Paper 115-31

Creating *AND* Importing Multi-Sheet Excel Workbooks the Easy Way with SAS®

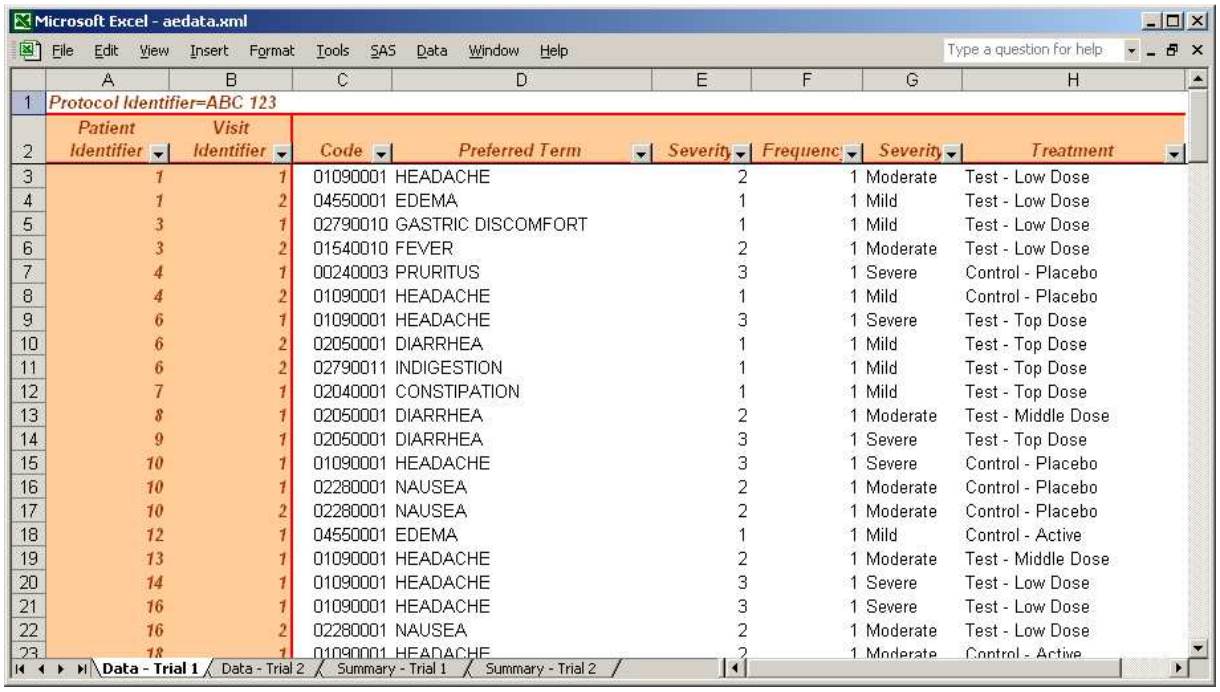
Vincent DelGobbo, SAS Institute Inc., Cary, NC

ABSTRACT

Transferring 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 new XML support in Base SAS 9.1 software to move data between SAS and Microsoft Excel (versions 2002 and later). You will learn step-by-step techniques to quickly and easily create multi-sheet Excel workbooks containing your SAS output. You'll also learn about converting your own multi-sheet Excel workbooks to SAS tables. The techniques described here can be used regardless of the platform on which SAS software is installed, including a mainframe! The use of SAS server technology is also discussed.

INTRODUCTION

This paper provides you with step-by-step instructions for using Base SAS 9.1 to create the Excel workbook shown in Figures 1 and 2 below.



Microsoft Excel - aedata.xml

File Edit View Insert Format Tools SAS Data Window Help — Type a question for help

	A	B	C	D	E	F	G	H
1	Protocol Identifier=ABC 123							
2	Patient Identifier	Visit Identifier	Code	Preferred Term	Severity	Frequenc	Severity	Treatment
3	1	1	01090001	HEADACHE	2	1	Moderate	Test - Low Dose
4	1	2	04550001	EDEMA	1	1	Mild	Test - Low Dose
5	3	1	02790010	GASTRIC DISCOMFORT	1	1	Mild	Test - Low Dose
6	3	2	01540010	FEVER	2	1	Moderate	Test - Low Dose
7	4	1	00240003	PRURITUS	3	1	Severe	Control - Placebo
8	4	2	01090001	HEADACHE	1	1	Mild	Control - Placebo
9	6	1	01090001	HEADACHE	3	1	Severe	Test - Top Dose
10	6	2	02050001	DIARRHEA	1	1	Mild	Test - Top Dose
11	6	2	02790011	INDIGESTION	1	1	Mild	Test - Top Dose
12	7	1	02040001	CONSTIPATION	1	1	Mild	Test - Top Dose
13	8	1	02050001	DIARRHEA	2	1	Moderate	Test - Middle Dose
14	9	1	02050001	DIARRHEA	3	1	Severe	Test - Top Dose
15	10	1	01090001	HEADACHE	3	1	Severe	Control - Placebo
16	10	1	02280001	NAUSEA	2	1	Moderate	Control - Placebo
17	10	2	02280001	NAUSEA	2	1	Moderate	Control - Placebo
18	12	1	04550001	EDEMA	1	1	Mild	Control - Active
19	13	1	01090001	HEADACHE	2	1	Moderate	Test - Middle Dose
20	14	1	01090001	HEADACHE	3	1	Severe	Test - Low Dose
21	16	1	01090001	HEADACHE	3	1	Severe	Test - Low Dose
22	16	2	02280001	NAUSEA	2	1	Moderate	Test - Low Dose
23	18	1	01090001	HEADACHE	2	1	Moderate	Control - Active

Data - Trial 1 | Data - Trial 2 | Summary - Trial 1 | Summary - Trial 2

Figure 1. Sample Multi-Sheet Excel Workbook (PRINT Procedure Output)

Figures 1 and 2 show two different worksheets of the same Excel workbook. The data describes adverse events for two different clinical trials of a fictitious drug. The worksheets named "Data – Trial 1" and "Data – Trial 2" represent the raw data, and were produced by running the PRINT procedure. The remaining two worksheets contain summarized data created by the TABULATE procedure (see Figure 2).

The columns "Patient Identifier" and "Visit Identifier" contain numeric data that represents IDs that correspond to the patient and visit number. The "Code" column is an identifier that corresponds to the adverse event that was experienced (the "Preferred Term"). Note that the data in the "Code" column contains leading zeroes. The other columns in this data should be self-explanatory.

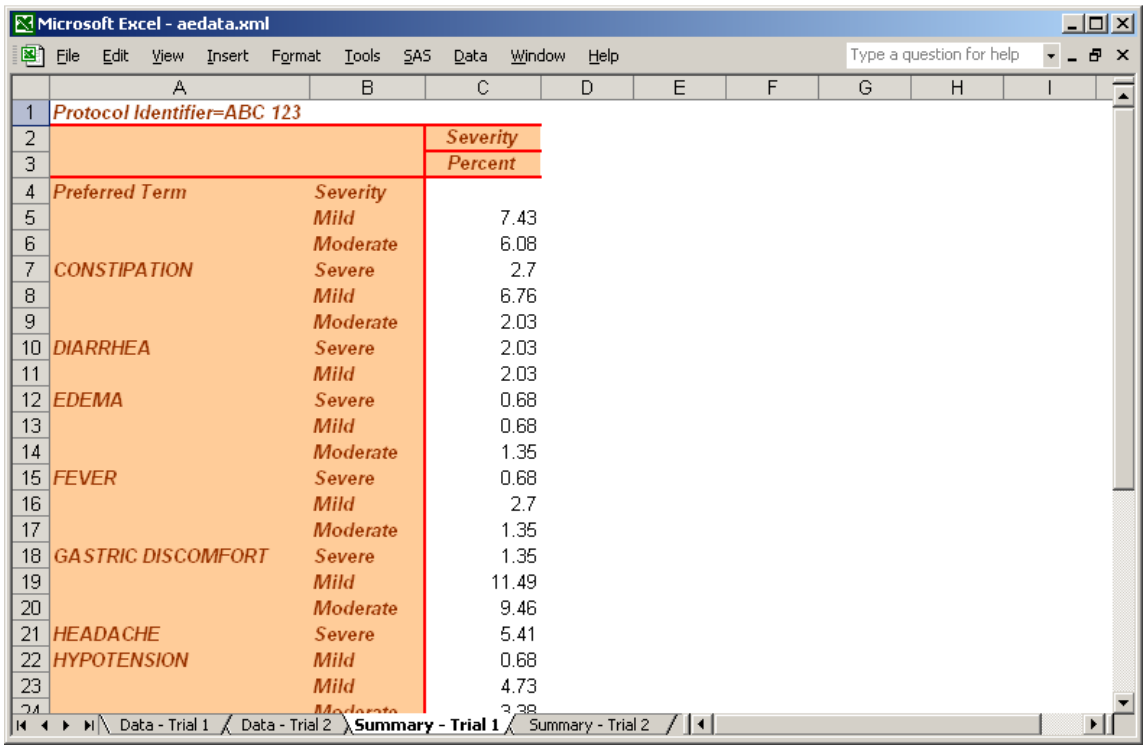


Figure 2. Sample Multi-Sheet Excel Workbook (TABULATE Procedure Output)

