

Using SAS[®] with XML to Create Custom Formatted Excel[®] Workbooks

James Van Campen, SRI International, Menlo Park, CA

ABSTRACT

This paper describes how to use SAS in conjunction with XML to produce custom formatted workbooks in Excel. A variety of methods for getting SAS output into Excel already exist. The primary advantage of the XML method outlined here is that custom workbook formatting can be specified directly from SAS. The method makes use of the Microsoft[®] XML Spreadsheet (XMLSS) schema, which permits specification of an Excel workbook from an XML file. The method is illustrated by creating an Excel workbook with multiple worksheets, in which each sheet is formatted differently. No prior knowledge of XML is assumed.

INTRODUCTION

Custom formatting of Excel workbooks has been described by some as the Holy Grail of SAS programming. Well, look no further, all ye knights of SAS. The Grail has been found, and it is XML. The method described in this paper is a consequence of three facts. First, it is easy to write SAS output to a text file. Second, XML files are text files. Third, Microsoft created XMLSS, which specifies how Excel workbooks can be written to and read from XML files. Thus, custom formatted Excel workbooks can be created from SAS by writing an XML file that conforms to XMLSS. This paper begins with an introduction to XML and XMLSS. Then the SAS code necessary to generate an XML file is discussed. Finally, a detailed example is given that illustrates a variety of formatting. The examples in this paper were done with SAS 9.1 and Microsoft Excel 2002 under the Windows XP operating system. Older versions of Excel that do not support XML can not be used with this method.

XML AND XMLSS

XML stands for Extensible Markup Language. In XML, nested markup tags are used to define a hierarchical structure for the data. XML can be used for just about any type of data, including Excel spreadsheets. Unlike HTML, XML has no predefined tags. XML schemas are used to define the tags, the nested structure, and all attributes. Schemas are a kind of XML file. The tags define what are called elements. Typically, elements are data. An element (parent element) may contain other elements (child elements). The parent-child relationship of the elements provides the hierarchical structure to the data. All XML files must be fully nested; that is, both the opening and closing tags of all child elements must be contained between the opening and closing tags of the parent element. All XML files have a root tag that contains all the other tags. Elements may have attributes. Typically, attributes are meta-data, such as formatting information. Some attributes are required and some are optional. XML parsers are very picky and will not parse a file that is not well formed and valid. Well formed means the file follows the XML syntax (e.g., the tags are properly nested, all required closing tags exist, etc.). Valid means the XML file conforms to the associated schema (e.g., the element names and attributes are consistent with the schema). An XML file can not be valid without also being well formed; however, a well-formed file is not necessarily valid.

Microsoft's XMLSS was developed to permit Excel workbooks to be saved as XML and XML files conforming to XMLSS to be opened as formatted workbooks. Excel is capable of opening any well-formed XML file; however, only valid files conforming to XMLSS can modify the default method of display. Details of XMLSS, including descriptions of the elements (tags) and attributes, can be found at the following website:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexcl2k2/html/odc_xmlss.asp

An example will be used here to illustrate the essentials of XMLSS. The contents of a simple Excel workbook, consisting of one worksheet with three rows and two columns, are shown in Figure 1. Default formatting is used throughout the worksheet, except for the column headings, which are bold.

Name	Age
Jack	9
Jill	7

Figure 1

The following XML file conforms to XMLSS and, when opened with Excel, has the same contents and formatting as Figure 1. The first tag is a special tag that indicates this is an XML file and specifies the XML version. The root element is Workbook. All the other elements in the file are contained between the opening and closing Workbook tags. The Workbook tag contains two attributes. The attributes specify namespaces where the elements and attributes are defined. A discussion of namespaces is beyond the scope of this paper, and the specification of the namespace attributes can be treated as boilerplate.

```

<?xml version="1.0"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet "
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">
  <Styles>
    <Style ss:ID="s21">
      <Font ss:Bold="1"/>
    </Style>
  </Styles>
  <Worksheet ss:Name="Sheet1">
    <Table>
      <Row>
        <Cell ss:StyleID="s21">
          <Data ss:Type="String">Name</Data>
        </Cell>
        <Cell ss:StyleID="s21">
          <Data ss:Type="String">Age</Data>
        </Cell>
      </Row>
      <Row>
        <Cell>
          <Data ss:Type="String">Jack</Data>
        </Cell>
        <Cell>
          <Data ss:Type="Number">9</Data>
        </Cell>
      </Row>
      <Row>
        <Cell>
          <Data ss:Type="String">Jill</Data>
        </Cell>
        <Cell>
          <Data ss:Type="Number">7</Data>
        </Cell>
      </Row>
    </Table>
  </Worksheet>
</Workbook>

