The Devil Is in the Details: Styles, Tips, and Tricks That Make Your Microsoft Excel Output Look Great!

Eric Gebhart, SAS Institute, Cary, NC

ABSTRACT

The Microsoft Excel XP tagset has given us the ability to create nicely formatted output for Excel, but it's not perfect and it's not always easy to get those last details ironed out. The tagset has over 50 options, and can also be controlled by styles in numerous ways.

This paper will show some of the best styles available as well as several tips and tricks to get your Microsoft Excel output looking just the way you want.

INTRODUCTION

Excel reports are in high demand, and although Excel provides a less than an optimal way of reporting, the need to create Excel reports remains. Even with the current quality of the ExcelXP destination, many of the processes necessary to create Excel reports require that someone spends time in Excel purgatory, changing sheet names, adding formats, putting in new borders, and so on.

This paper is all about how to get out of Excel purgatory. It takes some setup, but the ExcelXP destination can help you create a push-button process that generates reports just the way you want them.

The Excel tagset has come a long way since its beginnings a few years ago. It now works well for most quick reports. It can get the column widths and row heights right most of the time, and usually it looks pretty good depending upon the style you choose to use.

It's easy to get your reports looking 90% the way you want them to look. The last 10% is the hard part. The devil is in the details. The Excel tagset itself has over 50 options to control its behavior and settings. There are also a number ways to use styles to affect how the spreadsheet ultimately looks.

Because of this, using the ExcelXP destination can require a more intimate understanding of how it works, and in some cases why it must work the way it does. This paper will point out some ODS styles which look better with Excel, and will address the most common problems that continually come my way.

A FEW HEAVENLY STYLES

The first step towards heavenly Excel reports is to choose a good style. The Minimal style seems to be fairly popular, but there are others that look nicer and present a good foundation to build even better styles that will put your reports right through those heavenly gates.

Here are a few styles that look quite good with Excel output. If what you want is basic black and white, Statistical is a very good choice, It's nicer than the Minimal style. If you want a little gray the Printer style works well. The rest of these styles have a bit of color, the Journal, Meadow, and Seaside styles are subtle--they only use a little color in the header text, and sometimes a very light matching color for the background and borders. Analysis, Watercolor, and Sketch use louder background colors either for the headers or for the data.

These are only a few of the styles that I have chosen. In SAS® 9.2 there are over 50 to choose from. It is worth taking a look to see how they look in Excel. Because Excel has limited colors some styles look, well, like hell (BarretsBlue for example). But many of them look nice from the start. To see all of your style choices this simple bit of code will list them all:

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

This example shows how to use a few styles at once, it uses destination IDs to enable multiple ExcelXP destinations to be open at once.
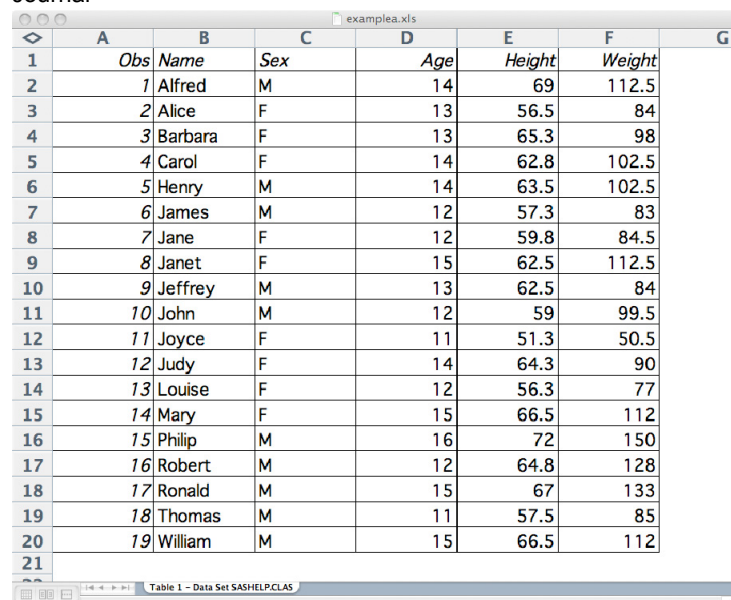
```
ods tagsets.Excelxp(a) file="examplea.xls" style=journal;
ods tagsets.Excelxp(b) file="exampleb.xls" style=meadow;
ods tagsets.Excelxp(c) file="examplec.xls" style=analysis;
ods tagsets.Excelxp(d) file="exampled.xls" style=statistical;
ods tagsets.Excelxp(e) file="examplee.xls" style=minimal;
ods tagsets.Excelxp(f) file="examplef.xls" style=printer;
ods tagsets.Excelxp(g) file="exampleg.xls" style=seaside;
ods tagsets.Excelxp(h) file="exampleh.xls" style=watercolor;
ods tagsets.Excelxp(i) file="examplei.xls" style=sketch;

proc print data=sashelp.class; run;

ods _all_ close;
```

