The Perfect Marriage: The SAS® Output Delivery System (ODS) and Microsoft Office

Chevell Parker, SAS Institute

ABSTRACT

A compatible relationship is one in which the associated entities complement each other's strengths and compensate for each other's weaknesses. This relationship is certainly the case when you combine the power of ODS and the flexibility of Microsoft Office. This paper describes how you can create outstanding spreadsheets via the harmonious union of ODS with Microsoft Office applications. The paper explores the relationships among ODS destinations (for example, ExcelXP and MSOffice2K) as well as custom tagsets (for example, the TableEditor tagset). In addition, the paper explains how these components enable you to generate highly presentational spreadsheets.

This paper covers the following topics:

- specific layout techniques such as paneling tables and graphics and positioning output in a worksheet
- common issues (related to styles, cell formatting, and so on) that occur when you export data
- customization of your output in Microsoft Word using the RTF destination and the tagset
- generated output that can be read into Microsoft Access and other Office applications

INTRODUCTION

One of the world's leading TV marriage counselors, Dr. Phil McGraw (Dr. Phil), is known for the following formula for a successful relationship: "The quality of a relationship is a function of the extent to which it is built on a solid underlying relationship and meets the needs of the two parties involved." (McGraw 2000) That sage advice refers to people, but the same can be said for other relationships. For example, the courtship between the SAS Output Delivery System (ODS) and Microsoft Office began with the release of SAS® 7.0. SAS 7.0 was the first release to incorporate the Output Delivery System. The first two ODS destinations were ODS HTML and ODS OUTPUT. The HTML destination enabled you to create customized, presentation-quality output that other formatting methods (for example, the CSV destination or the EXPORT procedure) did not offer for data exported to Microsoft Excel. The presentational output method enabled you to modify fonts and colors, use traffic lighting, add hyperlinks, and so on. Later, SAS 8.1 added the ODS RTF destination, which enabled the creation of output that was compatible with word-processing applications such as Microsoft Word. Then, SAS 8.2 added the CSV destination, which provided the ability to create CSV files from all procedures and from the DATA step. These solid underpinnings brought to fruition a true marriage between SAS and Microsoft Office. That marriage continues today with SAS 9.2, a relationship that is ever growing and changing to meet evolving market needs.

This paper is intended for all levels of users. It explores how you can take advantage of the marriage between SAS ODS and Microsoft Office to create effective, presentational output. After a brief overview about ODS tagsets, the paper discusses the following topics:

- Enhancing your presentation
    - creating and modifying styles
    - positioning output within a worksheet
    - paneling tables or graphics
    - inserting page breaks
    - resolving workbook size issues
    - resolving cell-formatting issues
- Exporting output to Microsoft Access and Microsoft Word

SAS ODS TAGSETS

Before delving too deeply, a discussion of tagsets is in order. Some of the examples in this document use the ODS Markup language to modify the Tagsets.ExcelXP or MSOffice2K tagsets.

A *tagset* is simply a collection of events that are triggered in a particular order. The implementation of tagsets revolutionized SAS because they provide complete control over tasks such as reordering, removing, hiding, and modifying output. The CSV, MSOffice2K, and ExcelXP destinations were created using ODS Markup language. SAS®9 and later includes the ODS Markup facility, which gives you access to all of the tags (or, tagsets) that are responsible for the ODS destination.

With the ODS Markup language, you can modify the tagset that is responsible for a particular ODS destination as well as add tagset options that enable user-generated modifications. You can even modify the tagset to create an entirely new ODS destination. The measured RTF destination, discussed later in "The ODS Measured RTF Tagset," is a new destination that was created this way. That destination is new in SAS 9.2. Because you now have the power to create the markup of your choice, you no longer have to wait, in some instances, for new releases of SAS software to obtain enhancements or fixes. You can make many of the enhancements or updates directly to the tagset. The abbreviated tagset examples that are used in this paper are available at http://www.sas.com/techsup/download/blind/sgf36.zip.

Tagsets play a vital role in exporting data to Excel. Once you compile a new tagset, it is installed in the SASUSER library, from which it is always available to you. To see a list all of the tagsets that are installed by SAS, submit the following TEMPLATE procedure:

```
proc template;
    list tagsets;
run;
```

The tagsets that generate output specifically for Excel are as follows:

- Tagsets.ExcelXP (hereafter referred to as ExcelXP)
- MSOffice2K
- CSV
- HTML3
- Tagsets.MSOffice2K_X (hereafter referred to as MSOffice2K_X. This is a sample tagset that adds options to the MSOffice2K destination). You can download these tagsets from support.sas.com/rnd/base/ods/odsmarkup.
- Tagsets.TableEditor (hereafter referred to as TableEditor). This sample tagset modifies and generates Excel output and enhances Web output). You can download this tagset from support.sas.com/rnd/base/ods/odsmarkup.

The following table lists the benefits of each of these tagsets:

Benefit	ExcelXP	MSOFFICE2K / HTML3	CSV	MSOFFICE2K_X	TableEditor
Creates Multiple Worksheets	✓			✓	✓
Creates Graphics		✓		✓	✓
Adds or Modifies Styles	✓	✓		✓	✓
Uses Options	✓		✓	✓	✓
Updates Worksheets					✓
Creates Output on all Operating Systems	✓	✓	✓	✓	✓

Table 1. Available Tagsets and Their Benefits

The next table lists available ODS destinations, their formats, and requirements.

ODS Destination	SAS® Release Required	File Format	Excel Version Required
ExcelXP	SAS 9.1	XML	2002+
MSOffice2K/HTML3	SAS 9.0	HTML	97+
HTML	SAS 7.0	HTML	97+
CSV	SAS 8.2	CSV	97+

Table 2. Available ODS Destinations with Formats and Software Requirements

ENHANCING YOUR PRESENTATION

This section presents examples that show you how to get the most out of your presentation style. The discussion presents examples that illustrate how to enhance your presentation by creating styles, highlighting rows and columns, positioning output in a worksheet, implementing page breaks, and more.

CREATING AND MODIFYING STYLES

If you're like a lot of people, you might find yourself wearing the same clothes and styles all the time. However, with your spouse's help, you learn how to diversify your wardrobe by adding more color and other styles.

This is no different from the selection of a style to highlight your data in Excel. There are over 50 styles and many colors to choose from, all of which are shipped with SAS. However, you need to look at the styles to determine which one to use or select as a base style for modification. For example, style elements that you might consider include the following:

- **colors:** You want to select a color that best highlights your output. In addition, you want to make sure that the color that you choose renders correctly in your audience's worksheet. In versions of Microsoft Excel prior to Excel 2007, colors are limited to those on the 56-color palette. Excel 2007 and later provide over 16 million colors.
- **fonts:** You want to make sure that the font complements your data and makes that data stand out.
- **borders:** For example, you should consider whether borders make the data easier or harder to read.
- **font sizes:** Your font size should appear the way your users expect, and it should enable them to see enough data without having to scroll or link to more pages.
- **gridlines:** You should consider whether gridlines will make your reports look the way customers expect (for example, a standard Excel worksheet that has gridlines by default).
- **row banding (highlighting)**: Consider whether alternating bands of color in your tables might make your report more readable.