```

Similar to HTML, XML tags begin with “<” and end with “>”. The element name is the first thing inside the tag. The name is followed by the attributes, if any. The first child element of Workbook is Styles. Each style is defined separately with Style tags. The Style element is a child of the Styles element. In this example, only one style is defined. The ID is a required attribute of the Style element and is set to “s21”. The “ss:” part of the attribute specification indicates the namespace where the attribute is defined. A default style can be defined, but it is not required; no default style is defined in this example. In the style defined, the Bold attribute of the Font element is set to “1” (True). Closing tags for elements differ from opening tags by preceding the name with a forward slash and not having any attributes specified. All non-empty elements must have both opening and closing tags for the XML file to be well formed. Empty elements may have only one tag. In those cases, the closing bracket is preceded by a forward slash. The Font element is an example of an empty element.

After the Styles element comes the Worksheet element. The worksheet name is a required attribute of the Worksheet element. In this case, it is set to “Sheet1”, the Excel default. The Table element is a child of the Worksheet element. The Column and Row elements are children of the Table element. In this example, there are no Column elements. The Cell elements are children of the Row elements, and two of the Cell elements have their StyleID attribute set to “s21” for the bold font style. The Data element is a child of the Cell element. Type is a required attribute of the Data element. Type can be set to “String”, “Number”, “DateTime”, “Boolean”, or “Error”. The data is written, without quotes, between the opening and closing data tags. In XML, white space is preserved. No spaces are allowed to precede or to follow numeric data. In fact, in XMLSS, no characters other than numerals and one decimal point are permitted in numeric data. This has important consequences when writing your SAS code, as will be seen in the next section. After all the rows are specified, the closing Table, Worksheet, and Workbook tags are written. The XML code above constitutes a well-formed and valid XML file conforming to XMLSS.

To create workbooks with formatting more complex than in this example requires using additional elements and attributes from XMLSS. One way to learn more about XMLSS is to go to the website specified above. Often the quickest and easiest way to learn about new elements and attributes in XMLSS is to create a workbook with the

formatting you desire and save it as XML. You can then open the XML file created by Excel with a text editor or your Internet browser and see the XML code. One thing to note is that the XML files created by Excel are quite verbose; that is, they contain a lot of nonessential XML code. The XML code shown in the example above is the minimum required to conform to XMLSS and specify the workbook. The XML file created by Excel when the Workbook was saved as XML contained about twice as much code.

SAS CODE FOR WRITING AN XML FILE

The SAS code for writing an XML file is in concept quite simple. You use a `_null_ DATA` step with `put` statements that write to a text file with a `.xml` extension. The devil is in the details. If your XML has one mistake (e.g., a space after numeric data, an omitted bracket, a misspelled attribute, etc.), then the file will not be read by the parser.

To create the XML file, you need to know when the first record is processed so the introductory XML code can be written. You also need to know when the last record is processed so the closing code can be written. Each worksheet in the workbook requires a separate `DATA` step, as will be illustrated in the example in the following section. All the `DATA` steps must write to the same XML file. SAS code for writing the XML file in the previous example is shown below.

```

data ages;
    input name $ age;
    datalines;
Jack 9
Jill 7
    ;
run;

data _null_;
    file "&output\Book1.xml";
    set ages end= last;
    if _n_=1 then put
        '<?xml version="1.0"?>' /
        '<Workbook    xmlns="urn:schemas-microsoft-com:office:spreadsheet" ' /
        '    xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet" >' /
        '    <Styles>' /
        '        <Style ss:ID="s21">' /
        '            <Font ss:Bold="1"/>' /
        '        </Style>' /
        '    </Styles>' /
        '    <Worksheet ss:Name="Sheet1">' /
        '        <Table>' /
        '            <Row>' /
        '                <Cell ss:StyleID="s21">' /
        '                    <Data ss:Type="String">Name</Data>' /
        '                </Cell>' /
        '                <Cell ss:StyleID="s21">' /
        '                    <Data ss:Type="String">Age</Data>' /
        '                </Cell>' /
        '            </Row>;
    put
        '    <Row>' /
        '        <Cell>' /
        '            <Data ss:Type="String">' name ' </Data>' /
        '        </Cell>' /
        '        <Cell>' /
        '            <Data ss:Type="Number">' age +(-1) ' </Data>' /
        '        </Cell>' /
        '    </Row>;
    if last then put
        '        </Table>' /
        '    </Worksheet>' /
        '</Workbook>' ;
run;