These are just a few styles that look reasonable when used with Excel. There are others that also look quite nice. All it takes is a little experimentation to see which style might work best for you.

Journal

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

examplea.xls

Table 1 – Data Set SASHELP.CLAS

Meadow

exampleb.xls

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

Table 1 – Data Set SASHELP.CLAS

Analysis

examplec.xls

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

Table 1 – Data Set SASHELP.CLAS

Statistical

exampled.xls

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

Table 1 – Data Set SASHELP.CLAS

Minimal

examplee.xls

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

Table 1 – Data Set SASHELP.CLAS

Printer

examplef.xls

	A	B	C	D	E	F	
1	Obs	Name	Sex	Age	Height	Weight	
2	1	Alfred	M	14	69	112.5	
3	2	Alice	F	13	56.5	84	
4	3	Barbara	F	13	65.3	98	
5	4	Carol	F	14	62.8	102.5	
6	5	Henry	M	14	63.5	102.5	
7	6	James	M	12	57.3	83	
8	7	Jane	F	12	59.8	84.5	
9	8	Janet	F	15	62.5	112.5	
10	9	Jeffrey	M	13	62.5	84	
11	10	John	M	12	59	99.5	
12	11	Joyce	F	11	51.3	50.5	
13	12	Judy	F	14	64.3	90	
14	13	Louise	F	12	56.3	77	
15	14	Mary	F	15	66.5	112	
16	15	Philip	M	16	72	150	
17	16	Robert	M	12	64.8	128	
18	17	Ronald	M	15	67	133	
19	18	Thomas	M	11	57.5	85	
20	19	William	M	15	66.5	112	
21							
22							

Table 1 – Data Set: SASHELP.CLAS

Seaside

exampleg.xls

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

Table 1 – Data Set SASHELP.CLAS

5

Watercolor

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

Table 1 – Data Set SASHELP.CLAS

Sketch

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

Table 1 – Data Set SASHELP.CLAS

SHEET INTERVAL, SHEET NAME, SHEET LABEL...

By default the ExcelXP destination creates a new worksheet for every table created in the SAS output. This behavior can be changed by using the Sheet_Interval option. Valid values are Table, Page, Bygroup, Proc and None. The worksheets can be controlled explicitly by specifying that sheet_interval='none' Every time you want a new worksheet in the workbook. Pairing the sheet_name option with sheet_interval gives absolute control over the worksheet tab names.

The most common devilish detail has to do with sheet_interval='bygroup', Each sheet_interval has its own way of creating worksheet tab names, but bygroup is special. When the sheet_interval is set to 'bygroup' the tab names become var=value, which is great, but a lot of the time, all you really want is the by value. There is a very easy solution. Use the Sheet_Label option. Regardless of the Sheet_interval setting, the value of Sheet_label is

incorporated into the worksheet tab name. When it is set, it replaces a portion of the tab name being generated according to the Sheet_interval setting. By specifying that Sheet_label=' ' (a space), that particular portion of the worksheet's tab name will simply go away. Spaces are invalid in a tab name, and the tagset automatically removes them when they occur. The end result when sheet_interval='bygroup' and sheet_label=' ' is you get a tab name with just the value of the variable.

Here is the code:
```
proc sort data=sashelp.class out=example;
    by age sex;

ods tagsets.Excelxp file="example1.xls" style=analysis
    options(sheet_interval = 'bygroup'
            sheet_label = ' ');

proc print data=example;
    by age sex;
run;

ods _all_ close;
```

Notice the nice tab names that are created for this workbook.

