Paper SAS177-2014

Secrets from a SAS Technical Support Guy: Combining the Power of the SAS® Output Delivery System with Microsoft Excel Worksheets

Chevell Parker, SAS Institute Inc.

ABSTRACT

Business analysts commonly use Microsoft Excel with the SAS® System to answer difficult business questions. While you can use these applications independently of each other to obtain the information you need, you can also combine the power of those applications, using the SAS Output Delivery System (ODS) tagsets, to completely automate the process. This combination delivers a more efficient process that enables you to create fully functional and highly customized Excel worksheets within SAS.

This paper starts by discussing common questions and problems that SAS Technical Support receives from users when they try to generate Excel worksheets. The discussion continues with methods for automating Excel worksheets using ODS tagsets and customizing your worksheets using the CSS style engine and extended tagsets. In addition, the paper discusses tips and techniques for migrating from the current MSOffice2K and ExcelXP tagsets to the new Excel destination, which generates output in the native Excel 2010 format.

INTRODUCTION

Let's face it, folks. Everyone loves knowing a secret! And while some secrets definitely should not be shared with anyone, other secrets should be shared with everyone. So the SAS Technical Support Guy is going to let you in on some secrets to help you more effectively debug, style, and enhance your Excel spreadsheets.

First, you will discover the answers to some common issues that users report when they try to generate Excel worksheets. Then the discussion offers tips in the following areas:

- automating SAS output in Excel worksheets by using ODS tagsets
- customizing and extending your worksheets using the CSS style engine and tagsets
- using the new (experimental) Excel destination

Note: Examples that are generated in this paper use the HTMLBLUE style.

COMMON ISSUES WITH GENERATING MICROSOFT EXCEL WORKSHEETS

SAS Technical Support receives some common questions and problems that are related to generating Excel worksheets. In the following sections, the SAS Technical Support Guy lets you in on the secrets for solving these issues. Knowing these secrets will save you time when you create your own worksheets. These issues are related mainly to the ExcelXP tagset, but some issues are related to the MSOffice2K destination.

REDUCING THE SIZE OF EXCEL WORKSHEETS

One of SAS Technical Support's most commonly received questions is about how to reduce the size of the files that are created with the ExcelXP destination. The ExcelXP tagset generates XML files specifically in the XML Spreadsheet (XMLSS) format. The XML file size can become quite large, which causes problems when you want to move or E-mail the file. There are several documented methods for overcoming large file size:

- resaving the file using Dynamic Data Exchange (DDE)
- using VBScript language (See SAS Note 43496, "Convert files created using an ODS destination to native Excel files." Available at **support.sas.com/kb/43/496.html**)
- using the TableEditor tagset
- using the experimental ODS Excel destination (available in the first maintenance release for SAS® 9.4 [TS1M1])

The new Excel destination enables you to generate output in the Microsoft Office Open XML format, which is the default file format beginning with Excel 2007. This format is discussed later in the document.

In SAS 9.2 and later, you can also manage the file size by using the ODS PACKAGE statement to place output in a ZIP file, as shown in the following code sample:

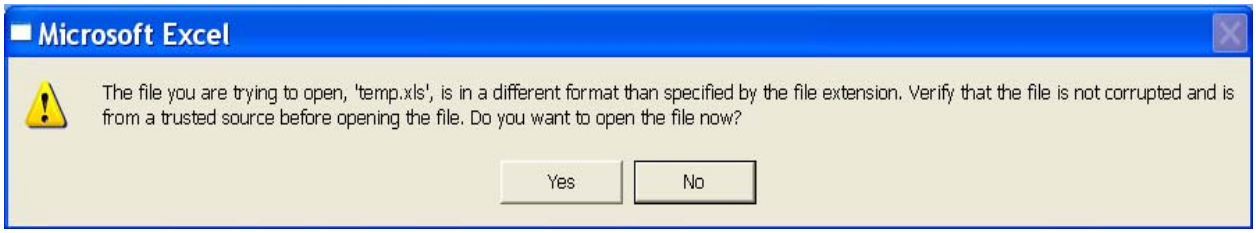
```
ods package open nopf;
ods tagsets.excelxp file="c:\temp.xml" package;

proc print data=sashelp.class;
run;

ods tagsets.excelxp close;
ods package publish archive properties (archive_name="file-name.zip"
                                        archive_path="c:\");
ods package close;
```

PREVENTING AND DIAGNOSING EXCEL DIALOG-BOX ERRORS

Microsoft Excel generates dialog-box errors when it finds a problem with an XML file that is open in Excel. These errors can occur for a variety of reasons. For example, warnings or errors from the Excel parser indicate serious problems with the file. The next sections discuss some of the more common errors you might encounter.

Security Warning

Beginning with Excel 2007, Microsoft introduced a new security feature that matches the content (the file format) of a file with the file's extension. If these two items do not match, Excel generates the message shown here in Display 1:



Display 1. Microsoft Security Message

The XLS extension is commonly used as the extension for output that is generated with the ExcelXP tagset. Although this extension is registered for Excel binary files, that use was never enforced prior to Excel 2007. You can circumvent this message either by modifying the Microsoft Windows Registry (check with your IT department before modifying the registry) or simply by adding the .XML extension to the file that is generated with the ExcelXP tagset. For details about how to modify the Windows Registry, see the Microsoft Knowledge Base note 948615, "Error opening file: 'The file format differs from the format the filename extension specifies." (**support.microsoft.com/kb/948615**)

Errors That Occur during Loading or Opening an XML File

When there is a problem with Excel loading or opening an XML file (such as a file that is generated with the ExcelXP tagset), Excel generates a log file that displays debugging information. This information generally details the reason the problem occurs, where the file resides, the specific tag and value in which the error occurs, and so on. (See Display 3, later in this paper.)

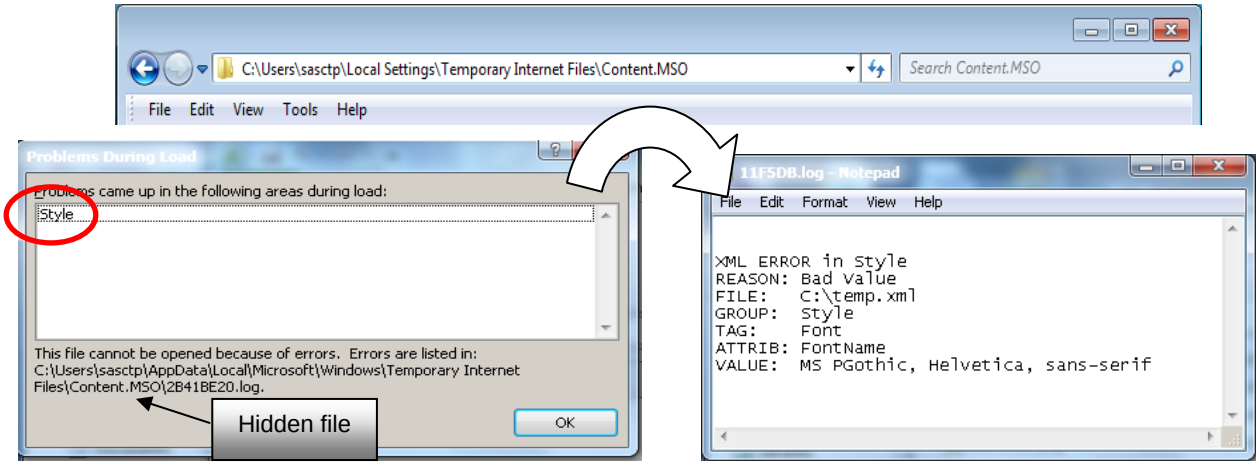
The dialog box shown later in Display 2 specifies the particular element that is involved. The more typical elements you will see listed in the dialog box are **`Style`**, **`Table`**, and **`WorksheetOptions`**. When the **`Style`** element is specified, it can indicate a number of problems, including problems that range from the addition of an invalid font to a worksheet to the addition of an invalid Excel format. When the **`Table`** element is specified, it is typically because a data value makes the file invalid. For example, special characters such as &, <, and > are invalid. When the **`WorksheetOptions`** element is specified, typically it is because an illegal character or a text string longer than 259 characters has been passed in the title or footnote.

When any of the problems occur that are discussed above, Excel generates the error log that you can use for debugging. That log file resides in the path that is specified at the bottom of the dialog box. (See Display 2). That path contains a hidden folder (**`Content.MSO`**) in both the Windows XP and Windows 7 operating environments.

You can find the hidden folder by pasting the following path into the Windows Explorer taskbar:

```
%homepath%\Local Settings\Temporary Internet Files\Content.MSO
```

A sample path with a hidden folder is shown in the taskbar below in Display 2:



Display 2. Excel Dialog Box

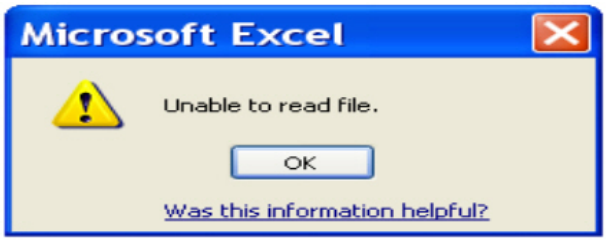
Display 3. Excel Log File

If you follow the tips in this section and you still cannot determine the source of the problem, it might be time to contact the SAS Technical Support Guy!