```

Before outputting the first row of data, the XML version tag, the opening workbook tag, the styles, the opening worksheet tag, the opening tables tag, and any titles or column headers must be output. The second `put` statement

outputs the data. After outputting the last row of data, the closing table tag, the closing worksheet tag, and the closing workbook tag must be output. If the workbook has multiple worksheets, the closing workbook tag is not written until the end of the final DATA step. In the first put statement, the text strings "Name" and "Age" are output as column headings. However, in the second put statement, the values of the variables, name and age, are output. Put statements may append a space to the variable values written. For string data, this is not a catastrophic problem; however, for numeric data, the space character makes the XML file invalid. To eliminate this problem, the put statement must be instructed to move the cursor back one space before writing the closing data tag. That is why "+(-1)" is written after the variable names. The forward slashes in the put statements move the cursor to the next line. In general, single quotes are used in the put statements since the XML attribute assignments require the use of double quotes. You must remove any SAS formats before outputting numeric data to the XML file. Commas and dollar signs in numeric data will make the XML file invalid.

EXAMPLE: WORKBOOK WITH MULTIPLE WORKSHEETS AND COMPLEX FORMATTING

In the following example, a workbook with two worksheets is created. The XML file is named SalesReport.xml. The first worksheet is named Sales_Summary and the second is named Sales_Data. The worksheets are created from SAS datasets with the same names. The Sales_Data dataset contains 130 records and has the variables salesperson, date, units, price, and sale. The Sales_Summary dataset contains 11 records and the variables salesperson, totalunits, totalsales, and averageprice. The last record of the Sales_Summary dataset has the totals for all salespersons.

The first worksheet, shown in Figure 2, contains the summary report. In addition, the page orientation will be set to landscape, and a footer will be added.

Sales Summary			
<u>Salesperson</u>	<u>Total Units</u>	<u>Total Sales</u>	<u>Average Price</u>
Baisden	1,446	\$40,897	\$28.28
Barat	1,861	\$58,504	\$31.44
Godard	2,014	\$63,388	\$31.47
McCracken	2,256	\$66,283	\$29.38
Miller	1,844	\$59,161	\$32.08
Radbill	1,995	\$62,088	\$31.12
Singer	1,967	\$55,344	\$28.14
Stevenson	1,697	\$56,775	\$33.46
Valdes	1,764	\$56,776	\$32.19
Williamson	1,592	\$47,466	\$29.82
Total	18,436	\$566,682	\$30.74

Figure 2

The formatting issues to be addressed for the first worksheet are as follows:

1. Setting row heights and column widths
2. Setting font type, size, and color
3. Merging cells
4. Setting background color
5. Justifying cell contents
6. Setting bold and underline
7. Setting numeric formats
8. Setting page orientation
9. Creating footers

The second worksheet contains the sales data. The data will be displayed as in Figure 3. In addition, the first row (column headings) will be repeated at the top of each page, and a header and footer will be added. For the sake of brevity, only the first 20 lines are displayed.

Salesperson	Date	Units	Price	Sale
Baisden	1/14/2002	79	\$25.00	\$1,975.00
Baisden	6/12/2002	61	\$33.00	\$2,013.00
Baisden	10/25/2002	227	\$28.00	\$6,356.00
Baisden	10/30/2002	212	\$26.00	\$5,512.00
Baisden	1/14/2003	49	\$22.00	\$1,078.00
Baisden	5/28/2003	168	\$22.00	\$3,696.00
Baisden	6/8/2003	63	\$34.00	\$2,142.00
Baisden	7/1/2003	76	\$37.00	\$2,812.00
Baisden	8/31/2003	46	\$37.00	\$1,702.00
Baisden	10/1/2003	269	\$30.00	\$8,070.00
Baisden	10/16/2003	49	\$25.00	\$1,225.00
Baisden	11/1/2003	85	\$34.00	\$2,890.00
Baisden	11/18/2003	62	\$23.00	\$1,426.00
Barat	2/12/2002	130	\$22.00	\$2,860.00
Barat	2/14/2002	104	\$22.00	\$2,288.00
Barat	2/28/2002	165	\$30.00	\$4,950.00
Barat	9/20/2002	271	\$33.00	\$8,943.00
Barat	10/4/2002	209	\$33.00	\$6,897.00
Barat	11/15/2002	231	\$33.00	\$7,623.00

Figure 3

The formatting issues to be addressed for the second worksheet are as follows:

1. Setting column widths
2. Setting bold
3. Justifying cell contents
4. Setting numeric and date formats
5. Setting gridlines
6. Creating footers and headers
7. Setting the first row to repeat at the top of each page

Let us now examine the SAS code for outputting an XML file that will specify the workbook. This program begins much like the program in the first example, except the numeric variables have their formats removed and additional namespace attributes are specified in the Workbook tag. In the first example, the numeric data was unformatted and the format removal step was omitted. The 'x' namespace is the only additional namespace used in the following program. However, all the namespaces are included for completeness.

