Paper TU12

Markup 101: Markup Basics

Cynthia Zender, SAS Institute, Inc., Cary, NC

ABSTRACT

Do you want to know what this "Markup" buzz is all about? Does it surprise you to know that the concept of "marking up" text for computers has been around since 1969? So what's all the excitement about? And what does SAS have to do with it?

INTRODUCTION

This paper provides an introduction to the concept of markup through concrete examples. The types of markup files which the author will discuss include: HTML, RTF, comma-delimited, tab-delimited, SYLK, WML, troff, LaTeX, Docbook, SGML and, yes, XML. We will compare the development of markup specifications with the development of WYSIWYG tools using a timeline of milestone development events. By the end of the presentation, you'll not only understand what the buzz is all about, you'll also understand how SAS can produce markup files and what kind of markup is possible. You will also receive a checklist of questions to ask at your company to help you figure out where to go next with markup and SAS.

WHAT IS MARKUP?

The concept of "markup" is as simple as a teacher with a red pencil "marking up" a student's paper or a copyeditor "marking up" page proofs with editing symbols. The key to making "markup" work is that the person marking up the content and the person processing the marked up content have both agreed on what the markup symbols mean. For example, "*stet*" written in the margin of a document means that something previously stricken out should be allowed to stay in the document. To download a printable page of proofing symbols from the American Heritage Dictionary, go to: http://www.montana.edu/commserv/cspublications/pdfs/editmarks.pdf

MANUAL MARKUP VS. COMPUTER MARKUP

The process of marking up page proofs or a manuscript is a manual process. In a computer file, the concept of markup has been extended to include formatting instructions inserted into a file or document text so that software can format the text or a printer can print the text or document. Generally, these formatting instructions take the form of tags or commands that start with a special character.

The "granddaddy" of all computer markup languages is SGML (Standard Generalized Markup Language). SGML causes a bit of confusion for programmers. SGML is not a compiled or procedural computer programming language in the same way that BASIC, COBOL or SAS are programming languages. Rather, SGML is actually a standard for how to describe a document. But, with SGML, the way to describe a document also involves a method for marking up the document content. And the method for marking up a document, involves using the SGML standard to define the kind of tags or elements that you will routinely use to always mark up documents of a certain type. If this sounds confusing and complicated, that's because SGML is all about grammar, semantics and definition. SGML is an ISO standard (ISO 8879:1986), which means that it went through a period of evaluation, design and definition before being adopted by the International Standards Organization. Interestingly enough, the "official" title of the SGML standard from the www.iso.org site is: "**Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML)"**.

SOME MARKUP EXAMPLES

What do markup files look like anyway? It depends. The one constant in markup files is that the information or content is usually readable. Sometimes, the formatting instructions that surround the text are easily decipherable, sometimes the formatting instructions seem opaque. Generally speaking, most markup files are designed to be used with an application, viewer or processor that understands what the formatting instructions mean and whether the formatting instructions are limited to a simple description of document structure or whether the formatting instructions also detail specific presentation characteristics.

In the following screen shots, I'll show SGML, HTML, RTF, SYLK, CSV, troff, and LaTeX markup examples. Most of these files, were either created with Notepad or created with an appropriate tool and then viewed with Notepad or a text editor. Files like the SGML, troff or LaTex examples might be passed to a typesetting program or printer,

while the HTML file would be viewed with a web browser. An RTF (Rich Text Format) file would be normally viewed with a word processing program. A CSV file (or SYLK file) could be opened with a spreadsheet or database program.

SGML example:

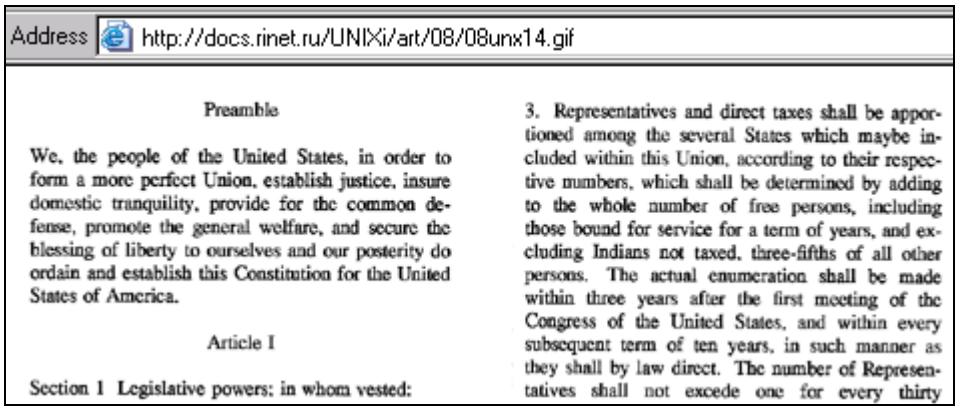
```
<!DOCTYPE sampleDocument SYSTEM "c:\admin\dtd\sample2.dtd" >
<sampleDocument lang=ENG>
<front><title>Paul's Usage of
<phrase lang=GRK>ta kat&omacr;tera</phrase>
("the lower parts of the earth") in Ephesians 4:9</title>
<author>Joseph DiVito</author></front>
<body>
<section><head>History of Scholarship</head>
<p>We will review the history of exegesis in two major phases.</p>
<section><head>The Church Fathers</head>
<p>The early church fathers understood Paul's phrase <phrase lang=GRK>ta
kat&omacr;tera</phrase> as a reference to the Netherworld, i.e., the <phrase
lang=LAT>Descensus ad Inferos</phrase> known from Sumerian, Akkadian, Homeric and early
Jewish literary traditions. This interpretation survives in the Apostles' Creed: "he
descended into Hell." See brief classical references in <citRef target=aland88>[Aland
1988:864]</citRef>, along with 1 Peter 3:19 and 4:6. The monograph of P. Benoit supplies a
thorough analysis: <title lang=FRE><phrase lang=LAT>Descensus Christi ad Inferos</phrase>
dans le Nouveau Testament</title>.</p></section> . . .<more SGML>. . .
</sampleDocument>
```

There is no screen shot of what this output looks like. This file might be passed to a publishing program.

