

Traffic Lighting Your Multi-Sheet Microsoft Excel Workbooks

the Easy Way with SAS®

Vincent DelGobbo, SAS Institute Inc., Cary, NC

ABSTRACT

"Traffic lighting" is the process of applying visual formatting to data. This paper explains how to use Base SAS®9 software to create multi-sheet Microsoft Excel workbooks (for Excel versions 2002 and later), and then traffic light the values that lie outside a range. You will learn step-by-step techniques for quickly and easily creating attractive, multi-sheet Excel workbooks that contain your SAS output using the ExcelXP ODS tagset. 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! Creating and delivering your workbooks on-demand and in real time using 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 or later to create an Excel workbook that contains two worksheets. The worksheets contain fictional lab result data for clinical trials "ABC 123" and "XYZ 987" (Figure 1), and data about the ranges used in the traffic lighting (Figure 2). Cells with non-white backgrounds are used to traffic light values that are outside the normal range. Excel formats, not SAS formats, are used to control the appearance of the date values. The date values are SAS date values that have been converted to Excel dates, without modifying the original SAS data. An explanation of the data used to create this workbook can be found in the section "Sample Data".

Treatment	Subject	Lab Site	Visit	Visit Date	Haemoglobin (g/L)	Haematocrit (%)	RBC (x10E12/L)	WBC (x10E9/L)	Abs. Lymphocytes (x10E9/L)	Abs. Monocytes (x10E9/L)
Control - Active	11	S11	1	17Jun2001	130	0.42	4	6.91	3.66	0.124
			2	19Jun2001	140	0.42	5	11.8	9.72	0.083
	13	G51	1	09Jun2001	130	0.38	4	10	6	0.1
			2	10Jun2001	90	0.28	3	6.1	3.172	0.061
	16	G51	1	07Jun2001	120	0.35	4	7.3	4.818	0
			2	09Jun2001	100	0.3	3	8.2	5.576	0.082
	17	A26	1	08Jun2001	150	0.45	5	11.76	8.41	0.09
			2	10Jun2001	130	0.4	4	18.61	15.6	0.06
	22	G51	1	10May2001	120	0.37	4	5.8	3.131	0.116
			2	12May2001	120	0.38	4	5.5	3.245	0.11
	24	G54	1	22Apr2001	130	0.38	4	5.9	2.95	0.354
			2	24Apr2001	120	0.35	4	8.1	5.103	0.486
	32	A92	1	21Jul2001	140	0.41	4	11	5.3	0.2
			2	23Jul2001	140	0.42	4	11.8	7.1	0.2
	36	A91	1	26Jul2001	140	0.41	5	8.1	5.994	0.081
			2	28Jul2001	130	0.37	4	8.7	5.915	0.087
	39	A92	1	29Sep2001	140	0.41	4	9.4	5	0.1
			2	01Oct2001	130	0.38	4	10.3	6.9	0.2
	48	G51	1	09Jun2001	110	0.34	4	5.6	2.856	0.28
			2	11Jun2001	90	0.28	3	9.6	7.776	0
	50	Q01	1	07Sep2001	140	0.41	4	7.6	4.68	0.078
			2	09Sep2001	130	0.37	4	12	8.64	0
	56	S11	1	25Aug2001	150	0.47	5	7.18	4.16	0.416
			2	27Aug2001	130	0.42	4	11	8.66	0.11
	64	S11	1	07Sep2001	120	0.36	4	3.4	1.326	0.068
			2	09Sep2001	100	0.3	4	9.4	7.896	0
	66	A92	1	28Jul2001	90	0.29	4	5.3	2.9	0.2
			2	30Jul2001	90	0.28	4	5.4	3.4	0.1
	71	A76	1	30Jul2001	140	0.47	5	7.68	4.76	0.07

Figure 1. Lab Results Worksheet of Multi-Sheet Excel Workbook Generated by the ODS ExcelXP Tagset

Range Flags	Color
Normal	
Low	
Clinically Significant Low	
High	
Clinically Significant High	

Figure 2. Range Flags Worksheet of Multi-Sheet Excel Workbook Generated by the ExcelXP Tagset

The remainder of this paper guides you through the steps used to create the Excel workbook shown in Figures 1 and 2 using Base SAS®9 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 "Traffic Lighting Your Multi-Sheet Microsoft 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 2003 SP2 software.

REQUIREMENTS

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 (z/OS, UNIX, etc.) 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.

LIMITATIONS

Because the ExcelXP ODS tagset creates files that conform to the Microsoft XML Spreadsheet Specification, you can create multi-sheet Excel workbooks containing the output from almost any SAS procedure. The exception is that the Microsoft XML Spreadsheet Specification does not support images, so the output from SAS/GRAPH® software procedures cannot be used (Microsoft Corporation 2001).

You can use ExcelXP tagset options with all procedure output, but ODS style overrides apply only to the PRINT, REPORT, and TABULATE procedures. These two techniques are discussed in the sections "Working with the ExcelXP Tagset Options" and "Using the XLSansPrinter Style", respectively.

SAMPLE DATA

Table 1 presents abbreviated information about the SAS table "LabResults" that is used to create the Excel workbook shown in Figure 1.

Variable Name	Variable Label	Variable Type	SAS Format	Typical Values
Protocol	Protocol Identifier	Character		ABC 123 or XYZ 987
Treat	Treatment	Character		Control - Active, Test - High Dose
Patient	Subject	Numeric		1 - 140
LabSite	Lab Site	Character	\$CHAR3.	S11, G51, A26
Visit	Visit	Numeric	1.	1 or 2
VisitDate	Visit Date	Numeric	YYMMDD10.	2001-06-17, 2001-06-19
Haemoglobin_Result	Haemoglobin*(g/L)	Numeric		60 - 160
Haemoglobin_Flag	(not applicable)	Numeric		0, 1, 2, 3, or 4
Haematocrit_Result	Haematocrit*(%)	Numeric		0.19 - 0.52
Haematocrit_Flag	(not applicable)	Numeric		0, 1, 2, 3, or 4
RBC_Result	RBC*(x10E12/L)	Numeric		2.0 - 6.0
RBC_Flag	(not applicable)	Numeric		0, 1, 2, 3, or 4
WBC_Result	WBC*(x10E9/L)	Numeric		2.14 - 22.8
WBC_Flag	(not applicable)	Numeric		0, 1, 2, 3, or 4

Table 1. Representative Data Values in the SAS Table "LabResults"

Variables whose names end in "_Result" contain the lab result values to traffic light. Table 1 lists 4 of the 10 such variables displayed in our Excel workbook. The background color of the cell is determined by the value of the corresponding "_Flag" variable. For example, the value of the variable "Haemoglobin_Flag" determines the background color for "Haemoglobin_Result".

The REPORT procedure is run against this data to create the worksheets named "16.2 ABC 123" and "16.2 XYZ 987" (Figure 1). The worksheet names are based on the International Committee on Harmonisation (ICH) Guideline E6 for clinical study report submissions. The guideline details the content and order of data collected in clinical trials. Patient data listings are found in section 16.2; hence, the naming of the worksheets in this paper.

Table 2 shows the background colors corresponding to the values of the "_Flag" variables.

Flag Value	Flag Meaning	Background Color
0	Normal	white
1	Low	#CCFFFF
3	Clinically Significant Low	#9999FF
2	High	#FFCC00
4	Clinically Significant High	#FF6600

Table 2. Flag Values and Their Meanings and Colors

The worksheet in Figure 2 serves as a legend, defining the meaning of the traffic light colors. The SAS table containing this data was created using the following code:

```
data Legend;
length LabFlag 8 Range $30;
LabFlag = 0; Range = 'Normal';          output;
LabFlag = 1; Range = 'Low';              output;
LabFlag = 3; Range = 'Clinically Significant Low'; output;
LabFlag = 2; Range = 'High';              output;
LabFlag = 4; Range = 'Clinically Significant High'; output;
label Range = 'Range Flags'
      LabFlag = 'Color';
run;
```

The PRINT procedure is used to display this range data in the worksheet named "Range Flags" (Figure 2).