In this example, 17 different styles are created for use in the two worksheets. Left-right justification, vertical alignment, and indenting are all controlled using the Alignment tag. Number formats are set with the Format attribute in the NumberFormat tag. Fontname, Size, Color, Bold, and Underline are all attributes of the Font tag. Cell background colors are set using the Color and Pattern attributes of the Interior tag. More than half of this program is devoted to establishing the styles.

```

data _null_;
  file "&output\SalesReport.xml";
  set datalib.sales_summary end= last;
  format _numeric_;
  if _n_=1 then put
    '<?xml version="1.0"?>' /
    '<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet" ' /
    '  xmlns:o="urn:schemas-microsoft-com:office:office" ' /
    '  xmlns:x="urn:schemas-microsoft-com:office:excel" ' /
    '  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet" ' /

```

```

'         xmlns:html="http://www.w3.org/TR/REC-html40" >' /
' <Styles>' /
'   <Style ss:ID="s21">' /
'     <Font ss:Bold="1"/>' /
'   </Style>' /
'   <Style ss:ID="s22">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <NumberFormat ss:Format="#,##0"/>' /
'   </Style>' /
'   <Style ss:ID="s23">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Bold="1"/>' /
'   </Style>' /
'   <Style ss:ID="s24">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <NumberFormat ss:Format="Short Date"/>' /
'   </Style>' /
'   <Style ss:ID="s25">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <NumberFormat
ss:Format="&quot;,$&quot;#,##0.00_); \(&quot;,$&quot;#,##0.00\)" />' /
'   </Style>' /
'   <Style ss:ID="s26">' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'   </Style>' /
'   <Style ss:ID="s27">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <NumberFormat ss:Format="#,##0"/>' /
'   </Style>' /
'   <Style ss:ID="s28">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <NumberFormat ss:Format="&quot;,$&quot;#,##0"/>' /
'   </Style>' /
'   <Style ss:ID="s29">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <NumberFormat ss:Format="&quot;,$&quot;#,##0.00"/>' /
'   </Style>' /
'   <Style ss:ID="s30">' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <Interior ss:Color="#CCFFFF" ss:Pattern="Solid"/>' /
'     <NumberFormat/>' /
'   </Style>' /
'   <Style ss:ID="s31">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <Interior ss:Color="#CCFFFF" ss:Pattern="Solid"/>' /
'     <NumberFormat ss:Format="#,##0"/>' /
'   </Style>' /
'   <Style ss:ID="s32">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <Interior ss:Color="#CCFFFF" ss:Pattern="Solid"/>' /
'     <NumberFormat
ss:Format="&quot;,$&quot;#,##0_); \(&quot;,$&quot;#,##0\)" />' /
'   </Style>' /
'   <Style ss:ID="s33">' /
'     <Alignment ss:Horizontal="Center"/>' /
'     <Font ss:Size="12" ss:Bold="1"/>' /
'     <Interior ss:Color="#CCFFFF" ss:Pattern="Solid"/>' /
'     <NumberFormat
ss:Format="&quot;,$&quot;#,##0.00_); \(&quot;,$&quot;#,##0.00\)" />' /
'   </Style>' /

```

```

' <Style ss:ID="s34">' /
' <Alignment ss:Horizontal="Center"/>' /
' <Font ss:Size="12" ss:Bold="1" ss:Underline="Single"/>' /
' </Style>' /
' <Style ss:ID="s35">' /
' <Alignment ss:Horizontal="Left"/>' /
' <Font ss:Size="12" ss:Bold="1" ss:Underline="Single"/>' /
' </Style>' /
' <Style ss:ID="s36">' /
' <Alignment ss:Horizontal="Left" ss:Indent="1"/>' /
' <Font ss:Size="12" ss:Bold="1"/>' /
' </Style>' /
' <Style ss:ID="s37">' /
' <Alignment ss:Horizontal="Center" ss:Vertical="Center"/>' /
' <Font ss:FontName="Times New Roman" ss:Size="20" ss:Bold="1"
ss:Color="#0000FF"/>' /
' <Interior ss:Color="#FF99CC" ss:Pattern="Solid"/>' /
' </Style>' /
' </Styles>' /

