Paper 13-25

XML and SAS®: An Advanced Tutorial

Greg Barnes Nelson, STATPROBE Technologies, Cary, NC

ABSTRACT

One of the goals for SAS applications developers has been to develop three-tier and n-tier applications where the application logic (business rules) is separate from the data, which, in turn, is isolated from the user interface. In a previous paper (Barnes Nelson, 1999)[1] we discussed how to implement this logic separation using the SAS Component Language. This paper extends that line of thinking by introducing SAS developers to XML**. eXtensible Markup Language**, or XML, is a protocol of sorts that can be described as a technique for separating data from its presentation. In this paper, we will discuss XML in the context of SAS applications and how it can be used in the preparation and presentation of data. We will explore some of the features of XML that makes it a good partner for SAS-based applications.

INTRODUCTION TO XML

XML has been referred as the next ASCII; kind of like HTML; a boon to corporate information exchange and technologies' next savior. This paper is an attempt to look beyond the hype and begin to understand the ideas behind XML, its purpose and implications for technology and business improvements. There have been 100's of articles written about its benefits, and yes we'll cover some of those here as well, however the focus of this paper is to discuss the role of XML in SAS applications. Specifically, we will discuss XML and related TLAs[♣]; the benefits of XML; common tasks that can benefit from an XML architecture; and wrap it all within the context of SAS. By the time we are finished here, you should benefit from the practical uses of XML and the SAS code that we used to produce them.

DEFINITIONS

As people are introduced to new technology, it is often confusing when terms are introduced before they are defined. Yet, when definitions come before they are used in context, the full benefit of the term is not realized. Throughout this paper, we will do our best to define terms within the context that they are used. We will also provide an annotated glossary at the end which, not only explains the term or phrase, but identifies its context and implications, if any. If you see a term in **bold** text, you can rest assured that we will provide greater depth to its meaning in the glossary at the end of this paper.

An obvious place to start in a paper about XML is with XML itself. XML stands for **eXtensible Markup Language**. XML is a language that is used to create other "languages". Many people have taken XML, for example, and created a standard "language" to describe an industry vocabulary (see for example, BizTalk.com and HRMML).

RELATIONSHIP TO SGML

XML has its roots in a more complicated **meta-language** called **SGML**, or the Standard Generalized Markup Language. SGML was formally approved as a standard in 1986 but had been in use at IBM for several years in the production of their technical publications. SGML is the father of another widely used standard that propelled the Internet into widespread adoption: **HTML**. SGML is a language that publishers, technical writers and library automation personnel have been using to create "documents" such as museum catalogs, technical publications and product catalogs from manufacturing specifications.

SGML is, by definition, a markup language. **Markup** refers to the additional information that is added to the text of a document to enrich either its meaning or presentation. In word processing packages, for example, markup is used to control the way information is presented (or how it looks when viewed or printed.) But we don't always have to use markup to control the way the document is presented. In the case of XML, markup is used to describe the actual contents – that is, to enrich the data by describing its use, context or definition.

RELATIONSHIP TO HTML

If we remember back to our first HTML lesson, you will recall that if you wanted to display the words "Hello World" in a browser, you had to markup the contents of the HTML with special tags.

```
<HTML>
   <HEAD>
       <TITLE>My First HTML Document</TITLE>
   </HEAD>
   <BODY>
       <P>Hello World</P>
   </BODY>
</HTML>
```

We needed that much HTML simply display the text "Hello World". We soon realized that by adding additional **tags**, we could change the characteristics of the text – making it bigger, bolded, blinking, and blue. The challenge that faced content providers soon became – "how do we write this simple passage without having to know the nuances of the markup language"? – the solution: the HTML editor. Not yet satisfied, we wanted it more interactive – the solution: JavaScript. Ah yes, now we need someone else to write the code – the solution: SAS HTML formatting macros. Faster, better, easier! – the solution: SAS/IntrNet (Application Dispatcher & htmSQL) and Java Server Pages.

HTML IS OLD, XML IS COOL!

HTML has and will continue to serve a very important role in delivering information across the globe. HTML won't go away anytime soon, just like printed books will not disappear. We accept HTML for the gifts that it has brought us – displaying static information on our web browsers. But what about my cell phone or PalmPilot™ - my webTV or web-enabled refrigerator? What about getting real data like stock quotes, movie times and SAS output! XML was designed from the ground up to help us describe, transfer and deliver data. Similar to its cousin, HTML, XML is a markup language. XML's markup doesn't tell us how the data should be presented, rather it tells us what it is, how it can be used – its role is to describe the content, rather than how to display it.

The Problem with HTML

HTML was designed primarily for delivering information over the web. It is, at its roots, a language used to describe what information will look like when rendered through a web browser. Both the content and formatting of the information is tied together in the HTML language tags.

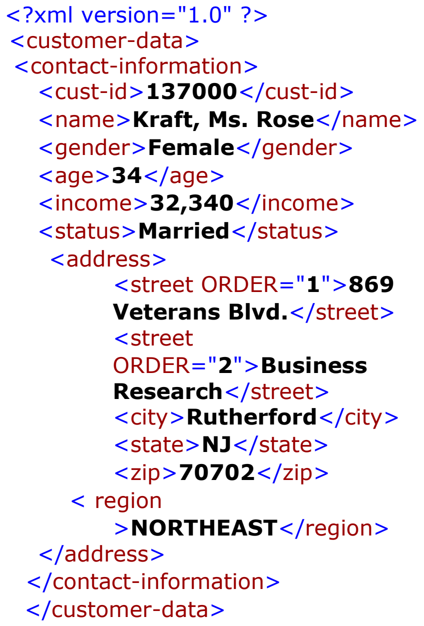
In HTML, we have a finite number of tags that we can use to markup our documents. These tags are used for controlling how the document should be presented. For example, in our previous example, we could have made the text "Hello World" bolded by added the <B> tag.

```
<P><B>Hello World</B</P>
```

In XML we are not limited by the number of tags that someone else has thought of to describe our data – that's our job! We use the tags as we see fit to describe the content and context of the data. For example, the following is XML document describes the first observations from some data that we may have about our customers.

[1] Barnes-Nelson, G.S. (April, 1999) Extending the Life of Your AF Application: Exploiting the Model-Viewer Paradigm. Invited paper at the annual convention of the International SAS Users Group (SUGI), Miami Beach, FL.