Troff example:

```
.ll 3i
.mk a
.ce
Preamble
.sp
We, the people of the United States, in order
to form a more perfect Union, establish justice, insure
domestic tranquility, provide for the common defense, promote
the general welfare,
and secure the blessing of liberty to ourselves and our posterity do
ordain and establish this Constitution for the United States of
America.
.sp
.ce
Article I
.sp
Section 1  Legislative powers; in whom vested:
.sp
All legislative powers herein granted shall be vested in a
Congress of the United States, which shall consist of a Senate
and a House of Representatives.
.sp
Section 2  House of Representatives, how and by whom chosen,
Qualifications of a Representative. Representatives and direct
taxes, how apportioned. Enumeration. Vacancies to be filled.
Power of choosing officers and of impeachment.
```

Unlike the SGML example, the troff example uses "dot" instructions like ".li", ".ce" and ".sp". This screen shot of the formatted troff output came from: http://docs.rinet.ru/UNIXi/art/08/08unx14.gif.

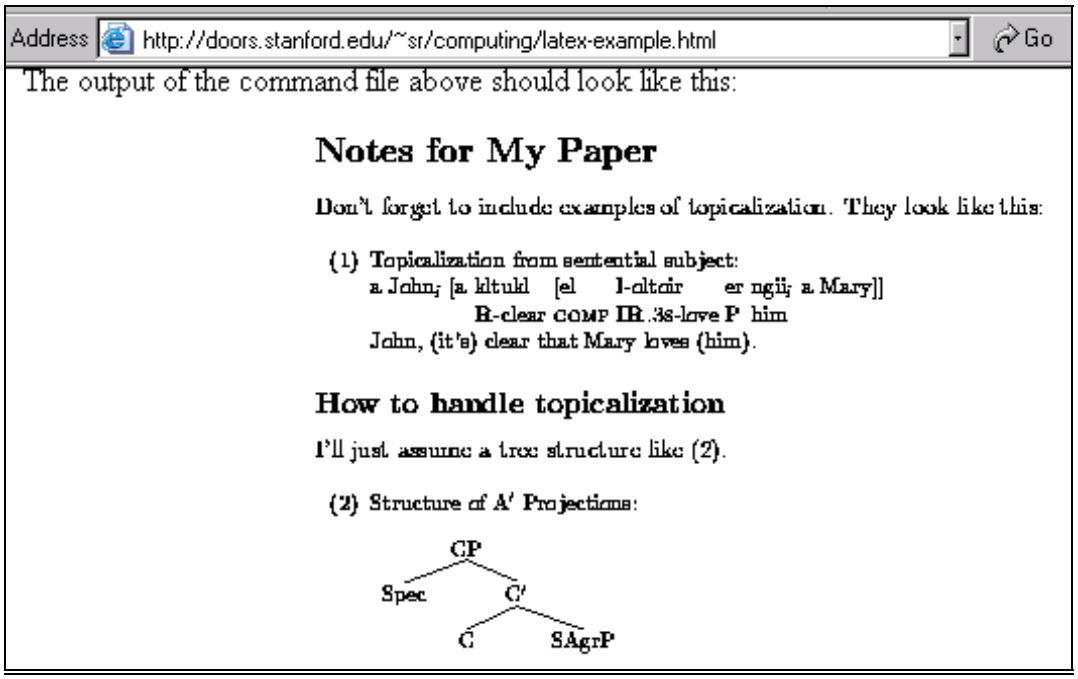


LaTex example:

To produce a simple LaTeX document, use an editor on turing (probably emacs), and make a file that looks like this:

```
\documentclass[12pt]{article}
\usepackage{lingmacros}
\usepackage{tree-dvips}
\begin{document}

\section*{Notes for My Paper}

Don't forget to include examples of topicalization.
They look like this:

{\small
\enumsentence{Topicalization from sentential subject:\\
\shortex{7}{a John$_i$ [a & kltukl & [el &
  {\bf l-}oltoir & er & ngii$_i$ & a Mary]]}
{ & {\bf R-}clear & {\sc comp} &
  {\bf IR}.{\sc 3s}-love   & P & him & }
{John, (it's) clear that Mary loves (him).}}
}

\subsection*{How to handle topicalization}

I'll just assume a tree structure like (\ex{1}).
```

Unlike either SGML or troff, LaTeX uses "slash" instructions, like "\section" or "\subsection". The following screen shot of the formatted LaTeX file came from http://doors.stanford.edu/~sr/computing/latex-example.html



The SGML, troff and LaTeX examples are all meant to be used by printers, typesetters or publishing programs. On the other hand, HTML, RTF and CSV are all examples of markup files that can be easily viewed or rendered with programs on a personal computer.

HTML example viewed in Notepad:

```
<html>
<head>
<title>HTML
Example</title>
</head>
<body>
<h1 align=center>
Hello World!
</h1>
</body>
</html>
```

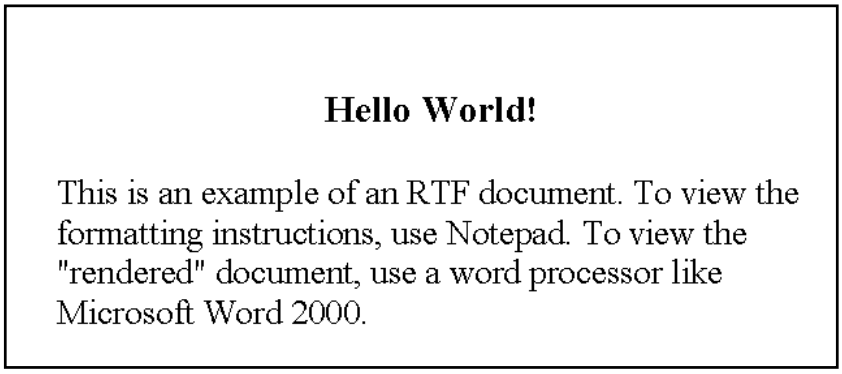
HTML Example viewed in Internet Explorer

Like SGML, the HTML file makes use of tags like <html> or <table> to delimit the content or text in the file. In the above example, we can see that the HTML specification allows for some definition of document structure with the use of the <head> and <body> tags and it also allows for the definition of text characteristics with the use of the <h1> tag that implies some automatic formatting (large bold font) and with the use of attributes (align=center) that further modifies the default behavior of the tag. Compare the relative simplicity of the HTML tags with the following RTF example:

Partial RTF example viewed in Notepad:

```
\pard\plain \qc
\li0\ri0\widctlpar\aspalpha\aspnum\faauto\adjustright
\rin0\lin0\itap0
\fs20\lang1033\langfe1033\cgrid\langnp1033\langfenp1033
{\b\fs44 Hello World!
\par
\par }
\pard \ql \li0\ri0\widctlpar\aspalpha\aspnum\faauto
\adjustright\rin0\lin0\itap0
{\fs40 This is an example of an RTF document  To view the
formatting instructions, use Notepad. To view the "rendered"
document, use a word processor like Microsoft Word 2000.
\par }}
```

RTF example viewed with Microsoft Word 2000:



The RTF file is rendered as a document, when viewed with a word processor. The formatting instructions for RTF files consist of slashes and curly braces which delimit the text.