The next step is to suppress the bylines and put in a nice title to replace the overly verbose byline. The Worksheet tab says the age, but all we really need to know is which table is which sex. Changing this, however, is problem full of gremlins, especially with the PRINT procedure. PROC PRINT does not provide byline metadata to the tagset like other procedures do, so the tagset has to figure out the by values by parsing the actual byline. That means that if you turn the bylines off, the output won't know when to create new worksheets and it won't know how to name them either. The ExcelXP tagset has an option, suppress_bylines, that is just for this specific problem. This option enables the tagset to get information from the bylines provided for PROC PRINT but then does not display them.

The next little gremlin to dispatch is to get the titles in the worksheet and in all the right places. The ExcelXP option, embedded_titles='yes', will cause the titles to be displayed above the tables. Without it, the titles would appear only when printing the workbook. But this wouldn't be a properly mischievous gremlin if that was all there was to it--there is more. PROC PRINT does its own paginating and it will display only the titles once for each pair of tables. In this case, what we really want is a title for each table. The solution is to use PAGEBY = SEX in PROC PRINT. PROC PRINT will then create a new page each time the value of sex changes. The ExcelXP tagset doesn't care about pages, it will do what it wants with the tables and the titles. Here is the code:

```
proc sort data=sashelp.class out=example;
    by age sex;

ods tagsets.Excelxp file="example1a.xls" style=analysis
    options(sheet_interval = 'bygroup'
            sheet_label = ' '
            embedded_titles = 'yes'
            suppress_bylines = 'yes');

title #byvar2 : #byval2;

proc print data=example;
    by age sex;
    pageby sex;
run;

ods _all_ close;
```
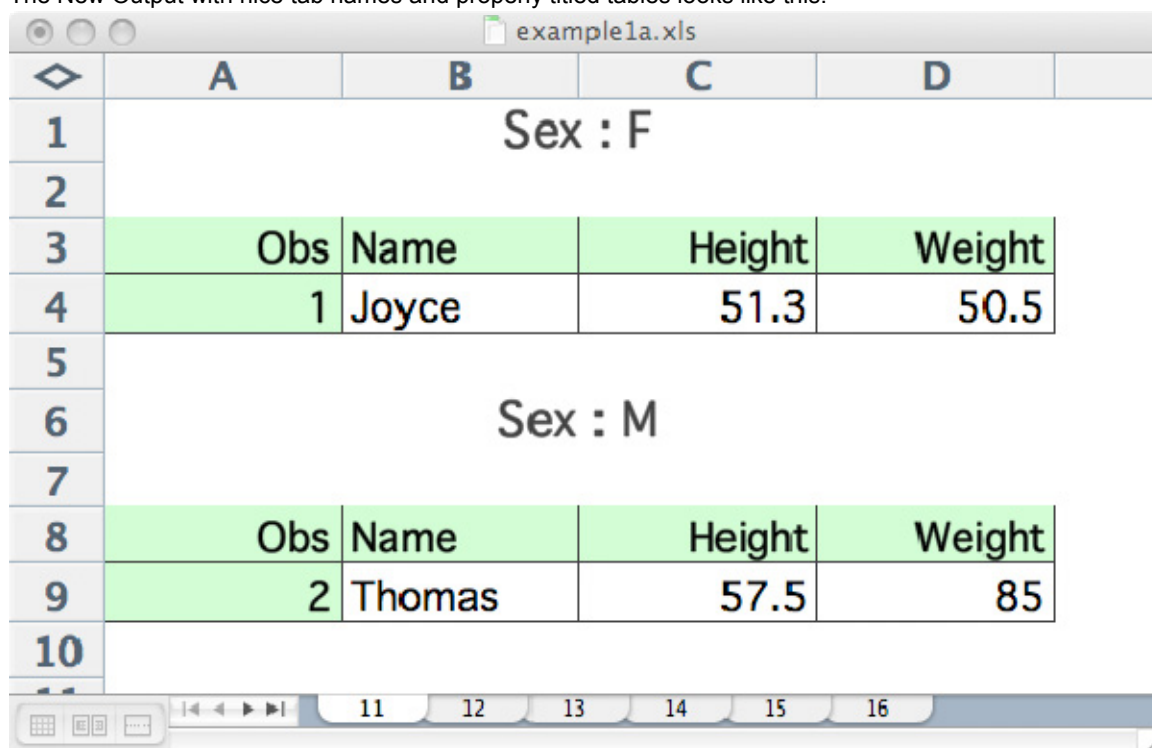
The New Output with nice tab names and properly titled tables looks like this.

