Paper SA03

Tips and Tricks for Creating Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®

Vincent DelGobbo, SAS Institute Inc., Cary, NC

ABSTRACT

Transferring SAS® data and analytical results between SAS and Microsoft Excel can be difficult, especially when SAS is not installed on a Windows platform. This paper discusses using the new XML support in Base SAS®-9 software to create multi-sheet Microsoft Excel workbooks (versions 2002 and later). You will learn step-by-step techniques for quickly and easily creating attractive multi-sheet Excel workbooks that contain your SAS output, and also tips and tricks for working with the ExcelXP ODS tagset. Most importantly, the techniques that are presented in this paper can be used regardless of the platform on which SAS software is installed. You can even use them on a mainframe! The use of SAS server technology is also discussed. Although the title is similar to previous papers by this author, this paper contains new and revised material not previously presented.

INTRODUCTION

This paper provides you with step-by-step instructions for using Base SAS 9.1 to create the Excel workbook shown in Figure 1.

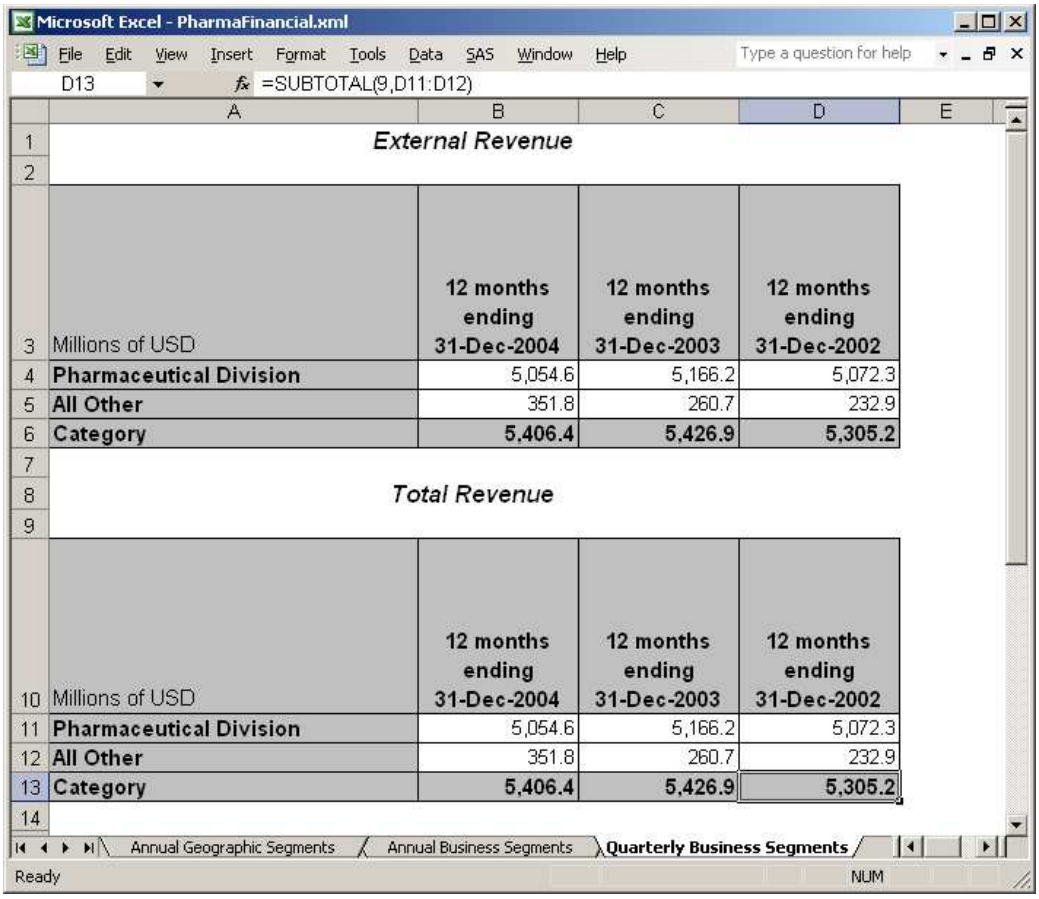
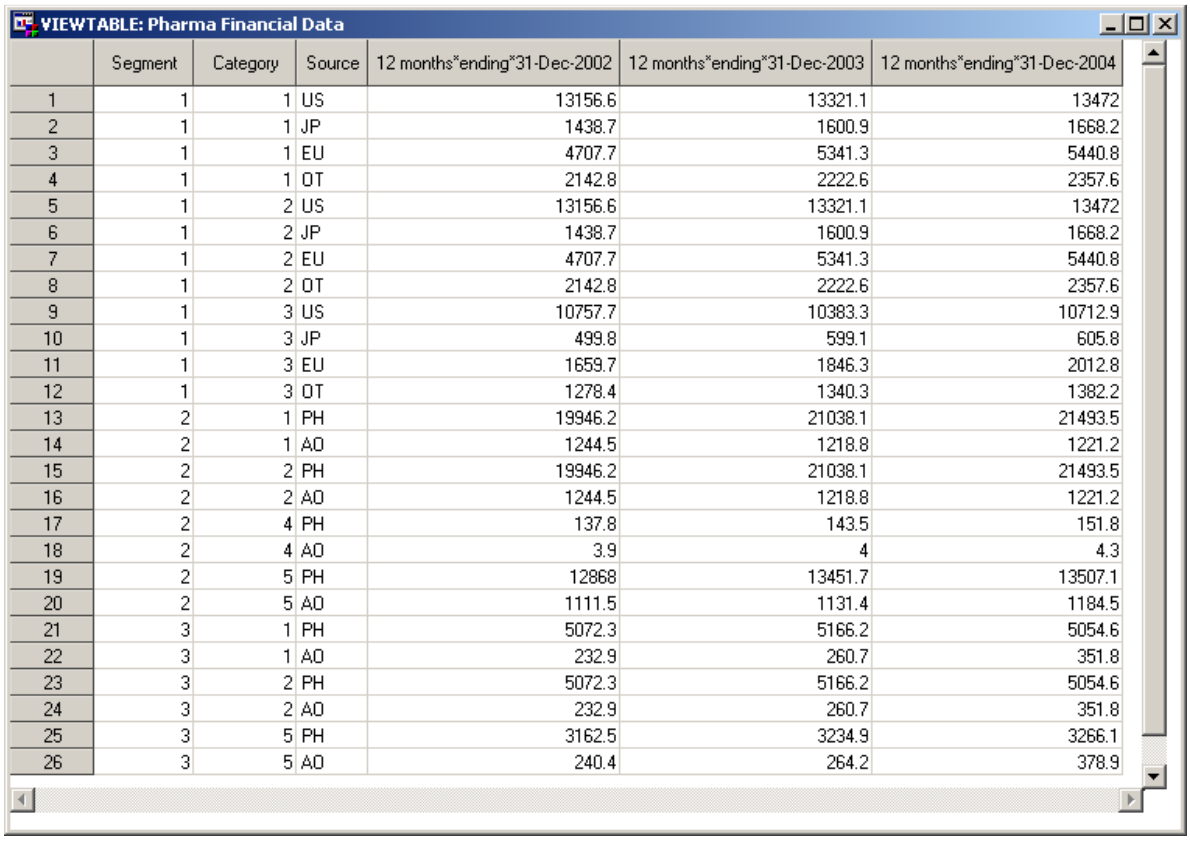