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 Figures 1 and 2 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 are the general ODS statements needed to generate XML output that is compatible with versions 2002 and later of Microsoft Excel:

```
❶ ods listing close;

❷ ods tagsets.ExcelXP file='file-name.xml' style=style-name ... ;
   * Your SAS procedure code here;

❸ ods tagsets.ExcelXP close;
```

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

The second ODS statement (❷) uses the ExcelXP tagset to generate the XML output and then store 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 other than Excel.
- You can take advantage of Web server authentication and security models.

Note: If you place the files where users 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** (Excel 2002 or 2003) or **Office Button** ➤ **Open** (Excel 2007)
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, except in step 2 you should either navigate to the desired file or type the path and file name in the **File name** field. You can also navigate to the file using Microsoft Windows Explorer, and then double-click the file to open it with Excel.

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)**. If you're using Excel 2007, click the Microsoft Office Button, and then select **Save As** ➤ **Excel 97-2003 Workbook**.

Sometimes you may encounter a "problem during load" error when you attempt to open the file with Excel. More information about this error can be found in the appendix in the section "Diagnosing Excel Load Errors".

SETTING UP THE ODS ENVIRONMENT

Our sample code employs a user-defined style named "XLsansPrinter" and a modified version of the ExcelXP tagset. The following statements define the location where the 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 of, 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 the access mode for `mylib.tmplmst` is "update", PROC TEMPLATE, used later in this paper, creates and stores 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 "Traffic Lighting Your Multi-Sheet Microsoft 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 need to submit this code 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 the "Setting up the ODS Environment" section, above.)

The ExcelXP tagset supports many options that control both the appearance and functionality of the Excel workbook. To see a listing of the supported options, submit the following 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 listing of the supported options 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 advantage of the features discussed in this paper, you must download a recent copy of the tagset and install it on your system as described previously. The version of the tagset used in this paper can be found in the download package on the SAS Presents Web site, as noted above. A recent version of the tagset is available from the ODS Web site (SAS Institute Inc. 2010. "ODS MARKUP Resources").

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, 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.

Use the ODS tagset named "style_popup" when you need to determine the attributes of style elements. The tagset creates an HTML file that, when viewed using the SAS Results Window or the Microsoft Internet Explorer Web

browser, displays style element information in popup windows (SAS Institute Inc. 2009. "Uses of the Results Window". *SAS® 9.2 Language Reference: Concepts*).

For example, to see the style attributes of the "header" style element of the "sansPrinter" style, follow these steps:

1. Submit this SAS code:

```
ods results;
ods tagsets.style_popup path='output-directory' file='temp.htm'
  style=sansPrinter;

  * Your SAS procedure code here;

ods tagsets.style_popup close;
```

2. View the HTML output using the SAS Results Window or Microsoft Internet Explorer.
3. Click on a column heading to display a popup window containing the style element name and style attribute information for that cell (Figure 3).

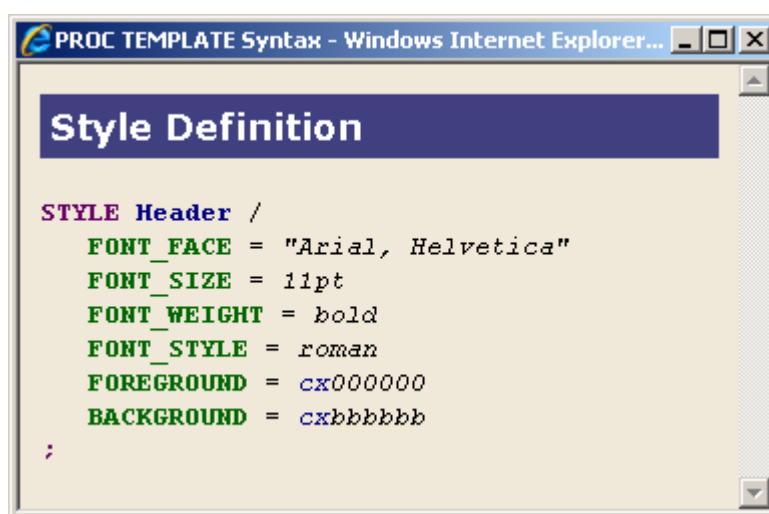


Figure 3. Style Attributes of the "header" Style Element

This style element contains the style attributes "font_face", "font_size", and so on. Use the TEMPLATE procedure to modify an existing style or to create a new style. The next section describes using the TEMPLATE procedure to create a new style by first copying an existing style, and then modifying the copy.

CREATING YOUR OWN STYLE: THE XLSANSPRINTER STYLE

The workbook shown in Figures 1 and 2 was created using a user-defined ODS style named "XLsansPrinter", which is based on the "sansPrinter" style that is supplied by SAS. 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 copy the "sansPrinter" style, change existing style elements, and create new style elements that will be used later. The complete code for creating the "XLsansPrinter" style can be found in the appendix of this paper, "Code for Creating the XLsansPrinter Style".

Although you can use the TEMPLATE procedure to create a new style without copying an existing style, it is usually easier to copy an existing style that is close to what you want, and then make modifications to the copy. The code below creates the user-defined "XLsansPrinter" style by copying the standard ODS style named "sansPrinter":

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

Because this TEMPLATE procedure code does not contain statements that change any style elements, the "XLSansPrinter" style is an exact copy of the "sansPrinter" style. That is, the "XLSansPrinter" style inherits all of the style elements and attributes of the parent style, "sansPrinter".

Style elements are created using the "style" statement, which follows this general format:

```
style new-style-element-name from existing-style-element-name /
  style-attribute-specifications;
```

The code below, when added to the initial definition of the "XLSansPrinter style" (above), illustrates how you can change the value of an existing style attribute, and also add new attributes.

```

  ❶          ❷
style header from header /
  font_size = 10pt
  just      = center
  vjust     = bottom;
```

The first occurrence of "header" (❶) indicates the name of the style element being created. The second occurrence (❷) specifies the name of an existing style element to use as the parent. New style elements inherit all of the style attributes of the parent element. The code above creates a style element named "header", which inherits all of the style attributes of the existing style element named "header" shown in Figure 3. Then the font size is changed from "11pt" to "10pt", and new style attributes are added to control the horizontal and vertical justification. All other style attributes are inherited from the existing "header" style element, and remain unchanged. Figure 4 is a graphic representation of the code above.

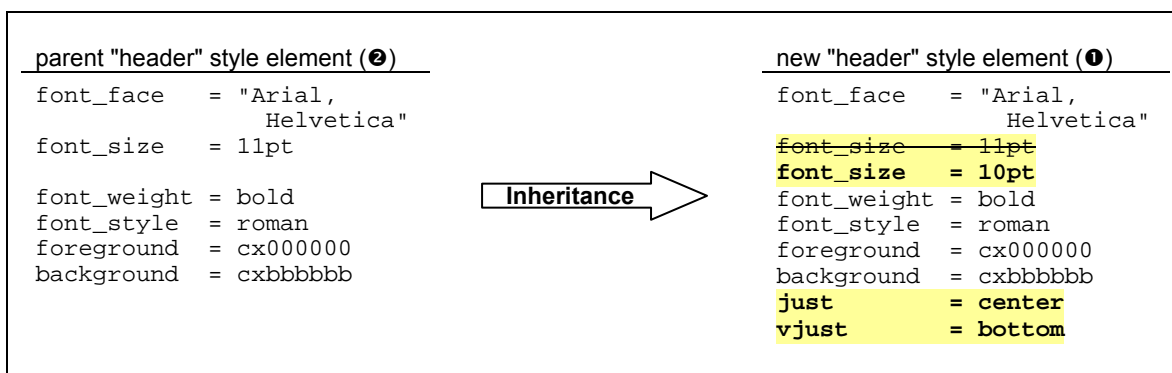


Figure 4. Creating the "header" Style Element

Similarly, the "notecontent" style element, which controls the appearance of lines inserted by the LINE statement of the REPORT procedure, is modified to ensure that border lines are displayed:

```
style notecontent from notecontent /
  bordertopwidth = 1
  borderbottomwidth = 1
  borderleftwidth = 1
  borderrightwidth = 1;
```

When you run the TEMPLATE procedure code provided in the appendix, you generate the "XLSansPrinter" style, and also create three user-defined style elements:

- **data_bold**
This style element has all the attributes of the "data" style element that is supplied by SAS, the difference being that text is bold. A style override is used later to apply the "data_bold" style element to the "Treat" column.
- **data_center**
Inherits all the attributes of the "data" style element that is supplied by SAS, but centers text within the cells. The "data_center" style element is later applied to the "Patient", "LabSite", and "Visit" columns using a style override.