Note: You should delete all files in the folder when you are finished in order to save disk space. You should also consider creating a shortcut to this folder. Having a shortcut enables you to monitor the folder to make sure it does not grow too large. Be aware that Excel 2013 sometimes generates empty log files, which you should also delete.

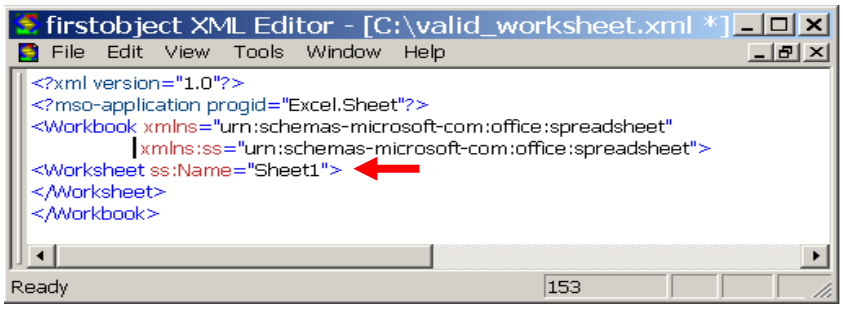
Error That Is Received When No Output Is Generated

If at least one procedure or DATA step does not render any output, the XML file that is generated is left incomplete and generates the message shown in Display 4.



Display 4. Message That Appears When No Output Is Generated

The XML in Display 5 shows the minimal XML syntax that is needed in order to generate a valid file that includes at least a single **<Worksheet>** tag.



Display 5. The Minimal XML Code That Is Required to Generate an XMLSS File

Using a tagset approach, there are two ways to prevent the error shown previously in Display 4 when no output is produced. You can either modify the ExcelXP tagset or add the **<Worksheet>** tag programmatically when no data exists. For code that uses the tagset approach, see the sample code that is available in **ftp://ftp.sas.com/techsup/download/base/SGF2014_examples.ZIP**.

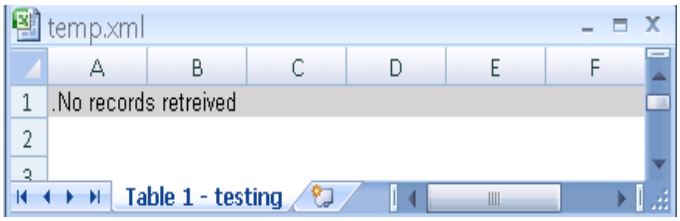
You can also use a coding option that does not use tagsets. This approach uses SAS macro language and the SQL procedure. PROC SQL generates an automatic macro variable named SQLOBS. The value for this macro variable is the number of records of output.

As shown in the following code, you can query SQLOBS so that it conditionally uses a DATA step to write a text string (**No records retrieved**) when no records exist.

```
ods tagsets.ExcelXP
    file="c:\temp.xml";
%macro test;
proc sql;
   select * from sashelp.class
   where age=-1;
run;
quit;

%if &sqlobs=0 %then %do;
data _null_;
   file print;
   put "No records retrieved";
run;
%end;
%mend;
%test
ods tagsets.Excelxp close;
```

Because no records are retrieved with an AGE value of -1 (as shown in the WHERE statement), no records are output in the Excel file.

Display 6 shows the text string that is generated in the Excel table by the code above.



Display 6. Worksheet with Text String Added by Using Macro

DISPLAYING HEADERS FROM DATA SETS WITH ZERO OBSERVATIONS

The example in the previous section generates a valid worksheet with alternate text when no records are produced. But sometimes, users need to see a view of the worksheet as it appears (with column headings, for example) even though no records are generated. So the example that follows generates the column headers when you have data sets with zero observations. This example builds on the previous example, which incorporates macro logic to check the number of observations in the data set before exporting the data.

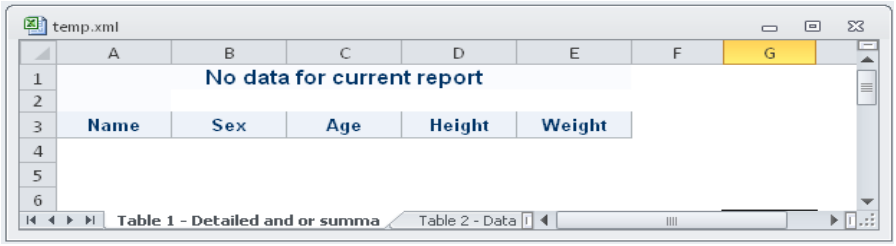
```
ods tagsets.excelxp file="c:\temp.xml" options(absolute_column_width="8"
                                               embedded_titles="yes");
%macro test(libref=,dsn=);
   %let rc=%sysfunc(open(&libref..&dsn,i));
   %let nobs=%sysfunc(attrn(&rc,NOBS));
   %let close=%sysfunc(CLOSE(&rc));

   %if &nobs ne 0 and %sysfunc(Exist(&libref..&dsn)) %then %do;
      proc print data=&libref..&dsn;
      run;
```

```
 %end;                                                    (code continued)
%else %do;
    proc sql noprint;
       create table temp as select name from dictionary.columns
             where libname=%upcase("&libref") and memname=%upcase("&dsn");
    run;
    quit;

    proc transpose data=temp out=temp1(drop=_label_ _name_);
       id name;
       var name;
    run;

    proc report noheader style(column)=header[just=center] nowd;
       title "No data for current report";
    run;
 %end;
%mend;
%test(libref=work,dsn=temp)
%test(libref=sashelp,dsn=class)
ods tagsets.excelxp close;
```

Zero observation data set

In this code:

- The macro conditionally executes the PRINT procedure or the PROC SQL step in which dictionary tables are used to retrieve the column names from the zero observation data set.
- The values (**Name**, **Sex**, **Age**, **Height**, and **Weight**) originally appear vertically in rows. However, the TRANSPOSE procedure rearranges those values into horizontal cells. This data is then generated by the REPORT procedure, as shown in Display 7.



Display 4. Headers Added When No Data Exists

MANAGING HOW TEXT WRAPS IN DATA CELLS

How text wraps is a source of mystery for many people who use the ExcelXP tagset. By default, the HTML-based output that is generated with the MSOffice2K destination generates output that is automatically fitted into the width of the cell. In the ExcelXP tagset, you control text wrapping with the WRAPTEXT= tagset option. This option, which was added in SAS 9.3, sets the AUTOFITWIDTH= XML attribute. The default value for the WRAPTEXT=is option is YES, which means that the data in the cells wraps to fit the column width. When you change the column width either with a style or with the ABSOLUTE_COLUMN_WIDTH= tagset option, data wrapping adjusts automatically. You can also set text wrapping with a style override by using the WRAP:YES | NO option within the TAGATTR= attribute, which can be applied in either a SAS procedure (the PRINT, REPORT, and TABULATE procedures) or a SAS template style (PROC TEMPLATE).

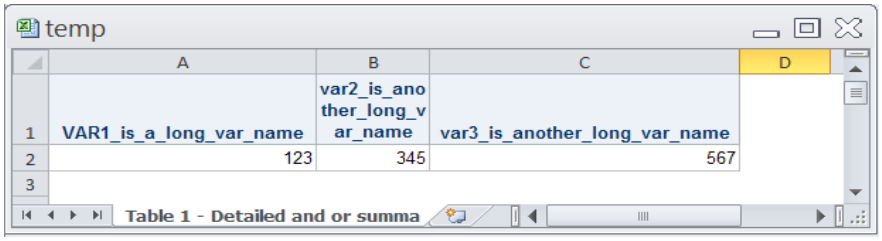
The following example contains variables that have long variable names and two of the variable names need to be wrapped.

```
data one;
   input  var1_is_a_long_var_name  var2_is_another_long_var_name
          var3_is_another_long_var_name;
   cards;
123 345 567
 ;
run;

ods tagsets.excelxp file="c:\temp.xml"
                         options(absolute_column_width="21,8,21")
                         style=htmlblue;

proc report data=one nowd;
   define var1_is_a_long_var_name / style(header)={tagattr="wrap:no"};
   define var3_is_another_long_var_name / style(header)={tagattr="wrap:no"};
run;

ods tagsets.excelxp close;
```

In this code:

- The column width is set with the ABSOUTLUTE_COLUMN_WIDTH= tagset option, and it uses the default wrapping behavior that is set by WRAPTEXT="YES". **Note**: Because WRAPTEXT="YES" is the default setting, it is not shown in the code sample.
- The TAGATTR= attribute is used in the REPORT procedure to selectively adjust the wrapping for two of the variables.

Display 8 shows the worksheet that is generated by the code above.