[♣] Three letter acronyms ✪



```
<?xml version="1.0" ?>
 <customer-data>
  <contact-information>
     <cust-id>137000</cust-id>
     <name>Kraft, Ms. Rose</name>
     <gender>Female</gender>
     <age>34</age>
     <income>32,340</income>
     <status>Married</status>
      <address>
             <street ORDER="1">869
             Veterans Blvd.</street>
             <street
             ORDER="2">Business
             Research</street>
             <city>Rutherford</city>
             <state>NJ</state>
             <zip>70702</zip>
         < region
             >NORTHEAST</region>
     </address>
   </contact-information>
   </customer-data>
```

XML Document 1. XML document produced with a SAS DATA STEP.

This XML document was produced dynamically from a SAS DATA Step[2]. But notice that instead of the usual tags like <B>, <P>, <A HREF> and so on, we have custom tags that we have used to describe our data. The first line tells us that we are dealing with an XML document. The second line has a tag called <customer-data>, which tells us that we are dealing with our customers. Within this, we find that we have some customer information, designated with the <customer-information> tag. Within each customer, we have collected a variety of information – for example, their name <name>, gender <gender>, income <income>, marital status <status>and another section that contains their address <address>.

Although not a terribly complicated example, we have exposed one of the key benefits of XML: simplicity. This document represents a hierarchy of information with easily understood patterns. Because of this simplicity, both humans and the computers can access it, understand it, and translate it into useful information.

In our example, the hierarchy dives only 3 levels deep: Customer Data, one or more customers (customer-information) and one more level for address information. There is no reason that an XML document could not represent a complex hierarchy with multiple, nested levels of information. We will explore a more complicated example later in this paper where we pull information from multiple tables to create a complete customer history profile.

INFORMATION TECHNOLOGY CHALLENGES

Despite incredible advances in technology, there are some persistent challenges that face us as technologists trying to solve real-world business problems.

XML will likely not be the technology that saves us from painstaking processes to make our data cleaner, more accessible or provide a richer context for information and its delivery across the web or across the room. However, if applied appropriately, XML can help solve some common barriers to productivity.

Multiple Views of the Same Data

A common challenge faced by many organizations requires data to be formatted differently depending upon its use. For example, account history or a customer's profile may be generated for the Customer Service department in order to interact with customers on the phone. The Finance department, however, may require a different view of the data in order to invoice the customer. The Sales and Marketing departments need yet a different view of the customer – requiring both granular data for each customer as well as highly summarized data for database marketing (buying history and cross-selling campaigns). Each department requires a similar, but different view of the data. In addition, data for each of these applications may be housed in different systems and defined somewhat differently.

XML can help us by providing a framework for understanding the customer in terms of a patterned hierarchy -- essentially giving us a common definition of a "customer". By defining a standard such as this, departments can exchange information about a customer easily and quickly -- regardless of where and how the data was stored. Additions or modifications to the source data have little impact on the XML document if care is taken to retain the way that the customer is defined according to the XML document. Because the data is separated from the way that it is presented, any changes in process or business rules, wont cause the systems to break -- especially when those systems cross-organizational boundaries. In addition, new views or representations of the data that are required can be generated without changing the underlying XML data.

We will explore later different methods to render or display XML data, but one of the benefits of XML is that once data has been delivered to the client application, it can be manipulated, transformed and presented in a variety of ways – all without having to request the XML document from the server a second or third time.

Application Integration

Implicit in the first point above is the idea that data can be integrated from disparate sources and/ or multiple applications. In our case, for example, we may have data that is housed in one or more source systems (e.g., ERP systems, billing systems, Sales-Force Automation, database marketing/ data mining databases). Despite this "separateness", data from these diverse sources and/ or applications can be brought together using a common meta-language that defines our customer. In our fictitious customer application, we have various applications connected over a network. When one application wants to access information from another application, an XML-formatted document is sent across the wire to the requesting application.

Because the definition of customer has been defined as a standard, information about the customer can be exchanged among companies much more easily as the mechanism for data interchange doesn't rely on, nor expose, the internal business systems. Data can be sent, for example, to a partner with only parts of the "customer" that they care to have the partner see. Invoicing might be shared with your billing supplier or the customer directly through the web, e-mail or other electronic means (PDAs, Cell Phones, etc.).

Information Optimization

Another business challenge that is often faced is the assimilation of huge amounts of unstructured data into meaningful context. As humans, we can process data very efficiently when the data doesn't appear to have a pattern to it. In our customer application, we may want to include all sorts of information about our customers from diverse data sources outside of our firewalls. Take for instance the following scenario:

As we cruise the web, we find an article about a new product that a potential customer is developing. Our company creates products and services that would be a good fit for this new potential client. As we read this article, we can parse the text, assimilate its meaning and make judgments about what we have read. Next, we decide to get an independent assessment of their company – we request information from Dunn & Bradstreet that will show us how they have done financially, who holds senior management positions, where they have offices, etc.

A trip to their web site affords us an opportunity to get another perspective: to get a sense of their culture – from the words they use to describe their company to the opinions they have about their own products. Finally, we incorporate some of what we have learned about this company into our own Sales Forces Automation system so that our sales representatives will be more knowledgeable as they interact with this new potential customer.

XML to the Rescue

Much of what we have described above can be best characterized

[2] See Appendix A for an example of producing XML using a SAS Data Step.

as unstructured data. A contemporary solution to this problem would be a Knowledge Management system. A non-technology solution would be to create new positions whose job would be to "surf and assimilate". But XML does offer some key advantages to this problem.

Smart Agents. Suppose instead of you sitting behind the terminal searching for documents, you tell the computer the kinds of things you were interested in and have a computer do the work for you – filtering, cataloging and storing the retrieved information. The XML paradigm keeps the information separate from the presentation rules, allowing for intelligent agents to scan through a document's content and ignore the style sheet if one is present.

Meaningful Searches. HTML-based search tools use keywords and text to manage the information about the millions of web sites in existence. XML-based search tools, however, use the inherent data structure in the XML documents themselves as well as the meta-data contained both in the XML document as well as the **Document Type Definition** (DTD). Given the amount of information on the web, technology that gives us more precise control over searches should help sift through information more cleanly than before. If we were to conduct a search on the web today – say on SAS – we would get between 268,000 and over a million hits depending on the HTML-based search engine we used. The topics returned range from SAS Institute to Scandinavian Airlines to Surfers Against Sewage. Because XML provides a context for our searching, we could specify SAS in the context of <company>, <software> or other relevant elements.

Granular Updates. Since the structure of an XML document is a known hierarchy of information, we are able to update information by sending only parts of the document each time there is a change. By using this feature of XML, we don't have to resend the entire hierarchy to the requesting application.