- `data_date9`
Converts SAS date values to Excel date values. This style element is based on the "data" style element that is supplied by SAS, and has the same appearance characteristics as that style element. The "tagattr" attribute applies an Excel format and instructs Excel to interpret the value in the cell as an Excel datetime value. This style element is later applied to the "VisitDate" column using a style override.

For more information about Excel formats, type "create a custom number format" into the Excel help system or refer to any of the innumerable books covering Excel.

At this point you should run the TEMPLATE procedure code found in the appendix of this paper to create the "XLSansPrinter" style on your system. You should need to perform this step only once because the "XLSansPrinter" style is stored in the directory corresponding to the MYLIB library, and all of your future SAS programs can access the style by adding 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. 2009. *SAS® 9.2 Output Delivery System: User's Guide*).

USING ODS TO CREATE THE MULTI-SHEET EXCEL WORKBOOK

By default, the ExcelXP tagset creates a new worksheet each time a SAS procedure creates new tabular output. The PRINT procedure creates the range flags worksheet and the REPORT procedure creates the two lab results worksheets.

SAS CODE TO CREATE THE EXCEL WORKBOOK

Below is an abbreviated listing of the *basic* SAS code used to create the Excel workbook. A full listing of the basic SAS code can be found in the appendix of this paper, "Basic Code for Creating the Excel Workbook".

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

title;
footnote;

* Create the range flags worksheet;

❷ proc print data=Legend noobs label;
    var Range;
    var LabFlag;
run; quit;

* Create the lab results worksheets;

❸ proc report data=sample.LabResults split='*' nowindows;
    by protocol;

    * 'ID' columns;

    column Treat Patient ... ;

    * Data columns with spanned headers;

    column ...

    * Hidden columns containing the range flags;

    column Haemoglobin_Flag Haematocrit_Flag ... ;

    * Dummy column to perform traffic lighting;

    column dummy;
```



```

* 'ID' columns;

define Treat      / display order;
define Patient    / display order;
... ;

* Data columns;

define Haemoglobin_Result / display;
define Haematocrit_Result / display;
... ;

* Hidden columns containing the range flags;

define Haemoglobin_Flag / display noprint;
define Haematocrit_Flag / display noprint;
... ;

* Dummy column for style assignment;

define dummy / computed noprint;

* Traffic light the data columns based on the hidden columns;

❷ compute dummy;
  * Place holder;
endcomp;

* Insert a blank line between treatments;

❸ compute after Treat;
  line ' ';
endcomp;

run; quit;

ods tagsets.ExcelXP close;

```

As you can see in the ODS statement (❶), the ExcelXP tagset generates the output, and the "XLSansPrinter" style controls the appearance of the output. PROC PRINT (❷) creates the first worksheet, which serves as a legend. (Code for creating the SAS table "Legend" is in the appendix.)

The multiple VAR statements (❸) enable us to apply an ODS style override to the "LabFlag" column, without affecting the "Range" column. The concept of style overrides is presented in the "Understanding and Using ODS Style Overrides" section.

Further examination of the code reveals that the REPORT procedure is run with a BY statement (❹) to create worksheets 2 and 3, one worksheet for each distinct value of the variable "Protocol". (The SAS table "LabResults" is included in the download package for this paper.)

The first COMPUTE block (❺) is used to traffic light the lab result data, and the second one (❻) inserts white space between the different treatment values.

Figure 5 displays the results of executing the basic SAS code (from the appendix), and opening the resulting "LabReport.xml" file with Excel. Notice that Figure 5 does not match Figures 1 and 2. The following problems are exhibited in Figure 5:

1. None of the values in any of the worksheets are traffic-lighted.
2. Data values in the "Treatment" column are not bold text.
3. Values in the "Subject", "Lab Site" and "Visit" columns are not centered.
4. The date values are displayed incorrectly.
5. Standard BY group text ("Protocol Identifier=ABC 123" or "Protocol Identifier=XYZ 987") precedes the REPORT procedure output and is obscured.
6. Unattractive, default worksheet names are used.
7. Some of the columns created by the REPORT procedure are too narrow.
8. If you were to scroll downward or to the right in the lab results worksheets, you will "lose" the column and row headers because by default, they are not "frozen".

Microsoft Excel - LabReport.xml

File Edit View Insert Format Tools Data SAS Window Help

Type a question for help

B2 0

	A	B	C	D	E	F	G	H	I	J	K	L
1	Range Flags	Color										
2	Normal	0										
3	Low	1										
4	Clinically Significant Low	3										
5	High	2										
6	Clinically Significant High	4										
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												
23												
24												
25												
26												
27												
28												
29												
30												
31												
32												

Table 1 - Data Set WORK.LEGEND Table 2 - Detailed and or summa Table 3 - Detailed and or summa

Ready

Microsoft Excel - LabReport.xml

File Edit View Insert Format Tools Data SAS Window Help

Type a question for help

E5 2001-06-17

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Protocol															
2																
3	Haematology															
4	Differential															
5	Treatme nt	Subject	Lab Site	Visit	Visit Date	Haemogl obin (g/L)	Haemato crit (%)	RBC (x10E12/ L)	WBC (x10E9/L)	Abs. Lymphoc ytes (x10E9/L)	Abs. Monocyt es (x10E9/L)	Abs. Neutroph ils (x10E9/L)	Abs. Eosinoph ils (x10E9/L)	Abs. Basophil s (x10E9/L)	Platelets (x10E9/L)	
6	Active	11	S11	1	17	130	0.42	4	6.91	3.66	0.124	2.49	0.539	0.076	170	
7				2	19	140	0.42	5	11.8	9.72	0.083	1.25	0.708	0.035	193	
8		13	G51	1	09	130	0.38	4	10	6	0.1	3.2	0.6	0.1	238	
9				2	10	90	0.28	3	6.1	3.172	0.061	2.379	0.488	0	189	
10		16	G51	1	07	120	0.35	4	7.3	4.818	0	1.97	0.438	0.073	321	
11				2	09	100	0.3	3	8.2	5.576	0.082	1.476	0.984	0.082	268	
12		17	A26	1	08	150	0.45	5	11.76	8.41	0.09	2.88	0.37	0.01	271	
13				2	10	130	0.4	4	18.61	15.6	0.06	2.09	0.86	0	257	
14		22	G51	1	10	120	0.37	4	5.8	3.131	0.116	1.856	0.638	0.058	204	
15				2	12	120	0.38	4	5.5	3.245	0.11	1.595	0.495	0.055	199	
16		24	G54	1	22	130	0.38	4	5.9	2.95	0.354	2.242	0.354	0.059	286	
17				2	24	120	0.35	4	8.1	5.103	0.486	2.106	0.405	0	290	
18		32	A92	1	21	140	0.41	4	11	5.3	0.2	4.8	0.7	0	233	
19				2	23	140	0.42	4	11.8	7.1	0.2	3.7	0.8	0	223	
20		36	A91	1	26	140	0.41	5	8.1	5.994	0.081	1.62	0.405	0.081	234	
21				2	28	130	0.37	4	8.7	5.915	0.087	2.001	0.609	0.087	212	
22		39	A92	1	29	140	0.41	4	9.4	5	0.1	3.2	0.4	0.7	268	
23				2	01	130	0.38	4	10.3	6.9	0.2	2.4	0.6	0.2	230	
24		48	G51	1	09	110	0.34	4	5.6	2.856	0.28	1.736	0.672	0.056	298	
25				2	11	90	0.28	3	9.6	7.776	0	0.768	1.056	0	262	

Table 1 - Data Set WORK.LEGEND Table 2 - Detailed and or summa Table 3 - Detailed and or summa

Ready

Figure 5. Range Flags (Top) and Lab Results (Bottom) Worksheets in the ODS ExcelXP-Generated Workbook

We will now change the basic SAS code to correct these problems.

USING THE XLSANSPRINTER STYLE

As mentioned earlier, the user-defined "XLSansPrinter" style is very similar to the standard ODS "sansPrinter" style, but the two styles do differ. For instance, the column headings use a font that is centered 10-point instead of left-justified 11-point, and cells generated as a result of LINE statements in PROC REPORT contain border lines on all sides. These differences are a result of the modifications made to the "header" and "notecontent" style elements which are supplied by SAS, as shown in the code in the appendix.

Additionally, the user-defined style elements "data_bold", "data_center", and "data_date9" are applied to specific parts of the output using ODS style overrides, which are covered in the following sections.

Understanding and Using ODS Style Overrides

The TEMPLATE procedure enables you to create new style elements. These style elements are then applied to specific parts of SAS output called *locations*. Figure 6 shows the locations that are pertinent to the PRINT and REPORT procedure output (SAS Institute Inc. 2008. "SAS[®]9 Reporting Procedure Styles Tip Sheet").

table			report		
obsheader	header	header	header	header	header
obs	column	column	column	column	column
obs	column	column	column	column	column
obs	column	column	summary	summary	summary
obs	column	column	lines		
obs	column	column	column	column	column
obs	column	column	column	column	column
bylabel	total	total	summary	summary	summary
grandtotal	grandtotal	grandtotal	lines		
n			lines		

Figure 6. Style Locations for the PRINT Procedure (Left) and the REPORT Procedure (Right)

By default, ODS applies the "header" style element to the "Header" location, and the appearance of the "Column" location is controlled by the "data" style element. In other words, the "header" style element controls the appearance of the column heading cells, and the "data" style element controls the appearance of the data cells. Changing the "header" style element, as we have done in the "XLSansPrinter" style, affects all procedure output that uses that style element. The same is true for the "notecontent" style element. Thus, you should carefully consider the impact of changing style elements that are part of a style supplied by SAS.

Style overrides are supported by the PRINT, TABULATE, and REPORT procedures, and can be specified in several ways, the two most common formats being:

- ❶ `style(location)=[style-attribute-name1=value1 style-attribute-name2=value2 ...]`
- ❷ `style(location)=style-element-name`

The first format (❶) uses individual style attributes defined inline. For example, the following code alters the attributes of the "Header" location:

```
style(Header)=[background=white font_size=10pt just=center]
```

While this is the most commonly used format, it has some disadvantages. To use the same style override for different variables, you must apply it in multiple places, making your SAS code harder to read and maintain. And, if you want to use the style overrides in other SAS programs, you must copy the list of attribute name/value pairs to the new code. Style overrides of this type should be avoided when possible. We use this method only to traffic light cells in the PRINT and REPORT procedure output.

The second format (❷) overcomes these problems. Using this format involves creating a new style element, setting the style attributes within the element, and then using the *style element* name in your style override. This results in code that is easier to read, maintain, and re-use.

As an example, suppose you want to change the appearance of the column headings in some, but not all, of your output. One solution is to create a new style element based on the original "header" style element, and use a style override to apply the new style element to the "Header" location as needed. The following code shows you how to create a new style element:

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

    style myheader from header /
      background = white
      font_size = 10pt
      just = center;
  end;
run; quit;
```

This code does not alter the "header" style element that is supplied by SAS (❷). Instead, it creates a new, user-defined style element named "myheader" (❶) that inherits all the attributes of the "header" style element. The code then changes the values of the "background", "font_size", and "just" attributes.

The user-defined style element must be specified as a style override before ODS will recognize it. The code below shows how to override the style element used for the "Header" location for the variables "name" and "age":

```
ods tagsets.ExcelXP style=MysansPrinter ...;
proc print data=sashelp.class noobs;
  var name age / style(Header)=myheader;
  var sex height weight;
run; quit;

proc report data=sashelp.class nowindows;
  column name age sex height weight;
  define name / style(Header)=myheader;
  define age / style(Header)=myheader;
run; quit;
ods tagsets.ExcelXP close;
```

Colors Supported by Excel

Excel versions 2002 and 2003 have a limited color palette. Figure 7 shows the default colors. If you plan to view your workbooks using one of these versions of Excel, choose colors that are listed in the default palette, otherwise Excel maps the unsupported color to a color that is supported.

However, specifying colors is not foolproof. Because each Excel user can customize the color palette, colors are not guaranteed to exist in all instances of Excel.

Note that Excel 2007 does not have this restricted color palette.

Black		#333399		#993300	
#333333		#666699		#993366	
Gray		Blue		#FF8080	
#969696		#0066CC		#FFCC99	
Silver		#3366FF		#FF99CC	
Teal		#00CCFF		Fuchsia	
#003300		#33CCCC		Red	
#333300		Aqua		#FF6600	
Green		#CCFFFF		#FF9900	
#339966		#99CCFF		#FFCC00	
Olive		#9999FF		Yellow	
#99CC00		#CCCCFF		#FFFF99	
Lime		#CC99FF		#FFFFCC	
#CCFFCC		Purple		White	
#003366		#660066			
Navy		Maroon			

Figure 7. Default Colors Supported by Excel Versions 2002 and 2003

Traffic Lighting the Range Flags Worksheet

The PRINT procedure creates the range flags worksheet shown in Figure 2. Initial execution of the code creates the worksheet shown at the top of Figure 5. In order to make Figure 2 match Figure 5, we need to traffic light the data in the "LabFlag" column. This can be accomplished using a style override.

The foreground and background colors of the cells are usually controlled by applying a style override to the "Column" location, using this general format:

```
var LabFlag / style(Column)=[foreground=color-value background=color-value];
```

This will not work in our case because *all* of the cells in the "LabFlag" column would have the same foreground and background color. Instead, we need the colors to change based on the value in the cell.

The code in the appendix creates the "FlagFmt" SAS format that controls the foreground and background color of the data cells. The format was constructed based on the criteria listed in Table 2 and is reproduced here:

```
proc format;
  value FlagFmt 0 = 'white'
                1 = '#CCFFFF'
                3 = '#9999FF'
                2 = '#FFCC00'
                4 = '#FF6600';
run; quit;
```

Specifying a format for *color-value* causes the current value of the data in the cell to be evaluated against the format to determine the attribute's value. For example, cells containing the value "1" have a light turquoise (#CCFFFF) foreground and background. Here is the resulting PROC PRINT code:

```
proc print data=Legend noobs label;
  var Range;
  var LabFlag / style(Column)=[foreground=FlagFmt. background=FlagFmt.];
run; quit;
```

Traffic Lighting the Lab Results Worksheets

Traffic lighting the lab result values using the REPORT procedure requires a slightly different technique. This traffic lighting uses a CALL DEFINE statement within a COMPUTE block. The CALL DEFINE statement assigns the "background" style attribute to the cell containing the result value. The general syntax of this statement is:

```
call define(column-id, 'attribute-name', attribute-value);
```

In the context of traffic lighting our data, this statement takes on the following form:

```
call define('Haemoglobin_Result', 'style', 'style=[background=color-value]');
```

where *color-value* is the desired color.

Recall that each lab result value has its own flag variable that is used to determine whether the result is in or out of range. We need to examine the value of the pertinent flag variable, and use that value to determine *color-value*. The "FlagFmt" format is used to determine *color-value* based on the flag value. For example, here is the code to traffic light "Haemoglobin_Result" cells:

```
if (Haemoglobin_Flag gt 0)
  then call define(Haemoglobin_Result,
                  'style',
                  'style=[background=' || put(Haemoglobin_Flag, FlagFmt.) || ']);
```

Table 3 shows the various style attribute values generated by the above code, based on the value of the "_Flag" variable.