Display 8. Modifying Wrapping Capabilities with the ExcelXP Tagset

UPDATING WORKBOOKS AND WORKSHEETS

Another common user question is how can you update output that is generated with the ExcelXP tagset? For example, how can you add a new worksheet within a workbook or add data to an existing worksheet? The answer to that question is that the ExcelXP destination and the new experimental Excel destination do not allow you to update a file.

You can update worksheets using a number of other well documented methods, including VBScript, PowerShell scripts, DDE, and the TableEditor tagset. For more information about updating workbooks and worksheets, see the section "Automating SAS Output."

HANDLING COMMON ISSUES WITH TAGSETS

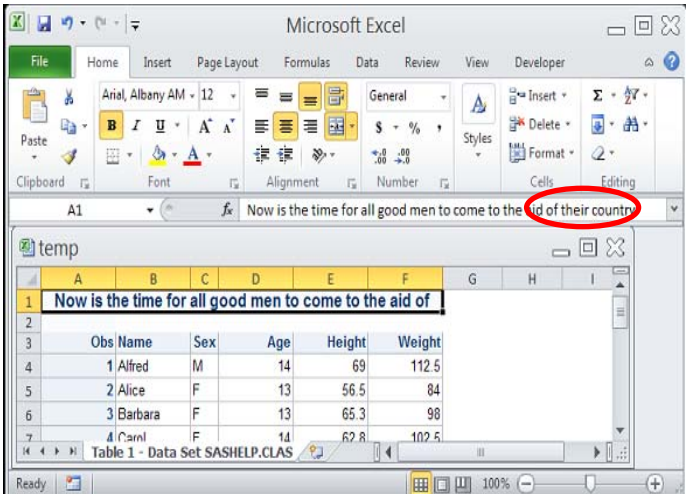
Some SAS users fall in love with a certain version of the ExcelXP tagset and install that same tagset version in subsequent SAS releases (for example, using tagset V1.94 with SAS 9.3). This is not a good practice because tagsets are not forward compatible. When this practice is used, various problems can occur (for example, style problems that include incorrect justification of cells or various warnings in the log.) Beginning with SAS 9.3, grammar changes to the ODS Markup language makes it more likely that an error will occur when you use a tagset that was created in an earlier SAS release.

SAS® Release	Tagset Version	Date
9.1.3	V1.28	08/29/2005
9.20M3	V1.94	09/09/2010
9.30M2	V.1.130	05/01/2011
9.40	V1.129	11/07/2011
9.40M1	V1.130	08/23/2013

Table 1. Tagset Version Based on SAS® Software Release

EMBEDDED TITLES

Prior to SAS 9.3, a frequent user question focused on how to use the ExcelXP tagset to display a complete title that appears truncated when the actual title is embedded in the worksheet. In SAS releases before 9.3, titles that span beyond the length of the Excel table in the worksheet are not fully displayed in the worksheet output, as shown in Display 9.



Display 9. Excel Table with a Truncated Title

Users commonly fix this problem by either manually expanding the row where the title appears or by selecting the title so that it appears in the title bar.

The ExcelXP tagset merges titles across cells that reach the full width of the Excel table. The XMLSS format that the tagset emulates is limited in that the merged titles cannot be automatically fitted when the column wraps. The XMLSS format also does not display data beyond the last column of the table, as shown above in Display 9.

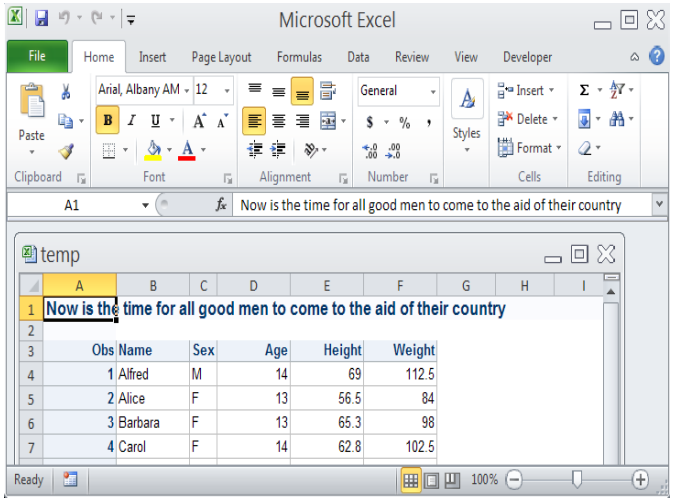
To display the complete title, you need to unmerge the title cell by left-justifying the title. You can left-justify the title in any of these three ways:

- Use the JUSTIFY= option in the TITLE statement.
- Use the JUST= option in the PROC TEMPLATE style.
- Use the NOCENTER option.

The JUSTIFY= option enables you to justify an individual title, whereas using the NOCENTER option or adding the JUST= option in a template style justifies all of the titles. This next sample code illustrates the use of the JUSTIFY= option.

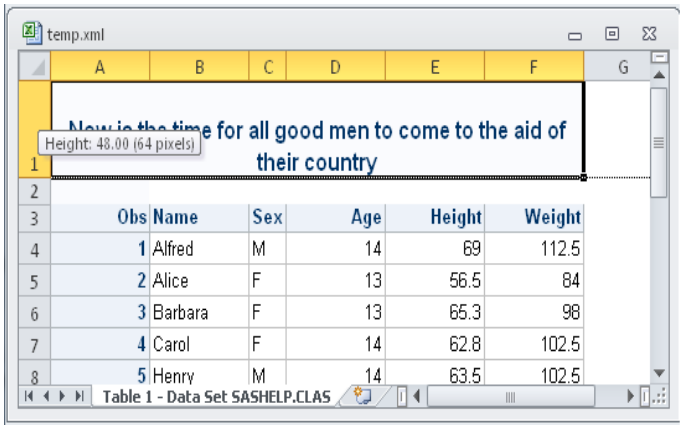
```
ods tagsets.excelxp file="c:\temp.xml" options(embedded_titles="yes");

proc print data=sashelp.class;
     title justify=l "Now is the time for all good men to come to the aid of
                      their country";

ods tagsets.excelxp close;
```

Display 10 shows the unmerged title cells and the complete title.



Display 10. Excel Table with a Left-Justified and Complete Title

Beginning with SAS 9.3, the default behavior for embedded titles changed. Titles now wrap to prevent losing the view of part of a title when the title is wider than the table. To compensate for the XMLSS format limitation of not fitting merged titles automatically, the ExcelXP tagset determines where a title should wrap based on a calculation that involves an estimated width of the title and the table. This width is not based on characters, so its use can lead to incorrect row heights within the title cells. This behavior is true especially with long titles that are close in length to the width of the table, as shown here in Display 11. Notice the extra space above the title.



Display 11. An Excel Table That Has Incorrect Row Heights

For the output above, you can reduce the extra space that appears when the title is wrapped by adding the ExcelXP tagset option WIDTH_FUDGE=".85". This option removes the additional space in the title by extending the width of the table. The ROW_HEIGHTS= option can also be used to set the height for the title.

The following sample code emulates the behavior of SAS prior to 9.3 that enables you merge title cells across a table in order to prevent the title from wrapping. This behavior alleviates the row-height problem by using the TITLE_FOOTNOTE_WIDTH= and the MERGE_TITLES_FOOTNOTES options to prevent the title from wrapping.

```
ods tagsets.excelxp file="c:\temp.xml" options(embedded_titles="yes"
                                               merge_titles_footnotes="no")
                                               style=htmlblue;

   proc print data=sashelp.class;
      title  j=l "Now is the time for all good men to come to the aid of
                  their country";
   run;

ods tagsets.excelxp close;
```

In this code:

- The TITLE_FOOTNOTE_ WIDTH= option enables you to merge the title cells based on the value that you assign to the option.
- The MERGE_TITLES_ FOOTNOTES= option merges the cells of a left-justified title. It also displays the complete title and prevents it from wrapping.

AUTOMATING SAS® OUTPUT

Automating current processes for how worksheets are generated makes your job more efficient and cost effective by reducing the time and staff that are needed to complete tasks. Rather than handling tasks individually, you can take care of the tasks all at once within SAS. If you are already generating reports in Excel, why not generate the reports from SAS instead and take advantage of both world-class analytics and graphics. The next sections discuss two main topics in which the processes are automated:

- automatically generating presentation-quality worksheets directly from SAS (for example, automating the process of generating pivot tables in Excel for interactive data analysis)
- enhancing generated Excel output from SAS

AUTOMATING PIVOT TABLE GENERATION FROM SAS®

The ability to generate pivot tables from Base SAS® software is made possible through the TableEditor tagset, which you can download from **support.sas.com/rnd/base/ods/odsmarkup/index.html**. Prior to the current release of the TableEditor tagset, you could generate pivot tables a couple of different ways. One method involves creating an HTML file that acts as the data source for a pivot table that is created with the TableEditor tagset. When the HTML file is generated, an **Export** button is added. When you click this button, the application exports both the HTML file as the source data and the pivot table, based on the options that you specify with the tagset. The other method involves pointing to a file that Excel can read. Such files include XML files that are generated with the ExcelXP tagset, HTML files that are generated with the MSOffice2K tagset, character-separated values (CSV) files, or native Excel files. This method also requires clicking an **Export** button that uses automation within the browser to launch the code that executes the pivot tables.