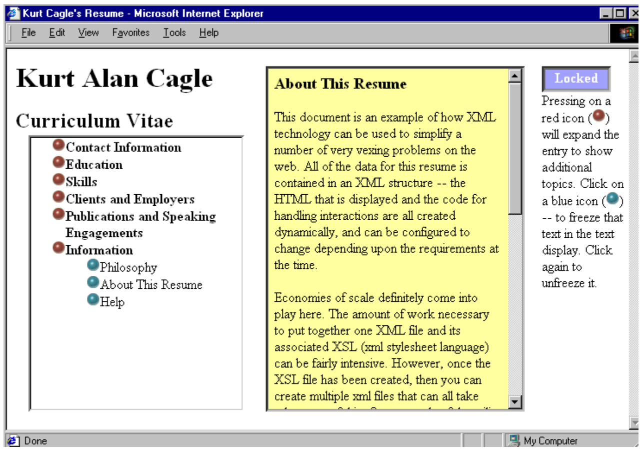
Technology Optimization

At the time of the writing of this paper, there are literally billions of pages of information available on the web on over millions of web sites globally. Most of the web pages are written using HTML. As we have learned earlier, HTML is a great tool for constructing documents to be displayed over the web. As you surf the web, you request a document from a web server using a Uniform Resource Locator (URL). Assuming the document can be located on the specific server, the page is downloaded to your browser for rendering. Once downloaded, the communication between your machine and the web server is essentially broken. That is, the relationship between the browser and the server is connectionless.

Continuing with our customer scenario, let's assume that we have downloaded a table of information that lists all of our customers, what products they have purchased, how much they have spent with us, where their home office is, who their sales rep is and so on. If we wanted to sort the information by any of the data that we have in the table, we would have to send a request back to the web server to have it re-processed and re-rendered. This approach places a tremendous burden on the web server to handle fairly simple requests. A more efficient approach would be to have the client machine handle the local manipulation of the data where it could be sliced-and-diced, sorted, filtered and rendered differently.

By combining XML and XSL (**eXtensible Stylesheet Language**), we can achieve this goal of local computation and manipulation (more on XLS later.) Once an XML document is downloaded, we can achieve this level of interactivity on the client side with a single request from the web server. Figure 1 shows an example of an interactive document that is built entirely of XML data delivered to the client by a single request to the server. For more information on this example, refer to the article entitle "Transform Your Data with XSL" found at

http://www.xml-zone.com/articles.asp



Kurt Alan Cagle

Curriculum Vitae

- Contact Information
- Education
- Skills
- Clients and Employers
- Publications and Speaking Engagements
- Information
  - Philosophy
  - About This Resume
  - Help

About This Resume

This document is an example of how XML technology can be used to simplify a number of very vexing problems on the web. All of the data for this resume is contained in an XML structure -- the HTML that is displayed and the code for handling interactions are all created dynamically, and can be configured to change depending upon the requirements at the time.

Economies of scale definitely come into play here. The amount of work necessary to put together one XML file and its associated XSL (xml stylesheet language) can be fairly intensive. However, once the XSL file has been created, then you can create multiple xml files that can all take

Pressing on a red icon (●) will expand the entry to show additional topics. Click on a blue icon (●) -- to freeze that text in the text display. Click again to unfreeze it.

Figure 1. Dynamic XML application.

In addition to the flexibility gained through local processing and manipulation, we can realize efficiency gains by off-loading the requests from the server. As the above example shows, data can be translated from this interactive version into a print version or XML only version so others who can process this XML can incorporate the data into their applications. In fact, there is a movement to unify the format of resumes so that human resources information can be shared universally (see the Human Resources Management Markup Language in the glossary for more information.)

UNDERSTANDING THE XML LANDSCAPE

So what? We have an XML document that is self-describing. We still can't print invoices, e-mail with it or build applications with it, right? What we have discovered thus far in our exploration of XML is the simple case of the XML document. The beauty of an XML document is that it can be *written*, *read* and *rendered* by countless applications. These three, oversimplified, tasks help define some of the technologies that surround XML. Before we discuss these three tasks and how to perform these tasks with the SAS System, it is important to understand the rules that govern XML documents.

Rules to Live By

Unlike HTML, XML has a rigorous set of rules that govern how a document should be constructed. A well-formed XML document is one that has all of the characteristics that it is supposed to have. There are a few simple rules about how an XML document should be constructed.

✓ Beginning and ending tags must match. That is, you cannot have an <ADDRESS> tag without one that ends the expression </ADRESS>.

✓ Elements can be nested within each other, but they cannot cross boundaries. For example, we can have

```
<Customer>
     <ID>109</ID>
     <Name>Greg Barnes Nelson</Name>
</Customer>
```

but not:

```
<Customer>
     <ID>109
     <Name>Greg Barnes Nelson</ID>
     </Name>
</Customer>
```

✓ XML tags are case sensitive. Each of the following, for example, is considered a different element.

```
<Customer> <CUSTOMER> <customer>
```

✓ Because each element must have a beginning and ending tag, empty elements are signaled by either a closing tag or a />. These two lines are equivalent:

```
<Customer/>
<Customer></Customer>
```

✓ Just like in HTML, there are some reserved characters that cannot be used. These include:

<	&lt;	&	&amp;
>	&gt;	"	&quot;
`	&apos;		

Each XML document must have a root element that denotes the top of the hierarchy or root. In our example <customer-data> represents the root element and cannot occur anywhere else in the document.

Assuming an XML document follows all of these rules, browsers or other parsers that can read XML are guaranteed to be able to read your document. This is one of the primary differences from HTML. Because all of these rules are adhered to, our XML document is said to be well-formed.

In addition to being well-formed, documents that have something called a **Document Type Definition** (DTD) and are well-formed are **valid**. Although it is not necessary that a document be valid (i.e., have a DTD), is it often useful to document them in a DTD so that other people or applications can benefit from knowing how they are supposed to be used.

A DTD essentially describes the document's rules – that is, which elements are present and the relationship among the elements. DTDs, although optional, help to validate the incoming XML document when the application doesn't have a built-in definition of the XML document.

Writing XML

As SAS applications developers, one of our primary tasks in the near future will be to learn how to create an XML document. Just as we have learned how to write HTML from our SAS applications, XML documents can be created programmatically using the SAS language.

In SAS Version 8.0 (M01), there exist several lightly documented methods for writing XML natively. We can write XML from a Data Step (see Appendices A and B for examples), from output that we generate using SAS' Output Delivery System (ODS) or using the Version 8 XML libname engine. Let's discuss each of these with an example of each.