_Flag Variable Value	Value of Style Attribute
0	Code not executed
1	style=[background=#CCFFFF]
3	style=[background=#9999FF]
2	style=[background=#FFCC00]
4	style=[background=#FF6600]

Table 3. Flag Values and Resulting Style Attribute Values

You could repeat the above code for each of the remaining result/flag combinations, or you can make use of arrays. Using arrays provides more compact code:

```
* Traffic light the data columns based on the hidden columns;

compute dummy;

array name(10) $31 ('Haemoglobin_Result' 'Haematocrit_Result' 'RBC_Result'
                  'WBC_Result' 'Lymphocytes_Result' 'Monocytes_Result'
                  'Neutrophils_Result' 'Eosinophils_Result' 'Basophils_Result'
                  'Platelets_Result');

array flag(10) Haemoglobin_Flag Haematocrit_Flag RBC_Flag
              WBC_Flag Lymphocytes_Flag Monocytes_Flag
              Neutrophils_Flag Eosinophils_Flag Basophils_Flag
              Platelets_Flag;

*;
* Loop over all the _Result columns ('name' array), and set the
* BACKGROUND style attribute based on the value of the corresponding
* _Flag column ('flag' array).
*;

do i = 1 to dim(name);
  if (flag(i) gt 0)
    then call define(name(i),
                    'style',
                    'style=[background=' || put(flag(i), FlagFmt.) || ']);
end;
endcomp;
```

Changing the Appearance of the Treatment, Patient, Lab Site, and Visit Columns

The values in the data cells of the "Patient", "LabSite", and "Visit" columns should be centered, but Figure 5 shows that they are not. The "XLSansPrinter" style contains the user-defined "data_center" style element that is exactly the same as the "data" style element that is supplied by SAS, except the justification is set to "center":

```
/* Used to center text */  
  
style data_center from data /  
    just = center;
```

The following PROC REPORT code centers the data values by applying the "data_center" style element, as a style override, to the "Column" location:

```
define Patient / display order style(Column)=data_center;  
define LabSite / display order style(Column)=data_center;  
define Visit / display order style(Column)=data_center;
```

Similar code causes the values in the data cells of the "Treat" column to be displayed with bold text:

```
define Treat / display order style(Column)=data_bold;
```

Working with SAS and Excel Dates

SAS and Microsoft Excel use different date systems. Consequently, you often encounter problems when Excel reads SAS output containing date values.

One way to correct this behavior is to write SAS code to convert numeric SAS date values to numeric Excel date values, but this approach is problematic because you must alter your original SAS data. While you can create a new SAS view or table that contains the new date values, this is a vector for errors and becomes inefficient as your data grows.

A better solution, one that does not require you to alter the underlying data, is to use a combination of SAS and Excel formats. First you specify a SAS format, and then you specify an Excel format using a style override. The SAS format changes what is physically written into the XML file, and the Excel format changes the way the value is displayed.

Because Excel expects dates to be represented using the ISO 8601 format (YYYY-MM-DD), SAS date values should be formatted using the YYMMDD or IS8601DA formats. Table 1 shows that the YYMMDD10. format is applied to the "VisitDate" column, providing Excel with dates in the correct format. No additional FORMAT statement is needed in this case.

The "XLSansPrinter" style contains the user-defined style element "data_date9", which inherits all the attributes of the "data" style element that is supplied by SAS. The "tagattr" attribute is used to apply an Excel format to the data, and also to instruct Excel to interpret the value as an Excel datetime value:

```
/* Used to convert SAS date values to Excel date values */  
  
style data_date9 from data /  
    tagattr = 'type:DateTime format:ddmmyyyy';
```

Table 4 shows how the date "June 9, 1997" would be displayed in Excel using different formats, including the format used by the "data_date9" style element ("ddmmyyyy"). Refer to Figure 1 to see other display values for the Excel "ddmmyyyy" format.

Excel Format	Display Value
ddmmyyyy	09Jun1997
m/d/yyyy	6/9/1997
mm/d/yyyy	06/9/1997
mm/dd/yyyy	06/09/1997
mmm d, yyyy	Jun 9, 1997
mmm, d, yyyy	June 9, 1997
m/d/yy	6/9/97

Table 4. Excel Formats and Corresponding Display Values for the Date June 9, 1997

The following PROC REPORT code applies, as a style override, the "data_date9" style element to the "Column" location of the "VisitDate" column:

```
define VisitDate / display style(Column)=data_date9;
```

Refer to SAS Usage Note 11206 for information about the ISO 8601 SAS date and datetime formats (SAS Institute Inc. 2010. "Usage Note 11206: IS8601* FORMATS and INFORMATS for DATE, TIME, and DATETIME"). To find out more about Excel date formats, type "display numbers as dates or times" into the Excel help system or refer to a book that covers this topic.

WORKING WITH THE EXCELXP TAGSET OPTIONS

As mentioned earlier, the ExcelXP tagset supports many 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 in an ODS statement using the OPTIONS keyword:

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

It is important to note that the value you specify for a tagset option remains in effect until the ExcelXP destination is closed, or the option is set to another value. Because 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 procedure code here;
ods tagsets.ExcelXP options(option-name='default-value') ... ;
* Other SAS procedure code here;
```

IMPORTANT NOTE

Tagset options are supported for **all** SAS procedure output, unlike ODS style overrides, which are supported only by the PRINT, REPORT, and TABULATE procedures.

Specifying the Name for the Range Flags Worksheet

ODS generates a unique name for each worksheet, as required by Microsoft Excel. Figure 5 shows the worksheet names that result from running the SAS code. There are, however, several tagset options that you can use to alter the names of the worksheets.

Use the SHEET_NAME option to explicitly specify a worksheet name. Recall that tagset options remain in effect until the ExcelXP destination is closed. Because we don't want the name of the range flags worksheet to affect the lab results worksheets, we reset the value after the PRINT procedure executes.

```
ods tagsets.ExcelXP path='...' file='...' style=XLsansPrinter;

title; footnote;

ods tagsets.ExcelXP options(sheet_name='Range Flags');

* Create the range flags worksheet;

proc print data=Legend ... ; run; quit;

* Need to reset the option value;

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

* Create the lab results worksheets;

proc report data=sample.LabResults ... ; run; quit;

ods tagsets.ExcelXP close;
```


Naming the Lab Results Worksheets Using the BY Group Values

The SHEET_INTERVAL option controls the interval at which SAS output is placed into worksheets, and the SHEET_LABEL option is used to specify the prefix to use for the worksheet names. When used together, the current value of the first BY group variable is used in the worksheet name. The following code causes the worksheet names to match those shown in Figures 1 and 2.

```
ods tagsets.ExcelXP path='...' file='...' style=XLsansPrinter;

title; footnote;

ods tagsets.ExcelXP options(sheet_name='Range Flags');

* Create the range flags worksheet;

proc print data=Legend ... ; run; quit;

* Need to reset the option value;

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

ods tagsets.ExcelXP options(sheet_interval='bygroup'
                             sheet_label='16.2');

* Create the lab results worksheets;

proc report data=sample.LabResults ... ; run; quit;

ods tagsets.ExcelXP close;
```

If you do not want any text to prefix the BY value, specify sheet_label=' ' (note the blank space between the quotation marks).

Suppressing the BY Line Text

BY line text appears in the lab results worksheets because the REPORT procedure is executed with a BY statement. The text is obscured because the first column is narrow, but it is also redundant because the BY value is displayed in the worksheet name. To omit the BY line text, specify the SUPPRESS_BYLINES option in the ODS statement that precedes the PROC REPORT code.

```
ods tagsets.ExcelXP options(sheet_interval='bygroup'
                             sheet_label='16.2'
                             suppress_bylines='yes');

* Create the lab results worksheets;

proc report data=sample.LabResults ... ; run; quit;
```

Do not attempt to use the NOBYLINE system option, as this disables BY group processing in the ExcelXP tagset when the SHEET_INTERVAL option is set to "bygroup".

Altering the Column Widths

Some of the column widths of the lab results worksheets are too narrow, causing data to be obscured, or unattractive wrapping of column headings (Figure 5). The ABSOLUTE_COLUMN_WIDTH option is used to correct this problem. The ExcelXP tagset uses the following formula to compute the approximate column width, sometimes resulting in less than perfect widths:

$$\text{width} = \text{PointSize} * \text{NumberOfCharacters} * \text{FudgeFactor}$$

The value of PointSize is computed based on the metrics of the font, and NumberOfCharacters is the value specified in the ABSOLUTE_COLUMN_WIDTH option. FudgeFactor has a default value of 0.75.

The ABSOLUTE_COLUMN_WIDTH option accepts a list of comma-separated values that specify the width, in characters, of the columns. If you specify fewer values than the number of columns in the worksheet, the values you specify will repeat, and are used for the remaining columns.

To approximate the values needed for the option, we start by specifying an arbitrary value of "10", which causes all of the columns to be the same size.