Now, however, updates to the TableEditor tagset enable you to fully automate the creation of pivot tables so that no human intervention is necessary. This process requires the generation of a separate file for the scripting, which was added to the HTML file that was discussed in the first method described in the previous paragraph. Then, this script file can be executed within the program. Automating the process this way enables you to perform tasks such as generating a report, analyzing data, and adding the ability create a drill-down report in the batch environment.

The next example examines how to automate the process of generating a pivot table within an existing report that is created with the ExcelXP tagset.

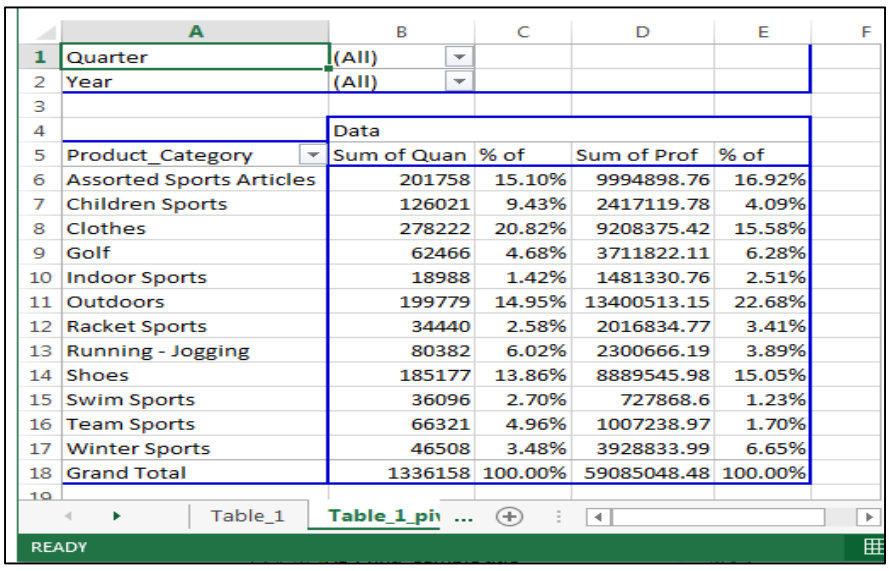
```
   /* Create a workbook to update. */

ods tagsets.ExcelXP file="c:\temp.xml" options(sheet_name="Table_1");
```

(*code continued)*

```
proc print data=sashelp.orsales noobs;
run;

ods tagsets.Excelxp close;
   /* Create and execute the script file that is generated */
   /* with the TableEditor tagset.                         */
ods noresults;
options noxwait noxsync;
ods tagsets.tableeditor file="c:\temp\temp.js"
options(update_target="c:\\temp.xml" open_excel="no" output_type="script"
        sheet_name="Table_1" pivotdata="quantity,quantity,profit,profit"
        pivotdata_stats="sum,sum,sum,sum" pivotcalc="none,total,none,total"
        pivotpage="year,quarter" pivotrow="product_category"
        pivotdata_caption="Sum of Quan,% of,Sum of Prof,% of"
        pivotdata_tocolumns="yes");

data _null_;
   file print;
   put "test";
run;

ods tagsets.tableeditor close;
x 'c:\temp\temp.js';
```

This code sample uses the UPDATE_TARGET= option to add the name of a file that is to be used for the source data. The sample also includes the SHEET_NAME= option, which determines which worksheet should get the data. The OUTPUT_TYPE= option is also used to determine that the file that is written is the scripting file (which is named in the FILE= ODS option).

Display 12 shows the pivot table that is generated with the code sample above.



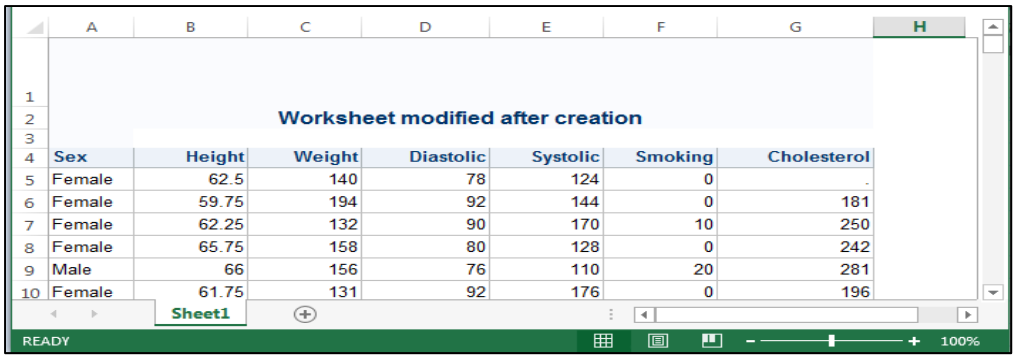
	A	B	C	D	E	F
1	Quarter	(All)				
2	Year	(All)				
3						
4		Data				
5	Product_Category	Sum of Quan	% of	Sum of Prof	% of	
6	Assorted Sports Articles	201758	15.10%	9994898.76	16.92%	
7	Children Sports	126021	9.43%	2417119.78	4.09%	
8	Clothes	278222	20.82%	9208375.42	15.58%	
9	Golf	62466	4.68%	3711822.11	6.28%	
10	Indoor Sports	18988	1.42%	1481330.76	2.51%	
11	Outdoors	199779	14.95%	13400513.15	22.68%	
12	Racket Sports	34440	2.58%	2016834.77	3.41%	
13	Running - Jogging	80382	6.02%	2300666.19	3.89%	
14	Shoes	185177	13.86%	8889545.98	15.05%	
15	Swim Sports	36096	2.70%	727868.6	1.23%	
16	Team Sports	66321	4.96%	1007238.97	1.70%	
17	Winter Sports	46508	3.48%	3928833.99	6.65%	
18	Grand Total	1336158	100.00%	59085048.48	100.00%	
19						

Display 12. Pivot Table That Is Generated as a Result of an Automated Process

ENHANCING GENERATED EXCEL OUTPUT

After you generate a workbook with the ExcelXP tagset or with any of the other ODS destinations, you might find that later requirements for the workbook exceed what is possible with the ExcelXP tagset or with any of the ODS destinations. For example, after you generate a workbook with the ExcelXP destination, you might be required to update the workbook by adding a new worksheet from the previous week's sales data. Or you might need to add the company logo and the accompanying graphics for the report. Or, you might want to add password protection or save the file in another format. The requirements mentioned here are not possible with the ExcelXP tagset because the XMLSS format does not support the tasks listed above. However, you can use any of the following tools to accomplish those and other tasks: DDE, VBScript language, PowerShell scripts, and the TableEditor tagset. You can also write your own script to accomplish some of these tasks, or you can use the %EXCEL_ENHANCE macro that is used in the sample code that follows. This macro, written by the SAS Technical Support Guy, is available at **`ftp://ftp.sas.com/techsup/download/base/SGF2014_examples.ZIP`**.

Display 13 shows a workbook, with one worksheet, as it appears when it is first generated. This is the worksheet that you want to modify.



Display 13. The Original XML Worksheet

The following sample code generates three worksheets and two images (a logo and a graph):

```
proc sgplot data=sashelp.heart;
   scatter x=diastolic y=systolic;
run;
options noxwait noxsync;
%include "c:\example2.sas";
%excel_enhance(open_workbook=c:\temp.xml,
            insert_workbook=c:\example_file.xml,
            insert_sheet=%str(Example_1,Example_2,Example_3),
            insert_image=%str(c:\sas.jpg#Sheet1!A1, c:\sgplot1.jpg#Sheet1!I4),
            create_workbook=c:\Combined_file.xml
            file_format=xlsx
            password=password);
```