CONTENTS, INDEX, FOOTNOTE, TITLE, AND A LINK BACK TO LIFE ABOVE

THE CONTENTS AND INDEX WORKSHEETS

When dealing with large workbooks it is sometimes nice to have a table of contents or an index of worksheets at the beginning of the worksheet. It is possible to do this with a simple REPORT procedure and a specially made data set, but if you already have your worksheet names the way you want them, it is much easier to use the contents and/or index options on the tagset. The Index option seems to be closer what most people want, as it provides a simple list

of worksheets in the first worksheet of the workbook. The Contents option creates a hierarchical table of contents similar to ODS HTML's table of contents or what you would see in the results viewer.
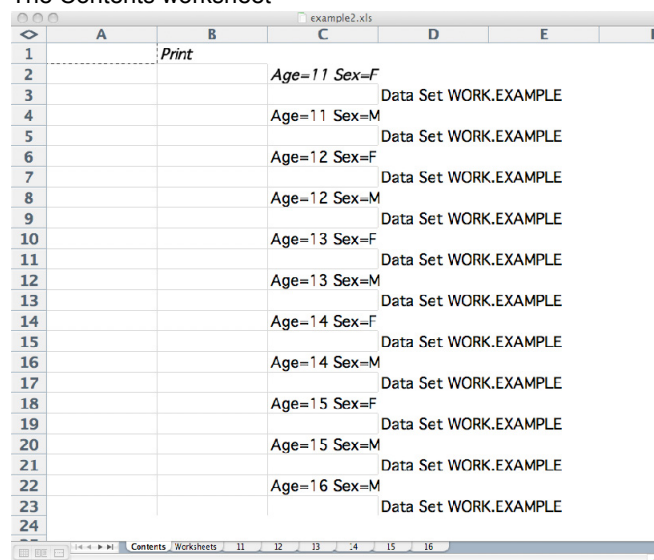
```
proc sort data=sashelp.class out=example;
    by age sex;

ods tagsets.Excelxp file="example2.xls" style=journal2
    options(sheet_interval = 'bygroup'
            sheet_label = ' '
            index='yes'
            contents='yes');

proc print data=example;
    by age sex;
run;

ods _all_ close;
```
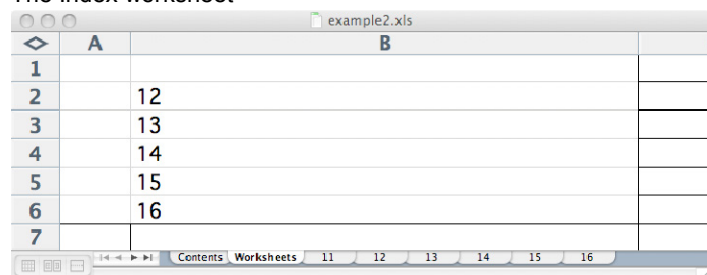
Each of the entries on these worksheets will take you to the corresponding worksheet.

The Contents worksheet

The Index worksheet

GETTING BACK

The Contents and Index worksheets solve only half of the problem of navigating a large workbook. After someone has navigated to the 40th tab of a workbook, they are stuck scrolling through tabs once again. There is an easy solution for this, which is to put a link in a footnote, title, or both! Just clicking on the title can bring you back to from hell and into the life above.