```

The first worksheet has four equal-width columns. Column width is set with the Width attribute of the Column tag. Notice the Span attribute is set to "3" in the Column tag. That means the following three columns get the same formatting. In the first row, the four cells are merged. The MergeAcross attribute is set to "3" in the first cell tag, which means the cell is merged with the following three cells. The StyleID for the cell is set to "s37", which corresponds to the style with the 20 pt Times New Roman font. Below the title row are the column headings. The code for the column headings is similar to that used in the first example. Since the bottom row of the table has different formatting from the other data rows, it is output in the third put statement with the closing code instead of in the second put statement with the main body of data. The Orientation attribute of the Layout tag is set to "Landscape" and the Data attribute of the Footer tag is specified. Both the Layout and Footer tags are children of the PageSetup element, which is in turn a child of the WorksheetOptions element. Note that no Workbook tag is written at the end of the first DATA step.

```

' <Worksheet ss:Name="SALES_SUMMARY">' /
' <Table>' /
' <Column ss:Width="156" ss:Span="3"/>' /
' <Row ss:Height="37">' /
' <Cell ss:MergeAcross="3" ss:StyleID="s37">' /
' <Data ss:Type="String">Sales Summary</Data>' /
' </Cell>' /
' </Row>' /
' <Row ss:Height="22">' /
' <Cell ss:StyleID="s35">' /
' <Data ss:Type="String">Salesperson</Data>' /
' </Cell>' /
' <Cell ss:StyleID="s34">' /
' <Data ss:Type="String">Total Units</Data>' /
' </Cell>' /
' <Cell ss:StyleID="s34">' /
' <Data ss:Type="String">Total Sales</Data>' /
' </Cell>' /
' <Cell ss:StyleID="s34">' /
' <Data ss:Type="String">Average Price</Data>' /
' </Cell>' /
' </Row>' ;
* Output all the data rows except the last which is formatted differently;
if not last then put
' <Row ss:Height="22">' /
' <Cell ss:StyleID="s36">' /
' <Data ss:Type="String">' salesperson +(-1) '</Data>' /
' </Cell>' /
' <Cell ss:StyleID="s27">' /
' <Data ss:Type="Number">' totalunits +(-1) '</Data>' /
' </Cell>' /
' <Cell ss:StyleID="s28">' /
' <Data ss:Type="Number">' totalsales +(-1) '</Data>' /

```

```

'           </Cell>' /
'           <Cell ss:StyleID="s29">' /
'               <Data ss:Type="Number">' averageprice +(-1) '</Data>' /
'           </Cell>' /
'       </Row>' ;
* The last row is output with different formatting;
if last then put
'       <Row ss:Height="22">' /
'           <Cell ss:StyleID="s30">' /
'               <Data ss:Type="String">' salesperson +(-1) '</Data>' /
'           </Cell>' /
'           <Cell ss:StyleID="s31">' /
'               <Data ss:Type="Number">' totalunits +(-1) '</Data>' /
'           </Cell>' /
'           <Cell ss:StyleID="s32">' /
'               <Data ss:Type="Number">' totalsales +(-1) '</Data>' /
'           </Cell>' /
'           <Cell ss:StyleID="s33">' /
'               <Data ss:Type="Number">' averageprice +(-1) '</Data>' /
'           </Cell>' /
'       </Row>' /
'   </Table>' /
'   <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">' /
'       <PageSetup>' /
'           <Layout x:Orientation="Landscape"/>' /
'           <Footer x:Data="&amp;L&amp;14James Van Campen&amp;R&amp;14SRI
International"/>' /
'       </PageSetup>' /
'   </WorksheetOptions>' /
' </Worksheet>' /;
run;

