Paper AD18

Comparison of Techniques for XML and RTF Tables

Linda Collins, PharmaStat LLC, Moraga, CA
Alan Hopkins, PharmaStat LLC, Moraga, CA

ABSTRACT

Easily interpretable statistical summary tables are important for communicating results of clinical trials. These tables typically contain simple statistics that must be obtained from more than one SAS procedure. Consequently, statistical programmers write specialized macros for obtaining appropriate statistics and then formatting those statistics into a table suitable for publication in a document. It is common for SAS programs to produce ASCII text files, with additional steps needed to incorporate this output into a larger document. Collectively this is a very labor intensive process We discuss a method of constructing tables with statistical information in two steps: (1) creation of a statistical results file with the needed statistics from SAS procedures, and (2) formatting the statistical analysis file into a table for incorporation into a document processing system using two different markup systems – rtf and xml.

BACKGROUND

Tables are typically produced in SAS by performing some preparatory data manipulations, then running one or more procedures to generate the statistics. The statistics are then 'printed' to an output file with appropriate formatting to produce a readable table.

ASCII or 'plain-text' outputs do not incorporate printing 'metadata' such as page margins, font type and size, or other special characteristics such as boldface, italics, or special characters. These characteristics are usually controlled by printing commands.

RTF output, by contrast, can incorporate such information as an intrinsic part of the document. SAS RTF templates, or specialized code within the program, give the user control over a wide variety of visual characteristics. The resulting file can be used as one of a series of appendices to a larger report, or cut and pasted into a larger document as an in-text table. For in-text tables, it is particularly convenient that the RTF output produced by many SAS procedures is in the form of a Microsoft Word table.

XML output shares many of the advantages of RTF output. Visual characteristics of the report can be stored in a related stylesheet document or imbedded as part of the document itself. XML has the further advantage of being a useful vehicle for assembling larger sets of interrelated documents such an electronic submission of data to regulatory agencies.

One simple way of creating a report with the statistical results file is to print the file using PROC REPORT. The results can be captured using ODS to get RTF or XML output. In our experience, the default RTF from ODS does not quite satisfy the rigorous publishing guidelines in some organizations. As a result, we one might customize the ODS output using PROC TEMPLATE to adjust style characteristics. It is also possible to use an analogous technique to create a publishable table with XML markup.

In addition to style specifications, PROC TEMPLATE can be used to create a table specification. The table specification allows the user to define characteristics of columnar data, store the specification in a template, and output the results with a fairly simple DATA _NULL_. We

compare the various methods of creating statistical tables and assess how well they integrate into downstream document publishing.

STATISTICAL RESULTS FILES

Palmer (2002) recommended creating an analysis data base for statistical results for later formatting in an xml publishing system. Harrell (2004) has also described a system for academic statisticians which combines S data frames into LaTeX documents. Our own SAS statistical macros (APT software, 2004) create a statistical results file which is used for input to publishing software. Our model for statistical tables (Hopkins and Collins, 2005) is based on 'segments' which describe analysis results for an analyte.

The key to a straightforward table-output system is a consistent format for the summarized data to be published. In the APT macro system, all statistical macro calls add records to a statistics dataset with a consistent structure. The user can request various kinds of univariate and comparative statistics, organized by 'segments'. A segment typically contains all the statistics needed for a single analysis variable. The statistics are generated by one or more SAS procedures (Univariate, FREQ), captured into SAS datasets, and then combined and reformatted into a standard statistical file structure. The macros used to do this allow the user to request a variety of formatting characteristics such as combinations of statistics to group on a row, the number of decimal places, or the use of punctuation such as parentheses and percent signs.

The statistical results file contains the following columns:

- By-Group Variables (optional)
- Segment Number
- Segment Label
- Sort Order
- Line Label
- COL*1* through COL*n* are the statistics for each report column.

In this standard structure, certain variables such as SEGLABL, LINLABEL, COL1 through COL*n* contain information that will be displayed on the page. Other variables, such as SEG and ORD1, are used internally in the printing routine to identify the logical grouping of lines and the order they should appear in.

Once all the desired statistics have been generated, the file is ready for output to one or more 'printing' destinations. In the 'printing' step, some overall information is supplied by macro parameters to describe other necessary information to document the analysis, e.g. table numbers and titles, etc.