The remainder of this paper will guide you through the steps used to create the Excel workbook shown in Figures 1 and 2. Additionally, you will learn a technique to import your own Excel workbooks into SAS tables. Both techniques use Base SAS 9.1 software. You can download a copy of the sample data and code used in this paper from the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Creating *AND* Importing Multi-Sheet Excel Workbooks the Easy Way with SAS".

The code in this paper was tested using SAS 9.1.3 Service Pack 3 and Microsoft Excel 2002 SP3 software.

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 des*tination* controls the type of output that's 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 Figures 1 and 2 was created using the ExcelXP ODS tagset and the SUGI31 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.

The ODS statements to generate XML compatible with versions of Microsoft Excel 2002 and later follow this general form:

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

The first ODS statement (❶) turns off the standard "line printer" ODS destination. We only want to generate XML output for use with Excel.

The second ODS statement (❷) generates the XML output and 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), a network-accessible disk, or a Web server, 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.

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's stored on a Web server, follow these steps:

1. In Excel, select **`File ➨ Open`**.
2. In the "File name" field, specify the full URL to the file you want to open. For example, **`http://Web-server/directory/aedata.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, except in step 2 you should either navigate to the desired file or type the filename and path 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 personal copy of the file in Excel format, select **`File ➨ Save As`** and then, from the "Save as type" drop-down, list choose **`Microsoft Excel Workbook (*.xls)`**.

SETTING UP THE ODS ENVIRONMENT

Our sample code uses a user-defined style named SUGI31 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's more efficient to create them once, and 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". Because ODS searches the PATH in the order given and because the access mode for `mylib.tmplmst` is "update", PROC TEMPLATE will store tagsets and styles in a file named `tmplmst.sas7bitm` in the directory that's associated with the MYLIB library.

You can think of the ODS PATH statement as functioning like the SAS FMTSEARCH option. You include the LIBNAME and ODS PATH statements in each SAS program that needs to use the tagsets and styles that you create.

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 "Creating *AND* Importing 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 only have to perform this step 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 a number of different 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, a copy of this information is available in the download package for this paper.

THE SUGI31 STYLE

The workbook shown in Figures 1 and 2 uses a color scheme that resembles the SUGI 31 conference materials and Web site. Both the colors and fonts are controlled by the ODS style named SUGI31. The complete code for creating this style can be found in the appendix in the section "Code for Creating the SUGI31 Style".

You simply run this code to create the SUGI31 style. You should have to perform this step only once. The SUGI31 style will be stored in the directory corresponding to the MYLIB library. All of your future SAS programs can access the style by specifying the correct LIBNAME and ODS PATH statements (see "Setting up the ODS Environment" above).

Because an in-depth discussion of creating and using ODS styles is beyond the scope of this paper, see the chapters about the TEMPLATE procedure in the ODS documentation (SAS Institute Inc. 1999, 2004).

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. Our sample code executes both the PRINT and TABULATE procedures twice. Each procedure execution results in the creation of one piece of tabular output being stored in its own worksheet.

SAS CODE TO CREATE THE EXECL WORKBOOK

Below is a listing of the SAS code used to create the Excel workbook:

```
options center;
title; footnote;

ods listing close;
ods tagsets.ExcelXP path='output-directory' file='aedata.xml' style=SUGI31;

  proc print data=sample.aedata noobs label;
    by protocol;
    where protocol eq 'ABC 123';
```

```
    id patient;
    id visit / style=rowheader_r_border style(header)=header_r_border;
    var aecode;
    var aetext--treat;
  run; quit;

  proc print data=sample.aedata noobs label;
    by protocol;
    where protocol eq 'XYZ 987';
    id patient;
    id visit / style=rowheader_r_border style(header)=header_r_border;
    var aecode;
    var aetext--treat;
  run; quit;

  proc tabulate data=sample.aedata;
    by protocol;
    where protocol eq 'ABC 123';
    var aesev;
    class aetext;
    class aesevc / style=rowheader_r_border;
    classlev aesevc / style=rowheader_r_border;
    table aetext*aesevc,aesev*pctn / box={label='' style=header_box};
    keyword all pctn;
    keylabel pctn='Percent';
  run; quit;

  proc tabulate data=sample.aedata;
    by protocol;
    where protocol eq 'XYZ 987';
    var aesev;
    class aetext;
    class aesevc / style=rowheader_r_border;
    classlev aesevc / style=rowheader_r_border;
    table aetext*aesevc,aesev*pctn / box={label='' style=header_box};
    keyword all pctn;
    keylabel pctn='Percent';
  run; quit;
ods tagsets.ExcelXP close;
```

As you can see from the fourth line, the ExcelXP tagset is used to generate the output, and the SUGI31 style is used to control the appearance aspects of the output. The PRINT procedure is run twice: once to display the data corresponding to the clinical trial identified as "ABC 123" and once for trial "XYZ 987". The TABULATE procedure is run once for each trial. Executing this code will result in the output of each procedure run being displayed in a separate worksheet. The SAS table "aedata" is included in the download package for this paper.

Notice that "style overrides" are being used in some cases, as evidenced by the use of the STYLE attribute. Although the SUGI31 style defines a particular appearance for your SAS output, you can change the attributes that are defined in the style at run time by using a style override.

Style overrides are supported by the PRINT, TABULATE, and REPORT procedures. Style overrides are best used for changing a small number of features because, if overused, they can result in excessive output file size. Style overrides can be specified in several ways:

❶ `style=`*`class-name`*
❷ `style=[`*`attribute-name1=value1 attribute-name2=value2 ...`*`]`
❸ `style=`*`class-name`* `[`*`attribute-name1=value1 attribute-name2=value2 ...`*`]`

The *`class-name`* is the name of a predefined style class you want to use and *`attribute-name`* is the name of an individual style attribute that you want to override.

The first format (❶) is the most efficient, as it has the smallest impact on output file size. You simply specify the name of the style class you want ODS to use. For example, in the code above, the style class named "header_box" controls the appearance of the "box cell" in the TABULATE procedure output. Refer to the code in the appendix to see the full definition of the "header_box" and other classes that are part of the SUGI31 style.

A less-efficient way of overriding the style for the box cell would be to use individual style attributes (❷) instead of a class name:

```
style=[just=center bordertopwidth=3 borderbottomwidth=3 borderleftwidth=0
       borderrightwidth=3]
```

Although useful, style overrides of this type should be avoided when possible because the additional XML data output can make an XML file dramatically larger than if it had been generated without the style overrides. The XML files generated by the ExcelXP tagset are already large in order to comply with the Microsoft XML Spreadsheet Specification ("XML Spreadsheet Reference", Microsoft Corp.) Inefficient use of style overrides will result in files that need more disk space and take longer to open in Excel.

Figures 3 and 4 show the result of executing the SAS code above and opening the file "aedata.xml" using Excel.