Figure 1. Multi-Sheet Excel Workbook Generated by the ODS ExcelXP Tagset

The workbook contains 3 worksheets with financial data for a fictional pharmaceutical company. Each worksheet contains more than one Excel table. Excel formats, not SAS formats, are used to control the appearance of the numeric data. Note that the revenue totals for each category in Figure 1 are not static values, but are Excel subtotals. For example, the value in cell D13 is not 5305.2, but instead is the Excel formula `=SUBTOTAL(9, D11:D12)`. The raw SAS data is shown in Figure 2.



VIEWTABLE: Pharma Financial Data

	Segment	Category	Source	12 months*ending*31-Dec-2002	12 months*ending*31-Dec-2003	12 months*ending*31-Dec-2004
1	1	1	US	13156.6	13321.1	13472
2	1	1	JP	1438.7	1600.9	1668.2
3	1	1	EU	4707.7	5341.3	5440.8
4	1	1	OT	2142.8	2222.6	2357.6
5	1	2	US	13156.6	13321.1	13472
6	1	2	JP	1438.7	1600.9	1668.2
7	1	2	EU	4707.7	5341.3	5440.8
8	1	2	OT	2142.8	2222.6	2357.6
9	1	3	US	10757.7	10383.3	10712.9
10	1	3	JP	499.8	599.1	605.8
11	1	3	EU	1659.7	1846.3	2012.8
12	1	3	OT	1278.4	1340.3	1382.2
13	2	1	PH	19946.2	21038.1	21493.5
14	2	1	AO	1244.5	1218.8	1221.2
15	2	2	PH	19946.2	21038.1	21493.5
16	2	2	AO	1244.5	1218.8	1221.2
17	2	4	PH	137.8	143.5	151.8
18	2	4	AO	3.9	4	4.3
19	2	5	PH	12868	13451.7	13507.1
20	2	5	AO	1111.5	1131.4	1184.5
21	3	1	PH	5072.3	5166.2	5054.6
22	3	1	AO	232.9	260.7	351.8
23	3	2	PH	5072.3	5166.2	5054.6
24	3	2	AO	232.9	260.7	351.8
25	3	5	PH	3162.5	3234.9	3266.1
26	3	5	AO	240.4	264.2	378.9

Figure 2. Raw SAS Data Displayed Using the SAS VIEWTABLE Application

The remainder of this paper will guide you through the steps used to create the Excel workbook shown in Figure 1 using Base SAS 9.1 software. You can download a copy of the code and data used in this paper from the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Tips and Tricks for Creating Multi-Sheet Excel Workbooks the Easy Way with SAS".

The code in this paper was tested using SAS 9.1.3 Service Pack 4 and Microsoft Excel 2002 SP3 software.

REQUIREMENTS

In order to use the techniques described in this paper, you must have the following software:

- Base SAS 9.1.3 or later, on *any* supported operating system and hardware.
- Microsoft Excel 2002 or later (also referred to as Microsoft Excel XP).
- An updated version of the SAS ExcelXP ODS tagset.

For more information about obtaining an updated version of the tagset, see "The ExcelXP Tagset" section later in this paper.

OUTPUT DELIVERY SYSTEM (ODS) BASICS

ODS is the part of Base SAS software that enables you to generate different types of output from your procedure code. An ODS *destination* controls the type of output that is generated (HTML, RTF, PDF, etc.). An ODS *style* controls the appearance of the output. In this paper, we use a type of ODS destination, called a *tagset*, that creates XML output that can be opened with Excel. This tagset, named ExcelXP, creates an Excel workbook that has multiple worksheets.

The Excel workbook in Figure 1 was created using the ExcelXP ODS tagset and the XLsansPrinter ODS style. The ExcelXP tagset creates an XML file that, when opened by Excel, is rendered as a multi-sheet workbook. All formatting and layout are performed by SAS; there is no need to "hand-edit" the Excel workbook. You simply use Excel to open the file created by ODS.

Here is the general form of the ODS statements that generate XML output that is compatible with versions of Microsoft Excel 2002 and later:

```
❶ ods listing close;
❷ ods tagsets.ExcelXP style=style-name file='file-name.xml' ... ;
    *  Your SAS code here;
❸ ods tagsets.ExcelXP close;
```

The first ODS statement (❶) closes the LISTING destination, which writes output to a listing file in batch mode, or to the Output window when SAS is run interactively. We only want to generate XML output for use with Excel.

The second ODS statement (❷) uses the ExcelXP tagset to generate the XML output and then stores the output in a file. The STYLE option controls the appearance of the output, such as the font and color scheme. To see a list of ODS styles that are available for use at your site, submit the following SAS code:

```
ods listing;
proc template; list styles; run; quit;
```

The third ODS statement (❸) closes and releases the XML file so that it can be opened with Excel.

Although you can store your output on a local disk (where SAS software is installed), or on a network-accessible disk, here are some good reasons to store your SAS output on a Web server:

- The files are available to anyone who has network access.
- The XML files can be accessed by Web-enabled applications besides Excel.
- You can take advantage of Web server authentication and security models.

Note: If you place the files where others can access them over a network, you should set file permissions to prevent accidental alteration.

OPENING ODS OUTPUT WITH EXCEL

To open an ODS-generated file that is stored on a Web server, follow these steps:

1. In Excel, select **File ➨ Open**.
2. In the **File name** field, specify the full URL for the file you want to open. For example, **http://*Web-server/directory/file-name*.xml**.
3. Click **Open** to import the XML file.

To open ODS-generated files from a local or network-accessible disk, follow the same steps with one exception: in step 2 you should either navigate to the desired file or type the path and file name in the **File name** field.