Here are two examples that show how to create a link back to the Contents worksheet and the Index worksheet. Both will return to cell A1 of their respective worksheets.

```
Title link="#Contents!A1" "Return to Contents";
Title1 link="#Index!A1" "Return to Index";

footnote link="#Contents!A1" "Return to Contents";
footnote1 link="#Index!A1" "Return to Index";
```

The devil is again in the details: adding these footnotes to your job won't do a thing unless you also add another option, embedded_footnotes='yes'. If you want titles, there is another option, embedded_titles='yes'. By default the titles and footnotes go to the print preview, not to the worksheet. If you want both, it's easiest to use embedded titles and footnotes and then use the print_header and print_footer options to set the headers and footers in the printer setup. Because of the way the PRINT procedure does paginating it is sometimes necessary to add some more tagset options or use the PRINT procedure's PAGEBY statement.

In this example, the links are placed in the titles, and to suppress the extra titles which would normally appear between each table, the ExcelXP option of embed_titles_once is used. This causes the tagset to suppress additional titles if they have already appeared on the current worksheet.

Expanding on the previous example we now have this code:

```
proc sort data=sashelp.class out=example;
     by age sex;

ods tagsets.Excelxp file="example3.xls" style=Journal3
    options(sheet_interval = 'bygroup'
            sheet_label = ' '
            index = 'yes'
            contents = 'yes'
            embedded_titles = 'yes'
            embed_titles_once = 'yes');

title link="#Contents!A1" "Return to Contents";
title2 link="#Worksheets!A1" "Return to Index";

proc print data=example;
     by age sex;
     pageby age;
run;

ods _all_ close;
```

DATES
Dates are something that inevitably come up and are somewhat difficult to manage when it comes to Excel. Our escape from Excel date purgatory requires a number of separate devilish details that need to be handled in concert.

The first problem is that Excel only understands datetime values. That means that SAS dates need to be converted to datetime values before Excel can do anything reasonable with them. The second problem is that Excel dates start at 0 on a different date than SAS dates do. So they have to be multiplied by 8640 before they will actually have the correct date in Excel. The third problem is that the resulting datetime value has to be formatted in ISO date format. A small detail of that is that there are two different formats to use depending upon your version of SAS. In SAS versions before SAS 9.2 the format is IS8601D. In SAS 9.2 that format is still allowed but the recommended format is E8601DT.

The fourth and fifth problems are that even after all of this, Excel won't know what to do unless we tell it explicitly what to do with the value. This corresponds to what would be done in the Format Cells dialog box in Excel. The type needs to be set to datetime and the format needs to be chosen or entered for that cell. The type and format are

specified either in the style definition or as a style override. Both values are placed in the tagattr style attribute. This is special behavior that the ExcelXP tagset has implemented, where the tagset looks in the tagattr style attribute for type, format, formula, rotate, and hidden keywords. For this example all we really need is Type and Format.