Related to the RTF format for word processing is the SYLK format for describing and exchanging spreadsheet data. You can save files from Excel or other spreadsheets as SYLK files or you could write a program to create output in SYLK format. In the same fashion,comma-delimited data is also used for loading data into a spreadsheet or database. Consider the following examples:

Partial SYLK file viewed in Notepad:

```
ID;PWXL;N;E
P;PGeneral
P;P0
P;P0.00
P;P#,##0
P;P#,##0.00
P;P#,##0_);;\(#,##0\)
P;P#,##0_);;[Red]\(#,##0\)
P;P#,##0.00_);;\(#,##0.00\)
P;P#,##0.00_);;[Red]\(#,##0.00\)
. . . more code . . .
F;SDM6;R1
C;Y1;X1;K"NAME"
C;X2;K"IDEA"
C;X3;K"LIMIT"
C;Y2;X1;K"Cindy"
C;X2;K"BOOKS"
C;X3;K60
. . . more code . . .
```

SYLK file viewed in Excel

	A	B	C
1	**NAME**	**IDEA**	**LIMIT**
2	Cindy	BOOKS	60
3	Sarah	JEWELRY	25
4	Lee	BIKE GEAR	75
5	Dan	SOFTWARE	50
6	Nicole	CLOTHES	30

CSV example viewed in Notepad:

```
"NAME","IDEA","LIMIT"
"CINDY","BOOKS","60"
"SARAH","JEWELRY","25"
"LEE","BIKE GEAR","75"
"DAN","SOFTWARE","50"
"NICOLE","CLOTHES","30"
```

CSV example viewed with Excel

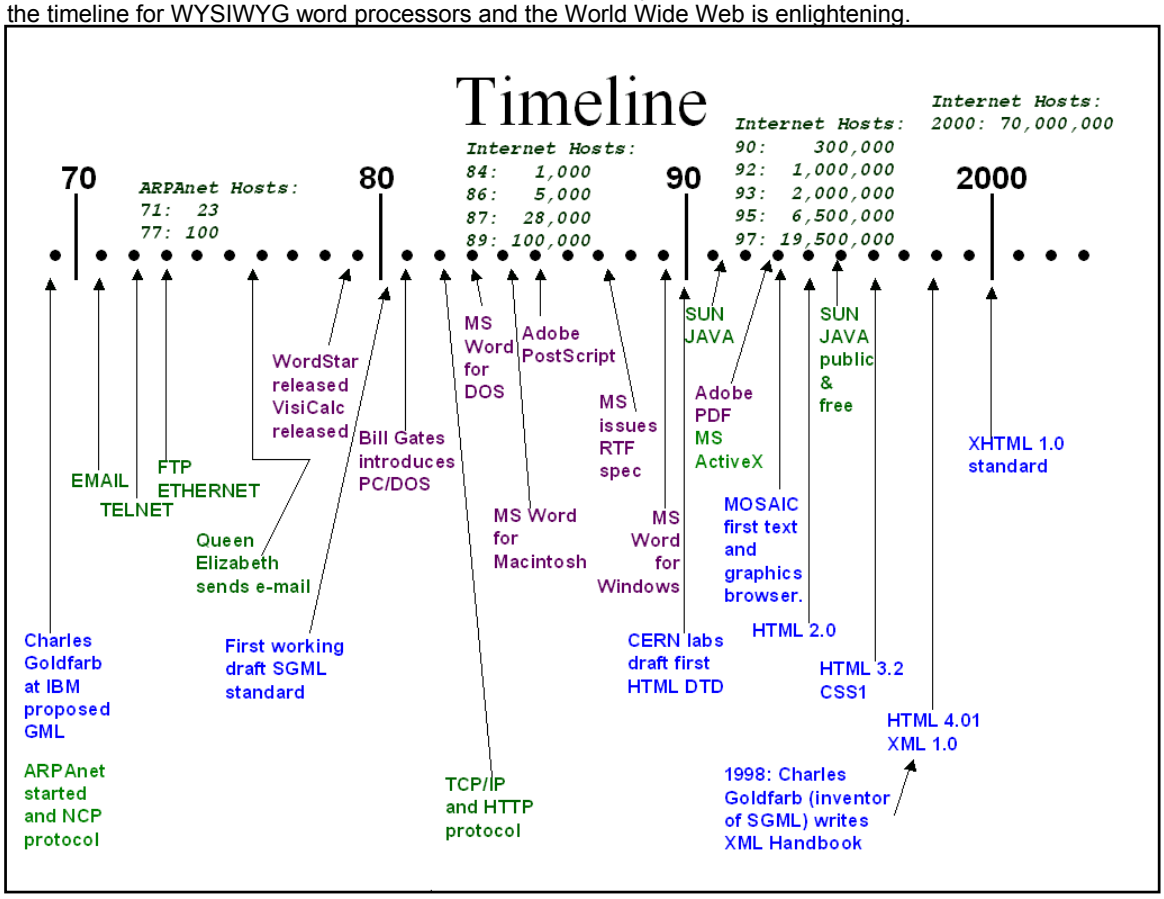
	A	B	C
1	NAME	IDEA	LIMIT
2	CINDY	BOOKS	60
3	SARAH	JEWELRY	25
4	LEE	BIKE GEAR	75
5	DAN	SOFTWARE	50
6	NICOLE	CLOTHES	30

Clearly, the CSV file is more readable than the SYLK file. Comma-delimited files can be used with spreadsheet programs, such as Microsoft Excel, Quattro Pro or Lotus or database programs such as Microsoft Access or Paradox.

Both the SYLK file and the CSV file are rendered as a table of rows and columns when viewed with Microsoft Excel. We expect these files to be rendered as shown in the screen shots. We've become so attuned to seeing the "rendered" output that the underlying formatting instructions recede in importance. But the fact is that somebody had to write the program code to drive the viewer, browser, spreadsheet and word processing programs that render all of these "marked up" files.

A MARKUP TIMELINE

SGML is the foundation on which HTML (Hypertext Markup Language) and XML (Extensible Markup Language) are built. But before we take a look at HTML, XML or other types of markup, the timeline of SGML compared to the timeline for WYSIWYG word processors and the World Wide Web is enlightening.