Excel will read and convert the XML file to the Excel format. After the conversion, you can perform any Excel function on the data. To save a copy of the file in Excel binary format using Excel 2002 or Excel 2003, select **`File`** ➨ **`Save As`** and then, from the **`Save as type`** drop-down list, select **`Microsoft Excel Workbook (*.xls)`**.

SETTING UP THE ODS ENVIRONMENT

Our sample code uses a user-defined style named XLsansPrinter and a modified version of the ExcelXP tagset. Use the ODS PATH statement to set the location where this style and tagset will be stored on your system:

```
❶ libname mylib 'some-directory'; * Location to store tagsets and styles;

❷ ods path mylib.tmplmst(update) sashelp.tmplmst(read);
```

The LIBNAME statement (❶) specifies where to store the user-defined tagsets and styles. Although you can temporarily store tagsets and styles in the WORK library, it is more efficient to create them once, and then store them in a permanent library so that you can reference them in other SAS programs.

The ODS PATH statement (❷) specifies the locations and the order in which to search for ODS tagsets and styles. Notice that the access mode for `mylib.tmplmst` is specified as `update` and the access mode for `sashelp.tmplmst` is specified as `read`.

ODS will search the path in the order given. Because PROC TEMPLATE, used later to create the tagsets and styles, will write in the first available location, and the access mode for `mylib.tmplmst` is `update`, the tagsets and styles will be stored in a file named `tmplmst.sas7bitm` in the directory that is associated with the MYLIB library.

Because ODS searches the path in the order given and the access mode for `mylib.tmplmst` is `update`, PROC TEMPLATE, used later in this paper, will store tagsets and styles in a file named `tmplmst.sas7bitm` in the directory that is associated with the MYLIB library.

THE EXCELXP TAGSET

Once you have issued the appropriate ODS PATH statement, you can import the modified version of the ExcelXP tagset and use it in your SAS programs. The version of the tagset used in this paper can be found in the download package on the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Tips and Tricks for Creating Multi-Sheet Excel Workbooks the Easy Way with SAS". The download package contains a file named ExcelXP.sas, which contains the SAS code for creating the ExcelXP tagset. Save a copy of this file, and submit the following SAS code to make the tagset available:

```
%include 'ExcelXP.sas';  * Specify path to the file, if necessary;
```

You should have to perform this step only once. The ExcelXP tagset will be imported and stored in the directory corresponding to the MYLIB library. All of your future SAS programs can access the tagset by specifying the correct LIBNAME and ODS PATH statements (see "Setting up the ODS Environment" above).

The ExcelXP tagset supports several options that control both the appearance and functionality of the Excel workbook. To see a listing of the supported options, submit this SAS code:

```
filename temp temp;
ods tagsets.ExcelXP file=temp options(doc='help');
ods tagsets.ExcelXP close;
```

The tagset information is printed to the SAS log. For your convenience, this information is included in the download package for this paper.

IMPORTANT NOTE

The version of the ExcelXP tagset that was shipped with Base SAS 9.1 has undergone many revisions since its initial release. In order to take advantages of the features discussed in this paper, you must download a recent copy of the tagset and install it on your system as described above. The version of the tagset used in this paper can be found in the download package on the SAS Presents Web site at (support.sas.com/saspresents). The most recent version of the tagset is available from the ODS Web site ("ODS MARKUP Resources", SAS Institute Inc.).

A BRIEF ANATOMY OF ODS STYLES

ODS styles control all aspects of the appearance of the output, and Base SAS software ships with over 40 different styles. A style is composed of *style elements,* each of which controls a particular part of the output. For example, all styles contain a style element named "header" that controls the appearance of column headings. Style elements consist of collections of *style attributes*, such as the background color and font size. The definition of the style element named "header" might look like this:

```
style header /
  font_face   = "Arial, Helvetica"
  font_size   = 11pt
  font_weight = bold
  foreground  = black
  background  = #bbbbbb;
```

As you can see, this style element contains the style attributes `font_face`, `font_size`, and so on. The TEMPLATE procedure can be used to modify an existing style or to create a completely new style from scratch. The next section describes using the TEMPLATE procedure to modify an existing style.

THE XLSANSPRINTER STYLE

Although the appearance of the workbook shown in Figure 1 closely resembles that which is generated using the standard ODS style named sansPrinter, the workbook was created using a custom ODS style named XLsansPrinter, which is based on the standard sansPrinter style. The "XL" in the XLsansPrinter name indicates that this style is intended for use with Microsoft Excel.

The XLsansPrinter style was created by using the TEMPLATE procedure to make three minor changes to the sansPrinter style. These changes result in three new style elements that will be used later, in the section "Changing the Appearance Using ODS Style Overrides and Excel Formats". The complete code for creating this style can be found in the appendix of this paper, in the section "Code for Creating the XLsansPrinter Style".

The new style element "header_id" was created based on the standard style element named "header". The "header_id" style element has all the same style attributes as its parent style element ("header"), except the font weight will not be bold. This style element will be used to control the appearance of the column header for the ID variable.

The two other new style elements "data_comma_1" and "header_comma_1" were created based on the standard style elements named "data" and "header", respectively. These style elements contain all the same style attributes as their parents, and have an additional style attribute named "tagattr". This attribute is used to apply an Excel format that causes data to be presented similarly to the SAS COMMAw.1 format. For more detailed information about Excel formats, refer to a book covering this topic, or type "create a custom number format" into the Excel help system.

Run the code in the appendix to create the XLsansPrinter style on your system. You should have to perform this step only once. The XLsansPrinter style will be stored in the directory corresponding to the MYLIB library. All of your future SAS programs can access the style when you specify the correct LIBNAME and ODS PATH statements (see the "Setting Up the ODS Environment" section).

Because an in-depth discussion of creating and using ODS styles is beyond the scope of this paper, see the chapter about the TEMPLATE procedure in the ODS documentation (SAS Institute Inc., "The TEMPLATE Procedure: Creating a Style Definition").

USING ODS TO CREATE THE MULTI-SHEET EXCEL WORKBOOK

By default, the ExcelXP tagset will create a new worksheet each time a SAS procedure creates new tabular output. Because our sample code uses BY statements, a new worksheet is created for each BY group, resulting in a total of 10 worksheets, each containing one Excel table. To create a more readable report, we will modify the code to output 3 worksheets with multiple tables on each.