These are all elements you might want to consider when you are selecting a style. To see a list of all of the styles shipped with SAS, use the LIST statement in the TEMPLATE procedure, as follows:

```
proc template;
    list styles;
run;
```

The following images show four of the popular styles that are used for worksheets:

Minimal   **Default**   **SASWeb**   **Normal**

To see a display of all of the styles that are shipped with SAS, see SAS Note 36900, "Instructions for viewing all of the style templates that are shipped with the SAS® System." (SAS Institute Inc. 2011) This note contains a sample program that you can use to create a file that displays all of the styles. Custom styles that you can use with the ExcelXP tagset are available at http://www.sas.com/techsup/download/blind/sgf36.zip.

You can create and modify styles with PROC TEMPLATE. In addition to PROC TEMPLATE, you can use the following tools to modify styles for MSOffice2K and HTML output:

- the TEMPLATE procedure
- inline formatting and the override feature that is available with the STYLE= option
- the STYLESHEET= option (which enables you to create or a use a cascading style sheet [CSS] file) and the CSSSTYLE= option (which enables you to use a CSS file to display output)

These features, with the exception of the STYLESHEET= option, can also be used by the ExcelXP destination.

In the following example, the PROC TEMPLATE and PROC REPORT steps modify the **Minimal** style.

Example

```
proc template;
    define style styles.test;
        parent=styles.minimal;
        style body from body     /  background=transparent;
        style table from table   /  borderwidth=0      cellspacing=0;
        style header from header /  font_weight=bold  font_size=10pt
                                    font_style=italic borderrightwidth=1
                                    bordertopwidth=1  borderbottomwidth=2;
        style data from data     /   bordertopwidth=0  borderbottomwidth=0
                                     borderleftwidth=1 borderrightwidth=1;
    end;
run;

ods tagsets.excelxp file="c:\temp\temp.xml" style=styles.test;

proc report data=sashelp.class nowd;
    define age / order;
    compute age;
        count+1;
        if mod(count,2) then call define(_row_,"style","style={background=#e0ffff}");
    endcomp;
run;

ods tagsets.excelxp close;
```

In This Example:

- PROC TEMPLATE uses STYLE statements to modify the borders, gridlines, and the fort style and font weight for the header.
- In the REPORT procedure, the DEFINE statement employs a compute block with the MOD function to determine which style and color to display on every other line.

Display 1 shows the output that is generated by this program.

Display 1. Using PROC TEMPLATE and PROC REPORT to Modify the Minimal Style

HIGHLIGHTING (BANDING) ROWS AND COLUMNS

If you have a large amount of data in the worksheet, you can make that that data more readable, visually, by using alternating colors for rows or columns (or groups of rows and groups of columns). As shown in the preceding example, you can implement this popular technique, called *banding*, by using a compute block with the MOD function in PROC REPORT. When you add or delete a row in PROC REPORT, the banding is not maintained.

Excel encompasses a similar technique that also uses the MOD function when you create a conditional format. In Excel, this technique is rule based, and you specify it based on the row number. Therefore, if you delete or add a row, the color banding does remain intact. So if you want to further modify the report in Excel, it might be better to create a conditional format in Excel.

The following conditional formatting element, with the MOD function embedded, supplies a style on every other row, based on the range that is specified. This example adds a style to the rows, but you can also add styles to columns.

```
<ConditionalFormatting xmlns="urn:schemas-microsoft-com:office:excel">
    <Range>R2C1:R20C6</Range>
    <Condition>
        <Value1> MOD(ROW(),2)=0</Value1>
        <Format Style="background-color:lightblue"/>
    </Condition>
</ConditionalFormatting>
```

The XML tagging in this example demonstrates a conditional format that is used within the XML Spreadsheet format. You can add multiple conditions within in the element, as well. The **<ConditionalFormatting>** element is included in the **<WorkSheet>** element of the XML markup. You can modify the MOD function value to display the style for any number of rows that you want. You can even specify the style based on the data value in the cell.

To implement Excel's MOD function into the tagset, you must generate the required XML Markup and supply it in the location that Excel expects. That location is in the markup's **<WorkSheet>** element. The tagset event that is created (Write_Conditional_Format) is triggered, or called, within the Worksheet_Head_end tagset event. The TEMPLATE procedure creates a new tagset (Tagsets.ExcelXP_Mod) that inherits all of the additional events from the ExcelXP tagset. To supply a style for the **<ConditionalFormatting>** tag, memory variables are created based on the options that are passed in the code. In this example, the style is to create specific background and foreground colors on every other row.

The following example is an abbreviated version of the tagset code. For this example (and all examples in this paper), you can download the full code from http://www.sas.com/techsup/download/blind/sgf36.zip.

Example

```
proc template;
    define tagsets.excelxp_mod;
        parent=tagsets.excelxp;
        define event worksheet_head_end;
        . . .more SAS statements. . .
        put "</x:WorksheetOptions>" NL;
            trigger write_conditional_format ;
            trigger write_autofilter;
    end;

    define event write_conditional_format;
        put "<ConditionalFormatting xmlns=""urn:schemas-microsoft-
             com:office:excel"">" NL;
        put "<Range>";
        put "R2C1:R" $worksheet_row "C" colcount;
        put "</Range>" NL;
        do / if any($options['FBANNER_COLOR_EVEN'],$options['BANNER_COLOR_EVEN']);
          put "<Condition>" NL;
          put "<Value1> MOD(ROW(),2)=0</Value1>" NL;
          put "<Format Style=""";
          put  "color:" $options['FBANNER_COLOR_EVEN'];
          put ";" background-color" $options['BANNER_COLOR_EVEN'];
          put """" "/>" NL;
          put "</Condition>" NL;
        done;

        ...more SAS statements...
    end;
run;
```

```
ods tagsets.excelxp_mod file="temp.xml"
options(fbanner_color_even="red"
        banner_color_even="#faf3d4"
        col_color_even="#faf3d4") style=normal;
proc print data=sashelp.class noobs;
run;

ods tagsets.excelxp_mod close;
```

In This Example:

- The PUT statement writes the **<ConditionalFormatting>** tags and the options parameters to the XML files.
- The MOD function is added to the **<Value1>** element as the first condition. When this condition is satisfied, the foreground or background colors are applied via the **<Format>** element. In this example, only a single condition is expressed. However, the complete, downloadable tagset for this example uses other options to alternate foreground and background colors.
- The $WORKSHEET_ROW and the COLCOUNT variables shown in this example are created in the parent tagset. Within the table, these variables build the range of rows and columns to which the formula should be applied.

This code generates the following report.