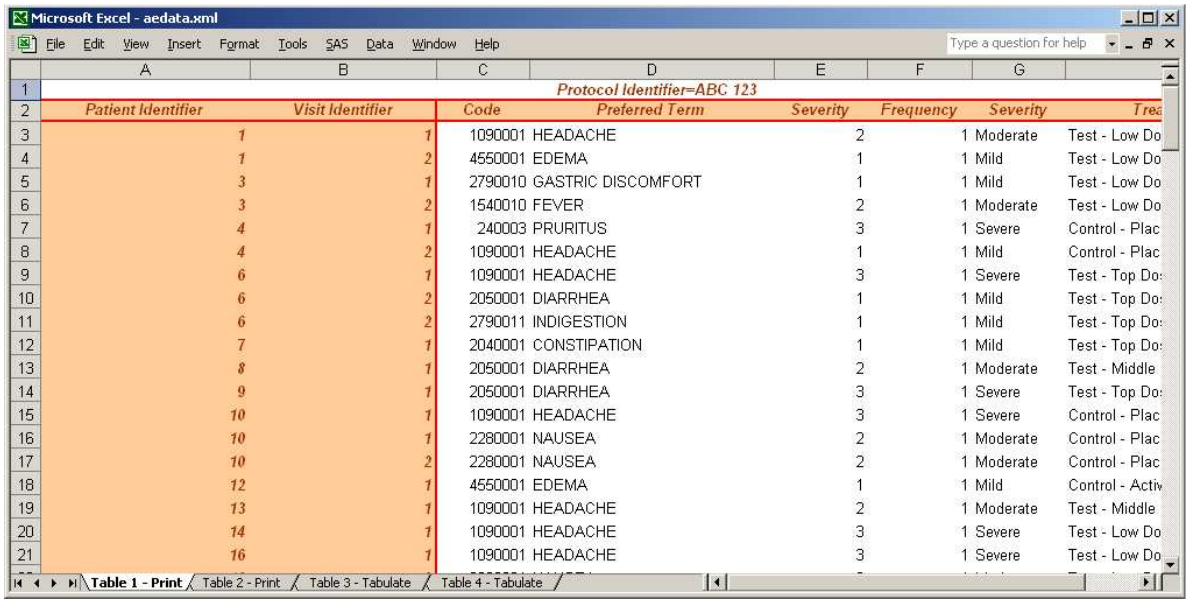


Figure 3. ODS ExcelXP-Generated Workbook (PRINT Procedure Output)

You will notice that the output shown in Figure 3 does not match the output shown in Figure 1. The problems exhibited in Figure 3 are:

1. The leading zeroes in the column labeled "Code" are missing.
2. Default worksheet names were used.
3. Some columns are wider than necessary.
4. The BY group label is centered, instead of being left-justified.
5. AutoFilters are missing from all columns.
6. If you were to scroll downward in this worksheet, you will "loose" the column headers because by default, they are not "frozen".

The second worksheet ("Table 2 – Print"), also created by the PRINT procedure, exhibits the same problems.

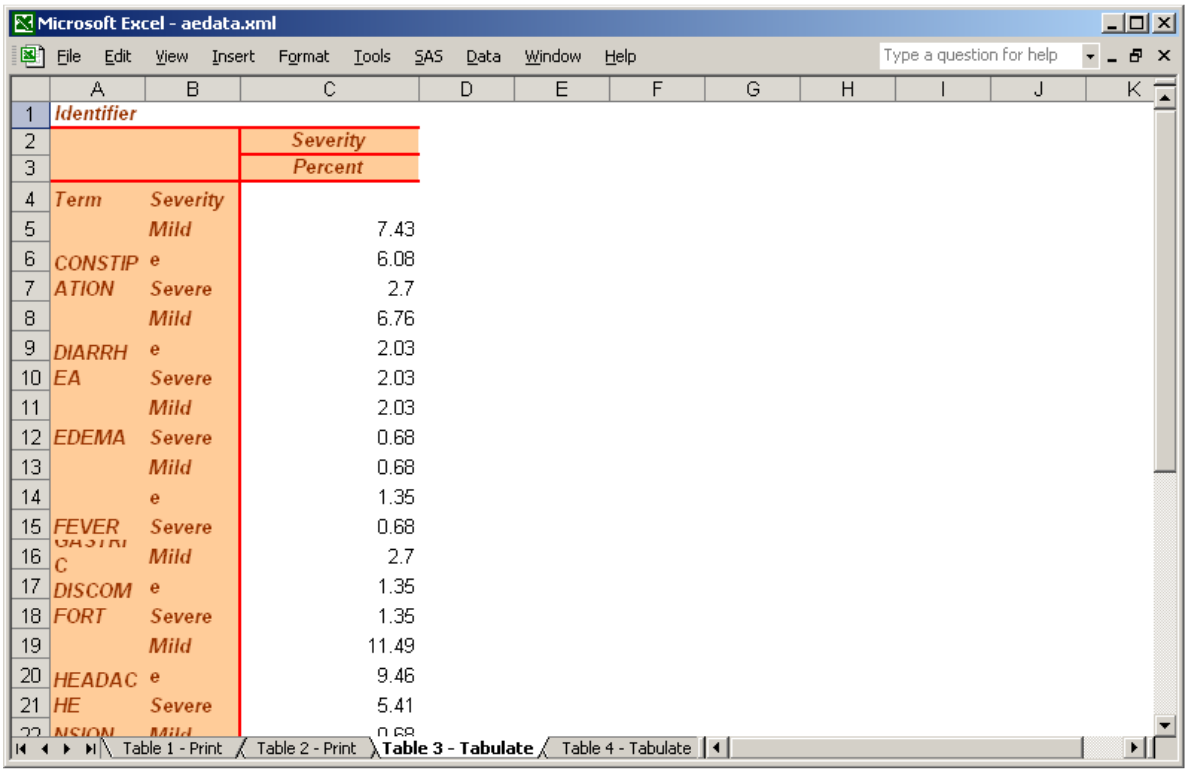


Figure 4. ODS ExcelXP-Generated Workbook (TABULATE Procedure Output)

Similarly, the output from Figure 4 does not match what is shown in Figure 2. The problems are:

1. Default worksheet names were used.
2. The BY group label is truncated.
3. Some columns are not as wide as necessary, resulting in the loss of data.

The fourth worksheet ("Table 4 – Tabulate"), also created by the TABULATE procedure, exhibits the same problems.

We will now make modifications to the SAS code shown above to correct these problems.

GETTING THE LEADING ZEROES BACK IN THE PRINT PROCEDURE OUTPUT

When Excel opened the file "aedata.xml", it examined the data in the "Code" column and determined that the Excel "General" format should be applied. This results in the loss of the leading zeroes. To force Excel to use a different format, use the ODS TAGATTR style element to specify the Excel format you want to use.

If you examine the code for the SUGI31 style in the appendix, you will notice that we defined a style class named "data_z8":

```
style data_z8 from data /
  tagattr='format:00000000';
```

The "data_z8" style has the same attributes as the "data" style, except that it explicitly specifies that Excel use its "00000000" format for the data. This Excel format is comparable to the SAS Z8 format. To include the leading zeroes, you must modify the PROC PRINT code shown earlier, and specify the "data_z8" style for the column that contains the numeric adverse event data:

```
var aecode / style(data)=data_z8;
```

This style override instructs ODS to use the style named "data_z8" for data cells.

ASSIGNING CUSTOM WORKSHEET NAMES

As mentioned earlier, the ExcelXP tagset supports a number of options that control both the appearance and functionality of the Excel workbook. The option that controls the worksheet name is SHEET_NAME. You simply specify the value you would like to use for the name of the worksheet.

In our sample, we want to assign a custom worksheet name for each sheet. Therefore, we must specify the SHEET_NAME option immediately before each worksheet is created. The listing below illustrates how to modify the SAS code to name each worksheet:

```
options center;
title; footnote;

ods listing close;
ods tagsets.ExcelXP path='output-directory' file='aedata.xml' style=SUGI31;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 1');

  * Proc PRINT #1 here w/ style override;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 2');

  * Proc PRINT #2 here w/ style override;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 1');

  * Proc TABULATE #1 here;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 2');

  * Proc TABULATE #2 here;

ods tagsets.ExcelXP close;
```

Each worksheet will now have the names shown in Figures 1 and 2, instead of the default names.