SAS CODE TO CREATE THE EXCEL WORKBOOK

Here is a listing of the *basic* SAS code used to create the Excel workbook:

```
  ods listing close;
❶ ods tagsets.ExcelXP path='output-directory' file='PharmaFinancial.xml'
    style=XLsansPrinter;

    title;
    footnote;

    *  Create a separate worksheet for each business segment;

❷   proc print data=mylib.PharmaFinancial noobs label split='*';
      where segment eq 1;  * Annual Geographic Segments;
      by segment category;
      id source;
      var val2004-val2002;
      sum val2002-val2004;
    run; quit;

    proc print data=mylib.PharmaFinancial noobs label split='*';
      where segment eq 2;  * Annual Business Segments;
      by segment category;
      id source;
      var val2004-val2002;
      sum val2002-val2004;
    run; quit;

    proc print data=mylib.PharmaFinancial noobs label split='*';
      where segment eq 3;  * Quarterly Business Segments;
      by segment category;
      id source;
      var val2004-val2002;
      sum val2002-val2004;
    run; quit;

  ods tagsets.ExcelXP close;
```

As you can see in this statement (❶), the ExcelXP tagset is used to generate the output, and the XLsansPrinter style is used to control the appearance of the output. The PRINT procedure (❷) is run with BY statements to create the worksheets. The SAS table "PharmaFinancial" is included in the download package for this paper.

Figure 3 shows the result of executing the SAS code above and opening the file "PharmaFinancial.xml" using Excel.

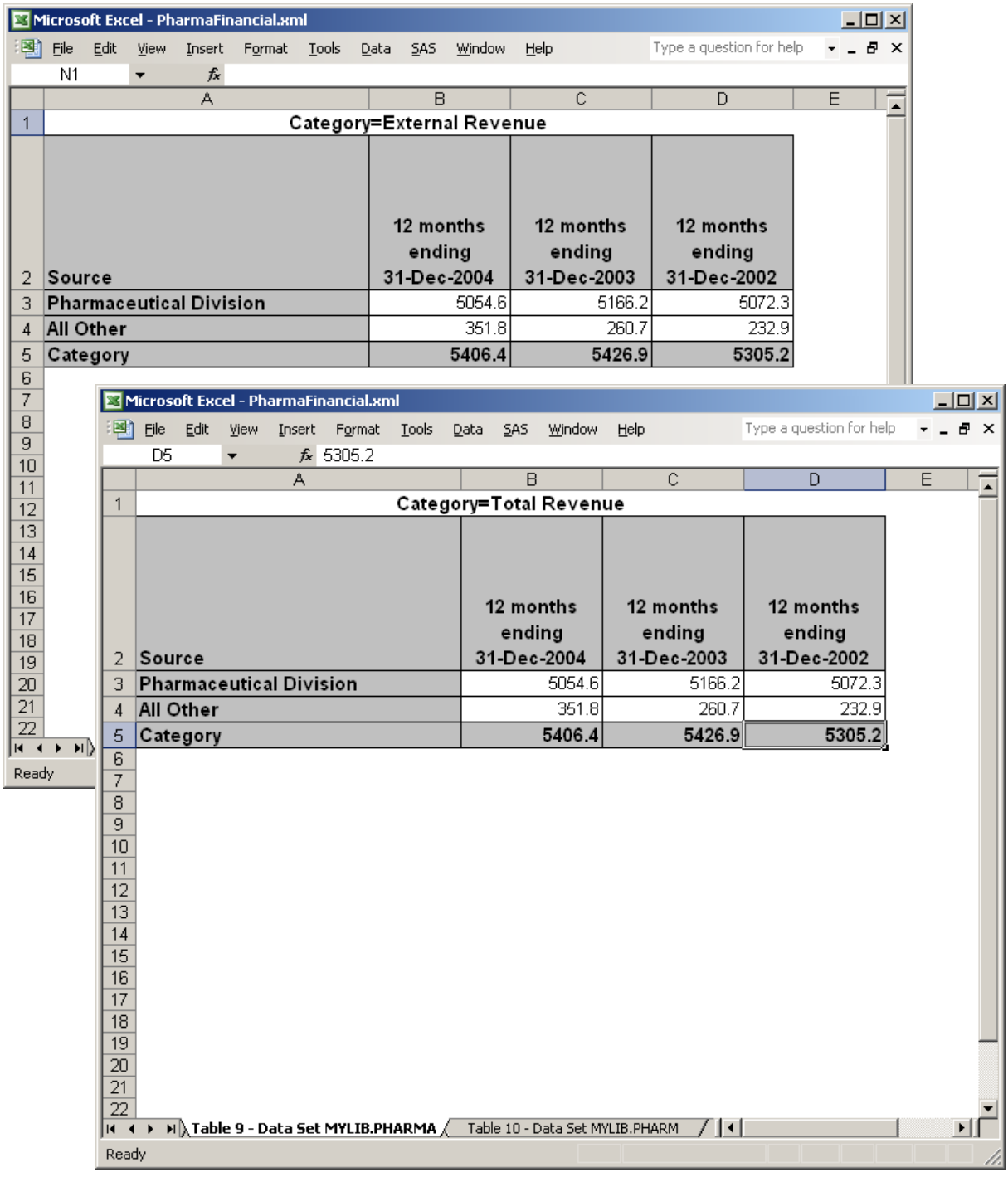


Figure 3. Two Different Worksheets of the ODS ExcelXP-Generated Workbook

Notice that Figure 3 does not match Figure 1. The problems exhibited in Figure 3 are:

1. The data for each BY group is in a separate worksheet, resulting in a total of 10 worksheets, instead of 3.
2. Default worksheet names were used, instead of the name of the corresponding business segment.
3. Standard BY group text precedes the Excel tables, instead of custom titles.

4. The Excel SUBTOTAL function is not used in the summary cells. For example, cell D5 contains the static text "5305.2" instead of the appropriate Excel formula.
5. Numeric values are not formatted using a comma for the thousands separator.
6. The heading for the "Source" column should be "Millions of USD" in light weight text.

We will now make changes to the preceding SAS code to correct these problems.

