This paper contains three parts. The first part, "An Introduction to SGML," provides an overview of SGML concepts. The second part, "SAS and SGML," describes some of the techniques that we have developed to convert SAS output to SGML. The third part, "A DTD for SAS Output," proposes a common DTD for SAS output which could be used to convert SAS output to SGML or other proprietary formats.

AN INTRODUCTION TO SGML

In December 1986, the International Standards Organization (ISO) issued a standard for document representation (ISO 8879). This standard is the Standard Generalized Markup Language (SGML).

SGML is an open standard. It is not the property of any vendor or government. As its name suggests, SGML is a language used to define markup systems that represent the content and structure of machine readable data. SGML markup systems usually do not contain information about document format.

Formatting provides clues about a document's structure. The size of a heading, indentation, page breaks tell the reader how a document is organized. SGML can also be very direct. Compare the markup used by a popular word processing program, a public domain composition tool, and SGML (Example 1). There is no doubt where a chapter or title begins in the SGML example.

In the past, SGML has been used to encode printed documentation, but today SGML applications also store and exchange many types of text-based information in electronic form. This information, which could be financial, legal, or multimedia, may never be printed. It is tagged and stored in SGML because of SGML's unique ability to identify and retrieve information.

SGML markup can be simple or complex. Because you can build your own markup system, you decide how much intelligence, value, and complexity to add to your data. This added value is limited only by your imagination.

Retrievability is not the only benefit that you can gain with SGML. Your data gains value because it is reusable and portable. SGML does not require any information about how a document is to be formatted. This allows you to move your documents easily between hardware platforms and operating systems. You can take advantage of the formatters or viewers available for a target platform to present the SGML document.

If you are about to select or convert to another publishing system, carefully consider the current state of document presentation technology. We are no longer in a paper-based world and we cannot be sure what presentation technologies will appear in the next five years.

In the 70's and early 80's batch composition was the leading choice for creating large documents. These documents were marked up with formatting codes and submitted to mainframe computers running formatting programs. The resulting typesetter driver files were then routed to specialized imaging equipment that produced fully paginated, camera-ready document pages. These systems were fast, efficient, and proprietary.

In the 80's, "what you see is what you get" or WYSIWYG desktop publishing systems emerged. This technology made it much easier for small companies to produce documentation.

Unfortunately, each publishing package vendor used a different set of formatting markup. These proprietary systems made it difficult and expensive to move documentation from one system to another. To make matters worse, as publishing systems became obsolete, companies were forced to convert their documents to a new formatting system. Although WYSIWYG systems are easy to use, and it is fun to see what your document will look like while you are creating it, they do not produce structured reusable documents.

Today, the advantages of SGML are becoming apparent. New vendors are appearing with products that support SGML. Vendors of proprietary systems are providing SGML extensions to import and export SGML (with varying degrees of success), or are advocating new portable (and still proprietary) systems for producing documentation.

Both format-based batch composition and WYSIWYG technologies primarily produce paper documents. They are proprietary which makes it difficult to reuse a document, or parts of it, in a different presentation media, or to move that document to another hardware platform or operating system.

SGML, on the other hand, enables you to markup your data in a manner that both adds to its value and ensures that it remains valid and usable in the future.

Is SGML the right choice for every document? In a word, no. The overhead of SGML cannot be justified for simple documents, such as a business letter created for a single use. However, if you publish a large number of similar documents or if you want publish your documents in a variety of formats, then SGML could be your publishing solution.

What are the Components of SGML Documents?

When you look at an SGML document that is marked up with a rich tag set for the first time, you may feel overwhelmed. The number of tags can be intimidating if you are accustomed to working with a format-based WYSIWYG product. Even though the application may hide the markup from you, it is still there. As Example 1 illustrated, it is not hard to read and understand SGML markup. If you do not want to see SGML tags while you work, there are authoring systems that hide the markup as well as most WYSIWYG systems. Most people who use such systems eventually choose to view the tags because they are good document navigation aids.