Initially, work on markup languages started at IBM (in 1969) with the design of a "generalized" markup language (GML), which would define documents by their structural components: sections, lists, paragraphs, headers, titles,

etc. The purpose for a generalized markup language was to allow people to define documents that could be printed and easily moved across platforms. The whole idea was to have a single document that could be rendered correctly on a printer, a typesetter or a computer terminal without being re-typed or re-created for each device. Thus, the document content or information would exist independently from the method used for input or the device used for output and the content or information would also be portable across different computer platforms.

We live in double-click computer world. When we double click on files now, our personal computers know what application has to be launched to open the file. When Charles Goldfarb, at IBM started working on the GML specification, it was in the Sixties when the personal computers, the Apple Operating System, Linux Operating System or Windows Operating System had not been invented yet. In fact, by the time the SGML standard was finalized (1986), personal computers were just becoming popular.

As personal computers rose in usage, the idea of WYSIWYG (What You See Is What You Get) tools like VisiCalc, WordStar, Lotus, Microsoft Word and Microsoft Excel seemed like they were going to make the concept of markup obsolete. But there were still cross-platform issues: how did you move a document created on UNIX to a Macintosh or Windows platform? Eventually WYSIWYG applications were enhanced so they could read files created on other platforms. The RTF specification for marking up word processing documents did not come into existence until several years after the advent of Microsoft Word. The RTF specification was designed as a way to make word processing documents portable across different computer platforms.

When the RTF specification was proposed, in 1987, there were only 28,000 Internet Hosts and Tim Berners-Lee at the CERN labs had just proposed the first draft of the HTML document type definition. At that time, the wave of the future seemed to be in integrated office applications and the paperless office, not integrated web server applications. Although it took 4 years (from 1990 - 1994) for the HTML 2.0 specification to be adopted, HTML revolutionized content delivery when it hit the World Wide Web. By the mid-nineties, there were over six million Internet Hosts. HTML as a means of delivering content was rising in prominence over proprietary word processing formats.
But companies had a huge investment in office integration applications. Spreadsheets and word processing documents are ubiquitous. Frequently, publishing content on a company's intranet meant having multiple versions of the content -- one for publishing on paper or for printing and another version for publishing to the Web in HTML format. Sound familiar? This is a perfect example of the old phrase that "history repeats itself" because again the issue is how to avoid having multiple copies of content in multiple formats and of needing to print, display and store the content with multiple devices or on multiple platforms.

So, we're back to the same problem that led Charles Goldfarb and IBM to come up with the GML and SGML standards. What's the answer this time around, 30 years later? We find a clue to the answer in the October 2002 press release by Microsoft that Jean Paoli was going to discuss XML integration into Office 11 at the 2002 XML Conference & Exposition.

XML stands for Extensible Markup Language and it is being proposed as the way to markup content and data in such a fashion that the content can be delivered in various forms by XML-compliant applications.

XML: EXTENSIBLE MARKUP LANGUAGE

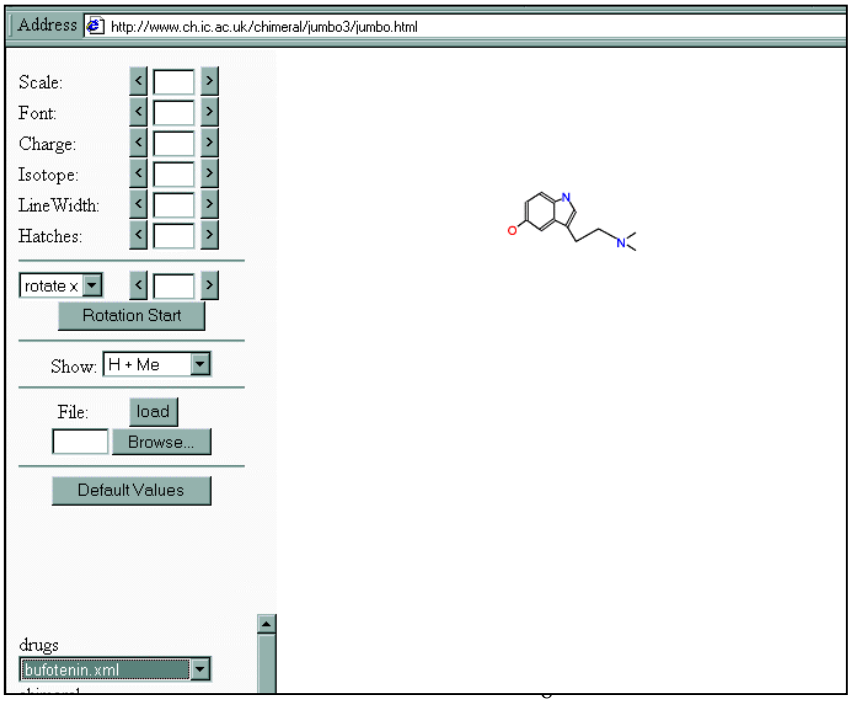
Extensible Markup Language is related to both SGML and HTML. It's related to SGML in that both HTML and XML are considered subsets of SGML. But HTML is a rather static SGML implementation -- with fixed tags and focus primarily on the presentation of content in a browser rather than the meaning or context for the content. XML , on the other hand, has little or no focus on presentation. XML is all about context and meaning. This allows people to use XML to create custom markup tags (or "tagsets") designed for their specific needs, instead of trying to force their application to fit within the constraints of HTML or their document within the constraints of a word processing program.

For example, imagine a bunch of chemists and chemistry teachers trying to design a web site that displayed the structure of molecules. Their choices for an HTML implementation might involve having a picture or image of every molecule and then displaying that static picture on demand. But that's a tedious process -- and there are a lot of molecules! Better would be a way to describe the molecules and the molecular structure and have a program draw the molecule dynamically for a chemist or student. That's not do-able with HTML. However, given the expertise to write the "drawing" program and some agreement on how to represent a molecule it is do-able with XML. In fact, the CML or ChemML standard and the JUMBO browser accomplish the above task. Consider the ChemML file that describes the bufotenin molecule:

Partial bufotenin.xml file:

```
<molecule convention="MDLMol" id="bufotenin" title="BUFOTENINE">
  <date day="22" month="11" year="1995">
  </date>
  <atomArray>
    <atom id="a1">
      <string builtin="elementType">C</string>
      <float builtin="x2">-1.6045</float>
      <float builtin="y2">0.1596</float>
    </atom>
    <atom id="a2">
      <string builtin="elementType">C</string>
      <float builtin="x2">-1.6045</float>
      <float builtin="y2">1.6675</float>
    </atom>
    <atom id="a3">
      <string builtin="elementType">C</string>
      <float builtin="x2">-0.525</float>
      <float builtin="y2">-0.3738</float>
    </atom>
    <atom id="a4">
      <string builtin="elementType">C</string>
      <float builtin="x2">-2.7679</float>
      <float builtin="y2">-0.6972</float>
    </atom>
    <atom id="a5">
      <string builtin="elementType">N</string>
      <float builtin="x2">-0.525</float>
      <float builtin="y2">2.1043</float>
    </atom>
```