ADJUSTING COLUMN WIDTHS FOR THE PRINT PROCEDURE OUTPUT

The ExcelXP tagset estimates the width of columns based on several factors. In general, the width of the column is determined by the length of the variable and the length of the variable label. The storage length for a given variable, assigned with the SAS LENGTH statement or attribute, often has the largest effect on the column width calculated by the tagset. In our case, the columns labeled "Patient Identifier" and "Visit Identifier" are wide because of the lengths of the column labels. You can get narrower columns by reassigning the column labels using a SPLIT character:

```
proc print data=sample.aedata noobs label split='*';
  by protocol;
  where protocol eq 'ABC 123';
  id patient;
  id visit / style=rowheader_r_border style(header)=header_r_border;
  var aecode / style(data)=data_z8;
  var aetext--treat;
  label protocol  = 'Protocol Identifier'
        patient   = 'Patient*Identifier'
        visit     = 'Visit*Identifier'
        aecode    = '*Code'
        aetext    = '*Preferred Term'
        aesev     = '*Severity'
        frequency = '*Frequency'
        aesevc    = '*Severity'
        treat     = '*Treatment';
run; quit;
```

Similarly, you would use the LABEL statement and the SPLIT option with the PRINT procedure code that prints the information for trial "XYZ 987".

Under some circumstances you may want to further adjust the width of the columns. One way to accomplish this is to use the WIDTH_FUDGE tagset option. The WIDTH_FUDGE option affects all columns, and is used to set the "fudge factor" that is used when the column width is calculated by the tagset. The default value for the “fudge factor” is 0.75. Setting the fudge factor to something greater than 0.75 results in wider columns, while specifying a value less than 0.75 results in columns that are narrower.

To adjust the column widths in the PRINT procedure output, simply add the WIDTH_FUDGE option to the existing tagset option list. Note that once you set a tagset option, it remains in effect until it is explicitly changed. Since we don't want to use WIDTH_FUDGE for the TABULATE procedure output, we need to specify the default value prior to the TABULATE procedure code. Thus, your SAS code should look as follows:

```
options center;
title; footnote;

ods listing close;
ods tagsets.ExcelXP path='output-directory' file='aedata.xml' style=SUGI31;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 1' width_fudge='0.7');

  * Proc PRINT #1 here w/ style override and new labels;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 2' width_fudge='0.7');

  * Proc PRINT #2 here w/ style override and new labels;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 1' width_fudge='0.75');

  * Proc TABULATE #1 here;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 2' width_fudge='0.75');

  * Proc TABULATE #2 here;

ods tagsets.ExcelXP close;
```

When run with the WIDTH_FUDGE additions, the code above should produce column headings and widths for the PRINT procedure output that match the columns shown in Figure 1.

There are times when these techniques fail to produce satisfactory column widths. One instance is when a variable is defined with a large value for the LENGTH statement or attribute, and the variable values do not contain much data. In this case, try using the ABSOLUTE_COLUMN_WIDTH tagset option, discussed in the section "Adjusting Column Widths for the TABULATE Procedure Output" below.

CHANGING THE BY GROUP LABEL PLACEMENT

The placement of the BY group labels is controlled by the CENTER/NOCENTER SAS system option. In many installations, this option is set to CENTER, resulting in the label placement you see in Figures 3 and 4. While the output shown in Figure 3 looks fine, you will notice that the BY group label associated with the TABULATE output (Figure 4) is truncated. Such truncation can happen any time the value of the label is long and the SAS output is narrow.

An easy way to correct this problem is to explicitly specify the NOCENTER option. In our sample code, change `options center` to `options nocenter`.

ADJUSTING COLUMN WIDTHS FOR THE TABULATE PROCEDURE OUTPUT

Correcting the column widths in the TABULATE procedure output requires a different approach because, in SAS 9.1, some procedures, including TABULATE, do not report enough information for the ExcelXP tagset to compute

accurate column widths. This lack of information means that correcting the column widths in the TABULATE procedure output could be a trial-and-error process.

To correct the column widths, use the ABSOLUTE_COLUMN_WIDTH tagset option. This option takes as input a comma-separated list of column widths, measured in characters. The tagset uses these column widths to calculate the approximate column width in points using this formula:

column width = (number of characters) * MAX(header point size, data point size) * (width fudge)

The "column width" is the computed width of the column in points, "number of characters" is the value specified in the ABSOLUTE_COLUMN_WIDTH tagset option, "header point size" and "data point size" are the point size of the header and data cell fonts specified in the ODS style definition, and "width fudge" is the value of the WIDTH_FUDGE tagset option.

By examining the TABULATE output, we find that the largest data value in the first column is 21 characters ("ESOPHAGEAL IRRITATION"), and the largest data values in the second and third columns are 8 characters each ("Moderate" and "Severity", respectively). Thus, the value specified for ABSOLUTE_COLUMN_WIDTH would be "21, 8, 8". You simply add this option to the existing list of options for your TABULATE output:

```
ods tagsets.ExcelXP options(sheet_name='Summary - Trial 1' width_fudge='0.75'
   absolute_column_width='21, 8, 8');

* Proc TABULATE #1 here;

ods tagsets.ExcelXP options(sheet_name='Summary - Trial 2' width_fudge='0.75'
   absolute_column_width='21, 8, 8');

* Proc TABULATE #2 here;
```

There are instances where the values you calculate could result in columns that are wider or narrower than necessary. If this is the case, make small adjustments to the values specified for the ABSOLUTE_COLUMN_WIDTH option until the column widths are acceptable.

ADDING FROZEN COLUMN HEADERS AND AUTOFILTERS TO THE PRINT PROCEDURE OUTPUT

When you scroll down in a worksheet that contains a lot of data, the column headings can scroll off the top of the viewable screen. The PRINT procedure output shown in Figure 1 contains approximately 150 rows, and scrolling past row 22 will move the headers off the screen. To correct this problem, use the FROZEN_HEADERS tagset option to "freeze" the rows you want to use as headers. The syntax of this option is FROZEN_HEADERS='*n*', where '*n*' represents the row you want to freeze. In our case, we want to freeze all rows prior to 2, inclusive, so we would use `FROZEN_HEADERS='2'`.

Excel has a wonderful feature known as an "AutoFilter". An AutoFilter allows you to filter, or subset, the data that is being displayed in a worksheet. A column of data containing an AutoFilter is indicated by an arrow button in the header cell of that column. For example, all columns in Figure 1 contain AutoFilters.

Suppose you want to view only records for patients that experienced a headache as an adverse event. You would click the AutoFilter button on the "Preferred Term" column, and select **HEADACHE**, as illustrated in Figure 5. All records in the worksheet that do not have "HEADACHE" as a value in the "Preferred Term" column will be hidden. The data is still present in the worksheet, but it is not displayed due to the filtering.

AutoFilter selections are additive. If you want, after filtering for "HEADACHE", to see only patients who experienced *severe* headaches, you would click the AutoFilter button on the "Severity" column and select **`Severe`**. To clear an AutoFilter, you would select **`(All)`** from the drop-down list.

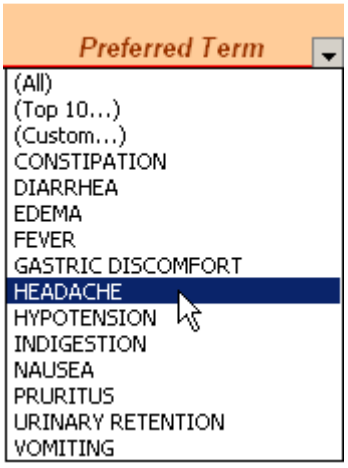


Figure 5. Excel AutoFilter

AutoFilters can be generated using the ODS tagset option named AUTOFILTER. You can specify values of "all", "none" or a range of columns. Since we want AutoFilters on all columns, we will specify the value "all". To freeze the headers and add AutoFilters to the PRINT procedure output, simply add the options to the existing ODS statement, making sure to turn the options off before the TABULATE procedure code:

```
ods tagsets.ExcelXP options(sheet_name='Data - Trial 1'  width_fudge='0.7'
   frozen_headers='2' autofilter='all');

  * Proc PRINT #1 here w/ style override and new labels;

ods tagsets.ExcelXP options(sheet_name='Data - Trial 2'  width_fudge='0.7'
   frozen_headers='2' autofilter='all');

  * Proc PRINT #2 here w/ style override and new labels;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 1' width_fudge='0.75'
     absolute_column_width='21, 8, 8' frozen_headers='no' autofilter='none');

  * Proc TABULATE #1 here;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 2' width_fudge='0.75'
     absolute_column_width='21, 8, 8' frozen_headers='no' autofilter='none');

  * Proc TABULATE #2 here;
```

PRINT ORIENTATION AND REPEATING COLUMN HEADER ROWS

At this point, with all the modifications, the code should produce the output shown in Figures 1 and 2. However, there are still some options you can set to improve the appearance of printed copies of the worksheets, notably: print orientation and repeating header rows.