The number of tags in an SGML document is directly related to the amount of unique information to be accessed in the document. With proper consideration given to the design of an SGML tag set, tags can be easily understood by people as well as machines. Proper design might not reduce the number of tags required, but it will make them easier to understand and use. Modern SGML authoring systems guide writers through the document structure and only present tags that are valid in the current context. The author can select tags from a list containing only valid tags in alphabetical order. As the content changes, so does the tag list. Some systems provide background parsers that prevent an author from inserting an invalid tag.

There is more to SGML than just a tag set. An SGML document, called a Document Instance (DI), does not stand alone. It requires three other components to be valid. These are the SGML...
declaration, the Document Type Definition (DTD), and the parser.

SGML Declaration

The SGML declaration tells the SGML application about the DTD and the document. It defines the document character sets, and the delimiters that contain the SGML tags. It also sets limits on variables used by the SGML parser. SGML declarations are not unique to single documents. Generally, one declaration can be used for every document created within a corporation.

Document Type Definition

The Document Type Definition (DTD) is the set of structure rules for a class of documents. These rules are expressed in the DTD as a tag set definition for the document class. Within each tag declaration a content model defines what the tag can contain. Content models are made up of other tags with notation that shows whether the tag is required or optional, the order of the tag, and the number of times the tag can occur. Here is a simple DTD that illustrates how the hierarchy of the document structure is defined.

```
<!DOCTYPE book [
  <!ELEMENT book (title, chap*)>
  <!ELEMENT chap (title, topic*)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT para (#PCDATA|title)>
  <!ELEMENT code (#PCDATA)>
]>  
```

This sample DTD describes the content model for a document. The first line defines the root, or DOCTYPE, of this class of document. It identifies the tag book that must contain the entire document. The actual definition of book is on the second line. The keyword ELEMENT identifies the character string "book" as a markup tag. The two hyphens separated by a blank following the tag name are markup minimization toggles. If both hyphens are present, as they are in this example, then there must be a start tag and an end tag placed in the document to indicate the range of the contents of the tag. If the second hyphen were replaced with an o (not a zero), the end tag could be omitted. This is allowed only when the content to be marked is obvious and unambiguous. An example that illustrates this follows:

```
<list>
  <item>this is the first list item</item>
  <item>this is the second list item</item>
  <item>this is the last list item</item>
</list>
```

The item end tags have been omitted in this example. It is easy to see that they were not necessary. Each item ends with the start of the next item or the end of the list.

It is also possible to omit the start tag. This is allowed only when the first hyphen in the element definition is replaced with an o. This situation follows the same rules as the last. The content of the document at the point where the tag is to be omitted must be obvious and unambiguous.

Continuing with the book tag definition, its content model is enclosed in parentheses. Content models are read from left to right and in this example, book must contain a title followed by one or more chap elements. The comma between title and chap is a sequence connector which indicates the required order in which the contents must occur. The plus sign following chap indicates that one or more chap tags are allowed in the context of book. The hierarchical nature of the DTD and the document it describes is visible in the way the elements (tags) are interrelated. The chap in the book definition is defined to contain a title and one or more topics. Topic is then defined to contain a title and zero or more paragraphs or topics. The zero or more iteration is indicated by the asterisk. Notice that the para tag and topic tag within the topic content model are connected with a vertical bar, and that the occurrence indicator (asterisk) is outside of the containing parenthesis. This means that the content of topic can contain multiple para tags and topic tags that can occur in any order, following the required title.

The topic element (tag) definition also illustrates recursion. A topic is allowed to contain another topic.

The basic building block for constructing DTD's is PC DATA. It stands for Parsed Character Data which is defined by the SGML standard to be data characters. PC DATA are the leaves of the document tree. With a few exceptions, all elements in a document eventually end up containing PC DATA. This is apparent in the example DTD as the element title has a content model of PC DATA. title is referenced in several other content models (book, chap, topic). This shows how the document elements that are closer to the root actually end up containing text.

Another important part of SGML tags is attributes. Attributes are qualifiers that indicate a property of an element, other than its type. Attributes are attached to SGML tags and can provide additional information about the content of the tags. For example here is a topic tag with attributes:

```
<title id=westl os=mvs review=2>
```

These attributes could provide a unique ID for cross referencing or retrieval by a document management system, tell what operating system the topic applies to, and track the topic as it is edited and reviewed. As you can see, attributes are a powerful feature that you can use to your advantage in many ways. Consider how useful they are in linking HTML documents (an application of SGML) on the World Wide Web.

