XML document is rather difficult in Java, and somewhat tricky in Oracle. In SAS it is a lead pipe cinch.

```
libname oracledb ORACLE
       user=scott
       password=tiger
       path='sample';

libname xmltrans XML
  ‘c:\My Documents\XML\all_users.xml’;

data xmltrans.all_users;
     set oracledb.all_users;
run;
```

Note that the input library in this case is Oracle—it could be SAS, MS/Access, or anything else that SAS offers a libname engine for. Right now (Version 8.2), SAS does not yet validate XML using DTDs, but it seems like this can only be a matter of time.

Parsing XML can be just as simple, as long as the document follows the required structure (see SAS, 2001d):

- The root enclosing element (top-level node) of an XML document is the document container. For SAS, it translates to a **library**.
- The nested elements occurring within the container (repeating element instances) [must] begin with the second-level instance tag.
- The repeating element instances must represent a rectangular organization. For a SAS data set, they become a collection of **rows** with a constant set of **columns**.

As of Version 8.2, if your XML document does not meet these requirements, it is necessary to turn to XSLT to transform it so that it does. However, SAS has promised that a future release will contain extension syntax that will allow the engine “to map XML tags into data sets (tables), columns, and rows. The new syntax contains elements of the W3C Xpath specification, which describes how to locate and access specific elements in an XML document.” (SAS, 2001d)

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 in version 8.2 has been enhanced to support a wide range of output options.

## USING THE OUTPUT DELIVERY SYSTEM TO CREATE XML FILES`

Version 8 introduced the ODS XML driver as an experimental driver, with no guarantee that the output would be valid XML. SAS There was only one DTD available, which produced a document in a single standard format (see the SAS R & D web site at www.sas.com/rnd/base/topics/odsxml/0005a.htm#.xml.v801xmldtd).

Release 8.2 provides the experimental ODS MARKUP statement. This allows the user to export a variety of markup languages, including HTML, XML, CSV, DTD, CSS and XSL (see SAS, 2001c). 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. The documentation for doing this has recently (6/22/2001) been made available at www.sas.com/rnd/base/topics/odstagsets. Users who are interested in pursuing this topic are encouraged to get in touch with the developers at ods@sas.com.

Creating an XML document using ODS is as simple as the following illustrates:

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

To view the resulting XML, see
www.sas.com/rnd/base/topics/odsmarkup/class.txt.

In order 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;
```

To view the resulting files, see
www.sas.com/rnd/base/topics/odsmarkup/statepopxml.txt
and
www.sas.com/rnd/base/topics/odsmarkup/statepopdtd.txt.

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.

## 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

"ODS HTML With Cascading Style Sheets." SAS: Cary, NC. 2001a. www.sas.com/rnd/base/topics/odscss.

"ODS XML Primer." SAS: Cary, NC. 2001b. www.sas.com/rnd/base/topics/odsxml.

"Using ODS to Export Markup Languages." SAS: Cary, NC. 2001c. www.sas.com/rnd/base/topics/odsmarkup.

"Why Can't SAS Import My XML Document?" SAS: Cary, NC. 2001d. www.sas.com/rnd/base/topics/sxle82/prod82.

"XML, ODS and The Markup Language" SAS: Cary, NC. 2001e. www.sas.com/rnd/base/topics/templateFAQ/xmlf.htm

Kent, Paul. "SAS Takes Advantage of <XML>". Formal Demo SUGI 26. SAS: Cary, NC. 2001. www.sas.com/usergroups/sugi/sugi26/presentations/xmlatsas.zip

Nelson, Greg Barnes. "XML and SAS: An Advanced Tutorial." Paper 13-25. *SUGI 25 Proceedings*. SAS: Cary, NC. 2000.

## CONTACT INFORMATION

Frederick Pratter
Computer Science Department
University of Montana
Missoula, MT 59812
pratter@cs.umt.edu