CONVERTING STATISTICAL RESULTS FILES TO PUBLISHABLE TABLES

ASCII output:

A finished statistical table of a clinical study might look like the following (data published by Diabetes Control and Complications Trial Research Group.,1993):

DCCT Study: Baseline Characteristics of Two Study Cohorts

	Primary Prevention		Secondary Intervention	
	Conventional (N=378)	Intensive (N=348)	Conventional (N=352)	Intensive (N=363)
Duration of IDDM (yrs)				
N	378	348	352	363
Mean (SD)	30.7 (16.7)	31.4 (16.5)	103.1 (44.4)	106.2 (45.2)
Median	27.0	29.0	103.5	112.0
Range	8.0 to 133.0	9.0 to 142.0	13.0 to 179.0	10.0 to 180.0
Glycosolated Hemoglobin (%)				
N	378	348	352	363
Mean (SD)	8.8 (1.7)	8.8 (1.6)	8.9 (1.5)	9.0 (1.5)
Median	8.5	8.6	8.7	8.8
Range	5.4 to 14.8	5.8 to 14.4	6.0 to 14.2	6.4 to 14.3
Presence of Clinical Neuropathy				
No	368 97.9%	329 95.1%	319 90.6%	328 90.6%
Yes	8 2.1%	17 4.9%	33 9.4%	34 9.4%
Presence of Clinical Neuropathy				
Chi-Square P-value (Conventional v. Intensive)	0.041		0.994	

The statistical results file would look something like the following (partial display):

seg	seglabl	ord1	linlabel	col1	col2
1	Duration of IDDM (yrs)	1	N	378	348
1	Duration of IDDM (yrs)	2	Mean (SD)	30.7 (16.7)	31.4 (16.5)
1	Duration of IDDM (yrs)	3	Median	27.0	29.0
1	Duration of IDDM (yrs)	4	Range	8.0 to 133.0	9.0 to 142.0
2	Glycosolated Hemoglobin (%)	1	N	378	348
2	Glycosolated Hemoglobin (%)	2	Mean (SD)	8.8 (1.7)	8.8 (1.6)
2	Glycosolated Hemoglobin (%)	3	Median	8.5	8.6
2	Glycosolated Hemoglobin (%)	4	Range	5.4 to 14.8	5.8 to 14.4
3	Presence of Clinical Neuropathy	1	No	368 97.9%	329 95.1%
3	Presence of Clinical Neuropathy	2	Yes	8 2.1%	17 4.9%
4	Presence of Clinical Neuropathy	1	Chi-Square P-value (Conventional v. Intensive)		0.041

(Note: linlabel value for the last record is word wrapped)

The table-generating macro performs some preprocessing to determine the spacing requirements for the report, and stores them in macro variables. It then performs a PROC REPORT like the following (simplified) example code which is set up for a plain ASCII output:

```
title1 "DCCT Study:  Baseline Characteristics of Two Study Cohorts";
  proc report data = bstats split='~' missing spacing = 1 nowindows headline list headskip ;

      column (“__” seg seglabl dummy linlabel
                  ( "Primary Prevention" col1 col2 )
                  ( "Secondary Intervention" col3 col4 ) ) ;

      define seg      /order order = internal      width = 0  spacing = 0 noprint ;
      define seglabl  /order order = data          width = 0  spacing = 0 noprint ;
      define dummy    /order order = internal      width = 15 spacing = 0 " " ;
      define linlabel /order order = internal flow width = 15 spacing = 0 " " ;
      define col1     / display width= 18 "   Conventional   ~     (N=&c1)     ~";
      define col2     / display width= 18 "    Intensive    ~     (N=&c2)     ~" ;
```

```
      define col3     / display width= 18 "   Conventional   ~     (N=&c3)     ~";
      define col4     / display width= 18 "    Intensive    ~     (N=&c4)     ~" ;

      compute before seglabl /;
          line @1 seglabl $94. ;
      endcomp ;
      break after seglabl / skip ;
   run ;
```

This printing step takes advantage of several useful PROC REPORT features.

- It uses spanned column headers to display the text of 'Primary Prevention' and 'Secondary Intervention',
  The overline over the column headings is also generated as a spanned column header by placing the "__" in the column statement with parentheses around all the variables it is to span.
- The 'headline' option in the procedure statement generates the underline under the column headings.
- The 'line' statement prints the text of the segment label on a separate line when the segment changes.
- A preprocessing step has put a split character (~) and imbedded spaces into the LINLABEL variable to control where the word wrapping occurs, and to indent by two spaces when it does. For this example, then, the record:

```
seg            seglabl            ord1 linlabel              col1              col2
4  Presence of Clinical Neuropathy   1  Chi-Square P-value                    0.041
                                        (Conventional v.
                                        Intensive)
```

is converted to:

```
seg            seglabl            ord1 linlabel              col1              col2
4  Presence of Clinical Neuropathy   1  Chi-Square P-value                    0.041
                                        ~  (Conventional v.
                                        ~  Intensive)
```

The split character and inserted spaces give a more readable indentation in the finished report.