Display 2. Output Showing Alternating Row Colors and Alternating Column Colors

SUPPLYING A DYNAMIC BAND VALUE

Now that you know how to use the MOD function to apply styles, you can apply band values of your choice (for example, for specific rows or columns). You can either hardcode the values, as shown in the previous example, or you can use conditional testing to specify a band value of your choice. In addition to using the same options as those used in the preceding example, this next example uses the BAND_VALUE= option, which enables you to choose specific band values.

```
ods tagsets.excelxp_mod file="c:\temp\temp.xml" style=sasweb
options(banner_color="lightblue" band_value="=MOD(ROW(),4)<2" );

proc print data=sashelp.class;
run;

ods tagsets.excelxp_mod close;
```

This code generates the following report with banding of specific rows:

	A	B	C	D	E	F	G	H
1	Obs	Name	Sex	Age	Height	Weight		
2	1	Alfred	M	14	69	112.5		
3	2	Alice	F	13	56.5	84		
4	3	Barbara	F	13	65.3	98		
5	4	Carol	F	14	62.8	102.5		
6	5	Henry	M	14	63.5	102.5		
7	6	James	M	12	57.3	83		
8	7	Jane	F	12	59.8	84.5		
9	8	Janet	F	15	62.5	112.5		
10	9	Jeffrey	M	13	62.5	84		
11	10	John	M	12	59	99.5		
12	11	Joyce	F	11	51.3	50.5		
13	12	Judy	F	14	64.3	90		
14	13	Louise	F	12	56.3	77		
15	14	Mary	F	15	66.5	112		
16	15	Philip	M	16	72	150		
17	16	Robert	M	12	64.8	128		
18	17	Ronald	M	15	67	133		
19	18	Thomas	M	11	57.5	85		
20	19	William	M	15	66.5	112		
21								

Display 3. Banding Specific Rows Using the BAND_VALUE= Option

POSITIONING OUTPUT WITHIN THE WORKSHEET

"No man is truly married until he understands every word his wife is NOT saying." – unknown author
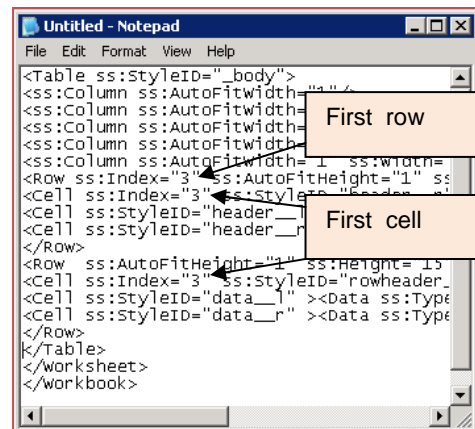
In a way, that is also true of the relationship between SAS and Microsoft Excel. You have to understand how to arrange output in Excel with. ODS and Excel allow some flexibility as far as the layout of tables or graphs in a worksheet. For more control, you can modify the ExcelXP and MSOffice2K tagsets, or you can use the sample TableEditor and MSOffice2K_X tagsets. For example, you can modify either the ExcelXP tagset or the MSOffice2K tagset such that you can place output in a specific row or column by adding an option to the tagset. You can implement the same behavior in the MSOffice2K_X and the TableEditor tagsets by using the WORKSHEET_LOCATION= option. In addition, you can add this option to the ExcelXP destination. The WORKSHEET_LOCATION= option takes either a single parameter or two values separated by a comma. The single parameter defaults to the starting row position, while the two values indicate the starting locations for a row and a column.

The WORKSHEET_LOCATON= option is implemented differently in each of the three tagsets. The TableEditor tagset places output in a specific row or column by modifying the Excel objects. The MSOffice2K_X tagset modifies the HTML, adding both the column and row location. As shown in Output 1, the ExcelXP destination modifies the XML by modifying the **ss:Index=** attribute in the first **<Row>** element of the table in order to begin the output in a specific row. To modify the column position, you add the **ss:Index=** attribute to the first **<Cell>** tag in each row of the XML. In the complete, downloadable tagset, you also have the ability to position titles and footnotes separately.

The following code example uses ODS Markup language to add the row and column positions to the tables.

Output 1. Modified XML Markup

Example

```
proc template;
  define tagsets tagsets.excelxp_mod;
    parent=tagsets.excelxp;
    define event row_start;
    put "<Row ";
    putq "ss:Index="scan($options['WORKSHEET_LOCATION'],1) / if $worksheet_row=1;
```

```
      ...more SAS statements...
end;
define event cell_start;
  start:
       ...more SAS statements...
  finish:
    break /if ^$$cell_start;
    open row;
    put "<Cell";
    putq " ss:Index=" scan($options['WORKSHEET_LOCATION'],2) / if cmp(colstart,"1");
       ...more SAS statements...
end;

data one;
   infile cards missover;
   input Business $26. var1 var2 var3 $;
   cards;
Current Assets
Cash in banks and in hand    39000  48000
Accounts receivable          60000  70000
Inventories                  75000  60000
prepaid operating expenses   18000  13000
Total                       192000 191000
Current Liabilities          10000 150000
Income tax payable           60000 800000
Dividends payable            80000 800000
Total                       150000 175000
Working capital              42000  16000
;
run;

proc template;
   define style styles.xlsstyle;
   parent=styles.journal;
   style table from table / borderwidth=0 cellspacing=0;
   end;
run;

ods tagsets.excelxp_mod file="c:\temp.xml" options(worksheet_location="5,2")
     style=styles.xlsstyle;

proc report data=one nowd  style(column)={tagattr="format:$#,###"};
cols business ("December 31" var1 var2);
define business / " " style(column)={tagattr="format:@*." just=l cellwidth=5in };
   define var1 / "1983";
   define var2 / "1984";
   compute business;
      if business in("Current Assets","Current Liabilities","Working capital") then
          call define(_col_,"style","style={ just=c font_weight=bold}");
           else if business in("Total") then
           call define(_col_,"style","style={pretext='    
                                                 '}");
           else call define(_col_,"style","style={ pretext='   '}");
   endcomp;
run;

ods tagsets.excelxp_mod close;
```
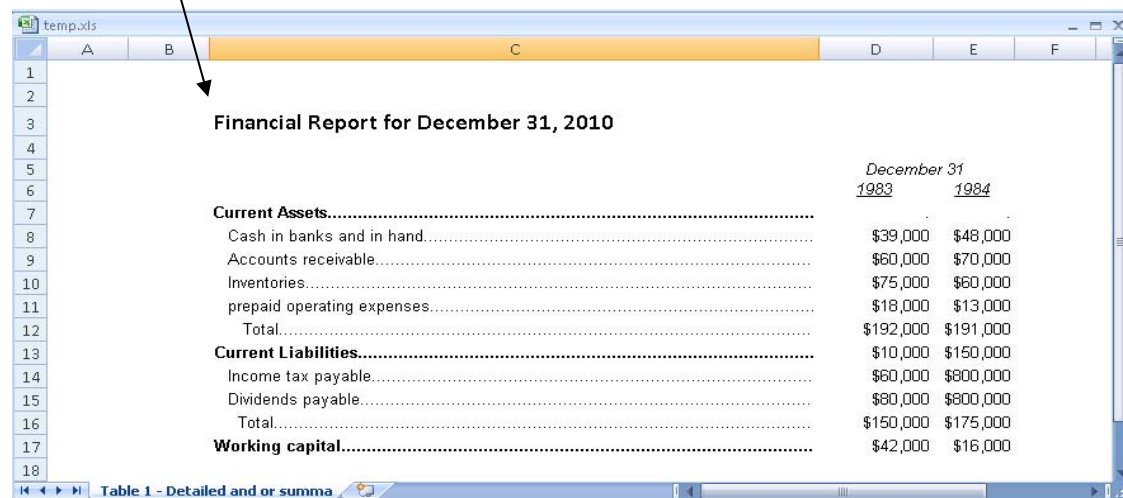
8

In This Example:

- The tagset starts by creating a new tagset that is derived from the ExcelXP tagset. The ROW_START event is modified by the **ss:Index=** attribute added to the **<Row>** element when the first parameter is passed in the WORKSHEET_LOCATON= option. This applies to the first **<Row>** element, which is conditionally added when the value of the memory variable $WORKSHEET_ROW is **1**.
- The column location is added by modifying the CELL_START event. The second parameter that is passed in the WORKSHEET_LOCATION= option is added to the **ss:Index=** attribute in the first **<Cell>** element of each observation. This attribute is conditionally written based on the value of the COLSTART variable, which iterates through the various fields in the data set.
- The VALUE_PUT event (not displayed in this partial code sample) adds the PRETEXT style variable, which enables you to add non-breaking space characters via the PRETEXT= attribute. This event indents values or other attributes in the output.