SGML tags may contain any number of attributes. When attributes are associated with tags, they are always attached to the start tags. The following example uses the same DTD that we looked at earlier. This time the DTD includes an attribute on title, a new tag, crossx, and the addition of crossx to the content model of para.

```
<DOCTYPE book []
  <!ELEMENT book (title, chap*)>
  <!ELEMENT chap (title, topic*)>
  <!ELEMENT topic (title, para|topic)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT para (#PCDATA|title)>
  <!ELEMENT code (#PCDATA)>
]>  
```

The id attribute associated with title has a default value of IMPLIED. This means that the id attribute is not required. The crossx xref attribute has a default value of REQUIRED. This means that if you place the crossx tag in a document, the attribute must contain a value. This makes sense because a cross reference without a target or destination is useless.

A document created with this enhanced tag set now has cross referencing or linking capabilities. An author can assign unique IDs to titles and can later refer to these titles with the crossx tag. Because titles are contained in topics and chapters the writer is
now able to refer the reader to a topic or chapter for additional information about a subject. In the example DTD, the CROSSREF can only be used in paragraphs. Here is an example of a cross reference:

```
<topic title_id="topic1">Mickey</topic>
<para>Walt Disney created Mickey Mouse in his early cartoons.</para>
```

```
<topic title_id="top2">Disney World</topic>
<crossref xref="top1">Mickey Mouse</crossref>
```

The output delivery application used to create the output from this SGML document could use the ID and IDREF information to make the cross reference or link. If the document is to be delivered on paper, the output delivery system might generate and insert text so that the paragraph looks like:

At Disney World, Mickey Mouse (see "Mickey" on page 7) is a recurring theme.

If the document were going to be delivered online, the same link information could be presented as a color change or underlining of the string "Mickey Mouse". If the reader clicks on the string, another window might pop up displaying the referenced text.

At Disney World Mickey Mouse, is a recurring theme.

Linking is not the only use for attributes. You can use attributes to convey characteristics about the data being marked up. You will see more examples of this in the proposed DTD for SAS output.

Parser

The final component required to produce a valid SGML document is a parser. A parser is a program that accepts the SGML declaration, the DTD and the document instance as input. It compares the use of the tags in the document to the rules established by the DTD and the declaration. If the document conforms to those rules it parses or it is a valid document. If the document violates these rules, the parser produces error messages. The errors must be corrected until the document can parse without errors. Fortunately, most SGML authoring systems parse the document as it is being written and prevent the creation of invalid documents.

In addition to producing error messages, parsers also output a validated intermediate form of the document instance called the Element Structure Information Set (ESIS). This intermediate form contains the full text of the document along with all the necessary tags. All markup that was omitted is automatically inserted by the parser, and all externally referenced text (known as entities in SGML) is inserted.

Without the parsing step it would be possible to create invalid documents. These non-conforming documents would corrupt the ordered data structures expected by other applications which access that data. Like falling dominos, problems in the document could lead to problems with the other applications. The parsing step is crucial to the success of any SGML document and its associated applications.

How Can You Use SGML applications?

Unlike proprietary formats, SGML documents are portable and reusable. SGML documents are associated with a DTD, which defines the rules that a computer application uses to access a document instance. The rules in the DTD allow applications to interpret the document, extract data, and exchange information with other applications to produce formatting code. The formatting code could be PostScript, for creating paper copies of the document, or other formats such as Rich Text Format (RTF) or Information Presentation Facility (IPF), that you could use to drive various online viewers. Because SGML is an open standard, you can write applications in any number of programming languages that can access the SGML data to produce the output that you want. The output format is determined by the formatting application. This separation allows you to reuse SGML documents to produce any form of output. As new output formats appear in the future, you can update your applications rather than converting your entire document database.

SAS AND SGML

The Publications Division at SAS Institute Inc. is in the process of converting all of its documentation into SGML. The driving force behind this conversion is the need to have portable and reusable documentation. We are developing applications which extract information from our SGML documents to create online help and manuals for all of the platforms supported by the SAS System. We will also produce books using the same pool of SGML documents. The conversion to SGML will enable us to take advantage of other information delivery methods that develop in the future.