GROUPING MORE THAN ONE EXCEL TABLE IN A WORKSHEET: AN INTRODUCTION TO TAGSET OPTIONS

As mentioned earlier, the ExcelXP tagset supports several options that control both the appearance and functionality of the Excel workbook. Many of these tagset options are simply tied straight into existing Excel options or features. Tagset options are specified on the ODS statement using the OPTIONS keyword:

```
ods tagsets.ExcelXP options(option-name1='value1' option-name2='value2' ...) ... ;
```

It is important to note that tagset options remain in effect until they are either turned off or set to another value. Since multiple ODS statements are allowed, it is good practice, in terms of functionality and code readability, to explicitly reset tagset options to their default value when you are finished using them. For example:

```
ods tagsets.ExcelXP options(option-name='some-value') ... ;
  *  Some SAS code here;
ods tagsets.ExcelXP options(option-name='default-value') ... ;
  *  Other SAS code here;
```

By default, the ExcelXP tagset will create a new worksheet each time a SAS procedure creates new tabular output. Because BY statements are used with the PRINT procedure, our Excel workbook will initially contain 10 worksheets instead of only the 3 shown in Figure 1.

The SHEET_INTERVAL tagset option can be used to control when the tagset creates a new worksheet. Setting this option to "none" serves two purposes:

- It suppresses the creation of new worksheets.
- If a worksheet is currently being created, it is "closed", and subsequent output is directed to a new worksheet.

In the code below, the first time the SHEET_INTERVAL option is set it creates the first worksheet, and the output from the first PRINT procedure code is directed to this worksheet. When the option is set the second time, the first worksheet is closed and a new worksheet is created containing the output from the second PRINT procedure code. Similarly, the third instance of the SHEET_INTERVAL option results in the creation of the third and final worksheet.

```
ods listing close;
ods tagsets.ExcelXP path='output-directory' file='PharmaFinancial.xml'
  style=XLsansPrinter;

  title;
  footnote;

  *  Create a separate worksheet for each business segment;

  ods tagsets.ExcelXP options(sheet_interval='none');

  proc print data=mylib.PharmaFinancial noobs label split='*';
    where segment eq 1;  * Annual Geographic Segments;
    ... ;
  run; quit;
```

```
ods tagsets.ExcelXP options(sheet_interval='none');

proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 2;  * Annual Business Segments;
   ... ;
run; quit;

ods tagsets.ExcelXP options(sheet_interval='none');

proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 3;  * Quarterly Business Segments;
  ... ;
run; quit;

ods tagsets.ExcelXP close;
```

Our workbook now has 3 worksheets instead of 10, with the first and third worksheet containing 3 Excel tables, and the second worksheet containing 4 Excel tables.

ASSIGNING CUSTOM WORKSHEET NAMES

ODS generates a unique name for each worksheet, as required by Microsoft Excel. Figure 4 shows the worksheet names that result from running the code to create multiple tables within a worksheet (see above). There are, however, several tagset options that you can use to alter the names of the worksheets. The SHEET_NAME option can be used to explicitly set the worksheet name. The following code will cause the worksheet names to match those shown in Figure 1.

Figure 4. ODS ExcelXP-Generated Worksheet Names

```
*  Create a separate worksheet for each business segment;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name= Annual Geographic Segments);

*  PROC PRINT code #1;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name=Annual Business Segments);

*  PROC PRINT code #2;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name=Quarterly Business Segments);

*  PROC PRINT code #3;
```

USING SAS TITLES INSTEAD OF BY LINE TEXT

BY line text such as "Category=External Revenue" and "Category=Total Revenue" precedes each table in our workbook (see Figure 3). To control the text preceding each table we can use standard TITLE statements. By default, SAS titles and footnotes appear as Excel print headers and print footers, which are displayed only when the Excel document is printed. To include the titles in the worksheets, use the EMBEDDED_TITLES tagset option, and the SUPPRESS_BYLINES option can be used to prevent the BY line text from being included in the worksheet.

To insert the current value of the "Category" BY variable into the title, specify `#BYVAL(Category)` in the text of the TITLE statement. Refer to the SAS documentation for the TITLE statement for more information about #BYVAL ("TITLE Statement", SAS Institute Inc.).

```
title '#BYVAL(Category)';
footnote;

*  Set global options;

ods tagsets.ExcelXP options(embedded_titles='yes' suppress_bylines='yes');

*  Create a separate worksheet for each business segment;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name= Annual Geographic Segments);

*  PROC PRINT code #1;

...
```

Notice that a new ODS statement was added to specify the EMBEDDED_TITLES and SUPPRESS_BYLINES options. These options could have been added to an existing ODS statement, but a new statement increases readability and also separates global options from options that pertain to specific worksheets.

The code above prints only a title at the top of each page. To cause PROC PRINT to create a title for each BY group, add a PAGEBY statement to each block of PRINT procedure code:

```
proc print data=mylib.PharmaFinancial noobs label split='*';
  pageby category;  * Force a new title for each value of Category;
  ... ;
run; quit;
```

ADDING EXCEL SUBTOTALS TO THE SUMMARY LINES

Using the AUTO_SUBTOTALS option causes the ExcelXP tagset to insert an Excel SUBTOTAL function into the summary cell when the SUM statement is used with PROC PRINT. This option works best when a BY statement is not being used. When a BY statement is used, the SUBTOTAL function is correctly used in the first summary row, but not in subsequent summary rows that represent grand totals or multiple BY variables. The code to include automatic subtotals is shown below.

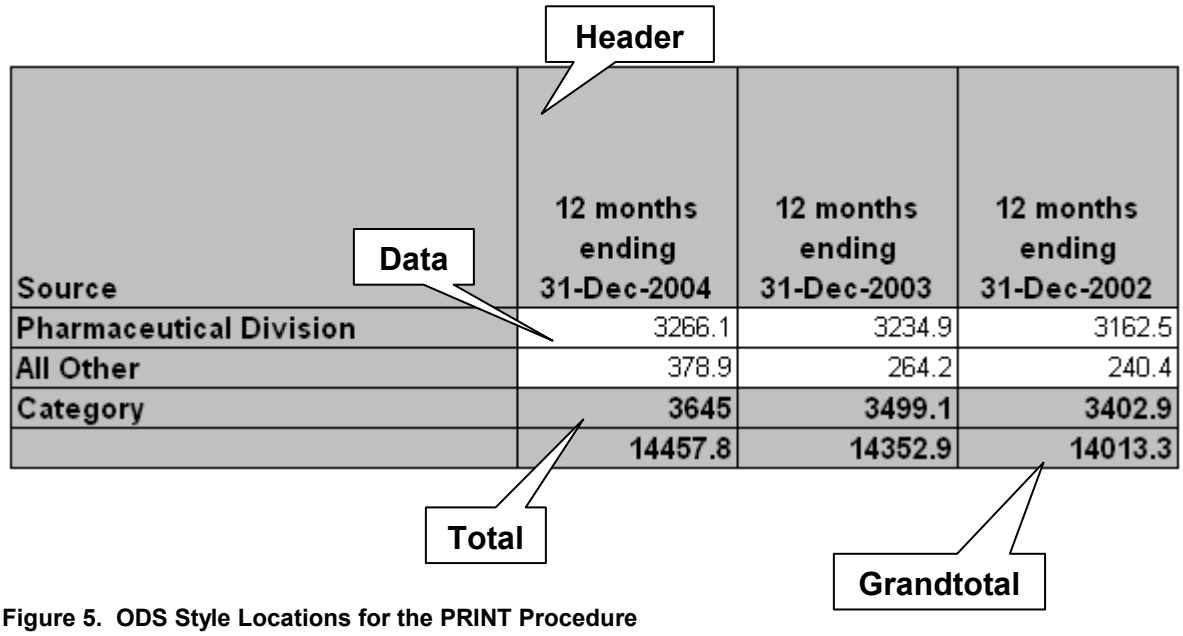
```
*  Set global options;

ods tagsets.ExcelXP options(embedded_titles='yes' suppress_bylines='yes'
                            auto_subtotals='yes');
```