Following this standard for representing molecule information, now allows programmers to code a viewer to enable the display of the bufotenin.xml file in a graphical format. One such browser program was the JUMBO plug-in for Internet Explorer. For more information about CML or other plugins, see:
http://www.xml-cml.org/information/position.html
When displayed using the Jumbo plug-in, the bufotenin.xml file looks considerably more usable:



This is just one example of what is possible with XML. Similar to ChemML is MathML, which provides a way for math expressions to be included in Web pages. Both of these XML tagsets highlight the requirements for using XML successfully:

1) A specific group or community had a specific need that could not be readily met with word processor documents or HTML documents;
2) That group or community used XML to define a markup language specification (called a DTD or XML Schema) that was designed to meet their needs;
3) An application or program was designed (and written) to display or process XML files that conformed to this agreed-upon specification.

CASE STUDY: LIST OF GIFT IDEAS

The molecule example is a specialized, scientific example. Let's focus now on a simpler, non-scientific information need. I have a list of gift ideas that I keep. It's very simple, just names, the kind of thing the person likes to get and a money amount. Here's the file with no markup at all:

```
CINDYBOOKS$60SARAHJEWELRY$25LEEBIKEGEAR$75DANSOFTWARE$50NICOLECLOTHES$30
```

Now, here's the file with line breaks and spaces added to the file.

```
CINDY BOOKS $60
SARAH JEWELRY $25
LEE BIKE GEAR $75
DAN SOFTWARE $50
NICOLE CLOTHES $30
```

So if I just put that gift idea list on a piece of paper and fold it into my wallet, I'll have a list and I can keep adding to it. But what do I do when I want to share the list with my husband or my brother? Now I have to type the list into e-mail and send it to them. Or type it into a spreadsheet and send the spreadsheet file to them. But my husband has an old Macintosh computer with ClarisWorks and my brother uses Quattro Pro. So I can't just send either of them an Excel file. So, I think I'll send them a comma-delimited file. Here's what my gift idea file looks like with commas and quotes added to it:

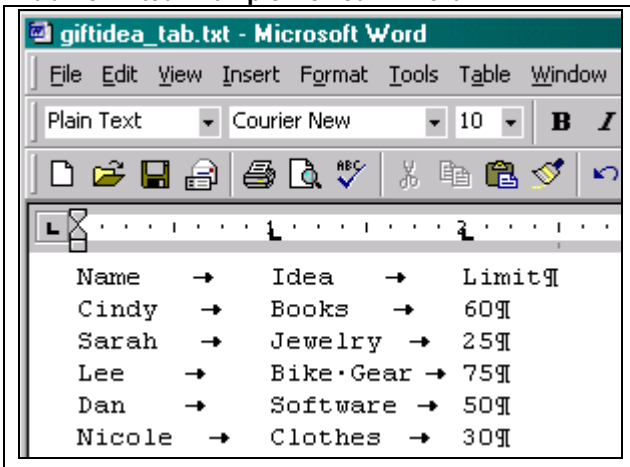
```
"CINDY","BOOKS","60"
"SARAH","JEWELRY","25"
"LEE","BIKE GEAR","75"
"DAN","SOFTWARE","50"
"NICOLE","CLOTHES","30"
```

What would make the file immediately usable in a spreadsheet program? Here's the file with the addition of some header information:

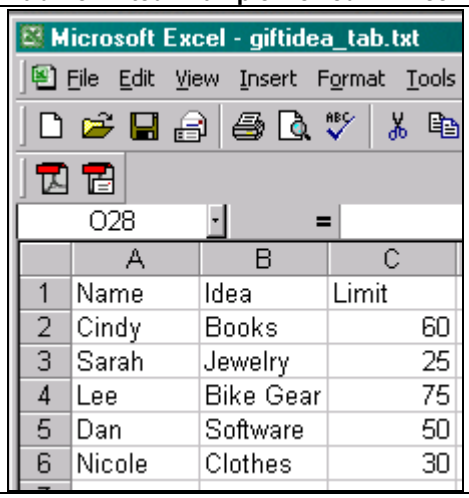
```
"NAME","IDEA","LIMIT"
"CINDY","BOOKS","60"
"SARAH","JEWELRY","25"
"LEE","BIKE GEAR","75"
"DAN","SOFTWARE","50"
"NICOLE","CLOTHES","30"
```

In fact, this is the file that we've already seen earlier in the paper and the screen shot seen earlier shows how the file looks when opened with Excel. This CSV file can be used with Excel, Lotus, Quattro Pro or ClarisWorks. And, given that I might want to do some analysis on my gift list, I could even read this file into SAS using either PROC IMPORT or a DATA step program. On the other hand, if I wanted to send a file that could be opened in either a word processor or a spreadsheet program, I might want to create a tab-delimited file. The hexadecimal character constant '09'x is interpreted as a tab for both Microsoft Word and Microsoft Excel. If I change my CSV file to have tab characters, then I could open the file in either Word or Excel as shown below.

Tab-Delimited Example viewed in Word:



Tab-Delimited Example viewed in Excel:



Last, but not least, if I wanted to share my gift idea file with other people and if I had my own web site, I could create an HTML file, or even an XML file. So I have a lot of methods for sharing my gift idea file. Let's look at the last two: HTML and XML.