The new techniques we are developing give us an opportunity to address the needs of SAS users who want to include SAS procedure output in their documents. The following sections of this paper describe some of the SAS applications we use to convert SAS output to SGML and introduce a proposed DTD for SAS output.

Using SAS Macros to Create SGML Output

Most people that we have talked to who have tried to convert SAS output to SGML start with the output file produced by PROC TABULATE (Example 2) or PROC REPORT. They convert the output file using the SAS DATA step or an SGML transformation utility to produce SGML. This method relies on the formatting clues in the output file, such as the FORMCHARS, to parse the content of the output. Even though the output file contains the output in the desired format, you will find it is easier to start the conversion from a SAS data set. You can build a data set that contains and identifies all the information that you need for a report with SAS procedures such as PROC SUMMARY, PROC MEANS, and PROC CONTENTS. Assuming that you are starting from a data set, let's examine how you can use SAS to convert the data set to SGML.

The most direct method for creating SGML output is with the DATA step and PUT statements, for example,

```
put '<ROW block="" count +col"">';
put '<ENTRY>' ID;
put '<ENTRY>' sex;
put '<ENTRY>' age;
put '<ENTRY>' result;
entry = left(put(blocktyp,tempcov.));
put '<ENTRY>' entry;
entry = left(put(newtrans,$tranfrm.));
put '<ENTRY>' entry;
```

Although this method is both flexible and easy to implement, you must create a SAS program for each report that you want to convert.
If you are converting several report formats to one DTD, you may want to consider the SAS macro facility. SAS macros make it easy to build a set of standard tools to convert a data set to SGML. You can either submit the macros before you start a conversion or store them permanently and call them with the autocall facility. Here is an example of a macro that creates tgroup element for a CALS table within a DATA step.

```sas
%macro tgroup (eols=,colsep=O,rowsep=O);
    put '<tgroup eols="' &cols" ' @;
    %if &colsep %then
        put ' colsep="l" @;
    %end;
    put '>
    %:if &rowsep %then
        do;
        put ' rowsep="l" ' @;
        %end;
    put '>
%mend;
```

To create a tgroup tag with the macro, name the attributes to be included in the tag with keyword parameters. The macro treats rowsep and colsep as toggles. They only produce attributes when they are passed to the macro. For example, the code

```sas
%tgroup(cols=11)
```

produces

```html
<tgroup cols="11">
```

and

```sas
%tgroup(cols=11,rowsep=O)
```

produces

```html
<tgroup cols="11",rowsep="O">
```

So far, this doesn't look very impressive, but when you use the TGROUP macro with macro variables, and include it in another macro then you can submit the following code to create the definitions for all the columns in a table.

```sas
%macro colspec (colname=,align=,colwidth=);
    put '<colspec colname="' &colname" ' @;
    %if &align ne %then
        do;
        put ' align="' &align" ' @;
    %end;
    %if &colwidth ne %then
        do;
        put ' colwidth="' &colwidth" ' @;
    %end;
    put '*-;;;.'
%mend;
```

```sas
%macro col_def;
    %tgroup(cols=&eolno)
    %do i=1 %to &colno;
        %colspec(colname=&&colnm&i,align=&&colj&i,colwidth=&&colw&i)
    %end;
    put '</tgroup>';
%mend;
```

You can exclude null values in the SGML output, as in the following.

```sas
%macro entry (align=,colname=,spanname=);
    put '<entry ' @;
    %if &align ne %then
        do;
        put ' align="' &align" ' @;
    %end;
    %if &colname ne %then
        do;
        put ' colname="' &colname" ' @;
    %end;
    %if &spanname ne %then
        do;
        put ' spanname="' &spanname" ', ,,' @;
    %end;
    put '>';
%mend;
```