You can hand-edit the Excel workbook to add extra SUBTOTAL functions as needed, or delete the excess summary rows if they are not needed. A method of using SAS macro code to run the PRINT procedure multiple times to avoid BY processing is discussed in the appendix in the section "Advanced Topic: Macro Solution to Avoid Extra Summary Lines".

CHANGING THE APPEARANCE USING ODS STYLE OVERRIDES AND EXCEL FORMATS

ODS style elements are applied to different parts of SAS output. These parts of the output are referred to as *locations*. Figure 5 shows the ODS locations pertinent to the PROC PRINT output.



Source	12 months ending 31-Dec-2004	12 months ending 31-Dec-2003	12 months ending 31-Dec-2002
Pharmaceutical Division	3266.1	3234.9	3162.5
All Other	378.9	264.2	240.4
Category	3645	3499.1	3402.9
	14457.8	14352.9	14013.3

Figure 5. ODS Style Locations for the PRINT Procedure

The "Header" location applies to all cells in the first row of the output. As you can see from Table 1, the "header" style element controls the appearance of this part of the output. Similarly, the "header" style element controls the appearance of the Grandtotal and Total locations. Finally, the appearance of numeric values in the "Data" location is controlled by the "data" style element.

Location Name	Default Style Element
Data	data (except for ID statement) rowheader (for ID statement)
Grandtotal	header
Header	header
Total	header

Table 1. ODS Style Locations and Their Default Style Elements

The last change we need to make to our program is to add ODS style overrides to change the appearance of the text and numbers in the locations listed in Table 1. The general syntax that we will use is:

```
statement-name variable-name / style(location-name)=style-element-name
```

Recall that the XLsansPrinter style contains new style elements that we can use:

```
/* Controls the appearance of the ID "header" cell */

style header_id from header /
  font_weight = light;

/* Assign an Excel format - used with "data" cells */

style data_comma_1 from data /
  tagattr = 'format:#,###.0';
```

```
/* Assign an Excel format - used with "total" and "grand total" cells */

style header_comma_1 from header /
  tagattr = 'format:#,###.0';
```

First, change the heading of the "Source" column to "Millions of USD" using the LABEL statement:

```
proc print data=mylib.PharmaFinancial noobs label split='*';
  where ...;
  by ...;
  id source;
  var val2004-val2002;
  sum val2002-val2004;
  label source = 'Millions of USD';
run; quit;
```

To change the appearance of the text "Millions of USD", apply a style override to the "Header" location:

```
proc print data=mylib.PharmaFinancial noobs label split='*';
  where ...;
  by ...;
  id source / style(header) = header_id;
  var val2004-val2002;
  sum val2002-val2004;
  label source = 'Millions of USD';
run; quit;
```

The style elements "data_comma_1" and "header_comma_1" use an Excel format to display numeric values using a comma for a thousands separator, and 1 decimal place. They are applied as style overrides:

```
proc print data=mylib.PharmaFinancial noobs label split='*';
  where ...;
  by ...;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;
```

THE FINAL SAS CODE

With all of these changes in place, the appearance of the PRINT procedure output matches Figure 1. The final SAS code should appear as follows:

```
ods listing close;
ods tagsets.ExcelXP path='output-directory' file='PharmaFinancial.xml'
  style=XLsansPrinter;

  *  Use the value of the BY group Category for the title;

  title '#BYVAL(Category)';
  footnote;

  *  Set global options;

  ods tagsets.ExcelXP options(embedded_titles='yes' suppress_bylines='yes'
                              auto_subtotals='yes');
```

```
*  Create a separate worksheet for each business segment;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name='Annual Geographic Segments');

proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 1;  * Annual Geographic Segments;
  by category;
  pageby category;     * Force a new title for each value of Category;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name='Annual Business Segments');

proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 2;  * Annual Business Segments;
  by category;
  pageby category;     * Force a new title for each value of Category;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;

ods tagsets.ExcelXP options(sheet_interval='none'
                            sheet_name='Quarterly Business Segments');

proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 3;  * Quarterly Business Segments;
  by category;
  pageby category;     * Force a new title for each value of Category;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;

ods tagsets.ExcelXP close;
```

SAS SERVER TECHNOLOGY

If you have licensed SAS/IntrNet software, you can incorporate dynamically generated SAS output into Excel by using the Application Dispatcher. You can also perform similar tasks with the Stored Process Server, which was new for SAS 9.1.

The Application Dispatcher and the Stored Process Server enable you to execute SAS programs from a Web browser or any other client that can open an HTTP connection to either of these SAS servers (which can run on *any* platform where SAS is licensed). The SAS programs that you execute from the browser can contain any combination of DATA step, PROC, macro, or SCL code. Thus, all the code that has been shown up to this point can be executed by either the Application Dispatcher or the Stored Process Server.

Program execution is typically initiated by accessing a URL that points to the SAS server program. Any parameters are passed to the program as name/value pairs in the URL. The SAS server takes these name/value pairs and

constructs SAS macro variables that are available to the SAS program.

Figure 6 shows a Web page that can deliver SAS output directly to Excel, using the Web browser as the client.