RTF output:
The same table rendered as an RTF output could look like:

	Primary Prevention		Secondary Intervention	
	Conventional (N=378)	**Intensive (N=348)**	**Conventional (N=352)**	**Intensive (N=363)**
Duration of IDDM (yrs)				
N	378	348	352	363
Mean (SD)	30.7 (16.7)	31.4 (16.5)	103.1 (44.4)	106.2 (45.2)
Median	27.0	29.0	103.5	112.0
Range	8.0 to 133.0	9.0 to 142.0	13.0 to 179.0	10.0 to 180.0
Glycosolated Hemoglobin (%)				
N	378	348	352	363
Mean (SD)	8.8 (1.7)	8.8 (1.6)	8.9 (1.5)	9.0 (1.5)
Median	8.5	8.6	8.7	8.8
Range	5.4 to 14.8	5.8 to 14.4	6.0 to 14.2	6.4 to 14.3
Presence of Clinical Neuropathy				
No	368 97.9%	329 95.1%	319 90.6%	328 90.6%
Yes	8 2.1%	17 4.9%	33 9.4%	34 9.4%

	Primary Prevention		Secondary Intervention	
	Conventional (N=378)	Intensive (N=348)	Conventional (N=352)	Intensive (N=363)
Presence of Clinical Neuropathy				
Chi-Square P-value (Conventional v. Intensive)		0.041		0.994

Several items must be added to – and subtracted from - the code to enable RTF characteristics. First, we set up code to generate a style sheet using PROC TEMPLATE. This is not required, as SAS will assume a default style for the RTF destination. However, it is often desirable to make modifications to the default style. In this case, we have set up a macro that allows us to control settings that we may want to vary.

```
%macro rtfstyle (
          name     = tempstyle ,
          fontname = Times New Roman ,
          fontsize = 10 ,
          cellpad  = 0.05 ,
          cellspac = 0.02 ,
          orient   = landscape
          );

options orientation=&orient ;
ods escapechar="^";

   proc template;
      define style &name ;
        parent=styles.rtf;              /* Base the style on the default RTF style */
        style Table from output /
              Rules=groups              /* Draw line between headings and detail   */
              Frame= box                /* Draw box around the entire table        */
              cellpadding = &cellpad.in
              cellspacing = &cellspac.in
              borderwidth = &cellspac.in
              ;
        style body from document /
              topmargin    = 1.25in     /* Control document margins */
              bottommargin = 1.00in
              leftmargin   = 1.00in
              rightmargin  = 1.00in
              ;
        style header from header /
              background=_undef_
              protectspecialchars=off
              asis=on
              ;

       replace fonts /
               'TitleFont2' = ("&fontname",&fontsize.pt,Bold Italic)
               'TitleFont' = ("&fontname",&fontsize.pt)
               'StrongFont' = ("&fontname",&fontsize.pt,Bold)
               'EmphasisFont' = ("&fontname",&fontsize.pt,Italic)
               'FixedEmphasisFont' = ("&fontname",&fontsize.pt,Italic)
               'FixedStrongFont' = ("&fontname",&fontsize.pt,Bold)
               'FixedHeadingFont' = ("&fontname",&fontsize.pt,Bold)
               'BatchFixedFont' = ("&fontname",&fontsize.pt)
               'FixedFont' = ("&fontname",&fontsize.pt)
               'headingEmphasisFont' = ("&fontname",&fontsize.pt,Bold Italic)
               'headingFont' = ("&fontname",&fontsize.pt,Bold)
               'docFont' = ("&fontname",&fontsize.pt)
               ;
      end;
   run;

%mend ;
```

The PROC REPORT code itself needs to be modified in several ways for RTF output:

```
ods rtf    file = "&progname..rtf"  style = tempstyle ;
ods eschapechar='^';

title1 "DCCT Study:  Baseline Characteristics of Two Study Cohorts";
proc report data = bstats split='~' missing spacing = 1 nowindows headline list headskip ;
       column /* ("__" */
              seg seglabl dummy linlabel
                    ( "Primary Prevention" col1 col2 )
                    ( "Secondary Intervention" col3 col4 )  /* ) */ ;


       define seg      / order order = internal      width = 0  spacing = 0 noprint ;

       define seglabl  / order order = data          width = 0  spacing = 0 noprint ;
       define dummy    / order order = internal      width = 15 spacing = 0
                         style(column)=[asis=off cellwidth=0.25in] " " ;

       define linlabel / order order = internal flow width = 15 spacing = 0
                         style(column)=[asis=off cellwidth=1.25in] " " ;

       define col1     / display width= 18 "   Conventional   ~     (N=&c1)     ~"
                         style(column)=[asis=on cellwidth=1.58in] ;

       define col2     / display width= 18 "    Intensive    ~     (N=&c2)     ~"
                         style(column)=[asis=on cellwidth=1.58in] ;
        {additional define statements}
       compute before seglabl / style=[cellwidth=0.25in] ;
           line @1 seglabl $94. ;
       endcomp ;
       break after seglabl / skip ;
   run ;
ods rtf close ;
```

The ODS RTF statement will redirect output to the RTF destination named in 'file='. The 'style=' specifications tells SAS that we want to use the style definition 'tempstyle' that we just defined.

What's changed in this code? We need alternative techniques for several features of this report.