SAS DATA STEP

In addition to providing a very powerful engine for creating complex documents, the DATA STEP will be perhaps the most familiar method to most SAS programmers. As depicted in Appendix B, we can programmatically create an XML document from a SAS data set. Despite its simplicity, we can extend this example to combine a much richer XML document by combining multiple data sets to create the patterned, hierarchical output that characterizes the XML document. In Appendix B, we find a more complicated example of this as we pull in data from multiple data sources.

In our simple case (see XML Document 1 shown earlier), we created a simple XML document that displayed our customer data with basic name and contact information as well as some brief demographics. But there is no reason that you couldn't extend the example to include past orders, billing and shipping address or marketing constructs such as Life-Time Value, Segmentation, etc. By using the power of the SAS Data Step, we can programmatically include additional content as well as metadata from SAS' dictionary tables and formats. The example shown below was created to show how we would create a complete customer history from multiple tables (see Appendix B.) Here we show an XML document that contains several customers, their contact information, demographics and all of their past invoices. We have collapsed the view to show only the high-level portion of the invoices. Below, we show an expanded invoice section.

```
<?xml version="1.0" ?>
<customer-data>
   <cust-info>
        <cust-id>137000</cust-id>
        <name>Kraft, Ms. Rose</name>
      <demographics>
           <gender>Female</gender>
           <age>34</age>
           <income>32,340</income>
           <status>Married</status>
        </demographics>
      <address>
           <street ORDER="1">869
               Veterans Blvd.</street>
           <street ORDER="2">Business
               Research</street>
           <city>Rutherford</city>
           <state>NJ</state>
           <zip-code>70702</zip-code>
           <region>NORTHEAST</region>
        </address>
      <invoices>
         <invoice INVOICE-ID=">107707"
               INVOICE-
               DATE="07MAR1994">
            .. .. .. ..
         <invoice INVOICE-ID=">135872"
               INVOICE-
               DATE="08AUG1994">
            .. .. .. ..
         <invoice INVOICE-ID=">243377"
               INVOICE-
               DATE="24APR1998">
        </invoices>
     </cust-info>
   <cust-info>
   .. .. .. ..
 </customer-data>
```

XML Document 2. Customer History XML document.

Here we show one particular invoice for August 8, 1994.

```
<invoice INVOICE-ID=">135872" INVOICE-
      DATE="08AUG1994">
<lineitems>
  <line-item ID="Item1">
      <product-code CAT="Toys">TY1200</product-code>
      <quantity>4</quantity>
   </line-item>
  <line-item ID="Item2">
      <product-code CAT="Toys">TY2100</product-code>
      <quantity>1</quantity>
   </line-item>
  <line-item ID="Item3">
      <product-code CAT="Toys">TY2300</product-code>
     <quantity>1</quantity>
   </line-item>
  <line-item ID="Item4">
     <product-code CAT="Toys">TY4100</product-code>
    <quantity>1</quantity>
   </line-item>
 </lineitems>
</invoice>
```

XML Document 3. Customer history XML document with expanded view of an invoice.

ODS

Starting with Version 7, SAS programmers were able to take full advantage of the Output Delivery System or ODS. The goal of ODS was to rules that governed what output should contain versus how it should be presented. Although experimental in Version 8.00, we can now direct any output that can be created and send it to an XML document. The default XML engine for ODS in Version 8.0 (M01) produces a proprietary XML document that adheres to SAS Institute’s Version 8.0 DTD. The code to create a sample XML document with PROC TABULATE is shown below. The XML document that is created as a result of this contains a tremendous amount of metadata, which can be used to describe the output and its potential relationship to other output objects.

```
ods xml file="c:\xmltabulate.xml";

  proc tabulate data=SASUSER.CLASS  ;
     table   ALL
        ,(sex age height weight) * n =' '
        ;
     class  sex age height weight;
  run;
ods xml close;
```

In Version 8.01, additional experimental engines will be developed that support the DocBook[3] standard as well as a few variants of HTML output (with CSS and a bare-bones HTML version). By combining SAS Institute’s raw XML documents with the appropriate DTD, we have a powerful method for transforming and filtering output in other applications that can read, write and render XML documents.

XML Libname Engine

Although experimental in Version 8.0 (M01), the XML LIBNAME engine provides an easy method of writing XML documents directly from libname references. Those tasks that you can perform with a standard libname such as updating data are available through this engine. The development at SAS Institute is ongoing in this area, but we can see a simple example of taking a SAS dataset and writing it out to an XML document.

```
libname sampdata 'C:\mylib' ;
libname DestXML XML 'output.xml';

data DestXML.dsetanything ;
  set sampdata.customer
       (label="My customer information");
  addr1=urlencode(addr1);
  label custnum ="Customer-Information";

run;
```

The first row of the XML document that is produced from these statements is presented below. We had to use the `urlencode` function in our code to encode any special XML element names such as the ampersand found in one of our addresses (see “Rules the Live By” section above for examples of these.) When we created the XML files, we specified a data set name just like we would for a permanenet SAS data set (`dsetanything`). This name is used in the XML as a descriptor for the document. However, with the default engine, which is generic mode XML[4], variable labels, formats and lengths are not written to the document. We can use a different engine by using the XMLTYPE= option and specify the values values of GENERIC|ORACLE or HTML or OIMDBM[5].

```
<?xml version="1.0" ?>
<TABLE>
    <ROW>
          <CUSTNUM>137000</CUSTNUM>
          <NAME>Kraft, Ms. Rose</NAME>

                <ADDR1>869%20Veterans%20Bl
                vd.%20%20%20%20%2</ADDR
                1>
          <ADDR2>Business Research</ADDR2>
          <CITY>Rutherford</CITY>
          <STATE>NJ</STATE>
          <PHONE>201-507-2211</PHONE>
          <REGION>NORTHEAST</REGION>
          <AGE>34</AGE>
          <INCOME>32340</INCOME>
          <MARRIED>1</MARRIED>
          <SEX>0</SEX>
          <ZIP>70702</ZIP>
      </ROW>
  .. .. ..
  </TABLE>
```

XML Document 5. XML document Created with the XML Libname Engine (Version 8.0 M01).

Reading XML

The primary value of an XML document is that it can be easily interpreted by someone else – especially by computer. The exchange of information through XML as the medium requires that the recipient be able to read, process and possibly transform the information into something usable. There are a variety of **XML parsers** available which can read and interpret an XML document (see for example IBM, Microsoft and Sun Microsystems.)

A parser, or engine that reads an XML document is typically embedded in an application. Its job is to read the document and convert the content into constructs that the application understands. For SAS application developers, we can write our own parser using SCL or Base SAS, but it would be much simpler to use an XML parser that was written by someone else. IBM’s parser is one of the most powerful and is available as a Java applet.