The output generated by PROC PRINT is wide, and should be printed in landscape orientation. In contrast, the TABULATE output should be printed in portrait orientation. Controlling page orientation is accomplished by adding the option `ORIENTATION='landscape'` to the ODS statements preceding the PRINT procedure code, and `ORIENTATION='portrait'` to the ODS statements that precede the TABULATE procedure code.

Just as frozen headers were helpful when scrolling through the PRINT procedure output, displaying the column headers on all pages of printed output is also desirable. You can accomplish this by setting the value of the ROW_REPEAT option to `'1-2'` on the ODS statements that control the PRINT procedure output, and `'none'` on the ODS statements that control the TABULATE output.

PRINT HEADERS AND FOOTERS

One final feature that might interest you is the ability to specify custom headers and footers at the top and bottom of the printed output. By default, the ExcelXP tagset uses the text of TITLE statements as print headers, and the text of FOOTNOTE statements as print footers. This behavior is controlled by the EMBEDDED_TITLES and EMBEDDED_FOOTERS tagset options.

Consider this change to our SAS code:

```
title '&CAdverse Event Data by Trial';
footnote '&LPrinted &D&RPage &P of &N';
```

The TITLE statement results in a print header with the text "Adverse Event Data by Trial" centered at the top of each page of the printed output. The special Excel control sequence "`&C`" causes all text following it to be centered.

The FOOTNOTE statement is a bit more complicated. First, the word "Printed" will appear left-justified at the bottom of the printed output, followed by the current date. The "`&D`" control sequence causes the current date to be printed. Next, the text "Page *P* of *N*" will be right justified, where *P* is the current page number, and *N* is the page count.

THE FINAL SAS CODE

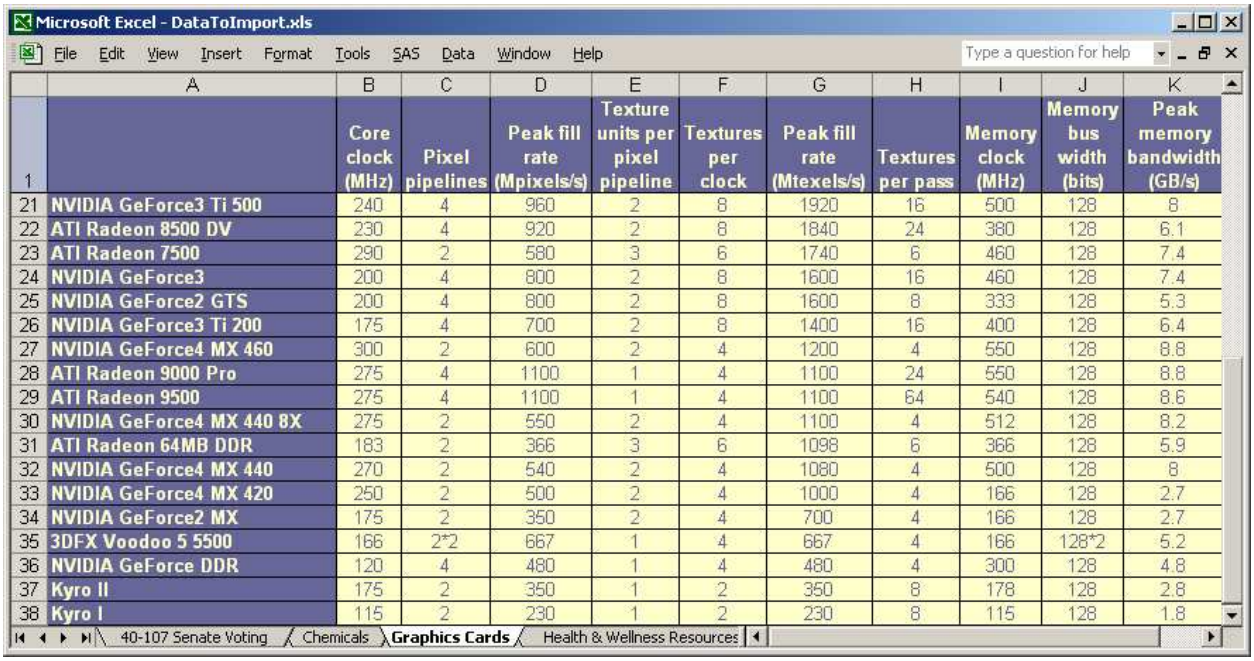
With all of the modifications in place, the code should appear as follows:

```
options nocenter;

title '&CAdverse Event Data by Trial';
footnote '&LPrinted &D&RPage &P of &N';

ods listing close;
ods tagsets.ExcelXP path='output-directory' file='aedata.xml' style=SUGI31;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 1'  width_fudge='0.7'
     frozen_headers='2' autofilter='all' orientation='landscape' row_repeat='1-2');

  * Proc PRINT #1 here w/ style override and new labels;

  ods tagsets.ExcelXP options(sheet_name='Data - Trial 2'  width_fudge='0.7'
     frozen_headers='2' autofilter='all' orientation='landscape' row_repeat='1-2');

  * Proc PRINT #2 here w/ style override and new labels;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 1' width_fudge='0.75'
     absolute_column_width='21, 8, 8' frozen_headers='no' autofilter='none'
     orientation='portrait' row_repeat='none');

  * Proc TABULATE #1 here;

  ods tagsets.ExcelXP options(sheet_name='Summary - Trial 2' width_fudge='0.75'
     absolute_column_width='21, 8, 8' frozen_headers='no' autofilter='none'
     orientation='portrait' row_repeat='none');

  * Proc TABULATE #2 here;

ods tagsets.ExcelXP close;
```

IMPORTING MULTI-SHEET EXCEL WORKBOOKS INTO SAS

Figure 6 shows the Excel workbook that we want to import into SAS. The workbook contains four worksheets. We want to import the data from each worksheet into a different SAS table, and use the name of the worksheet for the table name. The sections that follow explain how to save the workbook as an XML file, and then how to load the workbook into SAS using a provided SAS macro and SAS XMLMap, both designed specifically for Excel XML data. **All the necessary SAS code is supplied for you**.

To download the sample data, SAS macro, and SAS XMLMap, go to the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Creating *AND* Importing Multi-Sheet Excel Workbooks the Easy Way with SAS". There you'll find a download package that has all the files you need.



	A	B	C	D	E	F	G	H	I	J	K
1		Core clock (MHz)	Pixel pipelines	Peak fill rate (Mpixels/s)	Texture units per pixel pipeline	Textures per clock	Peak fill rate (Mtexels/s)	Textures per pass	Memory clock (MHz)	Memory bus width (bits)	Peak memory bandwidth (GB/s)
21	NVIDIA GeForce3 Ti 500	240	4	960	2	8	1920	16	500	128	8
22	ATI Radeon 8500 DV	230	4	920	2	8	1840	24	380	128	6.1
23	ATI Radeon 7500	290	2	580	3	6	1740	6	460	128	7.4
24	NVIDIA GeForce3	200	4	800	2	8	1600	16	460	128	7.4
25	NVIDIA GeForce2 GTS	200	4	800	2	8	1600	8	333	128	5.3
26	NVIDIA GeForce3 Ti 200	175	4	700	2	8	1400	16	400	128	6.4
27	NVIDIA GeForce4 MX 460	300	2	600	2	4	1200	4	550	128	8.8
28	ATI Radeon 9000 Pro	275	4	1100	1	4	1100	24	550	128	8.8
29	ATI Radeon 9500	275	4	1100	1	4	1100	64	540	128	8.6
30	NVIDIA GeForce4 MX 440 8X	275	2	550	2	4	1100	4	512	128	8.2
31	ATI Radeon 64MB DDR	183	2	366	3	6	1098	6	366	128	5.9
32	NVIDIA GeForce4 MX 440	270	2	540	2	4	1080	4	500	128	8
33	NVIDIA GeForce4 MX 420	250	2	500	2	4	1000	4	166	128	2.7
34	NVIDIA GeForce2 MX	175	2	350	2	4	700	4	166	128	2.7
35	3DFX Voodoo 5 5500	166	2*2	667	1	4	667	4	166	128*2	5.2
36	NVIDIA GeForce DDR	120	4	480	1	4	480	4	300	128	4.8
37	Kyro II	175	2	350	1	2	350	8	178	128	2.8
38	Kyro I	115	2	230	1	2	230	8	115	128	1.8

Figure 6. Excel Workbook with Data to Import into SAS

SAVING AN EXCEL WORKBOOK AS XML

You must save an Excel workbook as an XML file before you can import it into SAS. To do this, in Excel, select **`File ➨ Save As`**, and choose **`XML Spreadsheet (*.xml)`**. Specify a name and a directory for the file. For the purpose of this paper, we saved the XML in a file named "mydata.xml".