This code generates the following report.

Display 4. Positioning Rows and Columns in a Financial Report

PANELING TABLES OR GRAPHICS

Just as you can place tables in the row and column of your choice, you can also place multiple graphic images or multiple tables side by side (paneled output). To do this, you can use the MSOffice2K_X tagset, which is simply the MSOffice2K destination with modifications. The MSOffice2K_X tagset handles the graphics in slightly different ways than the MSOffice2K destination. Some of the differences address limitations in the way Excel handles HMTL. For example, if you have multiple tables or table tags within a worksheet, the application ignores the cell widths.

Attempting to add multiple graphics horizontally usually results in images that overlap in Excel. The tagset alleviates that problem by adding each image to individual table cells that are of a default width and height. The tagset uses the default width and height of the image for the width and height of the cell. You can modify these default values by using the GRAPH_HEIGHT= and the GRAPH_WIDTH= tagset options. The tagset and its options enable you to take advantage of many SAS graphics procedures that generate image files. You can also use the GKPI procedure to create dashboards or indicators that provide a quick visual status of your business activities.

In addition, you can use the tagset with the PANELCOLS= option to panel tables much the same way as you can panel graphics. You can use the PANEL_SPACE= option to add a blank cell between each table or graphic.

This technique works just as well with tables. When you combine tables and images, Microsoft Excel encounters a table tag and it resets the width of the cells for the images. In the following example each graphic is added in a table cell using the GRAPH_HEIGHT= and the GRAPH_WIDTH= options to specify the height and the width of the graphics.

```
goptions reset=all device=gif vsize=4in hsize=4in;
ods tagsets.msoffice2k_x path="c:\temp" file="temp.xls"
                                      options(graph_width="300" graph_height="300"
                                              panelcols="4" panel_space="0")
                                      style=sasweb;
proc gkpi mode=raised;
    dial actual=.70 bounds=(0 .25 .46 .65 .80 1) / target=.9  nolowbound
                    format="percent." label="North Sales";
run;
quit;

proc gkpi mode=raised;
    trafficlight actual=500 bounds=(1000 700 300 0) /
    colors=(green yellow red)  noavalue
    label="Actual Sales";
run;

quit;
...more SAS statements...

ods tagsets.msoffice2k_x close;
```
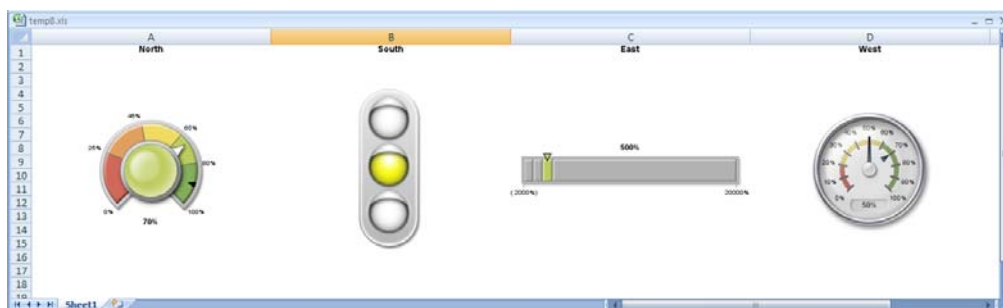
Display 5. Creating a Dashboard with Paneled Graphics

INSERTING PAGE BREAKS

Adding a page break in worksheet rows and columns indicates to Excel where it should physically generate a new page when the worksheet is printed. You use ODS Markup language to implement these breaks. In the ExcelXP tagset, you create page breaks by adding XML **<PageBreaks**> element after the closing **</WorkSheet>** tag. The WORKSHEET event triggers or calls the PAGEBREAK_ADD event that you create. You can employ this reusable tagging to override both the ExcelXP and the MSOffice2K destinations.

This ability to add physical page breaks is included in the downloadable sample MSOffice2K_X tagset, while the following example illustrates how to add breaks to the ExcelXP tagset.

The XML tags displayed in the image below instructs both the HTML and XML output to break at rows 10 and 20 in the output. The next example creates an event named PAGEBREAK_ADD that can be triggered from the Worksheet event which places the required XML after the closing **</WorkSheet>** tag. The code instructs the HTML and XML output to break at rows 3, 7, 11, and 15 in the first worksheet and rows 4 and 9 for the second.

Example

```
proc template;
  define tagsets.excelxp_mod;
    parent=tagsets.excelxp;
    define event worksheet;
      Start:
     ...more SAS statements...
     finish:
    ...more SAS statements...
      unset $byGroupLabel;
      trigger pagebreak_add;

  end;
```

```
define event pagebreak_add;
 do /if $options["PAGEBREAK_ROW"];
   set $pagebreak_row $options["PAGEBREAK_ROW"];

  do /if index($pagebreak_row,",");
     set $pbreak_value scan($pagebreak_row,1,",");
     eval $page_cnt 1;
     do /while ^cmp($pbreak_value," ");
        set $pbreak_row[] strip($pbreak_value);
        eval $page_cnt $page_cnt +1;
        set $pbreak_value scan($pagebreak_row,$page_cnt,",");
     done;
     else;
        set $pbreak_row[] strip($pagebreak_row);
     done;
  iterate $pbreak_row;
  put " <x:PageBreaks>" NL;
  do /while _value_;
     put "   <x:RowBreaks>" NL;
     put "   <x:RowBreak>" NL;
     put "   <x:Row>"_value_</x:Row>" NL;
     put "   </x:RowBreak>" NL;
     put "   </x:RowBreaks>" NL;
     next $pbreak_row;
  done;
  put " </x:PageBreaks>" NL;
done;
    end;
run;
```



```
ods tagsets.excelxp_mod file="c:\temp.xml" options(pagebreak_row="3,7,11,15");

proc print data=sashelp.class;
run;

ods tagsets.excelxp_mod options(pagebreak_row="4,9");

proc print data=sashelp.class;
run;

ods tagsets.excelxp_mod close;
```
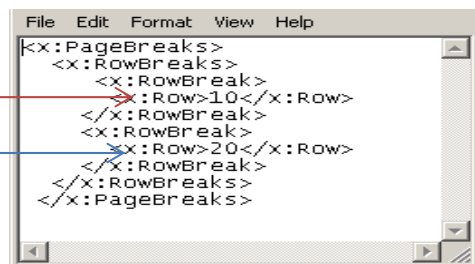
In This Example:

- This example adds the PAGEBREAK_ROW= option. It also checks to see whether a single value or multiple values separated by commas are specified for the PAGEBREAK_ROW= option. If multiple values are specified, then the values are added to the PBREAK_ROW dictionary variable. This process repeats until there are no more values to parse.
- By creating a tagset option, you can pass the values for the row breaks you want to the tagset. You can opt to manually specify a row for the page break, or you can specify a range of values.
- The ITERATE statement cycles through the PBREAK_ROW dictionary variable and supplies the value to the `<Row>` element with the special variable named _VALUE_. You can extend this process to the full tagset example, which repeats the page break for every character that you specify if the variable contains a single value. For example, if you specify the value **4**, the page break repeats after every 4 characters within the current table. This continues until there are no more rows.