```

The second worksheet is specified with a separate DATA step. To append the output to what has already been written to SalesReport.xml in the first DATA step, the mod option is used with the file statement. Again, the numeric variables have their formats removed. This table is three pages long. To have the first row (column headings) repeat at the top of each page, the NamedRange element must have the Name attribute set to "Print_Titles" and the RefersTo attribute set to "=SALES_DATA!R1". The NamedRange element is a child of the Names element. The code for the column headings and data is similar to what was done in the first example, with one exception. SAS date values are meaningless in Excel. Thus, to get date data into Excel from SAS, the dates must be output with the yymmdd10 format and have the string "T00:00:00.000" appended to make a datetime value that conforms to XMLSS. Outputting dates is one place where the "+(-1)" is not needed to move the cursor back a space. Headers are specified the same way as footers, except with a Header tag. The Gridlines element is specified inside the Print element. Finally, the closing Workbook tag is written.

```

data _null_;
file "&output\SalesReport.xml" mod;
set datalib.sales_data end= last;
format _numeric_;
if _n_=1 then put
'   <Worksheet ss:Name="SALES_DATA">' /
'       <Names>' /
'           <NamedRange ss:Name="Print_Titles" ss:RefersTo="=SALES_DATA!R1"/>' /
'       </Names>' /
'   <Table>' /
'       <Column ss:Width="90" ss:Span="4"/>' /
'   <Row>' /
'       <Cell ss:StyleID="s21">' /
'           <Data ss:Type="String">Salesperson</Data>' /
'       </Cell>' /
'       <Cell ss:StyleID="s23">' /
'           <Data ss:Type="String">Date</Data>' /
'       </Cell>' /
'       <Cell ss:StyleID="s23">' /
'           <Data ss:Type="String">Units</Data>' /
'       </Cell>' /

```

```

'          <Cell ss:StyleID="s23">' /
'              <Data ss:Type="String">Price</Data>' /
'          </Cell>' /
'          <Cell ss:StyleID="s23">' /
'              <Data ss:Type="String">Sale</Data>' /
'          </Cell>' /
'      </Row>' ;

put      '          <Row>' /
'          <Cell>' /
'              <Data ss:Type="String">' salesperson +(-1) '</Data>' /
'          </Cell>' /
'          <Cell ss:StyleID="s24">' /
'              <Data ss:Type="DateTime">' date yymmdd10.
'T00:00:00.000</Data>' /
'          </Cell>' /
'          <Cell ss:StyleID="s22">' /
'              <Data ss:Type="Number">' units +(-1) '</Data>' /
'          </Cell>' /
'          <Cell ss:StyleID="s25">' /
'              <Data ss:Type="Number">' price +(-1) '</Data>' /
'          </Cell>' /
'          <Cell ss:StyleID="s25">' /
'              <Data ss:Type="Number">' sale +(-1) '</Data>' /
'          </Cell>' /
'      </Row>' ;

if last then put
'      </Table>' /
'      <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">' /
'          <PageSetup>' /
'              <Header x:Data="&C&Amp;&quot;Arial,Bold&quot;SALES DATA"/>' /
'              <Footer x:Data="&R&Amp;&quot;Arial,Bold&quot;Page &P"/>' /
'          </PageSetup>' /
'          <Print>' /
'              <Gridlines/>' /
'          </Print>' /
'      </WorksheetOptions>' /
'  </Worksheet>' /
'</Workbook>' ;

run;

```

CONCLUSION

The method presented here is the first to make it possible to control all the formatting features of Excel directly from SAS without using ODS. As such, it is pretty cool. However, the method has some drawbacks. It is complex and requires learning about XML and XMLSS. The method is definitely not appropriate for producing small, one-of-a-kind workbooks. However, it can be a great time saver in situations where the same formatted Excel report is created repeatedly. The time invested in setting up the SAS code in those situations would be small compared with formatting each report by hand. The example programs here were written for illustrative purposes, and they are somewhat bulky. Much can be done to streamline and automate the code. For example, frequently used styles could be written in a separate file and called with the %include statement. The purpose of this paper was to bring to light another method of using SAS to create formatted output in Excel. Hopefully, the merits of the method are evident and further work will be done to refine the techniques presented here.

REFERENCES

Ted Conway, *Another Shot at the Holy Grail: Using SAS to Create Highly Customized Excel Workbooks*, SUGI 28
 Vincent DelGobbo, *From SAS to Excel Via XML*, NESUG 2004
 Peter Godard and Cyndi Williamson, *Using Microsoft Excel for Data Presentation*, WUSS 12

ACKNOWLEDGMENTS

The author would like to thank Patrick Thornton for proposing the idea for this paper. In addition, the author is grateful to Patrick for his advice, encouragement, and numerous stimulating conversations about SAS programming. Many thanks also go to Klaus Krause for his editing services. Any errors, omissions, or stylistic flaws in the paper are entirely my own.

CONTACT INFORMATION

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

James Van Campen
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025
Work Phone: 650-859-2906
Email: james.vancampen@sri.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.