To ensure that SAS can access the "mydata.xml" file, save the file on a network-accessible drive. Then you can access the file as if it were native to the operating environment where SAS is installed. Another solution is to save the file to a location that is under the control of a Web server. SAS will then be able to read the file using the URL access method, which is part of Base SAS. These techniques are useful if SAS and Excel are installed on different machines. If neither option is available to you, using FTP or some other method, move the file from the Windows machine to the machine that SAS is installed on.

If you place the files where others can access them, be sure to set file permissions to prevent accidental alteration.

ISSUES WITH AUTOMATIC DATA CONVERSION FROM EXCEL TO SAS

You need to be aware of some of the behaviors to expect when converting Excel data to SAS data. As mentioned earlier, when the worksheets in Figure 6 are imported into SAS tables, the name of the worksheet (for example, "Chemicals") is used as the SAS table name. However, the technique of using the worksheet name for the table name doesn't always work. For example, a worksheet named "40-107 Senate Voting" can't be used for the SAS table name because it begins with a number and contains invalid characters (the spaces and a dash). Therefore, the SAS code that loads the worksheet into SAS must convert the "4" and other invalid characters to an underscore (_). In this case, the resulting SAS table name would be "_0_107_senate_voting". In the same way, importing a worksheet named "Health & Wellness Resources" would result in a table named "Health___wellness_resources". In all cases, the *label* for the SAS table is set to the original value of the Excel worksheet name.

Attempting to use Excel column labels for the column names of a SAS table can result in a similar problem. Notice that none of the column labels in Figure 6 can be used as SAS column names because each column label contains invalid characters such as blanks, parentheses, and the forward slash (/). Additionally, the first column does not have a label. The SAS code that loads the worksheet into SAS must create valid column names by converting invalid characters to an underscore (_). SAS column *labels* are set to the original values of the Excel column labels.

Table 1 shows examples of how some of the Excel column labels in Figure 6 will be converted to SAS column names.

Excel Column Label	SAS Column Name	SAS Column Label
(blank)	A unique 32-character name starting with an underscore (_)	.
Pixel pipelines	Pixel_pipelines	Pixel pipelines
Peak fill rate (Mpixels/s)	Peak_fill_rate__Mpixels_s_	Peak fill rate (Mpixels/s)

Table 1. Conversion of Excel Column Labels to SAS Column Names

THE SAS XML LIBNAME ENGINE (SXLE) AND XMLMAPS

The SAS XML LIBNAME Engine can import an XML file into a SAS table or export a SAS table as an XML file. Thus, you can use the SXLE to exchange data between SAS and third-party, XML-aware applications such as Excel.

Although the SXLE has been available since SAS release 8.1, recent improvements have made it possible to precisely control how data is imported. The new SAS XMLMap enables you to map any XML element or attribute to a column or a row in a SAS table. The SXLE then uses the XMLMap to control how the XML data is imported into a SAS table. You can create the XMLMap by using the new XML Mapper (formerly known as XML Atlas), which provides a point-and-click interface.

IMPORTING EXCEL XML INTO SAS

Although you could use the SAS LIBNAME Engine and a SAS XMLMap that you created to import an Excel XML file into SAS, writing such code would be a daunting task, since the Excel XML format and the data conversion issues are quite complex. To save you the toil of having to write such complicated but needed code, SAS provides an XMLMap and SAS code specific to Excel to load the Excel XML into SAS tables. The SAS XMLMap and a SAS macro to import the Excel XML data are available as part of the download package for this paper.

Download the XMLMap and the SAS macro; make the two files available to the platform where SAS is installed. This paper assumes that you saved the SAS XMLMap in a file named "ExcelXP.map" and the SAS macro for loading the XML data in a file named "LoadXL.sas". The file "LoadXL.sas" contains a SAS macro named XLXP2SAS, which is used to import the Excel XML file into SAS tables.

The easiest way to explain how to use the macro is with a few examples. First, let's look at importing the XML workbook shown in Figure 6. This process will work for two scenarios:

1. SAS is installed on the same machine that has the XML file.
2. SAS is installed on a different machine or platform, but the XML file is accessible via a network drive.

To import the workbook into SAS, submit the following code, making sure to include the appropriate directory paths:

```
❶ %include 'LoadXL.sas';

❷ %xlxp2sas(excelfile=mydata.xml,
            mapfile=ExcelXP.map);
```

The first statement (❶) makes the XLXP2SAS macro available to SAS. The second statement (❷) imports the data from all the worksheets into separate SAS tables. By default, the SAS tables are created in the WORK library. You can control which library is used to store the SAS tables by specifying the LIBRARY argument in the XLXP2SAS macro. For example, to store the tables in the SASUSER library, submit this code:

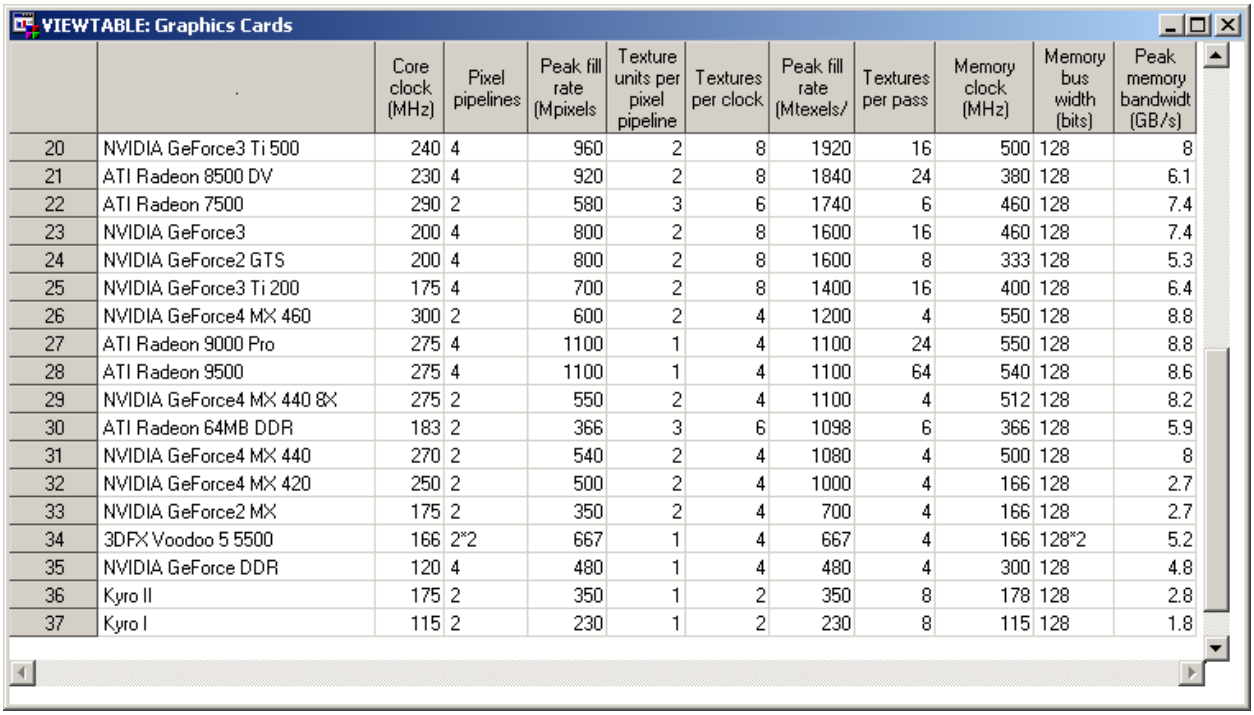
```
%xlxp2sas(excelfile=mydata.xml,
          mapfile=ExcelXP.map,
          library=sasuser);
```

Table 2 lists the SAS tables that were created as a result of importing the Excel workbook shown in Figure 6. Although the SAS table names might look a bit odd, the original worksheet names are used as SAS labels.