This code generates a report that contains inserted page breaks.

Display 6. Using the PAGEBREAK_ROW= Option to Insert Page Breaks

ODS OUTPUT AND FILE SIZE

According to a study in the *Journal of Economics and Human Biology* which compared 12,000 men and women aged 18 to mid-40s, statistics show that both men and women tend to gain weight after they get married. Sometimes, this is referred to this as the *honeymoon 15*.

In a way, the same is true for the relationship between ODS and Excel. When you generate output with the ExcelXP and MSOffice2K destinations, file size can become an issue. Because the files that are generated with ODS are neither true binary Excel files nor compressed Open XML (.XLSX) files, a comparison of the file sizes of ODS and Excel is not an equal comparison. As mentioned previously, ExcelXP generates XML, and MSOffice2K generates HTML. The more observations that you have, the more evident this issue can be because the files that are generated by ODS are text files that can be read by Excel. Not only do large files present a storage problem, they also take more time to open and load. The size problem is exacerbated in Excel 2007 because the row limit increased from 65,535 rows to over 1 million, and column limit increased from 255 columns to 16,384. Excel 2007 has new file formats (extensions); for example, the .XLSX format. **Note:** Currently, this format is not available with ODS. However, the addition of this file format is planned for a future release. The following table lists the new formats that are available in Excel 2007.

Extension	Type of File	Excel Version	Numeric Value
.XLSX	Open compressed XML	2007	51
.XLSB	Excel 2007 binary file	2007	50
.XLSM	Excel 2007 file (includes Excel macros)	2007	52
.XLS	Excel 2003 binary file	97-2003	-4143

Table 3. Excel File Extensions

TECHNIQUES FOR REDUCING WORKBOOK FILE SIZE

Several techniques are available to help you address problems with workbook file sizes. These techniques include the use of the following tools:

- the ODS CSV destination
- dynamic data exchange (DDE)
- the TableEditor tagset
- Visual Basic Scripting (VBSCRIPT)

The CSV Destination

Using the CSV destination is the most obvious solution because this destination contains only data and it generates a very small file. ODS can generate a CSV file from the DATA step and from almost all of the SAS procedures. Unlike

the MSOffice2K and ExcelXP destinations, the CSV destination does not offer presentation qualities for your data. If you create output that does not need such qualities, then the CSV destination is a good choice. With this destination, you do not have to worry about the variations in hardware or loading times.
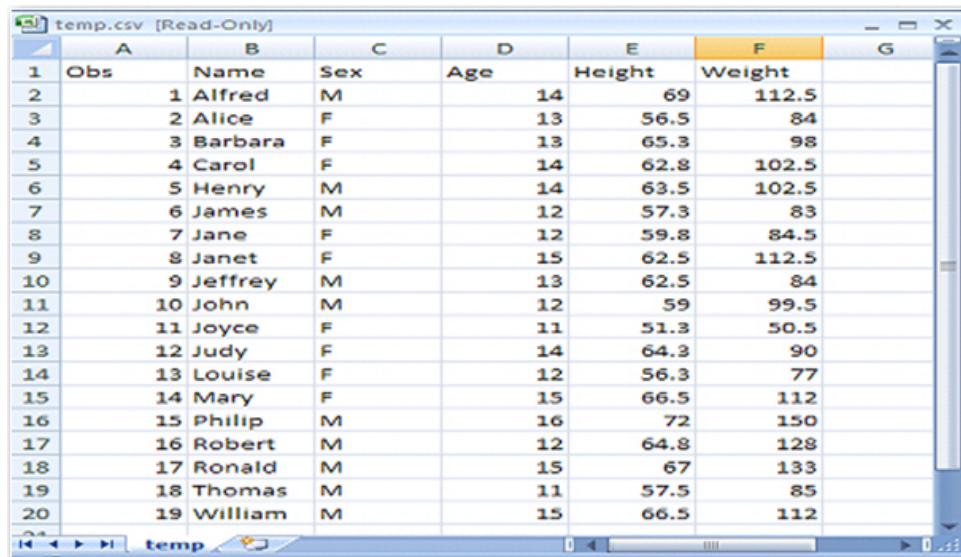
The following code is a simple example that uses the CSV destination.

```
ods csv file="c:\temp.csv";

proc print data=sashelp.class;
run;

ods csv close;
```

This code generates simple output similar to that in the following display.

	A	B	C	D	E	F	G
1	Obs	Name	Sex	Age	Height	Weight	
2	1	Alfred	M	14	69	112.5	
3	2	Alice	F	13	56.5	84	
4	3	Barbara	F	13	65.3	98	
5	4	Carol	F	14	62.8	102.5	
6	5	Henry	M	14	63.5	102.5	
7	6	James	M	12	57.3	83	
8	7	Jane	F	12	59.8	84.5	
9	8	Janet	F	15	62.5	112.5	
10	9	Jeffrey	M	13	62.5	84	
11	10	John	M	12	59	99.5	
12	11	Joyce	F	11	51.3	50.5	
13	12	Judy	F	14	64.3	90	
14	13	Louise	F	12	56.3	77	
15	14	Mary	F	15	66.5	112	
16	15	Philip	M	16	72	150	
17	16	Robert	M	12	64.8	128	
18	17	Ronald	M	15	67	133	
19	18	Thomas	M	11	57.5	85	
20	19	William	M	15	66.5	112	

temp.csv [Read-Only]   temp

Display 7. Simple Output Generated by the CSV Destination

Dynamic Data Exchange (DDE)

In Microsoft Windows operating environments, you can use DDE to save output files as Excel binary files, compressed XML , or the file format of your choice. Using DDE enables you to pass a command to save your files in the file format of your choice (Table 3). DDE also enables you to modify most parts of the worksheet and the workbook.

The following example uses DDE to save an XML file that is generated with the ExcelXP tagset to an Excel 2007 file format.