```
ods tagsets.ExcelXP options(sheet_interval='bygroup'
                             sheet_label='16.2'
                             suppress_bylines='yes'
                             absolute_column_width='10');
```

```
* Create the lab results worksheets;
```

```
proc report data=sample.LabResults ... ; run; quit;
```

Then we view the output using Excel (Figure 8).

Treatment	Subject	Lab Site	Visit	Visit Date	Haemoglobin (g/L)	Haematocrit (%)	RBC (x10E12/L)	WBC (x10E9/L)	Abs. Lymphocytes (x10E9/L)	Mon (x10E9/L)
Active	11	S11	1	17Jun2001	130	0.42	4	6.91	3.66	
			2	19Jun2001	140	0.42	5	11.8	9.72	
	13	G51	1	09Jun2001	130	0.38	4	10	6	
			2	10Jun2001	90	0.28	3	6.1	3.172	
	16	G51	1	07Jun2001	120	0.35	4	7.3	4.818	
			2	09Jun2001	100	0.3	3	8.2	5.576	
	17	A26	1	08Jun2001	150	0.45	5	11.76	8.41	
			2	10Jun2001	130	0.4	4	18.61	15.6	
	22	G51	1	10May2001	120	0.37	4	5.8	3.131	
			2	12May2001	120	0.38	4	5.5	3.245	
	24	G54	1	22Apr2001	130	0.38	4	5.9	2.95	
			2	24Apr2001	120	0.35	4	8.1	5.103	
	32	A92	1	21Jul2001	140	0.41	4	11	5.3	
			2	23Jul2001	140	0.42	4	11.8	7.1	
	36	A91	1	26Jul2001	140	0.41	5	8.1	5.994	
			2	28Jul2001	130	0.37	4	8.7	5.915	
	39	A92	1	29Sep2001	140	0.41	4	9.4	5	
			2	01Oct2001	130	0.38	4	10.3	6.9	
	48	G51	1	09Jun2001	110	0.34	4	5.6	2.856	
			2	11Jun2001	90	0.28	3	9.6	7.776	
	50	Q01	1	07Sep2001	140	0.41	4	7.8	4.68	

Figure 8. Lab Results Worksheet After Specifying ABSOLUTE_COLUMN_WIDTH='10'

The width of all the columns containing the lab result values are reasonable, but data in the "Treatment" column is still obscured, and the "Subject", "Lab Site", "Visit", and "Visit Date" columns are all wider than they need to be (see Figure 1). Also, comparing the column headings in Figures 1 and 8 shows that the column headings in Figure 8 are too tall. This is corrected using the AUTOFIT_HEIGHT option. The following code specifies more appropriate widths for each of the columns in the worksheet, and also corrects the row heights:

```
ods tagsets.ExcelXP options(sheet_interval='bygroup'
                             sheet_label='16.2'
                             suppress_bylines='yes'
                             absolute_column_width='15,7,7,7,7,10,10,10,10,
                                                    10,10,10,10,10,10'
                             autofit_height='yes'); * ignore line wrapping above;
```

```
* Create the lab results worksheets;
```

```
proc report data=sample.LabResults ... ; run; quit;
```

If desired, the ABSOLUTE_COLUMN_WIDTH option can be used to adjust the column widths in the range flags worksheet.

For an alternate technique of determining appropriate values for the ABSOLUTE_COLUMN_WIDTH option, refer to this author's earlier paper (DelGobbo 2009).

Frozen Column and Row Headings

When you scroll down in a workbook that contains a lot of data, the column headings can scroll off the top of the viewable screen. The REPORT procedure output shown in Figure 1 contains 15 columns and over 200 rows, and scrolling down past row 32 causes the column headings to move off the screen. You can correct this problem by using the FROZEN_HEADERS tagset option to "freeze" the rows you want to use as column headings.

Similarly, scrolling to the right past Excel column "K" causes the "Treatment", "Subject", "Lab Site", "Visit", and "Visit Date" columns to move off the left of the viewable screen. This problem is corrected by the FROZEN_ROWHEADERS option.

Valid values for both of these options are either "yes" or an integer. If you specify "yes", the tagset will attempt to automatically determine the columns or rows to freeze. If you do not like the results, you can explicitly specify the rows or columns to freeze. For example, to freeze rows 1-3, specify `frozen_headers='3'`. Adding these 2 options results in the following code:

```
ods tagsets.ExcelXP options(sheet_interval='bygroup'
                             sheet_label='16.2'
                             suppress_bylines='yes'
                             absolute_column_width='15,7,7,7,7,10,10,10,10,
                                                    10,10,10,10,10,10'
                             autofit_height='yes'
                             frozen_headers='yes' frozen_rowheaders='5');

* Create the lab results worksheets;

proc report data=sample.LabResults ... ; run; quit;
```

Specifying a value of "5" for FROZEN_ROWHEADERS ensures that Excel columns "A-E" are frozen when scrolling to the right.

THE FINAL SAS CODE

The final SAS code to create the output of Figures 1 and 2 follows:

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

title;
footnote;

* Create the range flags worksheet;

ods tagsets.ExcelXP options(sheet_name='Range Flags');

proc print data=Legend noobs label;
  var Range;
  var LabFlag / style(column)=[foreground=FlagFmt. background=FlagFmt.];
run; quit;

* Need to reset the option value;

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

```

* Create the lab results worksheets;

ods tagsets.ExcelXP options(sheet_interval='bygroup' sheet_label='16.2'
                           suppress_bylines='yes'
                           absolute_column_width='15,7,7,7,7,10,10,10,10,
                                                10,10,10,10,10,10'
                           autofit_height='yes'
                           frozen_headers='yes' frozen_rowheaders='5');

proc report data=sample.LabResults split='*' nowindows;
by protocol;

* 'ID' columns;

column Treat Patient LabSite Visit VisitDate;

* Data columns with spanned headers;

column ('Haematology' Haemoglobin_Result Haematocrit_Result RBC_Result WBC_Result
        ('Differential' Lymphocytes_Result Monocytes_Result Neutrophils_Result
         Eosinophils_Result Basophils_Result Platelets_Result
        )
);

* Hidden columns containing the range flags;

column Haemoglobin_Flag Haematocrit_Flag RBC_Flag
        WBC_Flag Lymphocytes_Flag Monocytes_Flag
        Neutrophils_Flag Eosinophils_Flag Basophils_Flag
        Platelets_Flag;

* Dummy column to perform traffic lighting;

column dummy;

* 'ID' columns;

define Treat / display order style(Column)=data_bold;
define Patient / display order style(Column)=data_center;
define LabSite / display order style(Column)=data_center;
define Visit / display order style(Column)=data_center;
define VisitDate / display style(Column)=data_date9;

* Data columns;

define Haemoglobin_Result / display;
define Haematocrit_Result / display;
define RBC_Result / display;
define WBC_Result / display;
define Lymphocytes_Result / display;
define Monocytes_Result / display;
define Neutrophils_Result / display;
define Eosinophils_Result / display;
define Basophils_Result / display;
define Platelets_Result / display;

* Hidden columns containing the range flags;

define Haemoglobin_Flag / display noprint;
define Haematocrit_Flag / display noprint;
define RBC_Flag / display noprint;
define WBC_Flag / display noprint;
define Lymphocytes_Flag / display noprint;
define Monocytes_Flag / display noprint;
define Neutrophils_Flag / display noprint;
define Eosinophils_Flag / display noprint;
define Basophils_Flag / display noprint;
define Platelets_Flag / display noprint;

```

```

* Dummy column to perform traffic lighting;

define dummy / computed noprint;

* Traffic light the data columns based on the hidden columns;

compute dummy;

array name(10) $31 ('Haemoglobin_Result' 'Haematocrit_Result' 'RBC_Result'
                  'WBC_Result' 'Lymphocytes_Result' 'Monocytes_Result'
                  'Neutrophils_Result' 'Eosinophils_Result' 'Basophils_Result'
                  'Platelets_Result');

array flag(10) Haemoglobin_Flag Haematocrit_Flag RBC_Flag
              WBC_Flag Lymphocytes_Flag Monocytes_Flag
              Neutrophils_Flag Eosinophils_Flag Basophils_Flag
              Platelets_Flag;