An HTML file is a possible choice for gift idea delivery, but for this method to work, I have to have access to a web server on which to store my HTML version of the file and the people who are going to need access to the file have to be able to get to my web site in order to retrieve the HTML file. These same requirements are also true of any XML version of my gift idea file. If I created an HTML document from my gift idea list, it might look something like this:

HTML Tags:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2
Final//EN">
<html>
<head>
<title>My Gift Idea List</title>
<link rel="stylesheet" href="mystyle.css">
</head>
<body bgcolor="#66ccff">
<div align="center">
<h1>My Gift Idea List</h1>
<table border="1">
<tr>
<th>Name</th><th>Idea</th><th>Limit</th></tr>
<tr>
<td>Cindy</td><td>Books</td><td>$60</td></tr>
<tr>
<td>Sarah</td><td>Jewelry</td><td>$25</td></tr>
<tr>
<td>Lee</td><td>Bike Gear</td><td>$75</td></tr>
<tr>
<td>Dan</td><td>Software</td><td>$50</td></tr>
<tr>
<td>Nicole</td><td>Clothes</td><td>$30</td></tr>
</table>
</div>
</body>
</html>
```

HTML File viewed in Internet Explorer:

Although the HTML file is very plain in definition, there is nothing in the file itself that identifies the fonts or colors to be used by my browser. The formatted browser view uses a second file, a cascading stylesheet (.CSS) file that is specified in the <head> section of my document with a <link> tag. We already know that I could read the CSV

or tab-delimited files into SAS. But, although it is possible, it's not likely that I would undertake writing a program to read this HTML file into SAS. While this HTML file might be possible to read into SAS, only the <title> or <h1> tags have any information about this file's intended usage. If I was very ambitious, I might decide to teach myself how to write an XML application using an XML file as shown below. What this means, however, is that first I have to design the tags that I will use to delimit the information or content in the file.

XML file created in Notepad

```
<?xml version="1.0"?>
<gifttable>
  <giftidea>
    <Name>Cindy</Name>
    <Idea>Books</Idea>
    <Limit>$60</Limit>
  </giftidea>
  <giftidea>
    <Name>Sarah</Name>
    <Idea>Jewelry</Idea>
    <Limit>$25</Limit>
  </giftidea>
  <giftidea>
    <Name>Lee</Name>
    <Idea>Bike Gear</Idea>
    <Limit>$75</Limit>
  </giftidea>
  . . . more tags . . .
</gifttable>
```

XML file viewed in Internet Explorer

```
<?xml version="1.0" ?>
- <gifttable>
  - <giftidea>
      <Name>Cindy</Name>
      <Idea>Books</Idea>
      <Limit>$60</Limit>
    </giftidea>
  - <giftidea>
      <Name>Sarah</Name>
      <Idea>Jewelry</Idea>
      <Limit>$25</Limit>
    </giftidea>
  - <giftidea>
      <Name>Lee</Name>
      <Idea>Bike Gear</Idea>
      <Limit>$75</Limit>
    </giftidea>
  + <giftidea>
  + <giftidea>
  </gifttable>
```

We can see from this example that the XML version of the gift idea file now contains some tags that describe the data. The XML processing instruction at the top of the file is required to tell the browser which level of XML is being used inside the file. The "root" or enclosing tag is the <gifttable> tag and each row in the file is delimited by the <giftidea> tag. Then, the <Name>, <Idea> and <Limit> tags delimit the gift information itself. But the view in the browser is almost the same as the view in Notepad because I have not made use of any other technology to transform the XML file into any other format (such as HTML). I can transform the way the XML file looks, however, if I use XSL (Extensible Stylesheet Language) to transform the XML from the "vanilla" XML format to HTML format. I can even add some bells and whistles to the transformed file:

XML file with XSL instruction added

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="GiftTransform.xsl"?>
<!--XML Demo-->
<gifttable xmlns="x-schema:GiftSchema.xsd">
 <giftidea>
  <Name>Cindy</Name>
  <Idea>Books</Idea>
  <Limit>$60</Limit>
 </giftidea>
 <giftidea>
  <Name>Sarah</Name>
  <Idea>Jewelry</Idea>
  <Limit>$25</Limit>
 </giftidea>
 <giftidea>
  <Name>Lee</Name>
  <Idea>Bike Gear</Idea>
  <Limit>$75</Limit>
 </giftidea>
 . . . more XML tags . . .
</gifttable>
```

XML Schema that defines tags:

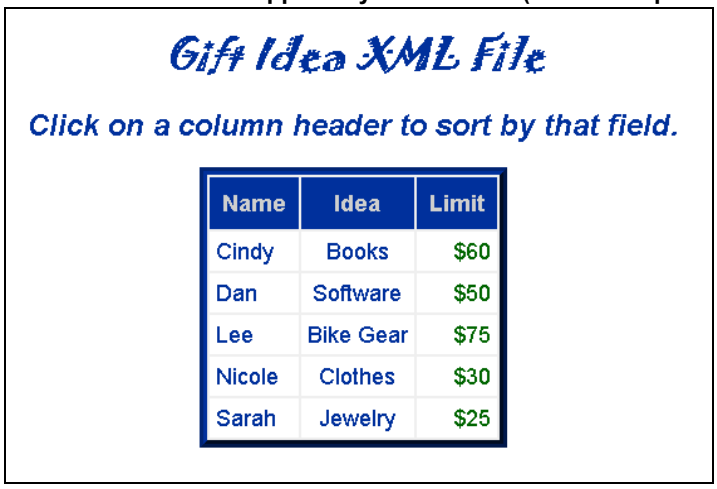
```
<?xml version="1.0"?>
<Schema name="GiftSchema" xmlns="urn:schemas-
microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
<ElementType name="Name" dt:type="string"/>
<ElementType name="Idea" dt:type="string"/>
<ElementType name="Limit" dt:type="string"/>
<ElementType name="giftidea" content="eltOnly">
<element type="Name" minOccurs="1"
maxOccurs="1"/>
<element type="Idea" minOccurs="1"
maxOccurs="1"/>
<element type="Limit" minOccurs="1"
maxOccurs="1"/>
</ElementType>
<ElementType name="gifttable" content="eltOnly">
<element type="giftidea" minOccurs="1"
maxOccurs="*"/>
</ElementType>
</Schema>
```