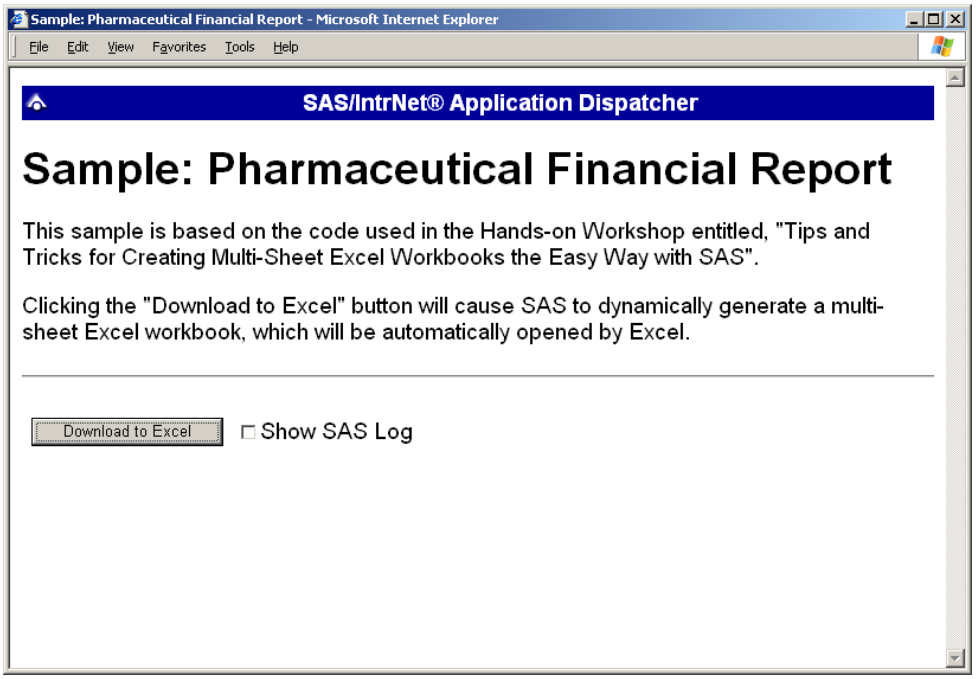


Figure 6. Web Page to Drive a SAS/IntrNet Application

Clicking the **`Download to Excel`** would result in the execution of a slightly modified version of the PRINT procedure code that we have been working with. The modifications are as follows:

```
%let RV=%sysfunc(appsrv_header(Content-type,application/vnd.ms-excel));
%let RV=%sysfunc(appsrv_header(Content-disposition,attachment; filename=
"PharmaFinancial.xls"));  * Ignore line wrapping;

options center;

ods listing close;

ods tagsets.ExcelXP file=_webout style=XLsansPrinter;
  *  Remainder of "final" SAS code;
ods tagsets.ExcelXP close;
```

The first APPSRV_HEADER function sets a special MIME header that causes the SAS output to be opened by Excel, instead of being rendered by the Web browser. This statement is required.

The second APPSRV_HEADER function sets a special MIME header that populates the file name field in the "File Download" dialog box. As you can see from Figure 7, the file name appears as "PharmaFinancial.xls". This header may cause problems with some versions of Excel, so be sure to test your applications before deploying them in a production environment. This statement is optional.

The reserved fileref _WEBOUT is automatically assigned by the SAS server, and is always used to direct output from the SAS server to the client. Modify your existing ODS statement to direct the output to this FILEREF, instead of an external disk file.

When you click the download button and are presented with the dialog box, you can click **Open** to immediately open your SAS output using Excel, or **Save** to save a copy for later use.

This is just one example of how you can dynamically deliver SAS output to Excel. For more detailed information and other examples, refer to the SAS/IntrNet Application Dispatcher or Stored Process documentation or both, as well as this author's earlier papers (DelGobbo, 2002, 2003, 2004).

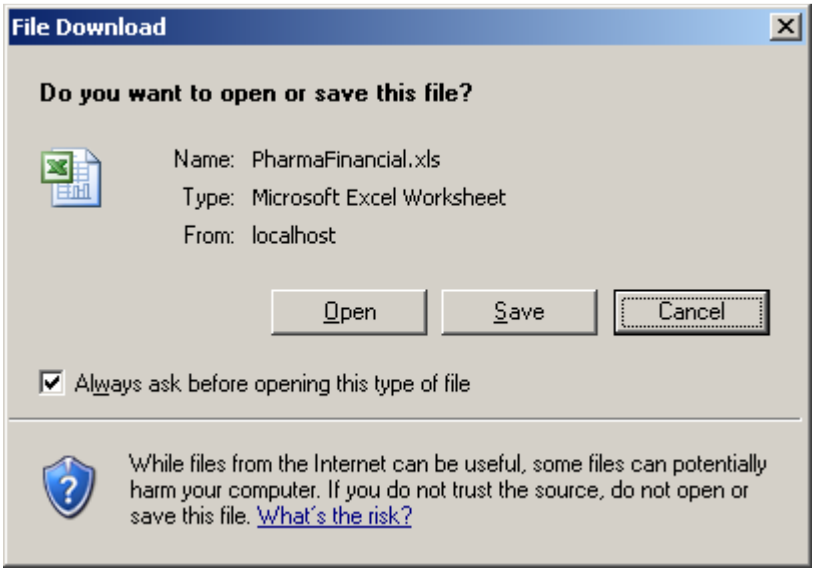


Figure 7. File Download Dialog Box

CONCLUSION

The SAS 9.1 ExcelXP ODS tagset provides an easy way to export your SAS data to Excel workbooks that contain multiple worksheets. By using ODS styles, style overrides, and tagset options, you can customize the output to achieve your design goals. The tagset complies with the Microsoft XML Spreadsheet Specification.

APPENDIX

CODE FOR CREATING THE XLSANSPRINTER STYLE

```
proc template;
  define style styles.XLsansPrinter;
    parent = styles.sansPrinter;

    /* Controls the appearance of the ID "header" cell */

    style header_id from header /
      font_weight = light;

    /* Assign an Excel format - used with "data" cells */

    style data_comma_1 from data /
      tagattr = 'format:#,###.0';

    /* Assign an Excel format - used with "total" and "grand total" cells */

    style header_comma_1 from header /
      tagattr = 'format:#,###.0';

  end;
run; quit;
```

ADVANCED TOPIC: MACRO SOLUTION TO AVOID EXTRA SUMMARY LINES

The AUTO_SUBTOTALS tagset option is used to insert an Excel SUBTOTAL function into summary cells created by the PRINT procedure. However, the SUBTOTAL function is inserted only into the cells corresponding to the first summary line, not the grand total summary line, or additional summary lines resulting from multiple BY groups. This behavior may be changed in a future version of the tagset, but currently you must hand-edit the Excel workbook to either add the SUBTOTAL function to the additional summary lines, or to delete the lines.