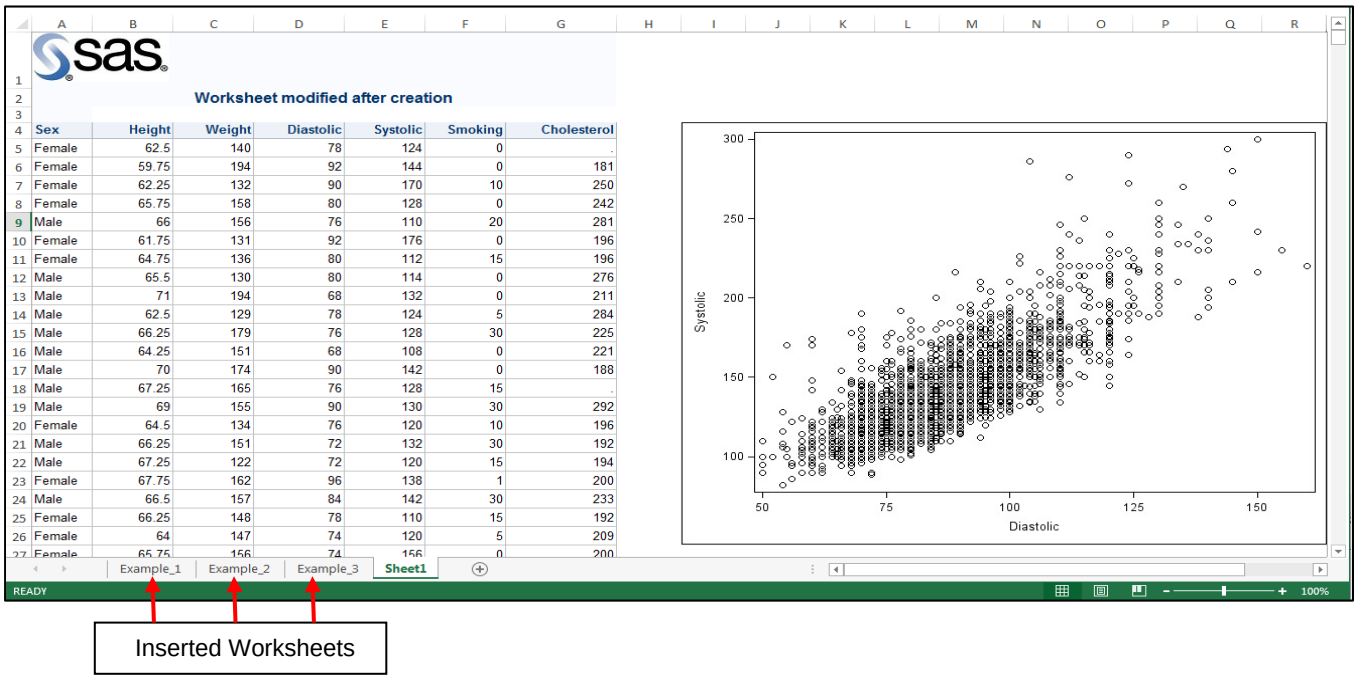
In this code:

- The SGPLOT procedure generates the graph.
- The OPEN_WORKBOOK= option in the %EXCEL_ENHANCE macro points to the file temp.xml that you want to modify.
- The INSERT_WORKBOOK= option points to the file that contains the worksheets that you want to insert.

(list continued)

- The INSERT_SHEET= option adds the worksheets into the current worksheet. Notice that the new worksheet names are embedded in the %STR macro function, which prevents the SAS macro from treating values that are separated by commas as separate parameters when the macro code is compiled and executed.
- The INSERT_IMAGE= macro parameter is an efficient way to add either a logo or a graph into the worksheet. In this parameter, the image descriptor has three parts: the path to the image, the sheet name to which the image should be written, and the range or location to which the image is written. Each image descriptor is separated by commas. Because of the embedded commas, this value is also added within the %STR() function. In the sample code, a logo is added to line1 (beginning with cell A1), of an existing worksheet and a graph to match the table is added to cell I4.

Display 14 shows the enhanced workbook that results from the previous code.



Sex	Height	Weight	Diastolic	Systolic	Smoking	Cholesterol
Female	62.5	140	78	124	0	.
Female	59.75	194	92	144	0	181
Female	62.25	132	90	170	10	250
Female	65.75	158	80	128	0	242
Male	66	156	76	110	20	281
Female	61.75	131	92	176	0	196
Female	64.75	136	80	112	15	196
Male	65.5	130	80	114	0	276
Male	71	194	68	132	0	211
Male	62.5	129	78	124	5	284
Male	66.25	179	76	128	30	225
Male	64.25	151	68	108	0	221
Male	70	174	90	142	0	188
Male	67.25	165	76	128	15	.
Male	69	155	90	130	30	292
Female	64.5	134	76	120	10	196
Male	66.25	151	72	132	30	192
Male	67.25	122	72	120	15	194
Female	67.75	162	96	138	1	200
Male	66.5	157	84	142	30	233
Female	66.25	148	78	110	15	192
Female	64	147	74	120	5	209
Female	65.75	156	74	156	0	200

Display 14. Workbook with Three Worksheets, a Graph, and a Logo Added

CUSTOMIZING WORKSHEETS AND EXTENDING THEIR FUNCTIONALITY

While it is nice to be able to make simple enhancements, your workbooks and worksheets will be much more effective when you can fully customize them both stylistically and functionally. The next sections explore how to maximize style and functionality by using cascading style sheets (CSS). A CSS permits maximum functionality as far as styling you output.

Topics in this section include the following:

- Modifying styles using a CSS: This topic delves into the use of CSS to achieve more advanced style generation. It also covers how the use of CSS differs from the use of the SAS TEMPLATE procedure
- Extending worksheet functionality: This topic discusses extending the functionality of worksheets by using preselected filters and filters.

MODIFYING STYLES USING CSS

The ability to create styles for output using SAS ODS has never been as robust and flexible as it is now. In terms of robustness, you can use either PROC TEMPLATE (available since the first release of SAS ODS in SAS® 7) or a CSS (available since SAS 9.2) to create overall styles for your output. Template styles (PROC TEMPLATE) are

already well documented in SAS documentation. However, the CSS is a newer method. Therefore, this section

focuses more on the use of a CSS. SAS 9.2 provides the ability to apply CSS formatted destinations such as PDF, RTF, and ExcelXP. And the HTML destination has always had the capability to apply CSS formatting by using the STYLESHEET= option in the ODS HTML statement. In SAS 9.2 and later, you apply a CSS by using the CSSSTYLE= in the ODS TAGSET statement. In terms of flexibility, you can also combine both CSS and template styles by using the IMPORT statement in PROC TEMPLATE. This method imports the CSS style.

There are some major differences between styles that are generated by PROC TEMPLATE and those that are generated with CSS. Template styles use a proprietary SAS file that is of member type ITEMSTORE. This file type can be used only in SAS, and it is not portable across operating systems. Styles that are generated with a CSS use a simple text file, which is portable across operating systems, releases, and applications that accept CSS.

The CSS style syntax consists of a list of rules. Each rule contains one or more selectors along with a declaration block that consists of the style properties and values.

The selector indicates to which part of the output the style applies. The most basic type of selector is the CSS class selector. A *CSS class selector* is indicated by prefixing the name of the class with a period (for example, **`.rowheader`**).

Beginning with SAS 9.3, many other types of selectors are supported by the CSS style engine, for example, ID, combinatory, universal, element-name, attribute, and pseudo-class selectors. The ExcelXP tagset supports only the class selectors that are supported by the CSS style engine. However, the new experimental Excel destination supports the other forms as well.

The following code is an example of content that appears in a CSS file.

```
.rowheader, .header {
                     background-color: #D8CFBC;
                     color:red;
                     font-family: sans-serif,Arial,Helvetica;
                     font-size: 11pt;
                     border:1px;
                    }
.data {
       background-color: #F5EFE1;
       color: #000000;
       font-family: sans-serif,Arial,Helvetica;
       font-size: 9pt;
       border:1px;
      }
.systemtitle {
              background-color: #000000;
              color: #FFFFFF;
              font-family: sans-serif,Arial,Helvetica;
              font-size: 11pt;
              border:1px;
             }
ods tagsets.excelxp file="c:\temp.xml" options(embedded_titles="yes")
                    cssstyle="c:\temp.css";
```

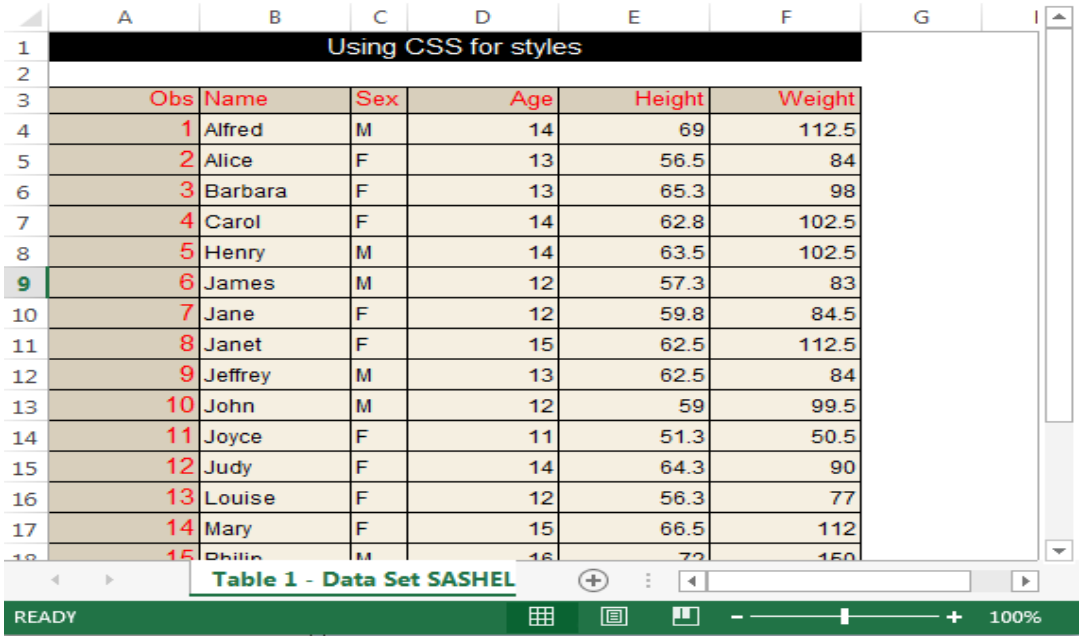
(code continued)

```
proc print data=sashelp.class;
   title "Using CSS for styles";
run;
ods tagsets.Excelxp close;
```

In this code:

- The .**rowheader** and **.header** class selectors are shown together on the same line, but they are separated by commas. These selectors are listed this way because they share the same style property. The **.rowheader** class selector adds style to the row header, and the **.header** class selector adds style to the column header.
- The file also contains the **.data** and the **.systemtitle** class selectors. The **.data** class selector adds style to the individual cells. The **.systemtitle** class selector adds style to the title.
- In addition, the CSSSTYLE= option is used in the ODS statement to import and use this CSS file for the output.