XSL file to transform from XML format to HTML format:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
        <html>
        <head>
        <link rel="stylesheet" href="giftstyle.css"/>
         <title>Gift Idea XML</title>
        </head>
        <SCRIPT>(insert Script for sorting)</SCRIPT>
        <body>
         <h1>Gift Idea XML File</h1>
         <h2>Click on a column header to sort by that field.</h2>
           <DIV id="listing"><xsl:apply-templates match="gifttable"/></DIV>
        </body>
        </html>
</xsl:template>
<xsl:template match="gifttable">
         <table align="center">
        <thead><tr>
        <th><DIV class="head" onClick="sort('Name')">Name</DIV></th>
        <th><DIV class="head" onClick="sort('Idea')">Idea</DIV></th>
        <th><DIV class="head" onClick="sort('Limit')">Limit</DIV></th>
        </tr></thead>
        <xsl:for-each select="giftidea" order-by="Name">
        <tr>
        <td><DIV class="name"><xsl:value-of select="Name"/></DIV></td>
        <td><DIV class="idea"><xsl:value-of select="Idea"/></DIV></td>
        <td><DIV class="limit"><xsl:value-of select="Limit"/></DIV></td>
        </tr>
        </xsl:for-each>
         </table>
</xsl:template>
</xsl:stylesheet>
```

Now, when I display the new XML file (giftidea_transform.xml) in the browser, I see a very different view. That is because the browser used the instructions in the XSL file to transform the content. In addition, because the XSL file inserted script language for sorting into the HTML file, this version of my gift idea list has some interactivity built in. A person can click on any of the header cells to sort the table by that cell (shown in the screen shot sorted by the Name column):

XML file with XSL file applied by the browser (Internet Explorer):



The ability to do this kind of transformation on an XML file is only part of the markup buzz. But as you can see, XML requires a steep learning curve because it is not one technology, but a family of technologies and new standards. The applications vary widely from the concept of ChemML to my personal GiftML XML tagset. And we still have to answer the question of where SAS fits in the whole picture.

SAS AND MARKUP

SAS intersects with markup in two ways: 1) through the SAS XML Libname engine (SXLE) to import from XML into SAS format or to export from SAS data set format to XML format or 2) with ODS MARKUP to create various types of markup files from procedure or data step output. The method that you use to create markup files depends on the disposition of the file. If you are publishing reports and document content, then you might use ODS HTML, ODS RTF, ODS CSV or ODS MARKUP to create marked up content. Other types of markup files that you can create include, but are not limited to: troff, LaTeX, DocBook, WML, HTML3, and Python. On the SAS web site, there is also a contributed SYLK tagset that you can use with ODS MARKUP to create SYLK files to open with a spreadsheet program.

If, on the other hand, you are dealing with XML as part of a data base or data exchange system, and you need to either create markup files from SAS data sets for data exchange or to import XML files into SAS data set format, you might use the SAS XML Libname engine.

The key to using either the SAS XML Libname engine or ODS for markup is to understand what set of markup tags you need to generate. In either case, the method by which tagsets are created for use with either the Libname engine or ODS MARKUP is with PROC TEMPLATE. Beginning with SAS 8.2 tagset templates are stored in the in the SASHELP.TMPLMST item store. You can use PROC TEMPLATE to modify these tagsets or to create your own tagset. But that's a topic for another paper. Suffice it to say that SAS has the means for you to write virtually any kind of markup that you need. The different situations that you could be faced with as a SAS programmer are that you could need to import an XML file into SAS (use the SAS XML Libname engine); you could need to export a SAS data set into XML format; or, you could need to produce SAS report output in XML or other markup format.

IMPORT FROM XML FORMAT INTO SAS

If you need to import a very rectangular XML file into a SAS data set, it is as easy as importing a CSV file or other "raw text" file into SAS. You could even use PROC COPY, but we'll show you the DATA Step method.

GIFTIDEA.XML

```
<?xml version="1.0"?>
<gifttable>
<giftidea>
<Name>Cindy</Name>
<Idea>Books</Idea>
<Limit>$60</Limit>
</giftidea>
<giftidea>
<Name>Sarah</Name>
<Idea>Jewelry</Idea>
<Limit>$25</Limit>
</giftidea>
<giftidea>
<Name>Lee</Name>
<Idea>Bike Gear</Idea>
<Limit>$75</Limit>
</giftidea>
 . . . more tags . . .
</gifttable>
```

DATA Step Program

```
libname gift xml 'giftidea.xml';
data work.sasgift;
  length Name $12 Idea $15 SpendLimit 8;
  set gift.giftidea;
  SpendLimit = input(Limit,comma6.);
  keep Name Idea SpendLimit;
  label Name = 'Name'
        Idea = 'Gift Idea'
         SpendLimit = 'Limit';
  format SpendLimit dollar6.;
run;
libname gift clear;

proc print data=work.sasgift;
title 'Import from XML into SAS';
run
```

If you receive an XML file that is not rectangular in structure, then you can use the XMLMap facility to provide the transformation required. Other SUGI presentations cover the use of the XML Libname engine and the XMLMap facility.

EXPORT FROM SAS TO XML FORMAT USING THE SAS XML LIBNAME ENGINE

If you need to export a SAS dataset to XML format, this too is possible with the SAS XML Libname engine. Types of markup that you can create are generic XML, Oracle-specific XML, or even HTML. You can use PROC COPY or a DATA Step program. Consider the following program that calculates a new variable and creates 3 markup files:

```
libname oraout xml 'c:\temp\orafile.xml' xmltype=oracle;
libname htmlout xml 'c:\temp\plainhtml.html' xmltype=html;
libname genout xml 'c:\temp\genxml.xml';

data oraout.gifttable
     htmlout.gifttable
     genout.gifttable;
  set work.sasgift;
  length NextYear 8;
  NextYear = SpendLimit * 1.05;
  format NextYear dollar6.;
run;

libname oraout clear;
libname htmlout clear;
libname genout clear;
```

The XMLTYPE option defines the kind of tagset or markup type that should be created. With XMLTYPE=ORACLE, SAS uses a predefined set of tags that conform to the Oracle XML specification. Without an XMLTYPE specified, SAS uses a generic set of XML tags to produce the output.

ROUTE SAS PROCEDURE OR DATA STEP OUTPUT TO MARKUP FORMAT

If your need is to create marked up output from SAS procedure or DATA Step output, then the basic ODS "sandwich" technique is where you start. You can create output that spans the continuum from the simple…

```
ods html3 file='simpleods.html' style=sasweb;

. . . sas code that produces output . . .

ods html3 close;
```

…to the complex:

```
proc template;
  define tagset tagsets.mymarkup;
  . . .more proc template code . . .
  end;
run;

ods tagsets.mymarkup file='mymarkup.???';

. . . sas code that produces output . . .