That is a lot of small details to get in a line so we can escape purgatory, but it's not too bad. Here is an example:

```
ods tagsets.excelxp file='example5.xls' style=analysis;

/* create a new DateTime column from the date column */
data buy;
    set sashelp.buy;
    datetime=date*86400;
run;

/* Pre-9.2, the format/informat name was IS8601DT . That name is still */
/* allowed in 9.2, although we are                                     */
/* now using the name E8601DT (E=Extended vs. B=Basic).                */


/* set the format for the DateTime coming from SAS */
/* and set the Excel type and format so Excel knows what to do with it */
proc print data=buy;
        format datetime e8601dt.;
    var datetime / style(data)={tagattr='type:DateTime format:YYYY-MM-DD'};
        var date;
    var amount;
run;

ods tagsets.excelxp close;
```
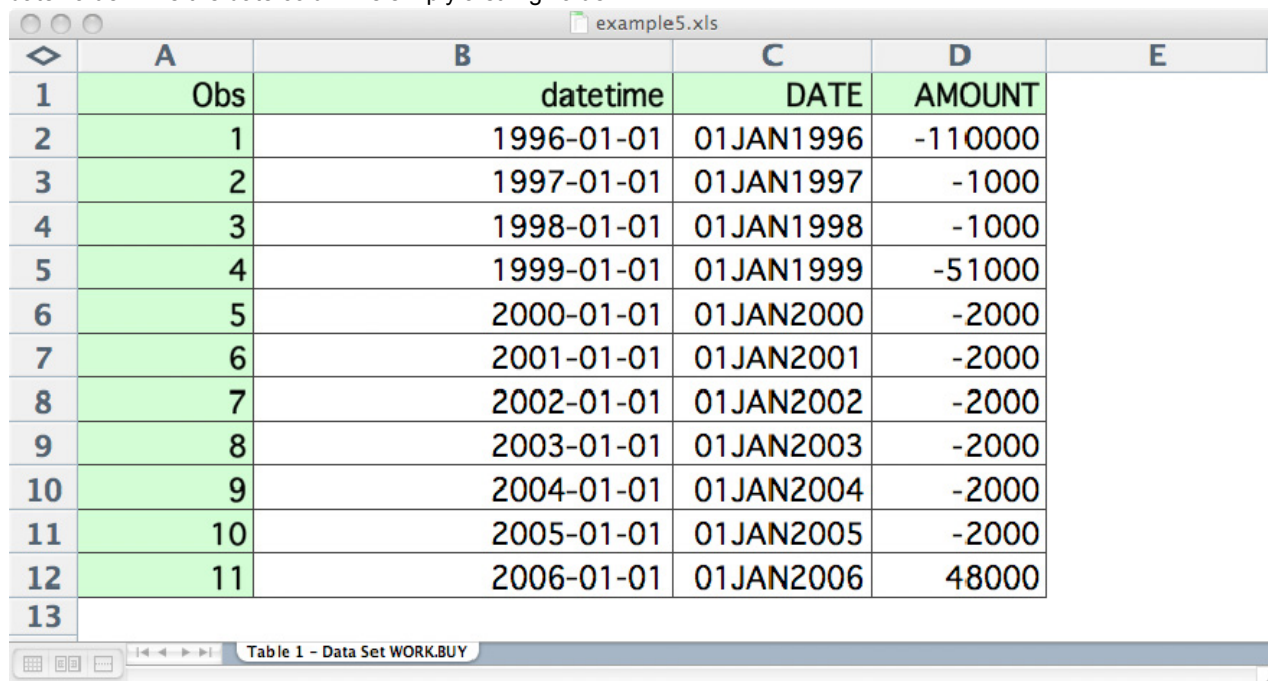
Both the datetime and date are shown in the output. The difference is that Excel sees the datetime column as a true date value while the date column is simply a string value.

	A	B	C	D	E
			example5.xls		
1	Obs	datetime	DATE	AMOUNT	
2	1	1996-01-01	01JAN1996	-110000	
3	2	1997-01-01	01JAN1997	-1000	
4	3	1998-01-01	01JAN1998	-1000	
5	4	1999-01-01	01JAN1999	-51000	
6	5	2000-01-01	01JAN2000	-2000	
7	6	2001-01-01	01JAN2001	-2000	
8	7	2002-01-01	01JAN2002	-2000	
9	8	2003-01-01	01JAN2003	-2000	
10	9	2004-01-01	01JAN2004	-2000	
11	10	2005-01-01	01JAN2005	-2000	
12	11	2006-01-01	01JAN2006	48000	
13					

Table 1 – Data Set WORK.BUY

OTHER FORMATS

Other Excel formats can be applied more easily. Mostly, all we need to do is give the Excel format to the tagset. The type is usually already set properly. One of the things that is important to understand is how Excel uses numbers. A number in Excel cannot have any symbols other than numbers and a decimal separator. That goes for currency values, too. So putting a Dollar format on your SAS data doesn't do a whole lot. What it does do is enable the tagset to detect that the value is currency and set the format for you. But the ExcelXP tagset must strip out any currency symbol, commas, percent signs, everything except the actual number. In the case of currency and percentages the tagset will set those formats for us, but if you desire anything fancy then it must be passed in as a style override much like was done in the last example with dates.

One nice thing about this is that all of those special things that can be done Excel formatting can be placed in your Excel format. The advantage over doing a format that way rather than with PROC FORMAT is that if anyone changes the Excel values, the Excel format will reflect those changes. For example, this format will make negative values turn red and add parenthesis around the value.

```
style = {tagattr='format:$#,##0_);[Red]\($#,##0\)'};
```

This format causes the cell to be a percentage with a single decimal.

```
style = {tagattr='format:0.0%'};
```

The key to making formats work is actually knowing how Excel formats work. It really doesn't have much to do with SAS. But once you know what you want, putting it into your SAS program will eliminate the pain of manually editing your Excel workbook later on.