Example

```
options noxwait noxsync;
filename cmds dde 'excel|system';

data _null_;
   file cmds;
   X=sleep(10);
   put "[open(""C:\temp.xml"")]";
   put '[ERROR("FALSE")]';
   put "[SAVE.AS(""C:\temp.xlsx"" ,51)]";
   x=sleep(2);
   put '[close("false")]';
   run;
```

In This Example:

- The FILENAME statement, along with the DDE triplet, allocates the Excel application.
- The DDE OPEN command opens the XML file that is created with the ExcelXP tagset.
- The DDE SAVE.AS command saves the XML file in the compressed XML format. This format is indicated by the value 51 in the SAVE.AS parameter.

By using DDE to save your workbook in a different file format (in this case, the .XLSX format), you are able to reduce the workbook size.

The TableEditor Tagset

You can use the sample TableEditor tagset in a manner similar to the way DDE works. That is, you can use the TableEditor tagset to open a file that you create with another tagset and then save that file to another file format. This tagset uses Excel objects and the UPDATE_TARGET=*workbook-name* option to open a specified file.

Example

```
   /* Create a sample files */
ods tagsets.excelxp file="c:\temp\temp.xml";

proc print data=sashelp.class;
run;

ods tagsets.excelxp close;

ods path(prepend) work.templat(update);
filename temp url
        'http://support.sas.com/rnd/base/ods/odsmarkup/tableeditor/tableeditor.tpl';
%include temp;

ods tagsets.tableeditor file="%sysfunc(getoption(work))\temp.html"
options(update_target="c:\\temp\\temp.xml"
        excel_save_file="c:\\temp\\test.xlsx"
        file_format="xlsx"
        open_excel="no"
        auto_excel="yes");

data _null_;
    file print;
    put "Save as Excel";
run;

ods tagsets.tableeditor close;
```

In This Example:

- This example uses the FILE_FORMAT= option to specify the extension (in this case, .XLSX) of the format in which you want your output formatted.
- The OPEN_EXCEL= option lets you specify whether or not you want your file to be opened in Excel.
- You can use the AUTO_EXCEL= option to update this information without clicking the button on the worksheet. Note that the file paths that are added in the options need double slashes when the paths already contain a single slash.

By changing the file format with the tagset, your workbook size is reduced.

Visual Basic Scripting

You can save files in the format of your choice by using simple scripting such as VBSCRIPT, which you can call from your program. If your program sends the XML files that you want to be converted to a particular directory, you can point the VBSCRIPT to this location. VBSCRIPT uses Excel objects to save this file in the format that you want, thereby reducing the size of the file.

The following example creates and executes a VBSCRIPT that opens all of the files in a directory that is specified with the DEFAULT= macro parameter. The script saves the files in compressed XML with the extension .XLSX in

Excel 2007 or the extension .XLS for Excel 97-2003. The script saves the files in compressed Open XML format with the extension of .XLSX in Excel 2007 (or in XLS format for Excel 97-2003).

This simple script makes it easy to open and save the workbook files in other formats, thereby reducing the workbook size.

Example

```
    /*******************************************************************/
    /* Sample script that opens all worksheets in a particular directory  */
    /* and saves them in a particular format.                          */
    /*******************************************************************/

  /* Create Sample files */
  ods tagsets.excelxp file="c:\temp\temp2_files\temp.xml" newfile=bygroup;

  proc print data=sashelp.class;
  by age;
  run;

  ods tagsets.excelxp close;

options noxsync noxwait;
ods tagsets.excelxp file="c:\temp\convert\temp.xml";

proc print data=sashelp.class;
run;

ods tagsets.excelxp close;

options noxsync noxwait;
%macro test(default=c:\temp\convert,store=c:\temp\temp.vbs,ext=xml);
    data _null_;
      file "&store";
      put "   Dim oXL ";
      put "   Dim oFolder";
      put "   Dim aFile";
      put "   Dim FSO";
      put "   Set oXL = CreateObject(""Excel.Application"")";
      put "   Set FSO = CreateObject(""Scripting.FileSystemObject"")";
      put "   oXL.DefaultFilePath = ""&default""";
      put "   oXL.DisplayAlerts = False";
      put "   if FSO.FolderExists(oXL.DefaultFilePath) then";
      put "       Set oFolder = FSO.GetFolder(oXL.DefaultFilePath)";
      put "       For each aFile in oFolder.Files ";
      put "           If Right(LCase(aFile.Name), 4) = "".&ext"" Then";
      put "             oXL.Workbooks.Open(aFile.Name)";
      put "               oXL.Visible = False";
      put "               if (oXL.Version) >= 12 Then" ;
      put "                   oXL.ActiveWorkBook.SaveAs Left(aFile, Len(aFile) - 4)
            & "".xlsx"",51";
      put "               Else";
      put "                   oXL.ActiveWorkBook.SaveAs Left(aFile, Len(aFile) - 4)
            & "".xls"",-4143";
      put "               End If";
      put "               oXL.ActiveWorkBook.Close SaveChanges = True";
      put "          End If";
      put "       Next";
      put "       Set oFolder = Nothing";
      put "   end if";
  /*  put "   FSO.DeleteFile(""&default\*.&ext""), DeleteReadOnly";*/
      put "   oXL.DisplayAlerts = True";
      put "   oXL.Quit";
      put "   Set oXL = Nothing";
```

```
 call system("&store");
run;
%mend;

%test()
```

In This Example:

- The EXT=XML parameter filters and opens all of the XML files in the directory. Then it saves them in the default format for the version of Excel that you have installed. Then, the XML files are deleted.
- The STORE= option points to the location in which you want to store the VBSCRIPT.
- The CALL SYSTEM routine calls and executes the script.

Display 7 lists the files that are saved. This display shows the list before the XML files are deleted.
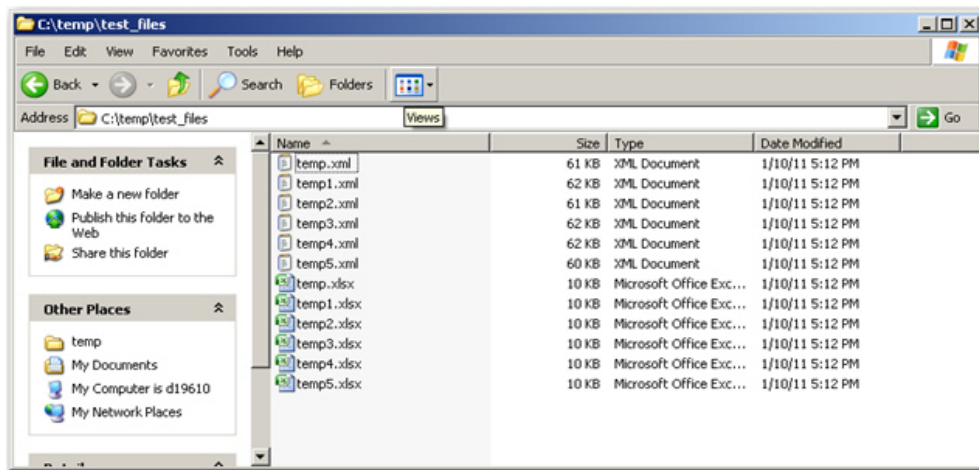
Display 8. List of the Files That Are Saved by the VBSCRIPT

CELL-FORMATTING ISSUES

In any successful marriage, communication is very important. Uncommunicated expectations cause friction and lead to complaints. For example, consider the classic toilet-seat example. One partner always leaves the toilet seat up, much to the chagrin of the other partner. Instead of just complaining, the unhappy partner should be specific and communicate his or her feelings and expectations.