ods tagsets.mymarkup close;
```

There are several "flavors" of HTML tagsets that SAS provides in the SASHELP.TMPLMST item store. Test out the HTML related tagsets and pick the one that suits your needs. Most of the HTML-related tagsets produce HTML 4.0 compliant markup. If you need to explicitly create HTML 3.2-compliant markup, then use the ODS HTML3 statement in SAS 9 (which produces the same kind of markup as with SAS 8).

There are also several flavors of CSV (comma-separated value) tagsets: CSV, CSVALL and CSVBYLINE. As the names of the tagsets imply, they either produce only the data and header values (CSV); the data and header values plus titles, notes and bylines (CSVALL) or just the data, headers and BYLINE information (CSVBYLINE). In addition, there are the more esoteric tagsets, like troff, LaTex, DocBook, and PYX to name just a few. There are also user-contribued tagsets, such as the SYLK tagset contributed by Jack Hamilton (a modified version of this tagset was used for the example in this paper).

And, with SAS 9, there are the Microsoft-related tagsets. The first is tagsets.msoffice2K that creates an HTML file conforming to the Microsoft Office specification. The other tagset is tagsets.ExcelXP that creates an XML file. The XML file conforms to the Excel XML specification and you must use Excel 2002 or higher to open the XML file created by this tagset.

All of this functionality is available to you with Base SAS. One advantage of the ODS MARKUP destination is that the developers can update tagsets or add tagsets without waiting for a new version of SAS to be released. If you have not already discovered the Base SAS community at support.sas.com, let me highly recommend these sites to you. For more information about the SAS XML Libname engine, go to: http://support.sas.com/rnd/base/topics/sxle90/ and for information about XML Atlas, go here http://support.sas.com/rnd/base/topics/sxle90/#atlas for information about a plug-in to help you generate XML Maps for use with the SAS XML Libname engine.

For more information about the ODS MARKUP tagsets, go to:
http://support.sas.com/rnd/base/topics/odsmarkup/markup_stmt.html To download updated tagsets for SAS 8.2, SAS 9.0 and SAS 9.1, visit this site: http://support.sas.com/rnd/base/topics/odsmarkup/
For more information about PROC TEMPLATE for tagset creation and modification, consult this site:
http://support.sas.com/rnd/base/topics/odsmarkup/tagsets.html There is a wealth of information and many examples at the above sites. They are invaluable references when starting out with tagsets.

CONCLUSION

The whole concept of markup and markup languages has been around since the 1960s. The current hot topic in the markup world is XML (Extensible Markup Language). SAS has the tools to do just about anything you need to do in the markup arena including, but not limited to XML markup. Back in the Sixties, the concept of SGML was intended to simplify information processing and information exchange. We're now in a new century and still reaching for the brass ring in terms of generating information in various formats. With SAS and all the various forms of Markup available, we're a little closer to reaching this goal. Not only does history repeat itself, but sometimes, as in the case of XML, we learn from the past and improve on it.

REFERENCES

Proceedings of the Annual Conference of the SAS Users Group International. Cary, NC: SAS Institute Inc.
Friebel, A: "<XML> at SAS® — A More Capable XML Libname Engine" SUGI 27
Gebhart, E: "ODS Markup: The Power of Choice and Change" SUGI 27

ACKNOWLEDGMENTS

Grateful thanks go to Chevell Parker, Eric Gebhart, Tony Friebel, Wayne Hester, David Kelley, Dan O'Connor and Sandy McNeill for patiently explaining the intricacies of ODS and how "it all depends on the destination." For editorial assistance and proofing, I would like to thank Jane Stroupe, Christine Riddiough, Michele Ensor and Linda Jolley.

CONTACT INFORMATION

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

Cynthia Zender
SAS Institute, Inc
Denver Regional Office
Work Phone: 303-290-9112 ext 1738
Email: Cynthia.Zender@sas.com

Before you get started with XML, in particular, you should use this checklist to help you zero in on which pieces of SAS you should use.

Getting Started With Markup	
Question/Issue	**Notes/Comments**
Who is your company Webmaster?	The webmaster is probably the person with the answers to the rest of these questions. Or the webmaster will know where to send you for information.
Is your company using XML files? Is your company in the evaluation process? Do you receive XML files that have to be transformed to some other format?	It's possible that your company just receives XML files in some format and those files need to be converted to some other format (like SAS). Your company could be involved in a B2B (business-to-business) relationship with another vendor and the XML is a behind-the-scenes part of the process.
If yes, are they using XML for data transfer or exchange?	Yes answer points to SAS XML Libname engine. If, on the other hand, your company is using XML for document or content transformation, the answer could point to ODS MARKUP.
Does your company belong to an industry or group that has published an XML DTD or XML Schema for a particular kind of markup?	You might have to create files that conform to this industry DTD or Schema. This means you'll probably use PROC TEMPLATE to define a custom tagset. Remember the ChemML tagset -- a specific industry may or may not have an application to process their marked up content. Perhaps your industry uses a standard format, such as the ORACLE format or the Excel XML specification.
Does your company use a data base system (like Oracle) that has XML import capability?	You might be able to use the XML Libname engine with one of the pre-defined XML tagsets
If your company is using XML, what other XML technologies are they using? XSL? XLink? XQuery? What other "web" technologies are they using? Java? ASP? ActiveX? Perl? PHP? Python?	If your company is using XML for content or report delivery, they may be using Java and Java Server Pages or ActiveX and Microsoft .NET technology to deliver content. If your company is using XML for data transfer, they may be using some other application to "hide" the underlying XML from most users.
If your company is not using XML, is there another kind of markup that is being used? If so, what kind? HTML, CSV? Other kinds? PDF files? PostScript files? RTF?	Your company may not be using XML, per se. SAS can still help you deliver content in web-friendly format. SAS delivers HTML 4.0 markup, CSV markup and XML markup via the ODS MARKUP facility. ODS RTF, ODS PDF (and other printer-family destinations) are considered non-markup destinations.
If your company is publishing content in HTML format, what version of HTML files is your company standard? (HTML 3.2 or HTML 4.0)	If you need to deliver HTML files, you have to know what level of HTML is acceptable for your company web site. SAS 8 produces HTML 3.2 markup; SAS 9 produces HTML 4.0 markup by default. With SAS 9, you can still produce HTML 3.2 markup by using the ODS HTML3 destination.
How do HTML files get to the company web server? What is the URL for your company web server? Does the webmaster allow you to write to the company web server? Do you have to put the files in a location where someone else transfers them to the web server?	Generally speaking, webmasters do not give many users write access to the company web server. Webmasters have opinions about who and what should be written to the company web server. If you are creating HTML reports for consumption either inside or outside your company, you should find out where the files need to live and the directory structure on the web server where they will ultimately reside. This will save you a lot of heartache if you have to subsequently create files with server references (HTTP URL: http://www.server.com/reptdir/acct/somefile.html) instead of physical file locations (c:\myreports\somefile.html)
Is your company using cascading style sheets?	If your company is already using CSS files, they have color and font standards already in place. Do not reinvent the wheel. Use ODS to access a corporate CSS file.
Do you have to deliver content in some format other than HTML? Is it important for your report users to edit the files you give them? Is it important for your report users to just print the files you give them?	If your users want to edit files, use ODS RTF or ODS CSV to deliver content. Be aware of the fact that many applications can open HTML files. If printing is important, consider ODS PDF. Remember that, generally speaking, PDF files cannot be edited after they are created.