Using the previous macro, you can submit

```sas
%entry()
```

to produce

```html
<entry>
```

or submit

```sas
%entry(align=left,colname=city)
```

to produce

```html
<entry align="left" colname="city">
```

Here is one final example to show how you could build the entire body of a table from a data set. You assign the contents of data set variables to sequentially-named macro variables. For example, the macro variable VARNM1 is assigned the name of the first data set variable, VARNM2 is assigned the name of the second data set variable, and so on. You can refer to the variables in a %DO loop to build a table row for each observation in a data set. For example, if the first variable name is IDNUM then this macro

```sas
%macro rows;
    put '<row>';
    %do i=1 %to &colno;
        %entry<para>
            %var:run&i
        </para><entry>
    %end;
    put '</row>';
%mend;
```

produces this statement.

```html
PUT '<entry<para> IDNUM @+(-1) '</para></entry>;
```

Using this method with four variables could produce output similar to the following.

```html
<row>
    <entry><para>1009</para></entry>
    <entry><para>15 GREENWOOD</para></entry>
    <entry><para>MORGAN, GEORGE B</para></entry>
    <entry><para>LOS ANGELES</para></entry>
</row>
```

Because the macro is in a DATA step, you can produce a row for every observation.
Including SAS Code and SAS Output in an SGML Document

The SGML standard provides a method for including data that requires special processing such as equations, graphics, music, or video. These data types must be declared in the DTD as NOTATION, which tells the SGML parser that the data is to be processed by another application. This enables you to include SAS code in an SGML document to automatically produce output. You can write an application to pass the SAS code in the SGML document to the SAS system to process when you format the document. The tables, reports, or graphics that SAS produces can then be included in the document as an SGML file entity.

PROPOSED DTD FOR SAS OUTPUT

Although you can convert SAS output to SGML, it would be simpler if SAS supported SGML output directly. This would enable us to run PROC TABULATE or PROC REPORT and include the resulting SGML output in your documents without a conversion.

There are several ways you could include SGML output produced by SAS in your documents. You could
- incorporate the DTD used for SAS output in your DTD
- include the SAS output using the SGML SUBDOC feature
- convert the SAS output to another format using the SAS DATA step or an SGML transformation utility.

To produce SGML from SAS output, you need a DTD that preserves as much information as possible about the original output such as the data set, statements, and statistics that produced the output. The DTD should also provide lots of handles for conversion to other output types (even proprietary formats). Toward that end, we would like to propose a DTD for SAS output.

The proposed DTD is presented in Example 3. Please look over the DTD and the example document instance and tell us what you think. Would you like the output to include an element that contains the SAS statements that generated the output? Do you need additional attributes on the rows, columns, or cells to identify the statistic and variable that produced them? We will pass your suggestions or comments on to those who are designing the next generation of SAS software.

HOW TO CONTACT US

To provide feedback about the proposed DTD for SAS output or to obtain more information, contact:

John Barnum, Technical Writer
sasbar@uax.sas.com
Craig Sampson, Senior Publishing Programmer
sascnp@uax.sas.com
or write us at:
SAS Institute
SAS Campus Drive
Cary, NC 27312

WHERE TO FIND MORE INFORMATION

To learn more about SGML, refer to the following:

Grunin, Lori (February 7, 1995), *Publish Without Paper!, PC Magazine*

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.
Example 1: Markup Systems

All three of these markup systems start a document with a chapter, title, topic and paragraph. Notice how the SGML markup explicitly defines the document structure rather than suggesting clues about it with formatting code.

RTF:

```rtf
\rtf1\ansi\deff12
\sectd{f\fonttbl{f12|fs\swiss Aria1;}}
{\stylesheet{\sO\snextO Base;}{\sl\fs36\snextl Banner Base;}}
\pard\sO{S{\footnote \$ Contents}}
{t{\footnote # Contents}}
{\plain }{\plainlfs48\b \tab \tab \tab My Book}\par
{\plain }{\plain\fs28\i My Chapter}\par
{\plain }{\tab }{\plain\fs28\i My Topic}\par
{\plain }{\tab This is the first paragraph in my topic. This example contains two sentences}\par\page
```

TeX:

```latex
\STAG\book()\STAG\DEFAULT{)(XLI(My-Book)\STAG\chapter()\STAG\DEFAULT{)(XLI(My-Chapter)\STAG\endDEFAULT{)(XLI(My-Topic)\STAG\enddefault{)(XLI(\parAf{)(XLI(This is the first paragraph in my topic. This example contains two sentences)\STAG\endparAf{})\STAG\endDEFAULT{)(XLI(\endbookO\bye
```

SGML:

```xml
<book>
  <title>My Book</title>
  <chap>
    <title>My Chapter</title>
    <topic>
      <title>My Topic</title>
      <para>
        This is the first paragraph in my topic. This example contains two sentences.
      </para>
    </topic>
  </chap>
</book>
```
Example 2: Sample Output from Proc Tabulate

This is a simple report produced by PROC TABULATE. Although the report is straightforward, the table includes spans, grouped headings, multiple groups.