- RTF output from PROC REPORT will not honor the width specification in characters on the define statements. For controlling width, we add explicit widths with the "style(column)=[cellwidth=*n.nn*in]". The 'ASIS=ON' portion of this statement tells SAS to honor the leading spaces in the data field.
- The overline and underline around the column headings must be suppressed and replaced with RULES=GROUPS in the RTF style template.
- RTF output from PROC REPORT will also not honor split characters. We replace the split character in the LINLABEL variable with RTF code for an imbedded hanging indent. In the preprocessing of the data file, instead of inserting split characters, we insert RTF control codes. The control code shown below tells Word to format the cell with a line indent (\li) of 0.25 inches (expressed in "twips" or 1/1440 of an inch) and a first-line indent (\fi) of -0.25 inches. The line

  ```
  ods escapechar='^';
  ```

  is needed for the beginning sequence ^R to be recognized as an RTF control code.

```
seg            seglabl            ord1 linlabel              col1               col2

4  Presence of Clinical Neuropathy   1  ^R"\li360\fi-360 "                        0.041
                                        Chi-Square P-value
                                        (Conventional v.
                                        Intensive)
```

There are several very useful differences to this type of output. When the output is opened by Mocrosoft Word, the output exists as a Word table, not just text. The table can be selected and

pasted into another document for use as an 'in-text' table, and column boundaries can be moved without retyping any of the internal text. Because we have set up the indentation as part of the cell characteristics, it will be maintained and word wrap correctly even if the table columns are resized.

There are also a couple of unsatisfying aspects to this approach. First, although the RTF specification has been published (documentation is available on the Microsoft web site), it is awkward to insert control codes into the dataset. The control codes mean that this print procedure cannot be used to simultaneously write output to multiple destinations (e.g. directing the same report to ASCII, PDF, XML, and RTF) because the other destinations would not interpret the control codes. This would be better handled by options in SAS, although as of this writing SAS does not support any such options.

Another problem is that SAS does not have complete control over how spacing in handled once the document is opened in Word. It is common for tables to 'break' into panels if the horizontal spacing is too tight. This results in an output table with the last few columns printed at the bottom of the page or on a following page in a separate block. For most clinical applications, this is unacceptable. One known problem in SAS 8.2 is that spanned columns (such as the spanned column headers) can cause SAS to ignore explicit column width settings. This further exacerbates the problem of panels breaking across pages. This problem has been alleviated in SAS 9.1.3. However, it is still not possible for SAS to force all columns to stay on the same page, and adjusting horizontal spacing is often a matter of trial and error.

CREATING STATISTICAL RESULTS FILES AS PUBLISHABLE TABLES IN XML

A similar technique can be used to produce a table in XML. This example uses two different types of XML destinations, XML and MARKUP. The syntax is slightly different, but the XML code produced is the same. In both cases, the PROC REPORT produces both a table file (the .xml output) and a style sheet (the .xsl output). The MARKUP example also causes the procedure to produce a 'document type definition' (.dtd) file.

```
ods xml    file = "&progname.1.xml" code="&progname.1.xsl"  style = docbook;
ods markup body = "&progname.2.xml" code="&progname.2.xsl"  style = docbook
           frame ="&progname.2.dtd"   tagset = default ;

title1 "DCCT Study:  Baseline Characteristics of Two Study Cohorts";
proc report data = bstats split='~' missing spacing = 1 nowindows headline list headskip ;
        {additional PROC REPORT statements as shown in RTF example}
run ;
ods xml close ;
```

The output produced, if viewed by any ASCII editor such as Notepad, looks like this (partial print):

```
<?xml version="1.0" encoding="windows-1252"?>
<odsxml>
<head>
<meta operator="Linda"/>
</head>
<body>
<proc name="Report">
<label name="IDX"/>
<title class="SystemTitle" toc-level="1">DCCT Study:  Baseline Characteristics of Two Study
Cohorts</title>
<branch name="Report" label="The Report Procedure" class="ContentProcName" toc-level="1">
<leaf name="Report" label="Detailed and/or summarized report" class="ContentItem" toc-level="2">
<output name="Report" label="Detailed and/or summarized report" clabel="Detailed and/or summarized
report">
<output-object type="table" class="Table">
  <style>
    <border spacing="1" padding="7" rules="GROUPS" frame="BOX"/>
  </style>
<colspecs columns="4">
<colgroup>
<colspec name="1" width="0" align="center" type="string"/>
```

```
<colspec name="2" width="0" align="center" type="string"/>
<colspec name="3" width="0" align="center" type="string"/>
<colspec name="4" width="0" align="center" type="string"/>
</colgroup>
</colspecs>
<output-head>
<row>
<data type="string" class="Header" row="1" column="1" column-end="2">
  <style>
    <span columns="2"/>
  </style>
<value></value>
</data>
<data type="string" class="Header" row="1" column="3" column-end="4">
  <style>
    <span columns="2"/>
  </style>
<value>Primary Prevention</value>
</data>
</row>
<row>
<data type="string" class="Header" row="2" column="1" column-end="2">
  <style>
    <span columns="2"/>
  </style>
<value></value>
</data>
<data type="string" class="Header" row="2" column="3">
<value>   Conventional   <br/>     (N=378)     </value>
```