Display 15 shows the results of using the CSS styles that are created by the previous sample code.



Using CSS for styles

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112

Display 15. An Excel Table with CSS Styles Applied

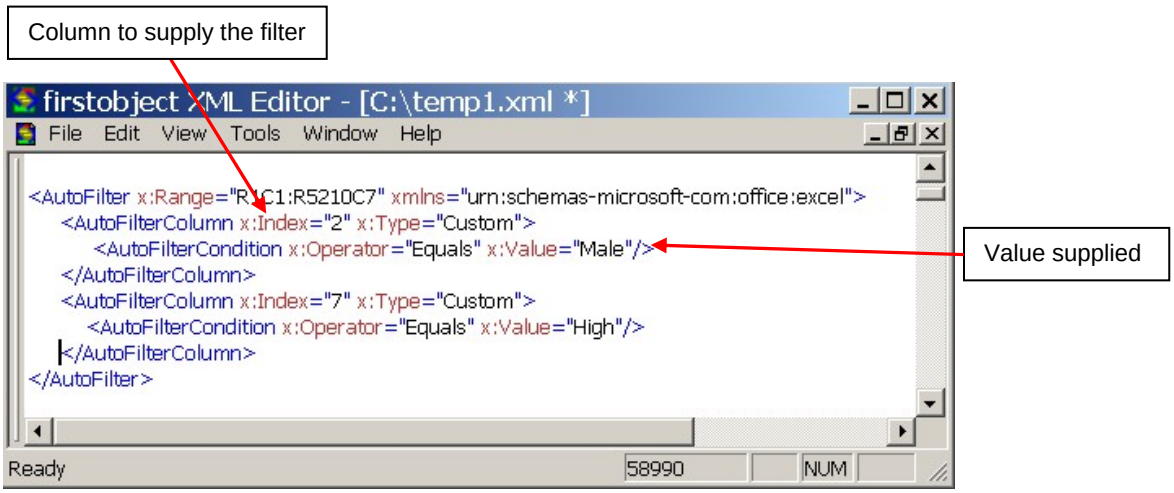
EXTENDING THE WORKSHEET FUNCTIONALITY

As mentioned in the previous section, you can modify the presentation of worksheets by using PROC TEMPLATE and CSS styles. In addition to modifying the appearance of worksheets, youc can also enhance their functionality (for example, by adding filters and preselected filters). *Filters* enable you to drill down in a worksheet and look at the data in separate views. You can easily add filters to the ExcelXP tagset by using the AUTOFILTER= tagset option.

Often, just using a filter is enough for most people. But, suppose you have a worksheet that contains clinical-trials data, and you want to send a user a copy of the worksheet that shows just the males with high-blood pressure. And you also want to be able to drill down into that information to see what factors in this group (such as weight and smoking) might influence the increase in blood pressure. For this type of task, you need to include preselected filters. *Preselected filters* enable you to limit the selection of data based on filters that you supply.

Currently, preselected filters are not included in the production version of the ExcelXP tagset. However, they can be included by modifying the ExcelXP tagset. That is, by modifying the XML that is generated through the tagset, you can specify parameters for a preselected filter (for example, you might want to add a column to a preselected filter). To do that, you add two new subelements (**<AutofilterColumn>** with the INDEX= attribute and **<AutofilterCondition>**) within the **<Autofilter>** element, which resides in the WRITE_AUTOFILTER event of the tagset. The **<AutofilterCondition>** element adds the condition that will be used for the preselected filter.

Display 16 shows the modified XML that is added to the **<Autofilter>** element in order to generate the preselected filters.



Display 16. Modified XML Code for Creating a Preselected Filter

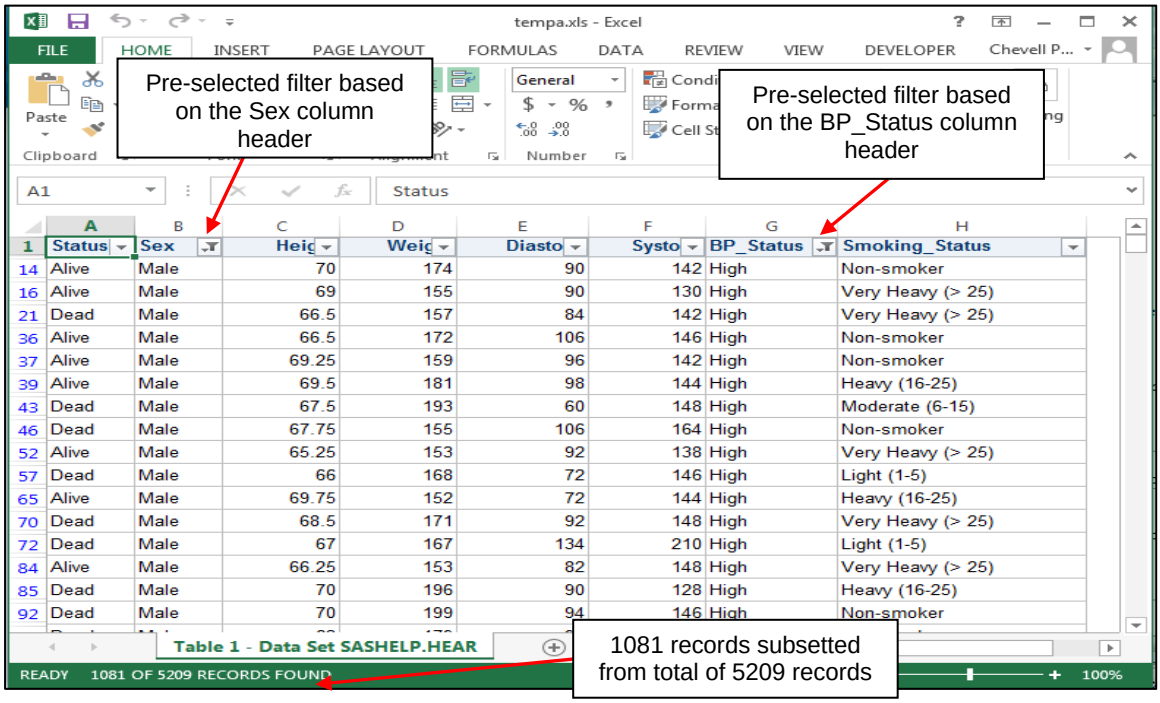
After you download, modify, and compile the ExcelXP tagset code (available at **ftp://ftp.sas.com/techsup/download/base/SGF2014_examples.ZIP**), you can use the following sample code to populate your filter.

```
ods tagsets.Excelxp_Mod file="c:\temp.xml"
                        options(autofilter_values="C2|Male,C7|High"
                                autofilter="yes") style=htmlblue;

proc print data=sashelp.heart noobs;
   var Status Sex Height Weight Diastolic Systolic BP_Status Smoking_Status;
run;

ods tagsets.ExcelXP_Mod close;
```

In this code:

- The AUTOFILTER= option is set to YES.
- The AUTOFILTER_VALUES= attribute specifies both the Excel column number and its related values. The column number and value are separated by a vertical bar (|). For example, column 2 (**C2**) is populated with the value **Male** and column 7 (**C7**) is populated with the value **High**. As you can see in Display 17, the worksheet lists 1081 of the total 5209 records.

Display 17 shows the output from the previous code sample.



	A	B	C	D	E	F	G	H
1	Status	Sex	Heig	Weig	Diasto	Systo	BP_Status	Smoking_Status
14	Alive	Male	70	174	90	142	High	Non-smoker
16	Alive	Male	69	155	90	130	High	Very Heavy (> 25)
21	Dead	Male	66.5	157	84	142	High	Very Heavy (> 25)
36	Alive	Male	66.5	172	106	146	High	Non-smoker
37	Alive	Male	69.25	159	96	142	High	Non-smoker
39	Alive	Male	69.5	181	98	144	High	Heavy (16-25)
43	Dead	Male	67.5	193	60	148	High	Moderate (6-15)
46	Dead	Male	67.75	155	106	164	High	Non-smoker
52	Alive	Male	65.25	153	92	138	High	Very Heavy (> 25)
57	Dead	Male	66	168	72	146	High	Light (1-5)
65	Alive	Male	69.75	152	72	144	High	Heavy (16-25)
70	Dead	Male	68.5	171	92	148	High	Very Heavy (> 25)
72	Dead	Male	67	167	134	210	High	Light (1-5)
84	Alive	Male	66.25	153	82	148	High	Very Heavy (> 25)
85	Dead	Male	70	196	90	128	High	Heavy (16-25)
92	Dead	Male	70	199	94	146	High	Non-smoker

Display 17. Excel Worksheet with Filters Applied

MOVING TO THE EXCEL DESTINATION

The new Excel destination, which is experimental in SAS 9.4 (TS1M1), generates output in Microsoft Office Open XML format. This format is new beginning with Microsoft Office 2007. Files formatted with the new Excel destination store data in a compressed ZIP file that contains various supporting files such as those displayed in Figure 2. The Excel destination encompasses the functionality of the MSOffice2K tagset (which is capable of including graphs and statistical graphs) along with the functionality of the ExcelXP tagset (which is capable of generating multiple worksheets per workbook). The Excel destination also includes the functionality of the CSV destination, which is capable of generating small files. However, the Excel destination does not generate files as small a CSV file.