Which parser you use will depend on your application. A likely scenario for SAS application developers would include a back-end or middle-ware parser would interpret the document and apply programming logic to load a database, construct an HTML page or render a PDF document.

Through the use of the XML Libname engine, we demonstrate below the conversion of an XML document to a SAS dataset. Once parsed, we can then use our traditional SAS programming constructs to develop robust client/ server or web-based applications. The general form of the XML syntax for reading an XML document is given here. (Remember, this is experimental!)

```
libname myXML XML 'C:\mylib\class.xml';
proc datasets dd=myXML; run;

data readback;
  set myXML.row;
run;

Proc print data=readback;
run;
```

Rendering XML

Although XML was designed primarily as a way to solve the problem of exchanging web documents, it is clear that XML has potential for solving other sorts of data exchange problems that are not limited to the web. In Version 8.1 of the SAS System, we will see support for WAP (**Wireless Application Protocol**) that will enable XML documents to be sent over wireless protocols to such devices as PDAs and cellular telephones.

Despite its youth, XML already has a rich set of tools, which allow us to render XML documents in a variety of ways. If our browser supports the viewing of XML documents, we can open them directly. IE 5 is currently the only browser that supports XML natively. Because of this fact, rendering HTML on the server and pushing it to the client will probably be the most common method of rendering XML until there is a larger base of browser support for XML.

Other types of transformations include:

- ✓ Converting XML into Scalable Vector Graphics (SVG) based on the W3C’s SVG markup language to produce pie charts and other graphical formats from XML documents.
- ✓ Rendering XML into PDF documents using James Tauber’s FOP (Formatting Objects into PDF).
- ✓ Converting XML documents into TeX files which can be

[3] For more information about DocBook, see http://www.oasis-open.org/docbook/

[4] This is compatible with the Oracle8i XML implementation.

[5] The OIMDBM is an industry standardization and attempts to express formatting information in its output. see http://www.mdcinfo.com/OIM/OIM10.html

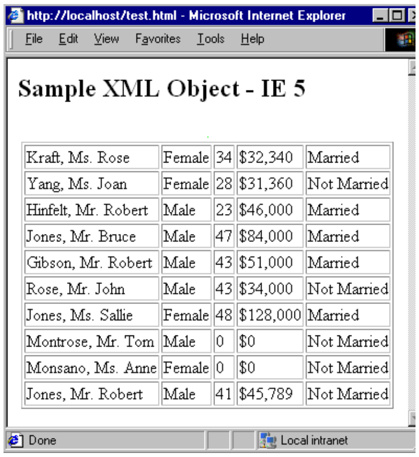
rendered on a variety of platforms.

- ✓ Converting XML into speech (VoiceXML) which can be used create audio tracks by a variety of software tools.
- ✓ Rendering XML into an HTML document.

Microsoft Internet Explorer 5.0

Because we are most familiar with the visual presentation of the web, we will explore rendering XML documents in a web browser using Microsoft's Internet Explorer (IE5). This first example shows the use of Microsoft's XML Data Source Object (DSO), which is a Java applet that can be used to bind an XML data source to a web page (in IE5).

The HTML used to produce this page is fairly straightforward. The key to this page is the APPLET reference where we pass it a parameter pointing it to the proper XML data source. This example shows XML's simplicity – we have not done anything else to this document to make it appear in the table except what you see below. The XML document referenced here is the same one that we showed in XML Document 1 above.


http://localhost/test.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

Sample XML Object - IE 5

Kraft, Ms. Rose	Female	34	$32,340	Married
Yang, Ms. Joan	Female	28	$31,360	Not Married
Hinfelt, Mr. Robert	Male	23	$46,000	Married
Jones, Mr. Bruce	Male	47	$84,000	Married
Gibson, Mr. Robert	Male	43	$51,000	Married
Rose, Mr. John	Male	43	$34,000	Not Married
Jones, Ms. Sallie	Female	48	$128,000	Married
Montrose, Mr. Tom	Male	0	$0	Not Married
Monsano, Ms. Anne	Female	0	$0	Not Married
Jones, Mr. Robert	Male	41	$45,789	Not Married

Figure 2. XML Document rendered through Microsoft's DSO.

```
<html><head></head><body>
<h2> Sample XML Object - IE 5</h2>
 <center>
<APPLET code=com.ms.xml.dso.XMLDSO.class id=Customer
width=0 height=0 MAYSCRIPT="true">
<PARAM NAME="url" VALUE="http://localhost/customer.xml">
</APPLET>
<table border=1 datasrc=#Customer>
  <tr>      <td><span datafld="name"></span></td>
            <td><span datafld="gender"></span></td>
            <td><span datafld="age"></span></td>
            <td>$<span datafld="income"></span></td>
            <td><span datafld="status"></span></td>
  </tr></table></center></body></html>
```

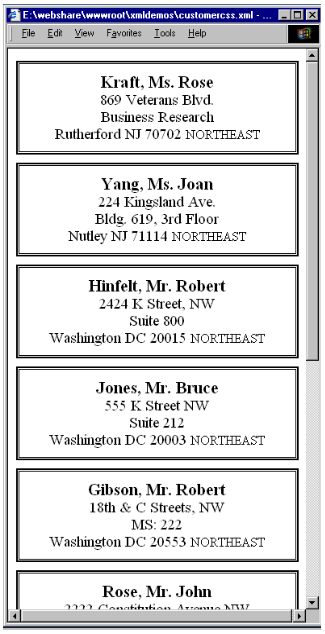
HTML Segment 1. HTML code for binding and XML data source to a web page.

Expanding on this simple example where we bind our customer XML document to a web browser, we could also display it in a form view with navigations that allow us to move through the dataset (forward, backward) as well as add/ modify and delete records. For more information on the XML **Data Source Object** (DSO), please refer to the glossary at the end. Here we provide references to on-line resources.

Doing it in Style

Those familiar with HTML may also be familiar with **Cascading Style Sheets** (CSS). CSS was an early attempt at separating the information contained in HTML from how it was presented. Instead of marking up a page header with font specifications that control the size, orientation and other font characteristics, we can use Cascading Style Sheets and a class definition to control this. This allowed us to create basic HTML documents whose look and feel could be controlled by an external style sheet.

XML has a similar construct for handling the complex presentation requirements of the web. In XML, one can either use Cascading Style Sheets (CSS) or the **Extensible Stylesheet Language** (XSL) to present data in a browser. Figure 3 shows an XML document formatted with a CSS (left) and an XSL (right).