What do we do with this? XML documents can be viewed by a number of editors and browsers, such as Internet Explorer, Netscape, XMLSpy by Altova, Epic by Arbortext, and others

There are two limitations to this output produced by SAS as of this writing. First, the XML file produced by SAS does not automatically contain a link between the document and the descriptive information contained in the DTD or the style sheet. Any viewing software needs at least one of these descriptive documents to allow it to interpret the XML table. We can correct this by editing the document and manually inserting the link. We edit the file 'example_report1.xml' and type in the second line (shown in boldface). The href= contains the name of the stylesheet document we want to reference:

```
<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="example_report1.xsl"?>
<odsxml>
<head>
```

This tells the XML editor that the document should be viewed using the style sheet file.
The second problem is that the tagsets used by SAS contain an error as of this writing. In the style sheet file, a tagset is missing a closing tag. We can correct this by editing the stylesheet file 'example_report1.xsl':

```
     <LINK REL="STYLESHEET" HREF="example_report1.xsl"/>
```

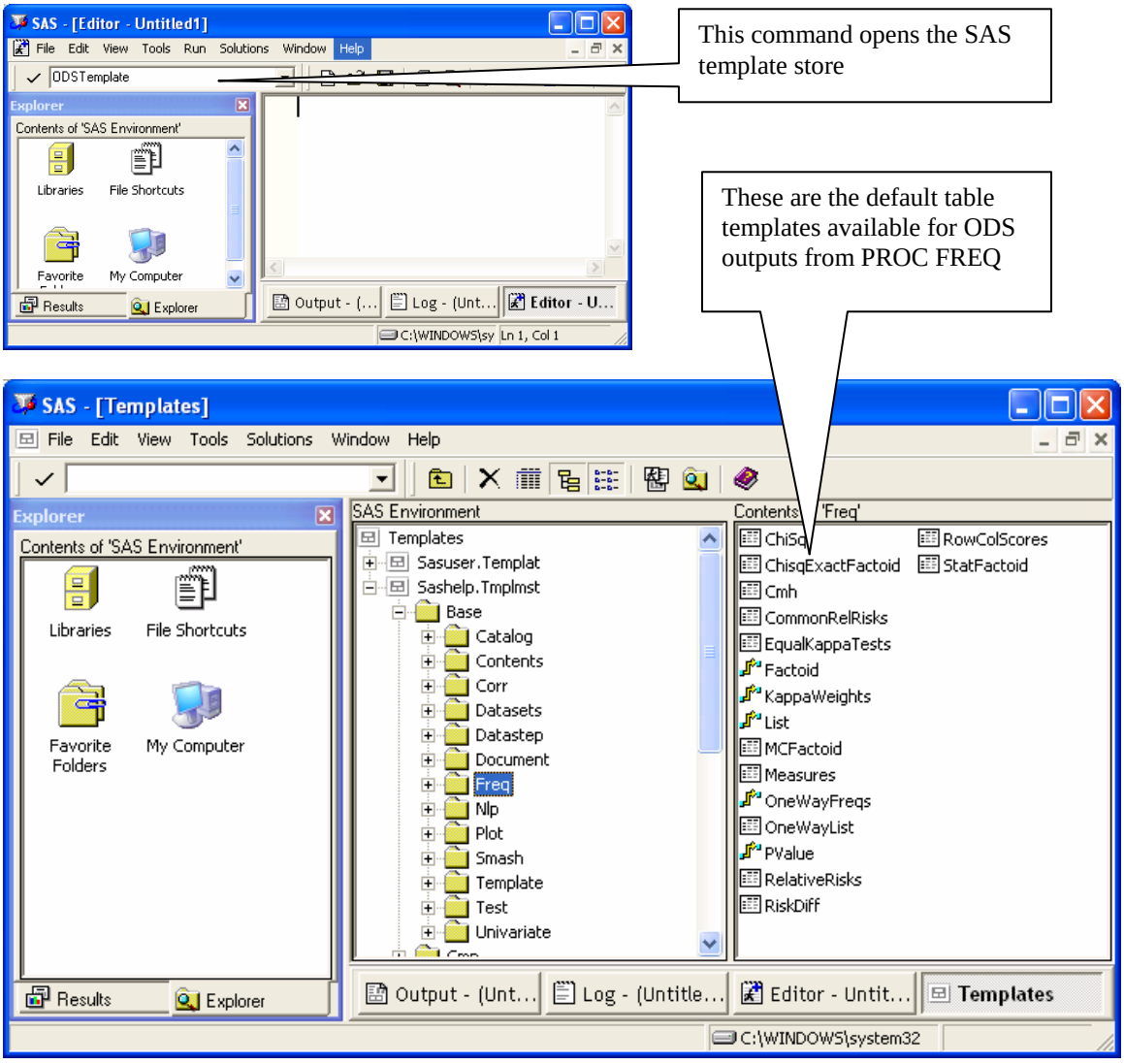
The tagset LINK is missing a closing forward slash. This is needed for this to be a 'well-formed' or valid XML document. Add it inside the closing bracket '>. Having made these two edits, we can open the file using Internet Explorer. Other XML editors or browsers could also be used.