<table>
<thead>
<tr>
<th>YEAR</th>
<th>DEPT</th>
<th>SALES</th>
<th>COST</th>
<th>PROFIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1981</td>
<td>MW03</td>
<td>3669.00</td>
<td>3594.00</td>
<td>73.00</td>
</tr>
<tr>
<td></td>
<td>MW101</td>
<td>8469.00</td>
<td>7425.00</td>
<td>1038.00</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>12132.00</td>
<td>11021.00</td>
<td>1111.00</td>
</tr>
<tr>
<td>1982</td>
<td>DEPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MW03</td>
<td>6396.00</td>
<td>4963.00</td>
<td>1433.00</td>
</tr>
<tr>
<td></td>
<td>MW101</td>
<td>13427.00</td>
<td>8019.00</td>
<td>5408.00</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>19823.00</td>
<td>12982.00</td>
<td>6841.00</td>
</tr>
<tr>
<td>1983</td>
<td>DEPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MW03</td>
<td>4258.00</td>
<td>3417.00</td>
<td>841.00</td>
</tr>
<tr>
<td></td>
<td>MW101</td>
<td>12724.00</td>
<td>10685.00</td>
<td>2039.00</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>16982.00</td>
<td>14102.00</td>
<td>2880.00</td>
</tr>
<tr>
<td>1984</td>
<td>DEPT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MW03</td>
<td>6345.00</td>
<td>5821.00</td>
<td>1324.00</td>
</tr>
<tr>
<td></td>
<td>MW101</td>
<td>11487.00</td>
<td>8759.00</td>
<td>2718.00</td>
</tr>
<tr>
<td></td>
<td>ALL</td>
<td>16982.00</td>
<td>14102.00</td>
<td>2880.00</td>
</tr>
</tbody>
</table>

Example 3: Proposed DTD for SAS output

Features of the SAS-out DTD:

```xml
<DOCTYPE SAS-out [>
  <!ELEMENT SAS-out (title,date?,titlegp?,table)+ --SAS output-->
  <!ELEMENT titlegp (title)+ --series of titles-->
  <!ELEMENT title (#PCDATA) --title-->
  <!ELEMENT date (#PCDATA) --date-->
  <!ELEMENT para (#PCDATA) --paragraph/running text content-->
  <!ELEMENT emph (#PCDATA) --emphasized text-->
  <!ELEMENT code (#PCDATA) --identify computer language-->
</SAS-out>
```

- **TITLEGP** supports one or more titles.
- **PARA** can contain text and code.
- **CODE** can contain SAS or host-specific code.
The SASPROC attribute identifies the procedure that produced the output. FRAME specifies the table borders. COLSEP and ROWSEP set the default horizontal and vertical rules for the table.

COL and SCOL support spanned and split columns.

ROW and SROW support split rows.

CELL can contain multiple paragraphs or, if you are creating a complex table, another table. CELL attributes to identify the data source, cell rotation, and alignment. COLSEP and ROWSEP can override the default horizontal and vertical rules set in TABLE.