If you do not want the additional summary lines, you can run PROC PRINT repeatedly without a BY statement to create the 10 Excel tables.

```
title 'External Revenue';
proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 1 and category eq 1;
  *  No longer need BY or PAGEBY;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;

title 'Total Revenue';
proc print data=mylib.PharmaFinancial noobs label split='*';
  where segment eq 1 and category eq 2;
  *  No longer need BY or PAGEBY;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;

*  Repeat similar code 8 more times;
```

Rather than duplicating the code for each BY group, a better solution is to create a SAS macro that uses a DO loop and the FIRSTOBS and OBS values corresponding to all the unique values of the original BY groups. Then call the macro from within the body of the main program:

```
ods tagsets.ExcelXP path='output-directory' file='PharmaFinancial.xml'
  style=XLsansPrinter;

  *  Titles and footnotes handled by macro code;

  ods tagsets.ExcelXP options(embedded_titles='yes'
                              auto_subtotals='yes');

  *  Create a separate worksheet for each business segment;

  ods tagsets.ExcelXP options(sheet_interval='none'
                              sheet_name='Annual Geographic Segments');

  *  Call the macro to create the first 3 tables;

  %PROC_PRINT(data=mylib.PharmaFinancial,
              where=segment eq 1,
              titlevar=category);
```

```
  ods tagsets.ExcelXP options(sheet_interval='none'
                              sheet_name='Annual Business Segments');

  *  Call the macro to create the next 4 tables;

  %PROC_PRINT(data=mylib.PharmaFinancial,
              where=segment eq 2,
              titlevar=category);

  ods tagsets.ExcelXP options(sheet_interval='none'
                              sheet_name='Quaterly Business Segments');

  *  Call the macro to create the last 3 tables;

  %PROC_PRINT(data=mylib.PharmaFinancial,
              where=segment eq 3,
              titlevar=category);

ods tagsets.ExcelXP close;
```

However, to use this macro implementation, you must first create the SAS table that contains the values of FIRSTOBS and OBS:

```
data table_info;
set mylib.PharmaFinancial;
by segment category;
length firstobs obs 8;
retain _all_; * General case, though not all will be meaningful;
if (first.category) then firstobs = _N_;
*  Keep 1 record per BY group with all the info we need for later;
if (last.category) then do;
  obs = _N_;
  output;
end;
run;
```

Figure 8 shows the resulting table, which contains one record for each Excel table that will be created.

VIEWTABLE: WORK.TABLE_INFO

	Segment	Category	Firstobs	Obs
1	Annual Geographic Segments	External Revenue	1	4
2	Annual Geographic Segments	Total Revenue	5	8
3	Annual Geographic Segments	Long-Lived Assets	9	12
4	Annual Business Segments	External Revenue	13	14
5	Annual Business Segments	Total Revenue	15	16
6	Annual Business Segments	Depreciation	17	18
7	Annual Business Segments	Operating Income/Loss	19	20
8	Quarterly Business Segments	External Revenue	21	22
9	Quarterly Business Segments	Total Revenue	23	24
10	Quarterly Business Segments	Operating Income/Loss	25	26

Figure 8. Partial View of the BY Group Information Displayed Using the SAS VIEWTABLE Application

Finally, create the PROC_PRINT macro to read the FIRSTOBS and OBS values from the table, and execute the PRINT procedure using these values. A portion of the macro code is shown below.

```
proc print data=&DATA(firstobs=&FIRSTOBS obs=&OBS) noobs label split='*';
  *  No longer need BY or PAGEBY;
  id source / style(header) = header_id;
  var val2004-val2002 / style(data)=data_comma_1;
  sum val2002-val2004 / style(total)=header_comma_1
                        style(grandtotal)=header_comma_1;
  label source = 'Millions of USD';
run; quit;
```

The complete code for the macro is in the download package on the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Tips and Tricks for Creating Multi-Sheet Excel Workbooks the Easy Way with SAS". The download package contains a file named "MakeXML-Macro.sas".

REFERENCES

DelGobbo, V. 2002. "Techniques for SAS® Enabling Microsoft Office in a Cross-Platform Environment". *Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference*, 27. CD-ROM. Paper 174. Available http://www2.sas.com/proceedings/sugi27/p174-27.pdf.

DelGobbo, V. 2003. "A Beginner's Guide to Incorporating SAS® Output in Microsoft Office Applications". P*roceedings of the Twenty-Eighth Annual SAS Users Group International Conference*, 28. CD-ROM. Paper 52. Available http://www2.sas.com/proceedings/sugi28/052-28.pdf.

DelGobbo, V. 2004. "From SAS® to Excel via XML". Available http://support.sas.com/saspresents.

SAS Institute Inc. "ODS MARKUP Resources". Available http://support.sas.com/rnd/base/topics/odsmarkup/.

SAS Institute Inc. "The TEMPLATE Procedure: Creating a Style Definition". Available http://support.sas.com/onlinedoc/913/docMainpage.jsp?_topic=odsug.hlp/a002565239.htm.

SAS Institute Inc. "TITLE Statement". Available http://support.sas.com/onlinedoc/913/docMainpage.jsp?_topic=lrdict.hlp/a000220968.htm

ACKNOWLEDGMENTS

The author would like to thank Chris Barrett of SAS Institute Inc. for his valuable contributions to this paper.

RECOMMENDED READING

Microsoft Corporation. "XML Spreadsheet Reference". Available http://msdn2.microsoft.com/en-us/library/aa140066(office.10).aspx.

SAS Institute Inc. "Application Dispatcher". Available http://support.sas.com/onlinedoc/913/docMainpage.jsp?_topic=dispatch.hlp/main_contents.htm

SAS Institute Inc. "The PRINT Procedure". Available http://support.sas.com/onlinedoc/913/docMainpage.jsp?_topic=proc.hlp/a000057825.htm.

SAS Institute Inc. "SAS Data Set Options". Available http://support.sas.com/onlinedoc/913/docMainpage.jsp?_topic=lrdict.hlp/a002295655.htm

SAS Institute Inc. "SAS Stored Processes". Available http://support.sas.com/rnd/itech/doc9/dev_guide/stprocess/index.html.

CONTACT INFORMATION

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

Vincent DelGobbo
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513

sasvcd@SAS.com

If your registered in-house or local SAS users group would like to request this presentation as your annual SAS presentation (as a seminar, talk, or workshop) at an upcoming meeting, please submit an online User Group Request Form (support.sas.com/usergroups/namerica/lug-form.html) at least eight weeks in advance.