The real value of an XML output, though, is not in viewing it as a standalone document. XML is increasing used as a vehicle for organizing large sets of interrelated documents, data, graphics, and other objects We anticipate that eventually electronic submissions of clinical data to regulatory agencies will use XML to organize clinical study reports, tables, graphical displays, and the underlying data in a form that can be navigated within a single tool. Many organizations now

do something like this by using document management software to link documents in PDF format. XML applications will be likely to improve on this process in the future. In such an application, the compiled set of document will most likely use a global DTD such as the CALS table model.

CREATING STATISTICAL TABLES IN XML AND RTF WITH TABLE TEMPLATES

PROC REPORT is not the only way to produce tabular output in SAS. By default, each procedure is associated with a 'table template', which is stored in the SAS 'template store'. We can also use the TEMPLATE procedure to write table definitions from scratch, or to modify existing ones. To see the default templates associated with SAS procedures, open the template store by typing in the command line of SAS the word 'ODSTemplate'. You can then navigate to the default template for various SAS outputs.



The default template for a procedure can be modified and used directly for output from that procedure, although we do not discuss that technique in detail here. The master template store can also be used as a set of models for writing our own template, which can be applied to our own reformatted statistics dataset using a DATA _NULL_.

Many SAS programmers remember applications using DATA _NULL_ with extensive and complicated line control and put statements. Using a table template, much of this can be reduced to the statement 'put _ods_'. In this example, we have used PROC TEMPLATE to set up a table template for our statistics file, and a the PUT ODS technique to write the file to our ODS destinations

The following sample code shows how a simple table template can be defined and used:

.

```
/* ---------------------------------------------------------------------------- */
/* Create table template                                                        */
/* ---------------------------------------------------------------------------- */
 proc template ;
        define table mytab.apt ;
            mvar colhd1 ch1 ch2  ;         /* Make macro variables available to template */
            column segl lin  c1 c2 ;
            header hdr1 hdr2a hdr2b   ;
            define header hdr1;
               text colhd1 ;
               style=header{font_size=11pt} ;
               start=c1 ;                  /* span columns c1 */
               end  =c2 ;                  /* through c2      */
               split = '~' ;
            end ;
            define header hdr2a;
               text ch1 ;
               style=header{font_size=10pt} ;
               start=c1 ;
               end  =c1 ;
               split = '~' ;
            end ;
            define header hdr2b;
               text ch2 ;
               style=header{font_size=10pt} ;
               start=c2 ;
               end  =c2 ;
               split = '~' ;
            end ;
            define segl ;
                generic=on ;
                header = " ";
                blank_dups=on;
                style = [cellwidth=2.0in] ;
                glue  = 325 ;
            end ;
            define lin ;
                generic=on ;
                header = " ";
                style = [cellwidth=1.0in] ;
                glue  = 325 ;
                flow ;
            end ;
            define c1 ;
                generic=on ;
                header = " ";
                style = [cellwidth=1.5in] ;
                glue  = 325 ;
            end ;
            define c2 ;
                generic=on ;
                header = " ";
                style = [cellwidth=1.5in] ;
                glue  = 325 ;
            end ;
        end ;
    run ;

/* ---------------------------------------------------------------------------- */
/* Redirect output to ODS destinations                                          */
/* ---------------------------------------------------------------------------- */
%rtfstyle (
        name     = tempstyle ,
        orient   = portrait ,
        fonttype = Courier New ,
        fontsize = 7
        ) ;
```

```
     ods xml    file = "exampleapt1.xml" code  ="exampleapt1.xsl"  style = docbook  ;
     ods markup body = "exampleapt2.xml" frame ="exampleapt2.dtd" code="exampleapt2.xsl"
                tagset = default ;
     ods rtf    file = "exampleapt.rtf"  style = tempstyle ;

/* ---------------------------------------------------------------------------- */
/* Generate the report                                                          */
/* ---------------------------------------------------------------------------- */

    %let colhd1 = Primary~Prevention ;
    %let ch1n   = 378 ;
    %let ch2n   = 348 ;
    %let ch1    = Conventional~(N=&ch1n) ;
    %let ch2    = Intensive~(N=&ch2n) ;

    data _null_ ;
        set bstats ;
        file print ods = (template = "mytab.apt"
                          columns=(
                              segl=seglabl(generic=on)
                              lin=linlabel(generic=on)
                              c1=col1(generic=on)
                              c2=col2(generic=on)
                              )
                          ) ;
        put _ods_ ;
    run ;
    ods xml close ;
    ods rtf close ;
    ods markup close ;
```

In the table template, the ‘mvar’ statement gives the template access to macro variable values. It is important to use this technique rather than the “*&macrovar*” syntax if the macro variable needs to be resolved at the time the report is produced, rather than at the time the table template is compiled (keep in mind that the table template may be stored in a template store and used for multiple reports). The ‘define header’ statements describe the column headers, and may incorporate spanned headers using the ‘start’ and ‘end’ statements.