SAS Table Name	SAS Label
Chemicals	Chemicals
Graphics_cards	Graphics Cards
Health___wellness_resources	Health & Wellness Resources
_0_107_senate_voting	40-107 Senate Voting

Table 2. SAS Tables Created from the Excel Workbook

Figure 7 shows a portion of the "Graphics Cards" table displayed using the SAS VIEWTABLE application. By comparing Figures 6 and 7, you can see that the XLXP2SAS macro successfully imported the "Graphics Cards" worksheet as a SAS table. For instance, the columns "Pixel pipelines" and "Memory bus width (bits)" were both correctly typed as character, because those columns contain data such as "2*2" and "128*2", respectively.



VIEWTABLE: Graphics Cards

	.	Core clock (MHz)	Pixel pipelines	Peak fill rate (Mpixels	Texture units per pixel pipeline	Textures per clock	Peak fill rate (Mtexels/	Textures per pass	Memory clock (MHz)	Memory bus width (bits)	Peak memory bandwidt (GB/s)
20	NVIDIA GeForce3 Ti 500	240	4	960	2	8	1920	16	500	128	8
21	ATI Radeon 8500 DV	230	4	920	2	8	1840	24	380	128	6.1
22	ATI Radeon 7500	290	2	580	3	6	1740	6	460	128	7.4
23	NVIDIA GeForce3	200	4	800	2	8	1600	16	460	128	7.4
24	NVIDIA GeForce2 GTS	200	4	800	2	8	1600	8	333	128	5.3
25	NVIDIA GeForce3 Ti 200	175	4	700	2	8	1400	16	400	128	6.4
26	NVIDIA GeForce4 MX 460	300	2	600	2	4	1200	4	550	128	8.8
27	ATI Radeon 9000 Pro	275	4	1100	1	4	1100	24	550	128	8.8
28	ATI Radeon 9500	275	4	1100	1	4	1100	64	540	128	8.6
29	NVIDIA GeForce4 MX 440 8X	275	2	550	2	4	1100	4	512	128	8.2
30	ATI Radeon 64MB DDR	183	2	366	3	6	1098	6	366	128	5.9
31	NVIDIA GeForce4 MX 440	270	2	540	2	4	1080	4	500	128	8
32	NVIDIA GeForce4 MX 420	250	2	500	2	4	1000	4	166	128	2.7
33	NVIDIA GeForce2 MX	175	2	350	2	4	700	4	166	128	2.7
34	3DFX Voodoo 5 5500	166	2*2	667	1	4	667	4	166	128*2	5.2
35	NVIDIA GeForce DDR	120	4	480	1	4	480	4	300	128	4.8
36	Kyro II	175	2	350	1	2	350	8	178	128	2.8
37	Kyro I	115	2	230	1	2	230	8	115	128	1.8

Figure 7. SAS Graphics Cards Table Created by Importing the Excel Workbook

Up to this point, the assumption is that the XML file resides on the same machine as SAS or was available via a network drive. However, if the XML file resides on the Web server of a remote machine, you can use the URL access method to retrieve the file by submitting the following code:

```
filename myxml URL 'http://Web-server/mydata.xml';
%xlxp2sas(excelfile=FILEREF:myxml,
          mapfile=ExcelXP.map);
```

The submitted code causes XLXP2SAS to contact the Web server to retrieve the XML file, rather than looking for it on a local disk. The FILEREF modifier can also be used with the MAPFILE argument to retrieve the SAS XMLMap from a Web server or other location. Documentation for the XLXP2SAS macro can be found in the appendix in the section “XLXP2SAS Macro”.

XLXP2SAS DRAWBACKS AND LIMITATIONS

There are issues surrounding XLXP2SAS that you should be aware of:

- XLXP2SAS creates temporary tables in the WORK library. These tables can get very large. However, unless your system is very low in disk space, this is generally not a problem because the temporary tables are automatically cleaned up after XLXP2SAS runs (unless the argument `CLEANUP=N` was specified).

- The data in every worksheet that you want to import must be fairly rectangular. That is, the data must already resemble a SAS table with rows and columns. Although the XLXP2SAS macro attempts to handle non-rectangular data by adding missing values, the results can be unpredictable if the data is too irregular.
- By default, XLXP2SAS runs with the argument HASLABELS=Y. This implies that *all* worksheets in a workbook have column labels in the first row. If *none* of your worksheets contain column labels in the first row, specify HASLABELS=N when you invoke XLXP2SAS. The HASLABELS argument applies to all worksheets in a workbook.
- If you specify HASLABELS=N, the column names in the SAS table(s) will be in the form "COLUMN1", "COLUMN2", "COLUMN3", and so on, and the respective column labels will be "Column 1", "Column 2" and "Column 3".

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 is new for SAS 9.1.

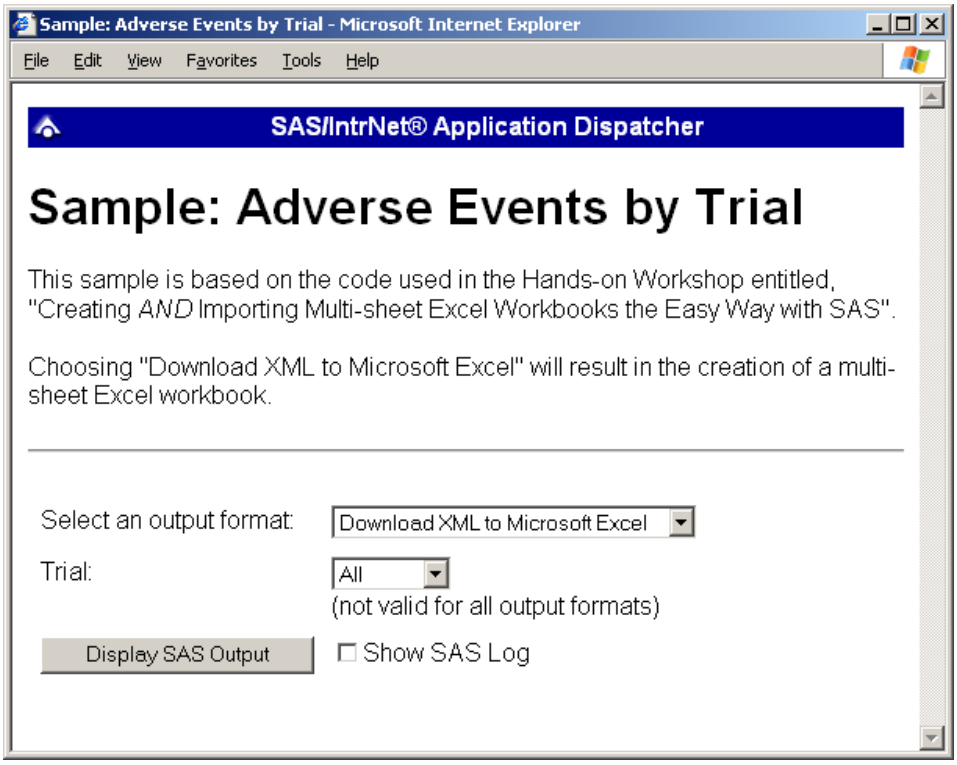


Figure 8. Web Page to Drive a SAS/IntrNet Application

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's been shown up to this point can be executed by using 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. 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 8 shows a Web page that could be used to deliver SAS output directly to Excel, using the Web browser as the client. Clicking the "Display SAS Output" button would result in the execution of a slightly modified version of the PRINT and TABULATE procedure code that we have been working with. The modifications are as follows:

```
%let RV = %sysfunc(appsrv_header(Content-type,application/vnd.ms-excel));
options nocenter;
title '&CAdverse Event Data by Trial';
footnote '&LPrinted &D&RPage &P of &N';
ods listing close;
ods tagsets.ExcelXP file=_webout style=SUGI31;
  * Remainder of "final" SAS code;
ods tagsets.ExcelXP close;
```

The 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. The special, reserved FILEREF (_WEBOUT) is automatically assigned by the SAS server, and is always used to direct output from the SAS server to the client.