*;
* Loop over all the _Result columns ('name' array), and set the
* BACKGROUND style attribute based on the value of the corresponding
* _Flag column ('flag' array).
*;

do i = 1 to dim(name);
  if (flag(i) gt 0)
    then call define(name(i),
                    'style',
                    'style=[background=' || put(flag(i), FlagFmt.) || ']');
end;

endcomp;

* Insert a blank line between treatments;

compute after Treat;
  line ' ';
endcomp;

run; quit;

ods tagsets.ExcelXP close;

```

SAS SERVER TECHNOLOGY

You can incorporate dynamically-generated SAS output into Excel by using the Application Dispatcher, which is part of SAS/IntrNet®. You can do the same with the SAS® Stored Process Server, new for SAS®9, and part of SAS® Integration Technologies.

Both of these products enable you to execute SAS programs from a Web browser or any other client that can open an HTTP connection to the Application Dispatcher or the SAS Stored Process Server (both of which can run on any platform where SAS is licensed). SAS software does not need to be installed on the client machine.

The SAS programs that you execute from the browser can contain any combination of DATA Step, procedure, macro, or SCL code. Thus, all the code that has been shown up to this point can be executed by both the Application Dispatcher and the SAS 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 9 shows a Web page that can deliver SAS output directly to Excel, using a Web browser as the client.

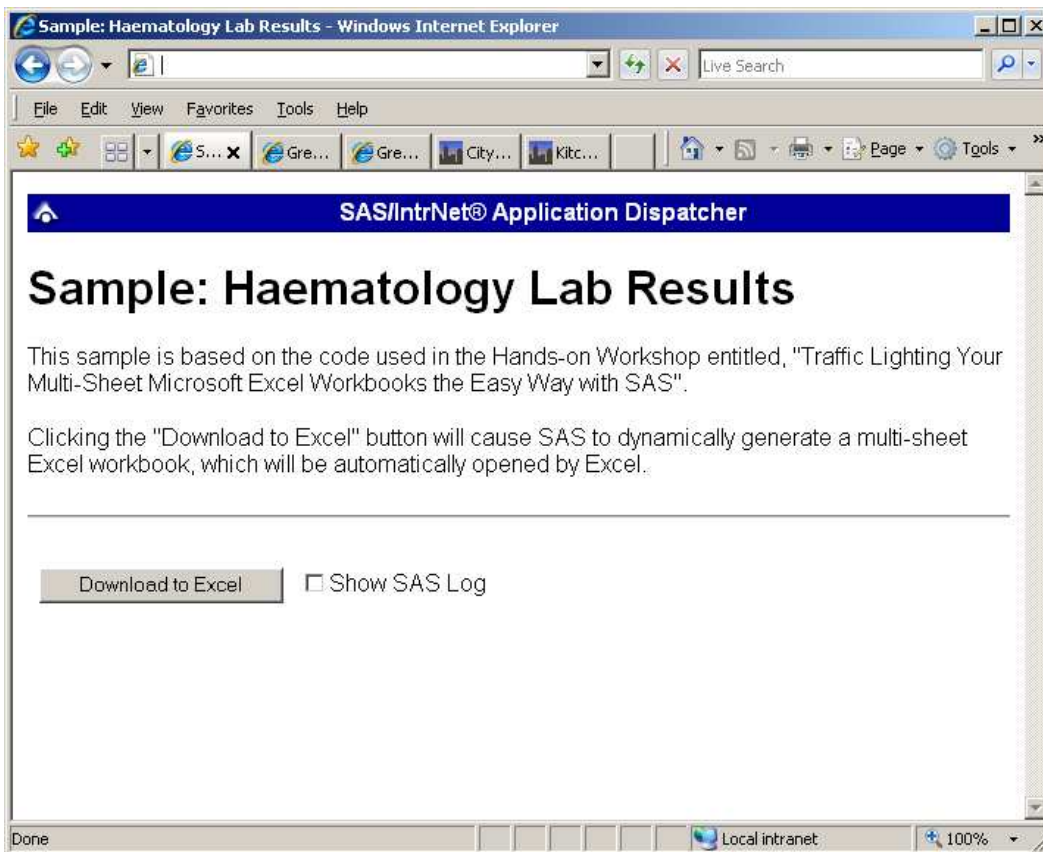


Figure 9. Web Page to Drive a SAS/IntrNet Application

Clicking "Download to Excel" executes a slightly modified version of the SAS code that we have been working on. 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=
"LabReport.xml")); * Ignore line wrapping;

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

The first APPSRV_HEADER function sets a 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 MIME header that populates the file name field in the "File Download" dialog box. As you can see from Figure 10, the file name appears as "LabReport.xml". This header might 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

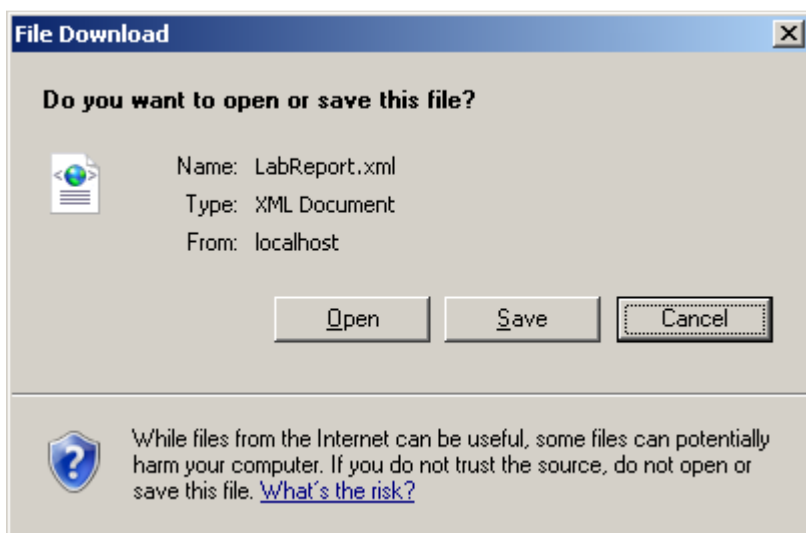


Figure 10. File Download Dialog Box

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 to Excel" button and are presented with the "File Download" dialog box (Figure 10), 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 and SAS Stored Process documentation (SAS Institute Inc. 2008. *SAS/IntrNet® 9.2: Application Dispatcher*; SAS Institute Inc. 2009. *SAS® 9.2 Stored Processes: Developer's Guide*), as well as this author's earlier papers (DelGobbo 2002-2004).

CONCLUSION

The SAS®9 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 a tagset that complies with the Microsoft XML Spreadsheet Specification, you can customize the output to achieve your design goals.

APPENDIX

CODE FOR CREATING THE XLSANSPRINTER STYLE

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

    /* Change attributes of the column headings */

    style header from header /
      font_size = 10pt
      just      = center
      vjust     = bottom;

    /* Change attributes of the blank lines */

    style notecontent from notecontent /
      bordertopwidth    = 1
      borderbottomwidth = 1
      borderleftwidth   = 1
      borderrightwidth  = 1;

    /* Used to create bold text */

    style data_bold from data /
      font_weight = bold;

    /* Used to center text */

    style data_center from data /
      just = center;

    /* Used to convert SAS date values to Excel date values */

    style data_date9 from data /
      tagattr = 'type:DateTime format:ddmmmyyyy';
  end;
run; quit;
```