After setting up our usual ODS redirection statements, the data step opens the statistics dataset for input , binds the table template as the format for its output, and produces the reports with the ‘put _ods_ ‘ statement. The resulting RTF file looks like this (XML output is similar):

		Primary Prevention	
		Conventional (N=378)	**Intensive (N=348)**
Duration of IDDM (yrs)	N	378	348
	Mean (SD)	30.7 (16.7)	31.4 (16.5)
	Median	27.0	29.0
	Range	8.0 to 133.0	9.0 to 142.0
Glycosolated Hemoglobin (%)	N	378	348
	Mean (SD)	8.8 (1.7)	8.8 (1.6)
	Median	8.5	8.6
	Range	5.4 to 14.8	5.8 to 14.4
Presence of Clinical Neuropathy	No	368 97.9%	329 95.1%
	Yes	8 2.1%	17 4.9%
	Chi-Square P-value		0.041

This technique has potential applicability for defining a standard table structure that could be used for any report, and storing it as either a master template or as a macro that sets up table definitions with certain overridable parameters. This can be a particularly powerful technique when a well-defined standard data structure for statistical output is available, such as the data structure used by the APT macros. The production of tables using the data _null_ would then require very little code and would be usable for multiple ODS destinations.

This technique does not, however, replicate all the features available in our PROC REPORT technique. We were able to mimic the spanned column header behavior of PROC report. We do not, however, have an analog to the COMPUTE block that allows certain lines to be written conditionally at the beginning or end of a group of records. The segment label, therefore, can only be written as a column, not as a LINE statement.

CREATING CUSTOM TAGSETS IN XML

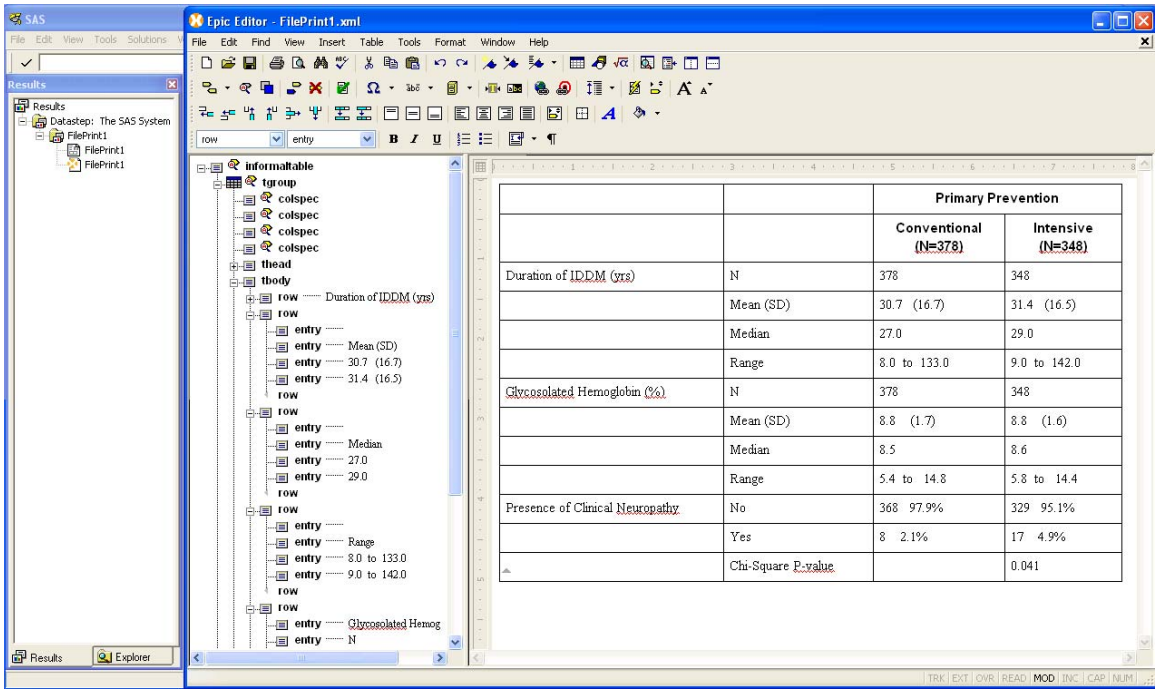
The XML created by ODS XML engine was meant to stand alone – a DTD and style sheet are required. We tested one alternative. We took the docbook tagset distributed with SAS 9.1.3 and created a custom apt_tagset which included only the tags necessary for creating OASIS-compatible table specifications (Walsh, 1999). When used in conjunction with the “mytab.apt” table template we created OASIS-compatible table descriptions.

```
proc template;
   define tagset apt_table;
      define event show_charset;
         set $show_charset "1" /if ^exists( suppress_charset);
      end;
      define event put_value;
         put VALUE;
      end;
      define event put_value_cr;
         put VALUE NL;
      end;
      define event dynamic;
         put "Content-Type: text/xml" NL NL NL;
      end;
      define event doc;
            trigger show_charset;
            put "<?xml version=""1.0""";
            putq " encoding=" encoding /if exists( $show_charset) ;
                                 put "?>" NL NL;
            /* For compatibility with Arbortext Epic editor */
            put "<!DOCTYPE book PUBLIC ""-//Arbortext//DTD DocBook XML V4.0//EN"" ""axdocbook.dtd"">";
            put NL NL;
      end;
        {additional tagset definition statements}
   end;
run;
   ods markup body = "exampleapt5.xml" tagset = apt_table ; /* Custom Tag set for processing tables */

/* ------------------------------------------------------------------------ */
/* Generate the report                                                      */
/* ------------------------------------------------------------------------ */

    data _null_ ;
        set bstats ;
        file print ods = (template = "mytab.apt"
                          columns=(
                              segl=seglabl(generic=on)
                              lin=linlabel(generic=on)
                              c1=col1(generic=on)
                              c2=col2(generic=on)
                              )
                          ) ;
        put _ods_ ;
    run ;
```