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 and/or Stored Process documentation, as well as this author's earlier papers (DelGobbo, 2002, 2003, 2004).

CONCLUSION

The SAS 9.1 ExcelXP ODS tagset complies with the Microsoft XML Spreadsheet Specification and provides an easy way to export your SAS data to Excel workbooks that contain multiple worksheets. Although you might initially encounter formatting problems when using this technique, by using ODS styles, style overrides and tagset options, you can correct these problems.

The SAS XML Libname Engine, SAS XMLMaps, and the XLXP2SAS macro greatly simplify the task of moving Excel XML-formatted data into SAS.

APPENDIX

CODE FOR CREATING THE SUGI31 STYLE

```
proc template;
  define style styles.SUGI31;
    parent = styles.Journal;

    /*************************************************************************
      Use approximate "SUGI reddish brown" cx993300 for some foreground elements.
      Use black for other foreground elements.
      Use approximate "SUGI tan" cxffcc99 for some background elements.
    *************************************************************************/

    replace colors /
      'docbg'     = cxFFFFFF
      'contentbg' = cxFFFFFF
      'contentfg' = cx000000
      'titlefg'   = _undef_
      'link2'     = cx0066AA
      'link1'     = cx004488
      'bg5'       = cxFFFFFF
      'bg4'       = cxFFFFFF
      'fg4'       = cx000000
      'bg3'       = cxFFFFFF
      'fg3'       = cx000000
      'bg2'       = cxffcc99   /*  Header and Row Header cells  */
      'bgA1'      = cxFFFFFF
      'fgA1'      = cxff0000   /*  Border lines  */
      'fg2'       = cx993300   /*  Header and Row Header cells  */
      'fg'        = cx993300;  /*  Titles and Footnotes  */

/*  Redefine some characteristics of some of the standard styles.  */

    style table from table /
      borderwidth = 0;

    style header from header /
      just              = center
      font_weight       = bold
      bordertopwidth    = 3
      borderbottomwidth = 3
      borderleftwidth   = 0
      borderrightwidth  = 0;
```

```
    style rowheader from rowheader /
      font_weight = bold;

/*  Controls border lines of some Header cells.  */

    style header_r_border from header /
      just              = center
      bordertopwidth    = 3
      borderbottomwidth = 3
      borderleftwidth   = 0
      borderrightwidth  = 3;

/*  Controls border lines of the TABULATE box cell.  */

    style header_box from header /
      just              = center
      bordertopwidth    = 3
      borderbottomwidth = 3
      borderleftwidth   = 0
      borderrightwidth  = 3;

/*  Controls border lines of some Row Header cells.  */

    style rowheader_r_border from rowheader /
      bordertopwidth    = 0
      borderbottomwidth = 0
      borderleftwidth   = 0
      borderrightwidth  = 3;

/*  Assigns an Excel format to certain Data cells.   */

    style data_z8 from data /
      tagattr='format:00000000';

  end;
run; quit;
```

Notice that *approximate* colors are used for titles, footnotes, header, and row header cells. Because Excel has a limited color palette and, by default, does not support all the exact colors used in the SUGI 31 conference materials and Web site, only Excel-supported colors were used. For further information about colors supported by Excel, refer to the section "Colors Supported by Excel" below.

COLORS SUPPORTED BY EXCEL

Figure A1 shows the colors that an unmodified version of Excel supports by default.

Because each Excel user has the ability to customize the color palette, choosing a color from this figure is not guaranteed to be supported in all instances of Excel.

For more information on working with colors in SAS and Excel, refer to this author's SUGI 30 paper (DelGobbo, 2005).

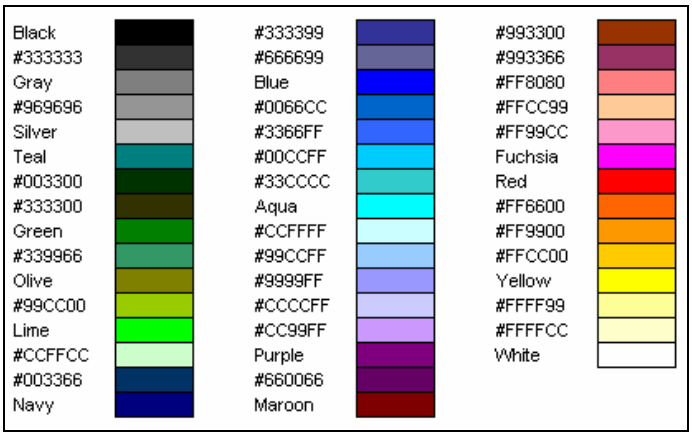


Figure A1. Default Colors Supported by Excel 2000 and Later

XLXP2SAS MACRO

The following table contains a list of the arguments that are supported by the XLXP2SAS macro. All arguments require a value, except where a default value is indicated.

Argument	Description	Default Value
EXCELFILE	Specifies the name and path for the Excel XML file that you want to import into SAS. Do not use quotation marks in this value. To specify a SAS FILEREF instead of a file, use `FILEREF:`*`fref`*.	
MAPFILE	Specifies the name and path of the SAS XMLMap for importing Excel XML files. Do not use quotation marks in this value. To specify a SAS FILEREF instead of a file, use `FILEREF:`*`fref`*. You can download a copy of the XMLMap from the SAS Presents Web site at support.sas.com/saspresents. Find the entry "Creating *AND* Importing Multi-Sheet Excel Workbooks the Easy Way with SAS".	
LIBRARY	Specifies the name of the SAS library where imported tables are stored.	WORK
HASLABELS	Specifies whether the Excel worksheets have column labels in the first row of the Excel table. This setting applies to all worksheets in a workbook. If set to `Y`, the labels are used for the SAS column names and labels. If your workbook does not have column labels in the first row of the Excel table, specify `N`.	Y
CLEANUP	Controls whether temporary SAS files are deleted and whether to de-assign FILEREFs that were used when importing the Excel data to SAS. FILEREFs that you explicitly assign with a FILEREF statement will not be de-assigned. To disable this feature, specify `N`.	Y
VERBOSE	Controls the level of debugging information written to the SAS Log. Specify `Y` to activate this feature.	N

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.

DelGobbo, V. 2005. "Moving Data and Analytical Results between SAS® and Microsoft Office". *Proceedings of the Thirtieth Annual SAS Users Group International Conference*, 30. CD-ROM. Paper 136. Available http://www2.sas.com/proceedings/sugi30/136-30.pdf.

Microsoft Corporation. "XML Spreadsheet Reference". Available http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnexcl2k2/html/odc_xmlss.asp.

SAS Institute Inc. 1999. "Chapter 5: The TEMPLATE Procedure". *The Complete Guide to the SAS Output Delivery System, Version 8.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2004. "Chapter 9: TEMPLATE Procedure: Creating a Style Definition". *SAS 9.1 Output Delivery System, User's Guide*. Cary, NC: SAS Institute Inc.

FURTHER READING

SAS Institute Inc. "Application Dispatcher". Available http://support.sas.com/rnd/web/intrnet/dispatch.html.

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

SAS Institute Inc. "The PRINT Procedure". *Base SAS® 9.1.3 Procedures* Guide. Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/base_proc_8417.pdf.

SAS Institute Inc. "The TABULATE Procedure". *Base SAS® 9.1.3 Procedures* Guide. Cary, NC: SAS Institute Inc. Available http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/base_proc_8417.pdf.

ACKNOWLEDGMENTS

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

CONTACT INFORMATION

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

Vincent DelGobbo
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000

sasvcd@unx.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.