BASIC CODE FOR CREATING THE EXCEL WORKBOOK

```
* Create a SAS table that is used for the range flags worksheet;

data Legend;
length LabFlag 8 Range $30;
LabFlag = 0; Range = 'Normal';          output;
LabFlag = 1; Range = 'Low';              output;
LabFlag = 3; Range = 'Clinically Significant Low'; output;
LabFlag = 2; Range = 'High';             output;
LabFlag = 4; Range = 'Clinically Significant High'; output;
label Range = 'Range Flags'
      LabFlag = 'Color';
run;

*;
* Create a format for traffic lighting, based
* on the lab flag values
*
* 0 : white      Normal
* 1 : #CCFFFF    Low
* 3 : #FFCC00    Clinically Significant Low
* 2 : #9999FF    High
* 4 : #FF6600    Clinically Significant High
*;

proc format;
value FlagFmt 0 = 'white'
              1 = '#CCFFFF'
              3 = '#9999FF'
              2 = '#FFCC00'
              4 = '#FF6600';
run; quit;

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

title;
footnote;

* Create the range flags worksheet;

proc print data=Legend noobs label;
var Range;
var LabFlag;
run; quit;

* Create the lab results worksheets;

proc report data=sample.LabResults split='*' nowindows;
by protocol;

* 'ID' columns;

column Treat Patient LabSite Visit VisitDate;

* Data columns with spanned headers;

column ('Haematology' Haemoglobin_Result Haematocrit_Result RBC_Result WBC_Result
        ('Differential' Lymphocytes_Result Monocytes_Result Neutrophils_Result
        Eosinophils_Result Basophils_Result Platelets_Result
        )
);
```



```

* Hidden columns containing the range flags;

column Haemoglobin_Flag Haematocrit_Flag RBC_Flag
      WBC_Flag      Lymphocytes_Flag Monocytes_Flag
      Neutrophils_Flag Eosinophils_Flag Basophils_Flag
      Platelets_Flag;

* Dummy column to perform traffic lighting;

column dummy;

* 'ID' columns;

define Treat      / display order;
define Patient    / display order;
define LabSite    / display order;
define Visit      / display order;
define VisitDate  / display format=yymmdd10.;

* Data columns;

define Haemoglobin_Result / display;
define Haematocrit_Result / display;
define RBC_Result         / display;
define WBC_Result         / display;
define Lymphocytes_Result / display;
define Monocytes_Result   / display;
define Neutrophils_Result / display;
define Eosinophils_Result / display;
define Basophils_Result   / display;
define Platelets_Result   / display;

* Hidden columns containing the range flags;

define Haemoglobin_Flag / display noprint;
define Haematocrit_Flag / display noprint;
define RBC_Flag         / display noprint;
define WBC_Flag         / display noprint;
define Lymphocytes_Flag / display noprint;
define Monocytes_Flag   / display noprint;
define Neutrophils_Flag / display noprint;
define Eosinophils_Flag / display noprint;
define Basophils_Flag   / display noprint;
define Platelets_Flag   / display noprint;

* Dummy column for style assignment;

define dummy / computed noprint;

* Traffic light the data columns based on the hidden columns;

compute dummy;

* Place holder;

endcomp;

* Insert a blank line between treatments;

compute after Treat;
  line ' ';
endcomp;

run; quit;

ods tagsets.ExcelXP close;

```

DIAGNOSING EXCEL LOAD ERRORS

When you open a malformed XML file using Excel, you are presented with an error dialog box containing a message like this:

```
This file cannot be opened because of errors. Errors are listed in: C:\Documents and Settings\userid\Local Settings\Temporary Internet Files\Content.MSO\404A54C0.log.
```

On most systems, the log file created by Excel is stored in a hidden directory, so you cannot navigate to it using Microsoft Windows Explorer. You can, however, navigate to the directory from a command prompt (cmd.exe) by typing this text and pressing the Enter key (ignore line wrapping):

```
%SYSTEMROOT%\explorer.exe /e, "%USERPROFILE%\Local Settings\Temporary Internet Files\Content.MSO"
```

The malformed XML is often created because of a typographical error in the name of the style element used in a style override, or by specifying a style element that is not part of the style definition. Here is an example of the first case, where the "data_bold" style element is defined in the "XLSansPrinter" style, but it is misspelled in the style override as "date_bold":

```
ods tagsets.ExcelXP file='LabReport.xml' style=XLSansPrinter ...;
...
define Treat / display order style(Column)=date_bold;
...
ods tagsets.ExcelXP close;
```

Here is an example of the second case, where the "data_bold" style element is spelled correctly, but it is not defined in the "sansPrinter" style that is supplied by SAS. The style element exists in the user-defined "XLSansPrinter" style, but you forgot to change the style name from "sansPrinter" to "XLSansPrinter":

```
ods tagsets.ExcelXP file='LabReport.xml' style=sansPrinter ...;
...
define Treat / display order style(Column)=data_bold;
...
ods tagsets.ExcelXP close;
```

In both instances, the Excel log file contains the following information, indicating that "unknown" is an invalid value for the "StyleID" attribute of the "Cell" tag:

```
XML ERROR in Table
REASON: Bad Value
FILE: C:\HOW\DelGobbo\LabReport.xml
GROUP: Row
TAG: Cell
ATTRIB: StyleID
VALUE: unknown
```

Open the file "LabReport.xml" using a text editor, and you will see instances of the incorrect value being used:

```
<Cell ss:StyleID="unknown" ... >
```

REFERENCES

- DelGobbo, Vincent. 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 at <http://www2.sas.com/proceedings/sugi27/p174-27.pdf>.
- DelGobbo, Vincent. 2003. "A Beginner's Guide to Incorporating SAS® Output in Microsoft® Office Applications". *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*, 28. CD-ROM. Paper 52. Available at <http://www2.sas.com/proceedings/sugi28/052-28.pdf>.
- DelGobbo, Vincent. 2004. "From SAS® to Excel via XML". Available at <http://support.sas.com/saspresents#excelxml>.
- DelGobbo, Vincent. 2009. "More Tips and Tricks for Creating Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®". Available at <http://support.sas.com/resources/papers/proceedings09/152-2009.pdf>.
- Microsoft Corporation. 2001. "XML Spreadsheet Reference". Available at [http://msdn2.microsoft.com/en-us/library/aa140066\(office.10\).aspx](http://msdn2.microsoft.com/en-us/library/aa140066(office.10).aspx).
- SAS Institute Inc. 2008. "SAS®9 Reporting Procedure Styles Tip Sheet". Available at <http://support.sas.com/rnd/base/ods/scratch/reporting-styles-tips.pdf>.
- SAS Institute Inc. 2008. *SAS/IntrNet® 9.2: Application Dispatcher*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/cdl/en/dispatch/59547/HTML/default/main_contents.htm.
- SAS Institute Inc. 2009. *SAS® 9.2 Output Delivery System: User's Guide*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/odsug/61723/HTML/default/a002565239.htm>.
- SAS Institute Inc. 2009. *SAS® 9.2 Stored Processes: Developer's Guide*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/stpug/61271/HTML/default/a003152554.htm>.
- SAS Institute Inc. 2009. "Uses of the Results Window". *SAS® 9.2 Language Reference: Concepts*. Cary, NC: SAS Institute Inc. Available at <http://support.sas.com/documentation/cdl/en/lrcon/61722/HTML/default/a003107493.htm#a003107660>.
- SAS Institute Inc. 2010. "ODS MARKUP Resources". Available at <http://support.sas.com/rnd/base/topics/odsmarkup/>.
- SAS Institute Inc. 2010. SAS Usage Note 11206. "IS8601* FORMATS and INFORMATS for DATE, TIME, and DATETIME". Available at <http://support.sas.com/kb/11/206.html>.

ACKNOWLEDGMENTS

The author would like to thank the following individuals for their valuable contributions to this paper: Chris Barrett of SAS Institute Inc.; Richard Allen of Peak Statistical Services; Nancy Brucken of i3 Statprobe; Susan Fehrer of BioClin Inc.; John King of Ouachita Clinical Data Services Inc.; and Alissa Ruelle of PharmaNet Inc.

RECOMMENDED READING

- DelGobbo, Vincent. 2006. "Creating AND Importing Multi-Sheet Excel Workbooks the Easy Way with SAS®". Available at <http://www2.sas.com/proceedings/sugi31/115-31.pdf>.
- DelGobbo, Vincent. 2007. "Creating Multi-Sheet Excel Workbooks the Easy Way with SAS®". Available at <http://www2.sas.com/proceedings/forum2007/120-2007.pdf>.
- DelGobbo, Vincent. 2008. "Tips and Tricks for Creating Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®". Available at <http://www2.sas.com/proceedings/forum2008/192-2008.pdf>.
- DelGobbo, Vincent. 2010. "An Introduction to Creating Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®". Available at <http://www.sas.com/reg/gen/corp/867226?page=Resources>.
- DelGobbo, Vincent. 2010. "Vince DelGobbo's ExcelXP Tagset Paper Index". Available at <http://www.sas.com/events/cm/867226/ExcelXPPaperIndex.pdf>.

CONTACT INFORMATION

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

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.

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

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