E:\webshare\wwwroot\xmldemos\customercss.xml - ...
File Edit View Favorites Tools Help

Kraft, Ms. Rose
869 Veterans Blvd.
Business Research
Rutherford NJ 70702 NORTHEAST

Yang, Ms. Joan
224 Kingsland Ave.
Bldg. 619, 3rd Floor
Nutley NJ 71114 NORTHEAST

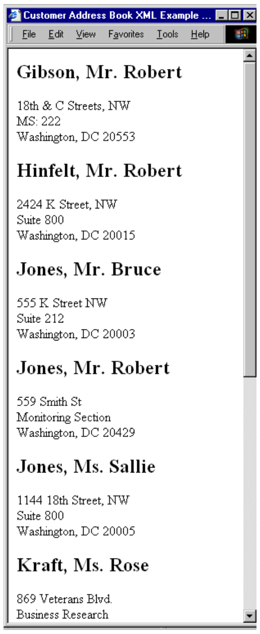
Hinfelt, Mr. Robert
2424 K Street, NW
Suite 800
Washington DC 20015 NORTHEAST

Jones, Mr. Bruce
555 K Street NW
Suite 212
Washington DC 20003 NORTHEAST

Gibson, Mr. Robert
18th & C Streets, NW
MS: 222
Washington DC 20553 NORTHEAST

Rose, Mr. John

XML Document rendered with a CSS.


Customer Address Book XML Example ...
File Edit View Favorites Tools Help

Gibson, Mr. Robert

18th & C Streets, NW
MS: 222
Washington, DC 20553

Hinfelt, Mr. Robert

2424 K Street, NW
Suite 800
Washington, DC 20015

Jones, Mr. Bruce

555 K Street NW
Suite 212
Washington, DC 20003

Jones, Mr. Robert

559 Smith St
Monitoring Section
Washington, DC 20429

Jones, Ms. Sallie

1144 18th Street, NW
Suite 800
Washington, DC 20005

Kraft, Ms. Rose

869 Veterans Blvd.
Business Research

XML Document rendered with an XLS.

Figure 3. XML Document rendered with Cascading Style Sheets versus an Extensible Style Sheet.

Despite the fact that Cascading Style Sheets were designed for HTML, it is just as good at formatting XML documents for the web. The XML document that was used was identical to that described in XML Document 1 referenced earlier in this paper. The only difference was the addition of a header describing where we can find the CSS. The CSS can be found in Appendix D.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="Labels.css"?>
<customer-data> and so on . . .  . . .
```

XLS (extensible Style Sheet) is a style sheet technology designed specifically for XML. With XSL, we are able to control our document's presentation much more than before. As demonstrated here (one in Figure 3-Right and again in Figure 4), we have two XML documents that are identical except for the reference to its XSL style sheet. Note the clear separation of the content (XML document) from its presentation (through its XSL reference.)

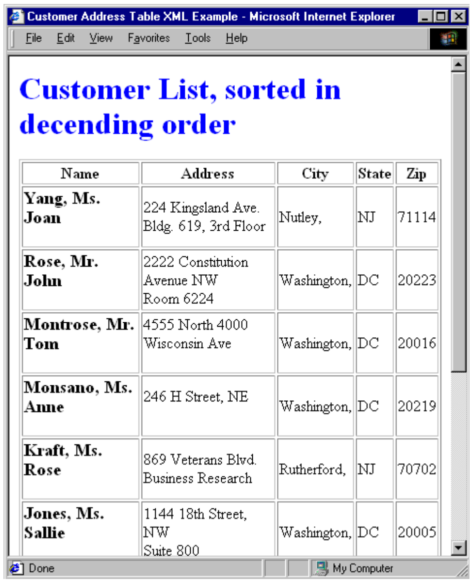
The XML header for this document is shown here:

```
<?xml version="1.0" ?>
<?xml-stylesheet href="Labels.xsl" type="text/xsl"?>
```

This second rendering below shows the exact same document – the only difference: the XML header points us to a different XSL document.

```
<?xml version="1.0" ?>
```

```
<?xml-stylesheet href="Table.xsl" type="text/xsl"?>
```


Customer Address Table XML Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Customer List, sorted in decending order

Name	Address	City	State	Zip
Yang, Ms. Joan	224 Kingsland Ave. Bldg. 619, 3rd Floor	Nutley,	NJ	71114
Rose, Mr. John	2222 Constitution Avenue NW Room 6224	Washington,	DC	20223
Montrose, Mr. Tom	4555 North 4000 Wisconsin Ave	Washington,	DC	20016
Monsano, Ms. Anne	246 H Street, NE	Washington,	DC	20219
Kraft, Ms. Rose	869 Veterans Blvd. Business Research	Rutherford,	NJ	70702
Jones, Ms. Sallie	1144 18th Street, NW Suite 800	Washington,	DC	20005

Figure 4. XML Document rendered with Table.xls.

Transformation Through XSL and XSLT

In addition to being able to render XML documents, XML provides us with a rich toolset for transforming documents as the style sheet is applied. Through the use of both XSL and XSTL (**XML Transformation Language**) we can move text from one place in an XML document to another (for example, moving the value of first name and last name around); sorting elements (see, for example Figures 3 and 4 – note the order of the names in each); generating text and performing calculations (such as the count of all of the line items in an invoice); and numbering items in a list. Both XSL and XSTL provide a rich set of tools for manipulating and managing the information contained within an XML document.

CONCLUSION

XML brings a tremendous amount of power and flexibility to both client/ server and web-based applications. As we have seen in this paper, there are a number of compelling benefits to both developers and to the organizations they serve. For the SAS developer, XML offers a rich toolset for communication across application and organizational boundaries. We are able to structure data in the context of meaningful hierarchy in a way that other people and software can easily understand. Because of its patterned structure, data can be shared literally with any application or user that requires it. As data and its corresponding structure changes, the XML document and its structure change with it.

THE FUTURE OF XML AND SAS

Although no production SAS Institute applications have been delivered to date using XML, we can easily incorporate XML into our applications by using common programming elements such as SCL and the DATA Step or combining other technologies in concert with the SAS System.

As more vendors, like SAS Institute, incorporate XML into both applications and lower level language support, we should expect to see a wide variety of uses for XML in our applications. In the short term, we can expect to see more experimental engines in BASE SAS with ODS and the LIBNAME engines in Version 8.1.

Potential Applications

Beyond simple tasks like reading and writing, we would hope to see a variety of application level support for XML. Here are just a few ideas.

Native SAS XML Tree-Viewer. Having a built-in editor for viewing/ editing XML documents that have been created within SAS. For example, one could pull up an XML document from the File -> Open Menu to display a tree-view for editing/ viewing the XML structure.

SAS/IntrNet Extensions. For web-based applications, native support for both parsing XML and writing XML to the Application Dispatcher sessions for use in subsequent pages would be a logical extension of this technology. The XSL language is common in many regards to what we see in htmSQL as it handles a records sets in the {eachrow} directive. I would expect native htmSQL to be able to navigate an XML document hierarchy.

Messaging Transport. With respect to messaging, we should be able to expect will provide a transport mechanism (both receiving and sending) XML-based messages between both SAS and non-SAS based clients and servers.

Balanced Scorecard. For applications, like Balanced Scorecard, we could use XML to deliver metrics from throughout the organization in a single, common format.

In general, XML can be used wherever data is being exchanged between multiple applications and/or organizations to facilitate the understanding and assimilation of the data in its new context.

By allowing SAS developers to fully exploit the benefits of XML, the SAS System will continue its leadership role in the integration, analysis, presentation and decision support for years to come.

ACKNOWLEDGMENTS

The author would like to sincerely thank several people for their rule in gentle and thoughtful review of this manuscript. Specifically, we would like to thank Ian Whitlock, Don Henderson, Chris Olinger and Anthony Friebel for their support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