The markup created by SAS is an output file that may be viewed directly from a XML editor as shown below. Alternatively, these XML tables could be stored and links built from XML documents to create dynamic documents.



SUMMARY

Creating statistical tables involves at least two steps: first, generating the statistics and second, formatting the statistics for publishing in a document. We have found that SAS can create statistical files from multiple SAS procedures. We have tested two markup languages for incorporating the results into documents – Rich Text Format (RTF) and XML. Using SAS ODS and PROC TEMPLATE we were able to create publishable tables (with some difficulty) for both output destinations. Among the computing approaches considered, none was completely satisfactory.

The choice of markup language will be dictated by the authoring environment for the publication of results. Clearly RTF is the preferred choice for creating MS Word documents. We learned however, that the connection between SAS and RTF, although powerful in many respects, can be complex and require considerable tuning. Some table characteristics such as breaking of panels for horizontal spacing, are hard to control via SAS code. Finally, some characteristics that we would like to control, such as cell indentation, do not have 'hooks' in SAS at this time.

XML has a lot of appeal because of its ability to integrate data with text. The information can be processed and transformed easily. Th OASIS table models is simple and straight forward. However, creating XML makes sense when using a XML authoring environment which is not common in the industry today. XML authoring is more complicated than simple word processing involving document schemas, style sheets and tags. Significant technical support is definitely required to get started in the XML environment.

The scenario considered in this paper represents an elementary approach to the dissemination of statistical results – generate tables, then write the report. Another approach is to integrate the data analysis and the document by embedding the analysis within the document itself. We should be working toward the simultaneous documentation of process, workflow, and generation

of statistical results. This is called *Literate Statistical Practice* by Rossini and Leisch (2003). Literate statistics discourages the construction of many little unconnected pieces and focuses on producing a representation of the whole data analysis project. Understandable, documented, reproducible analysis increases the comprehension of others and is critical for acceptance of results especially in a regulated environment.

Choice of the publishing environment will be a critical choice in the ability to document statistical analyses in the future. Overall we feel the industry has a long way to go to achieve a goal of literate statistical practice.

REFERENCES

APT Software Library (2004). Copyright © Hygia Biostat, Inc., Oakland, CA.

Diabetes Control and Complications Trial Research Group. (1993) The effect of intensive treatment of diabetes on the development and progression of long-term complications in insulin-dependent diabetes mellitus. *New England Journal of Medicine* 329:977-986.

Hopkins, A. and Collins, L. (2005) Statistical Table Specifications and Automatic Code Generation using XML. PharmaSUG 2005 Proceedings.

Palmer, Michael. (2002) "XML Erases the Barriers between Data and Documents in Clinical Research". Zurich Biostatistics, Inc.

Harrell, F.E. Statistical Tables and Plots using S and LATEX, February 15, 2004. http://biostat.mc.vanderbilt.edu/twiki/pub/Main/StatReport/summary.pdf.

Rossini, A.. and Leisch, F. (2003) Literate Statistical Practice. Berkeley Electonic Press. http://www.bepress.com/uwbiostat/paper194

Walsh, N. (1999). "Organization for the Advancement of Structured Information Standards" Technical Memorandum TR 9901. www.oasis-open.org/specs/tm9901.htm