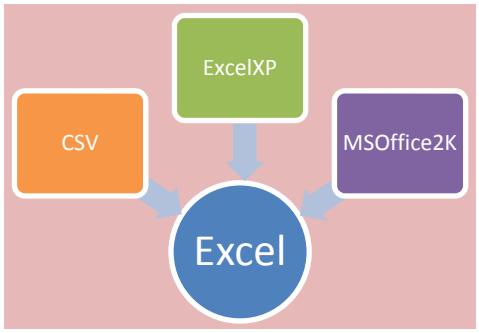


Figure 1. The Excel Destination Encompasses Features from the CSV, ExcelXP, and MSOffice2K Tagsets

Some of the greatest benefits of the Excel destination are the following:

- Output that is generated with the Excel destination can be up to 75% smaller than that generated with other tagsets (such as the ExcelXP tagset).
- Graphics, as well as tables and text, can be stored in the Office Open XML format.

(list continued)

- The Excel destination combines the functionality of the previous tagsets that are used to created worksheets.
- The Excel destination is more fully integrated into ODS (like all the other ODS destinations) than the ExcelXP tagset. The integration is possible because many of the capabilities that required special accommodation in ExcelXP are now inherent in the Excel destination.
- Output can be read easily by other software and applications (for example, the IMPORT procedure).
- You can add formulas using the A1 notation as well as R1C1 notation.

The next sections explore the Office Open XML file format, the ODS and tagset options, new features and functionality, and other information you need in order to migrate to the new Excel destination. You will find the Excel destination to be most like the ExcelXP tagset in terms of functionality and options but with some differences that are also explored in the next sections.

UNDERSTANDING THE MICROSOFT OFFICE OPEN XML FORMAT

Microsoft Office Open XML is an XML-based format for spreadsheets, presentations, charts, and Word documents. This format is the default files format for Office applications such as Excel, PowerPoint, and Word. Starting in SAS 9.4, the ODS PowerPoint destination is the first destination to take advantage of the Open XML format. The Excel destination started using the format in the first maintenance release for SAS 9.4 (TS1M1). Although the XMLSS format that is generated by the ExcelXP destination is simpler, the Open XML format that is generated by the Excel destination is more robust. The Open XML format contains more files as well as more functionality that is needed for worksheets. Open XML is also more cost effective in that its compressed files use less bandwidth when you send them in e-mail.

To accommodate the requirements for the Open XML file format, the Excel destination uses the Lua programming language rather than the ODS Markup language. Other new destinations in SAS 9.4 (HTML5, EPUB, and PowerPoint destinations), also use the LUA language. The change in architecture and programming language means that you can no longer modify or view the tagset in the tagset library as you have in previous releases. However, you can still modify tagsets (ExcelXP, MSOffice2K, CVS, and so on) that were created using the ODS Markup language. Open XML also incorporates better error-handling logic. When problems occur with a file, the format attempts to repair the file so that it can be displayed.

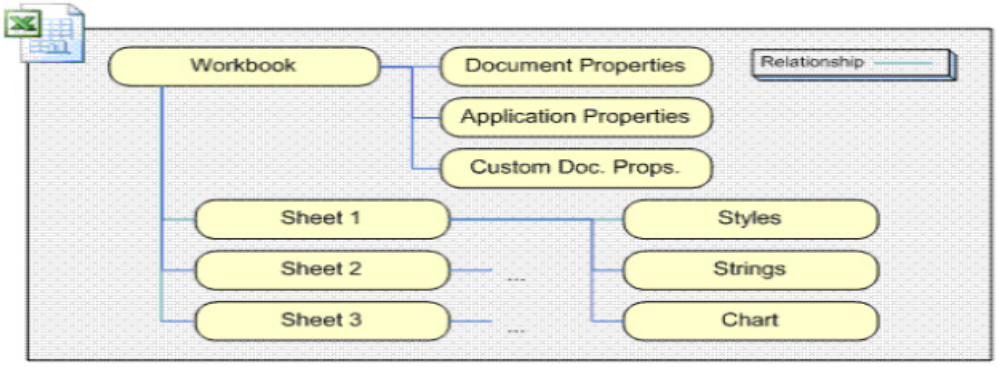


Figure 2. The Structure of a Microsoft Office Open XML Archive ZIP File

FEATURES OFTHE NEW EXCEL DESTINATION

The new Excel destination has much of the same functionality as the ExcelXP tagset, and it includes some new features as well. Most of the existing options and features in the ExcelXP tagset have been carried forward into the Excel destination, and the Excel destination includes some additional enhancements. You will find that the Excel destination works more like the other ODS destinations than the ExcelXP destination ever did, because of better technology and better ways of handling tasks (for example, the Excel destination measures table cells automatically, eliminating the need to adjust the width of the cells manually with tagset options).

The next sections discuss ODS statement options that not valid in the Excel destination, new and deprecated options in the Excel destination, the ability to add graphs and statistical graphs, and automatic cell formatting in the Excel destination.

ODS Statement Options That Are Not Available in the Excel Destination

The GPATH=, PATH=, and BASE= options that are available to both the ExcelXP tagset and the MSOffice2K destination are not valid for the Excel destination. These ODS statement options are used mainly for web output. Because the file that is generated with the Excel destination is a self-contained ZIP archive, the GPATH=, PATH=, and BASE= options are no longer necessary. As a result, the options are not included in the Excel destination. Other options (for example, the FRAME= and DATA= options) for the ExcelXP tagset are no longer valid as well. A syntax error is generated when these options are added to the new Excel destination. So the code in the Excel destination should be simply the following:

```
ods excel file="temp.xlsx";
```

Deprecated and New Excel Destination Options

As mentioned previously, most of the ExcelXP tagset options are included in the new Excel destination. However, some of the existing options have been deprecated in the Excel destination because it inherently handles many tasks that were controlled by options in the ExcelXP tagset. **Note:** This list currently reflects the options that are deprecated or new in the experimental release. This list will change as the Excel destination evolves.

Table 2 lists the options that are new and deprecated in the Excel destination.

ExcelXP Tagset Option	Status in Excel
ABSOLUTE_COLUMN_HEIGHT	New
ASCII_DOTS	Deprecated
AUTOFIT_HEIGHT	Deprecated
AUTO_SUBTOTALS	Deprecated
CONFIGURATION_FILE	Deprecated
CONFIGURATION_NAME	Deprecated
CONTENTS_WORKBOOK	Deprecated
CONVERT_PERCENTAGES	Deprecated
CURRENCY_SYMBOL	Deprecated
DEBUG_LEVEL	Deprecated
DECIMAL_SEPARATOR	Deprecated
DEFAULT_COLUMN_WIDTH	Deprecated
MERGE_TITLE_FOOTNOTES	Deprecated
MINIMIZE_STYLE	Deprecated

ExcelXP Tagset Option	Status in Excel
MISSING_ALIGN	Deprecated
NUMERIC_TEST_FORMAT	Deprecated
PAGE_ORDER_ACROSS	New
PAGEBREAKS	Deprecated
ROW_HEIGHT_FUDGE	Deprecated
ROW_HEIGHTS	Deprecated
SKIP_SPACE	Deprecated
START_AT	New
TAB_COLOR	New
THOUSANDS_SEPARATOR	Deprecated
TITLE_FOOTNOTE_WIDTH	Deprecated
WIDTH_FUDGE	Deprecated
WIDTH_POINTS	Deprecated
WRAPTEXT	Deprecated

Table 2. New and Deprecated Options in the Excel Destination

Many of the deprecated options (for example CURRENCY_SYMBOL= and DECIMAL_SEPARATOR=) are handled by the locale that is set for your SAS software. Other options (for example, WRAPTEXT= and AUTOFIT_HEIGHT=) are handled internally, by default, so there is no need to use those options.