```
<?xml version="1.0" ?>
<business-card>
    <author>
        <name>
             <first-name>Greg</first-name>
             <middle-initial>S.</middle-initial>
             <last-name>Barnes Nelson</last-name>
         </name>
         <title>Director</title>
        <company>
             <comp-name>STATPROBE Technologies</comp-name>
             <web-site>www.statprobetechnologies.com</web-site>
            <address>
                 <street>117 Edinburgh South, Suite 202</street>
                 <city>Cary</city>
                 <state>NC</state>
                 <zip-code>27511</zip-code>
             </address>
         </company>
        <contact-methods>
             <business-phone>919-465-0322 x351</business-phone>
             <business-fax>919.465.0323</business-fax>
             <b-email>greg.barnesnelson@statprobe.com</b-email>
             <personal-email>gregbn@ix.netcom.com</personal-email>
         </contact-methods>
      </author>
  </business-card>
```

…or simply email me at greg.barnesnelson@statprobe.com.

APPENDICES

Appendix A. Producing a simple XML document from a SAS Data Set

Simplecustomerview.sas ....

```
filename outxml "c:\simplecustomer.xml";

Data _null_;
  file outxml;
  set sampdata.cust10 NOBS=Lst;
  length gender $6. marital_status $11.;

  %let tab="     ";
  addr1=htmlencode(addr1);
  addr2=htmlencode(addr2);

          if sex=0 then gender='Female';
```

```
                   else gender='Male';

         if married=0 then marital_status="Not Married";
                    else marital_status="Married";

         if _n_=1 then do;
                  put '<?xml version="1.0" ?>';
                  put '<customer-data>';
         end;

         put '<contact-information>';

          put &tab'<cust-id>' custnum '</cust-id>';
          put &tab'<name>'   name '</name>';
          put &tab'<gender>' gender '</gender>';
          put &tab'<age>'    age '</age>';
          put &tab'<income>' income '</income>';
          put &tab'<status>' marital_status '</status>';
          put &tab &tab '<address>';
          put &tab &tab '<street ORDER="1">'
              addr1 '</street>';
          put &tab &tab '<street ORDER="2">'
               addr2 '</street>';
          put &tab &tab '<city>' city '</city>';
          put &tab &tab '<state>' state '</state>';
          put &tab &tab '<zip-code>' zip '</zip-code>';
          put &tab &tab '<region>' region '</region>';
          put &tab &tab '</address>';

         put '</contact-information>';

         if _n_ = lst then do;
                  put '</customer-data>';
         end;
 run;
```

Appendix B. Producing a simple XML document from Multiple Data sets

CustomerInvoiceView.sas ....

Because of the length of this program, the code for this example can be found on-line at http://www.statprobetechnologies.com/XML

Appendix C. Cascading Style Sheet used to format an XML document

Labels.css ....

```
contact-information {
 display: block;   width: 350px;   padding: 10px;   margin-bottom: 10px;
border: 4px double black;   background-color: white;   color: black;   text-
align: center;}

name {
 display: block;  font-family: Times, serif;  font-size: 16pt;
 font-weight: bold;}

street {
 display: block;  font-family: Times, serif;  font-size: 14pt;}
city, state, zip-code {
 display: inline;  font-family: Times, serif;  font-size: 14pt;}

cust-id, gender, age, income, status {
 display: none;}
```

Appendix D. eXtensible Style Sheet used to format an XML document as Mailing labels

Labels.xls...

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <xsl:template match="/">
  <html><head><title>Customer Address Book XML
Example</title></head>
   <body bgcolor="#FFFFFF">
  <xsl:for-each order-by="+ name" select="customer-data/contact-
information">
     <xsl:apply-templates select="name"/>
      <xsl:for-each select="address">
        <xsl:apply-templates select="street"/>
        <xsl:apply-templates select="city"/>
        <xsl:apply-templates select="state"/>
        <xsl:apply-templates select="zip-code"/>
      </xsl:for-each>
    </xsl:for-each>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="name"><h2><xsl:value-of/></h2>
 </xsl:template>

 <xsl:template match="street"><xsl:value-of/><br/>
 </xsl:template>

 <xsl:template match="city"><xsl:value-of/>,
 </xsl:template>

 <xsl:template match="state"><xsl:value-of/>
 </xsl:template>

 <xsl:template match="zip-code"><xsl:value-of/><br/>
 </xsl:template>

</xsl:stylesheet>
```

Appendix E. eXtensible Style Sheet used to format an XML document as a Table

Table.xls ....