ALIGN provides justification as well as alignment on a character.

```xml
<ELEMENT table - - (title,col+,row*)
   --SAS canonical tables consist of-->
<!ATTLIST table id ID #REQUIRED --identify table--
sasproc CDATA #IMPLIED --created by SAS procedure--
frame (top|bottom|topbot|all|sides|none) #IMPLIED
   --border rules--
colsep NUMBER #IMPLIED --vrule on columns right--
rowsep NUMBER #IMPLIED --hrule below row--
orient (port|land) #IMPLIED --table orientation on pg--
pwwide NUMBER #IMPLIED
   --0=fit to text column, 1=fit to pg-->

<!ELEMENT col - - (scol)*
   --indicate a column--
<!ATTLIST col id ID #REQUIRED --identify the column--
pc CDATA #IMPLIED --reference to parent column--
colsep NUMBER #IMPLIED --vrule on this columns right-->

<!ELEMENT scol - - (col,col*)
   --split or sub column--

<!ELEMENT row - - (cell|srow)*
   --indicate a row--
<!ATTLIST row id ID #REQUIRED --identify the row--
pr CDATA #IMPLIED --reference to parent row--
rowsep NUMBER #IMPLIED --hrule below cell-->

<!ELEMENT srow - - (row,row*)
   --split or sub row--

<!ELEMENT cell - - (para|table)
   --contents of a table cell--
<!ATTLIST cell pr CDATA #REQUIRED --reference parent row--
pc CDATA #REQUIRED --reference parent column--
id ID #IMPLIED --cell id (link cell or header cell)--
localhd CDATA #IMPLIED --reference a local header--
content (head|data) "data" --contents of cell--
colsep NUMBER #IMPLIED --vrule on cells right--
rowsep NUMBER #IMPLIED --hrule below cell--
rotate NUMBER "0" --rotate cell contents--
valign (top|bottom|middle) #IMPLIED --vert justification--
align (left|right|center|justify|char) #IMPLIED
   --hor justification--
charoff NOTOKEN #IMPLIED --character alignment offset--
char CDATA #IMPLIED --alignment character-->

757
Example 4: PROC TABULATE Output Tagged in SAS-out

This is a document instance of the DOCTYPE SAS-out and shows how the PROC TABULATE output in Example 2 is tagged.

**COL** defines the four main columns for the report. Notice how column 1 is divided into two sub columns by **SCOL**.

**ROW** and **CELL** define the table body. The **CONTENT** attribute identifies the cell as a row or column head. The attributes **pr** and **pc** identify the parent of each cell.

**CELL** tags define the subcolumn heads "YEAR" and "DEPT" for the classes.
CELL tags define the row head, the sub-row head, and the contents of the body cells in the first group.

```xml
<table>
  <row id="r3" rowsep="l">
    <cell id="stub3" pr="r3" pc="cla" localhd="stub2" content="head">
      <para>1981</para>
    </cell>
    <srow>
      <row id="r3a" rowsep="l">
        <cell id="s-stub3" pr="r3a" pc="c1b" localhd="s-stub2" content="head">
          <para>MW203</para>
        </cell>
        <cell pr="r3a" pc="c2" localhd="s-stub3" content="data">
          <para>3669.00</para>
        </cell>
        <cell pr="r3a" pc="c3" localhd="s-stub3" content="data">
          <para>3596.00</para>
        </cell>
        <cell pr="r3a" pc="c4" localhd="s-stub3" content="data">
          <para>73.00</para>
        </cell>
      </row>
      <row id="r3b" rowsep="l">
        <cell id="s-stub3b" pr="r3b" pc="c1b" localhd="s-stub2" content="head">
          <para>NE101</para>
        </cell>
        <cell pr="r3b" pc="c2" localhd="s-stub3b" content="data">
          <para>8463.00</para>
        </cell>
        <cell pr="r3b" pc="c3" localhd="s-stub3b" content="data">
          <para>7425.00</para>
        </cell>
        <cell pr="r3b" pc="c4" localhd="s-stub3b" content="data">
          <para>1038.00</para>
        </cell>
      </row>
      <row id="r3c" rowsep="l">
        <cell id="s-stub3c" pr="r3c" pc="c1b" localhd="s-stub2" content="head">
          <para>ALL</para>
        </cell>
        <cell pr="r3c" pc="c2" localhd="s-stub3c" content="data">
          <para>12132.00</para>
        </cell>
        <cell pr="r3c" pc="c3" localhd="s-stub3c" content="data">
          <para>11021.00</para>
        </cell>
        <cell pr="r3c" pc="c4" localhd="s-stub3c" content="data">
          <para>1111.00</para>
        </cell>
      </row>
    </srow>
  </row>
</table>
</SAS-out>

759