This anecdotal situation is similar to the way things work between ODS and Microsoft Office. For example, consider a situation in which you export data to Excel using an ODS destination. The output might not be displayed in Excel the way you expect unless you communicate to Excel how the output should appear. Without this communication, the expected output might not occur, for the following reasons:

- Leading and trailing zeroes are omitted in numbers.

- Numbers larger than 13 digits are displayed in scientific notation.

- Numeric values longer than 15 digits lose precision.

- Numbers connected by dashes might be converted to a date.

- Specifying the COMMA format in SAS might not carry through in Excel.

If an Excel format is not applied to a cell when the worksheet is opened, there is an unrealistic expectation that Excel will do what you consider to be correct when it displays the cell value. In the absence of a specified Excel format, Excel uses the **General** format for cell values. This format is similar to the ANY format in SAS.

Which method that you use to apply Excel formatting in SAS varies, depending on the destination that you use. For example, the ExcelXP destination uses the attribute TAGATTR="format:*excel-format*". The MSOffice2K and the HTML destinations use the attribute HTMLSTYLE="mso-number-format:*excel-value*". Both of these destinations enable you to pass Excel formatting.

CUSTOM FORMAT OPERATORS AND GUIDELINES

Every custom format is made up of four sections: formats for positive, negative, zero, and text values. If only one section is specified, then Excel assumes that it is for the positive value. You do have the ability to control these values individually. Consider the following illustration:

This illustration shows the four pieces that make up a format:

- a format with the thousands separator for the positive value.
- a negative value that is to be displayed in red, with the thousands separator and two decimal places. The 0 fill value is formatted with two decimal places in red and within parentheses.
- a zero format with two decimal places.
- a text format.

If only section one is specified, then the application uses that section for all cases. The following table displays an abbreviated list of some of the most commonly used custom-format operators.

Character	Description
0	Pads the value with zeros.
#	Does not display extra zeros.
?	Leaves a space for insignificant zeros.
. (period)	Displays a decimal number.
%	Multiplies by 100 and displays the value as a percentage.
, (comma)	Uses a thousands separator.
Text Code	**Description**
\character	Displays the character that you specify.
"text"	Displays the value as text.
*	Repeats a character to fill the format.
_ (underscore)	Skips the width of the next character.
@	Is a text placeholder.
Date Code	**Description**
M/D/YYYY	Formats a date as Month/Day/Year.
D-MMM-YY	Formats a date as Day/Month/Year.
AM/PM a.m./p.m.	Formats time based on the 12-hour clock.
Miscellaneous	**Description**
[BLACK], [BLUE],…[COLOR n]	Display the characters in the specified colors. n is a value from 1 to 56.

Table 4. Common Custom Formats

APPLYING EXCEL NUMBER FORMATS

As mentioned earlier, the application of the Excel formats differs depending on the destination. For the ExcelXP tagset, you use the FORMAT parameter in the TAGATTR="*string*" attribute. For the MSOffice2K destination, you use the Microsoft cascading style sheet property **mso-number-format** in the HTMLSTYLE="*string*" attribute. The CSV destination does not allow styles, therefore permits only limited formatting. Consider the following table, which has not been formatted:

Obs	leading	trailing	scientific	condition	repeat	thousands	label
1		1	12345.1	1.23457E+17	100	200	1000	1.2
2		2	23455.2	3.33434E+17	200	300	2000	1.46
3		3	12344.1	1.23457E+17	300	500	3000	1.2

Output 2. Table before Formatting

The following example uses the ExcelXP and MSOffice2K destinations to apply number formats to cells within this table. In addition to illustrating general formatting, this example also demonstrates the use of formatting to handle common problems, such as retaining leading and trailing zeros, preventing numbers from displaying in scientific notation, and retaining comma delimiters. As shown in the following example, you can also modify the data types by using the attribute TAGATTR="*type*:*String*", which is the same as adding the text format.

Example

```
data one;
    input leading trailing scientific $18. condition repeat thousands label;
    cards;
0001 12345.10 123456789101112145 100 200 1000 1.2
0002 23455.20 333434455555544334 200 300 2000 1.46
0003 12344.10 1234567789101112145 300 500 3000 1.2
;
run;

ods tagsets.excelxp file="temp.xml" style=styles.normal;

proc print data=one;
    format leading z4.;
    var leading    / style(data)={tagattr="format:@"};
    var trailing   / style(data)={tagattr="format:#####.#0"};
    var scientific / style(data)={tagattr="type:String"};
    var condition  / style(data)={tagattr="format:[green][>=300];
                                            [yellow][<200]#,###;
                                            [red]#,###"};
    var repeat     / style(data)={tagattr="format:$**#,###"};
    var thousands  / style(data)={tagattr="format:#,###"};
    var label      / style(data)={ tagattr='format:#.##\m'};
run;

ods tagsets.excelxp close;

    /* MSOffice2K */
ods msoffice2k file="temp.xls" style=styles.normal;

proc print data=one;
    format leading z4.;
    var leading /   style(data)={htmlstyle="mso-number-format:\@"};
    var trailing /  style(data)={htmlstyle="mso-number-format:\#####.\#0"};
    var scientific / style(data)={htmlstyle="mso-number-format:\@"};
    var condition / style(data)={htmlstyle="mso-number-format:[green][>=300];
    [yellow][<200]#,###;[red]#,###"};
    var repeat /     style(data)={htmlstyle="mso-number-format:\$**\#\,###"};
    var thousands /  style(data)={htmlstyle="mso-number-format:\#\,###"};
    var  label /     style(data)={htmlstyle="mso-number-format:\#\.##\\m"};
run;

ods msoffice2k close;
```

After the formatting is applied, the table appears as follows:

Obs	leading	trailing	scientific	condition	repeat	thousands	label
1	0001	12345.10	123456789101112145	100	$***********200	1,000	1.2m
2	0002	23455.20	333434455555544334	200	$**********300	2,000	1.46m
3	0003	12344.10	123456789101112145	300	$***********500	3,000	1.2m

Output 3. Table after Formatting

The following table contains an abbreviated list of common Excel formats and custom formats that you can use with the ExcelXP and the MSOffice2K destinations.

Format (ExcelXP)	Format (MSOffice2K)	Original Value	Value After Format
General	General	01234	1234
General Number	Fixed	01234	1234.00
Euro Currency	\£#\,\#\#0\	1234	£1,234
Currency	Currency	1234	$1,234.00
Percent	Percent	12	1200.00%
Scientific	Scientific	12345	123E+04
@	\@	01234	01234
####	\####	1234	1234
0 0000	0\.0000	.01234	0.1234
#,##0	\#\.,\##0	1234	1,234
#,##0.00	\#\,##0\.00	1234	1234.00
#,##0_);[Red](#,##0)	\#\,##0_);[Red](\#\,\##0	-12345	(1,2345)
$*#,##0_	\$**\#\,###	12345	$*******1,2345
0%	0\%	10	100%
0.00%	0\.00\%	10	100.00%

Table 5: Excel Formats

EXPORTING OUTPUT TO MSACCESS

Microsoft Access is a database management system that enables you to manage data (for example, adding, deleting, sorting, and querying data). Access also enables you to create forms and sophisticated reports from the data in your database. You can create reports interactively either by using the Access Report Wizard or by creating the output in the application's Design view. The Access application enables you to import and export data, and the application is fully integrated with XML. XML provides the ability to describe your data as well as to define the structure of your data by specifying an XML schema.