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
 <xsl:template match="/">
  <html><head><title>Customer Address Table Example</title></head>
   <body bgcolor="#FFFFFF">
<h1> <font color="blue">Sorted in decending order</font> </h1><p/>
<table align="center" border="1">
        <tr>
         <th>Name</th>
         <th>Address</th>
         <th>City</th>
         <th>State</th>
         <th>Zip</th>
        </tr>
<xsl:for-each order-by="- name" select="customer-data/contact-
information">
        <tr>
     <td><xsl:apply-templates select="name"/></td>
      <xsl:for-each select="address">
        <td><xsl:apply-templates select="street"/></td>
        <td><xsl:apply-templates select="city"/></td>
        <td><xsl:apply-templates select="state"/></td>
        <td><xsl:apply-templates select="zip-code"/></td>
      </xsl:for-each>
        </tr>
    </xsl:for-each>

        </table>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="name"><h3><xsl:value-of/></h3>
</xsl:template>

 <xsl:template match="street"><xsl:value-of/><br/>
 </xsl:template>

 <xsl:template match="city"><xsl:value-of/>,
 </xsl:template>

 <xsl:template match="state"><xsl:value-of/>
 </xsl:template>

 <xsl:template match="zip-code"><xsl:value-of/><br/>
 </xsl:template>
</xsl:stylesheet>
```

APPENDIX B. ANNOTATED GLOSSARY AND BIBLIOGRAPHY

This glossary provides some of the most common terms and phrases used in the context of creating, processing and rendering XML documents. An up-to-date version of this table can be found at http://www.statprobetechnologies.com/XML

Term	Acronym	Description/ Purpose	References
Cascading Style Sheet	CSS	CSS is a style sheet language designed as an early attempt to help developers separate the content in HTML from the way that it was presented. CSS can use tag definitions such as <H1> to apply formatting across a document or to sections that are specifically identified within a document. CSS can be used to format an HTML document as well as an XML document.	http://webreview.com/guides/style/style.html http://www.w3.org/Style/
Data Source Object	DSO	Data Source Objects (DSOs) are objects that can imbed structured data, including XML, into HTML pages. This is a proprietary technology that is embedded into Microsoft's browsers.	http://msdn.microsoft.com/workshop/Author/databind/datasources.asp
Dynamic HTML	DHTML	Dynamic HTML provides a mechanism to control client- or browser-based interactivity through programmatic access to the HTML elements. Programs are typically written in client-side scripting languages like JavaScript and VBScript	http://msdn.microsoft.com/library/backgrnd/html/msdn_dynhtml.htm
Document Object Model	DOM	The document object model is a standard objected oriented application programming interface (API) that gives developers programmatic control of XML document content, structure, formats, etc. The XML DOM is a recommendation from the W3C and provides access through scripting languages like VBScript and JavaScript.	http://www.w3c.org/TR/REC-DOM-Level-1 http://www.w3.org/DOM/
Document Type Definition	DTD	The document type definition defines the valid syntax for a class of XML documents. It provides a list of the element names, which elements can appear in combination with which other ones and so on.	http://www.whatis.com/dtd.htm
eXtensible Markup Language	XML	XML is the new language of the web. It provides a framework for delivering structured data from a wide-variety of applications to the desktop for local computation and presentation. It is an ideal format for exchanging data within and between applications and organizations.	http://www.ucc.ie/xml/faq.html http://www.oasis-open.org/cover/xml.html http://www.w3.org/XML http://msdn.microsoft.com/xml/general/xmlfaq.asp http://www.alphaworks.ibm.com http://www.xml.com/pub/98/10/guide0.html http://www.webdeveloper.com/html/html_xml_1.html
Extensible Stylesheet Language	XSL	A working draft that describes a language that can be used to provide flexible document presentation rules. Similar to CSS, it applies formatting rules to a document that is separate from the XML document itself. Written in XML, XSL contains instructions for getting data out of a document and converting it into another.	http://www.xmlmag.com/upload/free/features/xml/1999/01win99/kc2win99.asp http://www-4.ibm.com/software/developer/education/transforming-xml http://www.w3.org/Style/
Human Resources Management Markup Language	HRMML	HRMML, created by Structure Methods, Inc., is an application whose aim is to unify the manner in which human resource information is represented and shared.	http://www.structuredmethods.com/hrxml
Hyper Text Markup Language	HTML	The language of the web. HTML is used to format documents so that they can be viewed over the web through a browser such as Internet Explorer or Netscape.	For the differences between XHTML and HTML see: http://webreview.com/wr/pub/1999/07/16/feature/index2.html

Term	Acronym	Description/ Purpose	References
Metadata		"Data about data". Metadata is that which describes data in terms of its meaning or use. In SAS, we have the SQL dictionary tables, which describe tables, variables, etc. These dictionary tables could be described as metadata.	http://www.mdcinfo.com/
Meta-Language		A language that is used to describe or create other languages. XML isn't really a language at all, but rather a language the allows people to create languages.	http://www.whatis.com/meta.htm
Standard Generalized Markup Language	SGML	SGML is the international standard for defining the structure and content in electronic documentation. XML is a simplified version of SGML and was designed to take some of the complexity out of SGML for web-delivery.	http://www.whatis.com/sgml.htm
Tags		In a document markup language a tag is used to describe some information. For example, the tag <b> tells us that what follows should be bolded. In XML, we create our own tags like <customer> or <invoice>.	http://k12unix.larc.nasa.gov/training/tags.html
Valid		In addition to being well-formed, a valid XML document has a DTD or XML schema that defines its content and use.	http://www.xml.com/pub/98/10/guide3.html
Well formed		An XML document is said to be well formed if it follows all of the rules defined by the XML specification. Although well formed documents essentially follow the rules, they don't have a DTD associated with them.	http://msdn.microsoft.com/xml/general/what-is-xml.asp
Wireless Application Protocol	WAP	A new technology, originating in Europe, which maintains the specifications for how data should be encoded, transferred and processed with wireless applications. An example would be data transferred from a web server to a PalmPilot™.	http://www.wapforum.org/ http://www.wapforum.org/faqs/index.htm
XHTML	XHTML	XHTML is a specification approved by the W3C that allows users to create documents in HTML, but use some XML tags for embedding things like mathematical equations.	http://www.w3c.org/TR/xhtml1 http://webreview.com/wr/pub/1999/07/16/feature/index.html?wwwrrr_19990716.tx
XML Linking Language	Xlink	A draft language specification that describes how XML documents can be linked to one another (similar to HTML's linking <A HREF>.)	http://www.w3.org/TR/1998/WD-xptr-19980303
XML Namespaces	Namespaces	Currently a recommendation from the W3C as a standard to prevent the overlap of names used by different software vendors and/ or applications. For example, we may use the element name <status> to mean marital status whereas in our invoice, the name <status> may refer to whether the account is active or not. Namespaces help provide context to the names of the elements used.	http://www.w3c.org/TR/REC--xml-names
XML Parser		An XML parser reads a string of XML data and generates a structured tree (hierarchy). Some parsers validate the document against a DTD or schema.	http://www.zdnet.com/pcmag/features/xml98/parsers.html
XML Schema		Similar to an XML DTD but its rules are more rigorous. An XML schema provides a definition for how an XML document should be interpreted.	http://www.schema.net
XML Transformations	XSLT	A working draft from the W3C that describes ad language that can be used to transform XML content from one data format to another.	http://www.xmlmag.com/upload/free/features/xml/1999/01win99/kc2win99.asp