The following example demonstrates the use of some of the new Excel destination options that are not present in the ExcelXP tagset. This example also uses the ODS TEXT= option that is valid with the MSOffice2K destination but not with the ExcelXP destination.

```
ods excel file="c:\temp.xlsx" options(start_at="B5" tab_color="red"
                                      absolute_row_height="15"
                                      embedded_titles="yes");

ods text="Sales report for company X";

proc print data=sashelp.orsales;
   title "Sample title showing new features";
run;

ods excel close;
```

In this code:

- The Excel destination's START_AT= option positions the worksheet text starting in column B and row 5.
- The TAB_COLOR= option adds a highlight color (red) to the tab.
- The ABSOLUTE_ROW_HEIGHT= option adds a default height for all cells in this example.
- The ODS TEXT= statement adds supplementary text to the worksheet.

The code sample results in the worksheet shown here in Display 18:

temp.xlsx

Sample title showing new features of the Excel destination

Sales report for company X

Year	Quarter	Product Line	Product Category	Product Group	Number of Items	Profit in USD	Total Retail Price in USD
1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.90
1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.40
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.80
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334	7136.80	14337.20
1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303	7163.00	12996.20
1999	1999Q1	Children	Children Sports	N.D. Gear, Kids	755	19153.05	34250.50
1999	1999Q1	Children	Children Sports	Olssons, Kids	209	1975.35	3339.30
1999	1999Q1	Children	Children Sports	Orion Kid's Clothes	14	288.80	580.40
1999	1999Q1	Children	Children Sports	Osprey, Kids	454	7334.70	13219.60
1999	1999Q1	Children	Children Sports	Tracker Kid's Clothes	1243	21847.85	40049.50

Table 1 - Detailed and-or su

Display 18. Excel Worksheet That Is Styled by Using Two New Options in the Excel Destination

Automatic Cell Formatting in the Excel Destination

There are some common issues related to cell formatting that can occur when output is exported to an Excel spreadsheet. Problems can occur when you do not apply an Excel format. By default, Excel uses its General format. However, this format can cause display problems in output that is generated by the ExcelXP, MSOffice2K, and CSV destinations. For example, output might appear with leading zeros removed, or long values might appear in scientific notation.

The new Excel destination alleviates these problems because it performs an automatic mapping of SAS formats to Excel formats. This automatic behavior saves you a lot of time and effort because you do not have to learn the Excel custom formatting that is required in order to display the output that you want.

But what do you do if you want to use a different format outside of automatic mapping? With the Excel destination, you can override the default automatic mapping of cell formats by using the TAGATTR= attribute with the FORMAT parameter (as you can in the ExcelXP tagset), or you can use the HTMLSTYLE= attribute with the CSS style property **`mso-number-format`**.

The following code uses the Excel destination (which automatically maps from SAS formats to Excel formats) to apply formats to all of the numeric variables in the code:

```
data one;
   zero=0001;
   dollar_val=123456;
   percent_val=.80;
   comma_val=123456;
   number=12345;
run;

ods excel file="c:\temp.xlsx";
```

(code continued)

```
proc print data=one;
   var _numeric_ ;
   var number / style(data)={tagattr="format:[red]#,###"};
   format zero z4. dollar_val dollar6.0 percent_val percent5.2
          comma_val comma. number 4.;
run;

ods excel close;
```

In this code, the variable NUMBER uses both the automatic mapping as well as a style override that changes the color to red and that applies the desired format to the output.

Display 19 shows output that maintains leading zeros and long values because of the automatic format mapping.

Display 19. Output That Is Generated with the New Excel Destination

ADDING GRAPHS TO WORKSHEETS WITH THE EXCEL DESTINATION

In addition to providing more efficient data storage and automatic format conversion, the Excel destination also enables you to add graphs to a multisheet workbook. The Excel destination is the first true ODS destination that allows images and tables to co-exist while it generates multiple worksheets per workbook. This capability is not possible with the ExcelXP destination. You can add graphs in the MSOffice2K destination, but you cannot generate multiple worksheets per workbook.

With the Excel destination, you can generate graphs in different ways:

- with one of the SAS/GRAPH® procedures (device-based graphics)
- with SAS ODS Graphics (template-based graphics, via the Graph Template Language [GTL])
- statistical graphics procedures such as the SGPLOT, SGPANEL, and SGSCATTR procedures

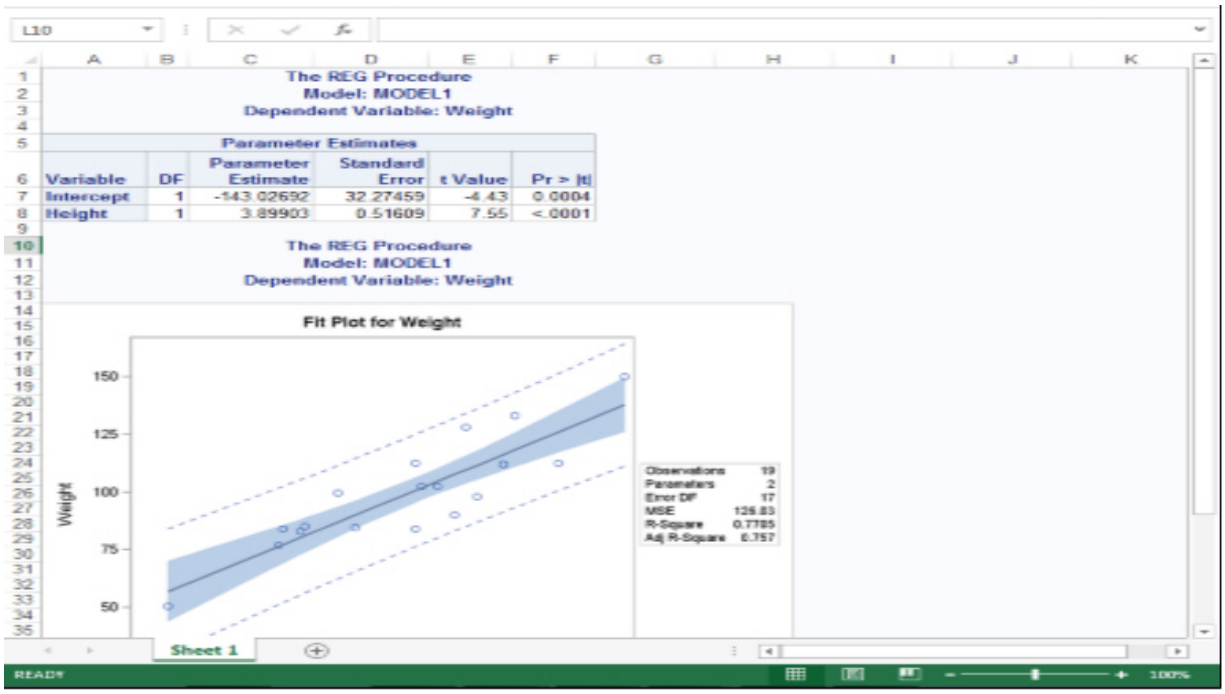
The following example uses the Excel destination along with the REG procedure, which generates output both a table and a graph to the worksheet.

```
ods select fitplot parameterestimates;
ods excel file="c:\temp.xlsx" options(sheet_interval="none");

proc reg data=sashelp.class;
   model weight=height;
run;
quit;

ods excel close;
```

In this code:

- The ODS SELECT statement adds the FITPLOT and PARAMETERESTIMATES objects.
- The SHEET_INTERVAL="NONE" option places both the table and graph on the same worksheet.

Display 20 shows the result of this code sample.



Display 20. Excel Worksheet That Contains Both a Table and a Graph

CONCLUSION

Using SAS ODS in combination with Microsoft Excel enables you to leverage the power of SAS while taking advantage of Excel's ability to generate fully functional and presentation-quality spreadsheets. In this paper, the SAS Technical Support Guy first discusses some of the common issues that are associated with using ODS to generate worksheets using ODS and solutions to those issues. The paper also discussed some of the methods for automating certain worksheet tasks such as generating pivot tables and adding images to the worksheets. The discussion also covered enhancement of worksheet styles using CSS and extended functionality in the tagset. Finally, the SAS Technical Support Guy offered features and tips about moving to the new, experimental Excel destination. The SAS Technical Support Guy hopes that these secrets that have been shared can help you generate powerful spreadsheets for your users.

ACKNOWLEDGMENTS

I would like to thank Susan Berry and Kevin Smith for their assistance with this paper.

RECOMMENDED READING

- Eberhardt, Peter and Louanna Kong. 2012. "The Armchair Quarterback: Writing SAS® Code for the Perfect Pivot (Table, That Is)." *Proceedings of the SAS Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings12/146-2012.pdf**.
- Parker, Chevell. 2010. "Using SAS Output Delivery System (ODS) Markup to Generate Custom PivotTable and PivotChart Reports." *Proceedings of the SAS Global Forum 2010 Conference.* Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings10/003-2010.pdf**.
- Smith, Kevin D. 2011. "Unveiling the Power of Cascading Style Sheets (CSS) in ODS." *Proceedings of the SAS Global Forum 2011 Conference.* Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings11/297-2011.pdf**.
- Parker, Chevell. 2011. "The Perfect Marriage: The SAS Output Delivery System and Microsoft Office." *Proceedings of the SAS Global Forum 2011 Conference.* Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings11/250-2011.pdf**.

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: **Chevell.Parker@sas.com**
Web: **support.sas.com**