At times, you might want to store your data in the Access application. You can export data directly to Access, from which your users can take advantage of the full features available in the application. Those features include the following:

- report generation
- form design and implementation

- the ability to generate PivotCharts and PivotTables

- the ability to import and export XML

- the ability to interact with many other applications

You can take advantage of these features by creating XML within SAS and importing the data into Access. Or, you can go the other way and generate XML within Access and then read it into SAS.

The following example leverages the power of SAS to create an XML file from a SAS data set before importing the data into Access. Rather than using ODS, this example uses a close cousin, the XML LIBNAME engine, in a LIBNAME statement.

Example

```
libname temp xml "c:\temp\temp.xml" xmltype=msaccess  xmlmeta=schemadata;

data temp,class;
    set sashelp.class;
run;
```

In This Example:

- With the XML LIBNAME engine, you use the XMTYPE=MSACCESS option in a LIBNAME statement. This coding generates the XML that the Access application expects.

- This example also generates an embedded schema using the XMLMETA=SCHEMADATA option.

- You import the file via the Microsoft Access application, as follows:

   1. Open an existing database or create a new database.
   2. Under **Tables Tools**, right-click the database and select **Import ►XML File**, as shown in the following display:
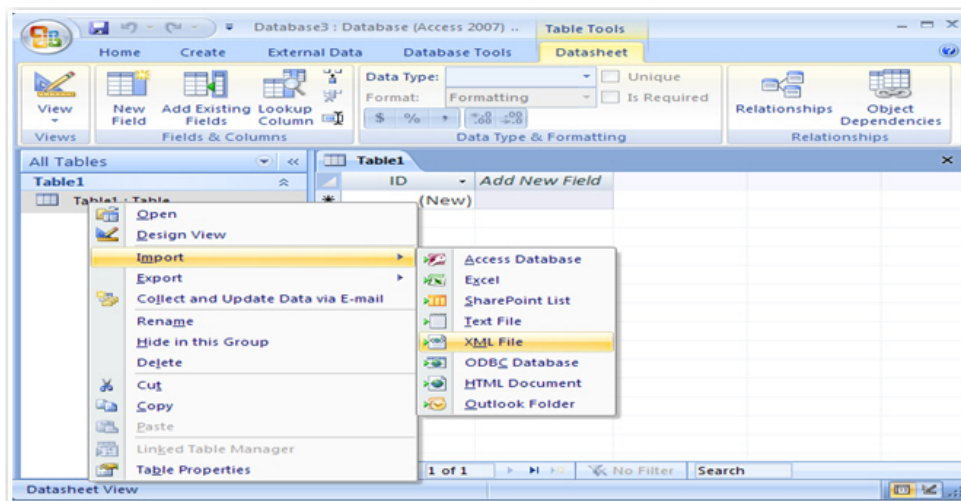
Display 9. Importing an XML File into Microsoft Access

After you import the XML file, you can use either the Access Report Wizard or the interactive Design view to generate very sophisticated reports.

EXPORTING OUTPUT TO MICROSOFT WORD

The ODS RTF destination was added to the ODS family in SAS 8.1. This destination creates output that can be rendered by word-processing packages such as Microsoft Word. As is the case with other destinations, the RTF destination enables you to modify formatting features such as fonts, colors, links, and so on. Like the MSOffice2K and HTML destinations, the RTF destination enables you to use styles that are generated with PROC TEMPLATE,

cascading style-sheet styles (using the CSSSTYLE= option), and styles that are created directly in certain procedures using the STYLE= option. In addition, this destination also uses specific RTF control words that enable you to modify the RTF output. This functionality lets you make all types of modifications, so long as you are familiar with the native RTF language that is responsible for those modifications.

The RTF destination enables you to generate output that can be viewed as well as edited. This output does not define the vertical measurement of the page. Rather, the output is passed to Microsoft Word, which is responsible for determining the end of the page. However, ODS does control the horizontal measurement of the output. When you generate files with ODS RTF, the file size is different than it would be if you had used Word. This difference

occurs because the RTF files are not the native .DOCX files. Generating output via the RTF destination is simple, as shown in the following example:

Example

```
ods rtf file="c:\temp.rtf";

proc print data=sashelp.class;
run;

ods rtf close;
```

THE ODS MEASURED RTF TAGSET

The ODS RTF tagset, which is new in SAS 9.2, measures output both vertically and horizontally. Therefore, you do not have problems with positioning titles, footnotes, and column headers vertically on a page because SAS handles the vertical measurement. The RTF tagset is fully customizable because all of the tags that are required for RTF output are stored in the RTF tagset. This ability to customize gives you tremendous flexibility. Not only can you insert the native RTF (as mentioned earlier with the traditional RTF), but you also have full control of how the output is generated.

The RTF tagset also has built-in, predefined options that provide you with access to some of the most requested functionality. For example, the tagset contains options that enable you to rotate column headers, pass information to the section information, and more. If the RTF tagset does not have functionality that you want, you can simply add it via a tagset modification. To see all of the options that are defined in the tagset, submit the tagset option DOC="HELP'.

The following example rotates the column headers and instructs the application not to create a section after each piece of output:

Example

```
ods tagsets.rtf file="c:\temp.rtf"
                options(doc="help"
                sect="none"
                trhdr="\trrh800"
                trowhdrcell="\cltxbtlr");

proc print data=sashelp.class style(obsheader header)={just=l};
run;

ods tagsets.rtf close;
```

This code generates a report that has the column headers rotated 90 degrees and that is not divided into sections. Output 4 shows only a part of the full report:

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0

Output 4. Report (Partial) That Is Created When You Use Specific Options with the RTF Tagset

CONCLUSION

A perfect marriage comprises a number of characteristics and behaviors, including complementary abilities, diversity, compromise, and communication. As this paper has shown, these same characteristics and behaviors help to create a perfect match between SAS ODS and Microsoft Office. The paper offers examples to illustrate the features and tools inherent in these two applications, and it uses the examples to equip you with the skills you need in order to develop compelling spreadsheets and applications that are both useful and aesthetically pleasing.

REFERENCES

McGraw, Phillip C., Ph.D. Relationship Rescue: A Seven-Step Strategy for Reconnecting with Your Partner. New York, NY Hyperion.

SAS Institute Inc. 2011. SAS Note 36900, "Instructions for viewing all of the style templates that are shipped with the SAS® System." Cary, NC: SAS Institute Inc. Available at support.sas.com/kb/36/900.html.

SAS Institute Inc. 2011. SAS Note 38282, "Instructions for adding Microsoft Excel grid lines". Available at support.sas.com/kb/38/282.html.

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

Chevell Parker
SAS Institute Inc.
SAS Campus Drive:
Cary, NC 27513
E-mail: support@sas.com
Web: support.sas.com