This example also shows how to use a formula along with a format. Additionally, it also shows a few other things like Auto filters so the spreadsheet can be filtered to see just what you want. It also has frozen headers and frozen columns so that you can see the headers no matter how far you scroll to to the right or down. One last detail is to add some nice labels with split characters in the so that the headers will be more readable by humans, but I suppose you don't really need to go this far unless your boss is the devil incarnate.

```
   /*-- Create a difference column --*/
   /*--------------------------------*/
data prdsale;
  set sashelp.prdsale;
  Difference = actual-predict;
run;

proc sort data=prdsale;
  by country region division year;
run;


ods tagsets.excelxp file="examlple6.xls" style=meadow
                    options(autofilter='1-3' frozen_headers='2'
                            frozen_rowheaders='4' auto_subtotals='yes');


   /*-------------------------------------------------------*/
   /*-- Use Excel formulas to represent computed cells,  --*/
   /*-- and use an Excel format to force Excel to show   --*/
   /*-- negative currency values in red and with the     --*/
   /*-- format ($nnn).  In the formula below, the RC     --*/
   /*-- value corresponds to the cell relative to the    --*/
   /*-- current cell.  For example, RC[-2] means "2      --*/
   /*-- cells to the left of the current cell".  Any     --*/
   /*-- valid Excel formula can be used, and the formula --*/
   /*-- used here matches the computation performed      --*/
   /*-- in the DATA step that created the column.        --*/
   /*-------------------------------------------------------*/
```

```
title2 'Print of data using tagattr with formats';
title3 'predict & actual - ';
title4 'difference - ';
title5 'Sums - ';
title6 'Adding labels to prodtype, predict and actual';

 proc print data=prdsale noobs label split='*';
   id country region division;
   var prodtype product quarter month year;

   var predict actual / style={tagattr='format:$#,##0_);[Red]\($#,##0\)'};

   var difference /
         style={tagattr='format:$#,##0_);[Red]\($#,##0\) formula:RC[-1]-RC[-2]'};

   sum predict actual difference /
         style={tagattr='format:$#,##0_);[Red]\($#,##0\)'};;

   label prodtype = 'Product*Type'
         predict  = 'Predicted*Sales*For Area'
         actual   = 'Actual*Sales*Amount';
 run;


   ods tagsets.excelxp close;
```

It is difficult to see in a static image, but this fairly wide table has been scrolled so that only the headers and the last three columns are showing. Filters have also been applied to show Germany, West and Education. The nice split headers are readable and the fancy Excel format that turns negative numbers (red) can be seen in the last column.

	A	B	C	D	I	J	K
1	Country	Region	Division	Product Type	Predicted Sales For Area	Actual Sales Amount	Difference
842	GERMANY	WEST	EDUCATION	FURNITURE	$739	$982	$243
843	GERMANY	WEST	EDUCATION	FURNITURE	$683	$593	($90)
844	GERMANY	WEST	EDUCATION	FURNITURE	$610	$702	$92
845	GERMANY	WEST	EDUCATION	FURNITURE	$248	$528	$280
846	GERMANY	WEST	EDUCATION	FURNITURE	$530	$873	$343
847	GERMANY	WEST	EDUCATION	FURNITURE	$889	$301	($588)
848	GERMANY	WEST	EDUCATION	FURNITURE	$245	$769	$524
849	GERMANY	WEST	EDUCATION	FURNITURE	$473	$724	$251
850	GERMANY	WEST	EDUCATION	FURNITURE	$938	$466	($472)
851	GERMANY	WEST	EDUCATION	FURNITURE	$150	$774	$624
852	GERMANY	WEST	EDUCATION	FURNITURE	$772	$111	($661)
853	GERMANY	WEST	EDUCATION	FURNITURE	$201	$954	$753
854	GERMANY	WEST	EDUCATION	FURNITURE	$239	$597	$358
855	GERMANY	WEST	EDUCATION	FURNITURE	$637	$649	$12
856	GERMANY	WEST	EDUCATION	FURNITURE	$938	$3	($935)
857	GERMANY	WEST	EDUCATION	FURNITURE	$788	$731	($57)

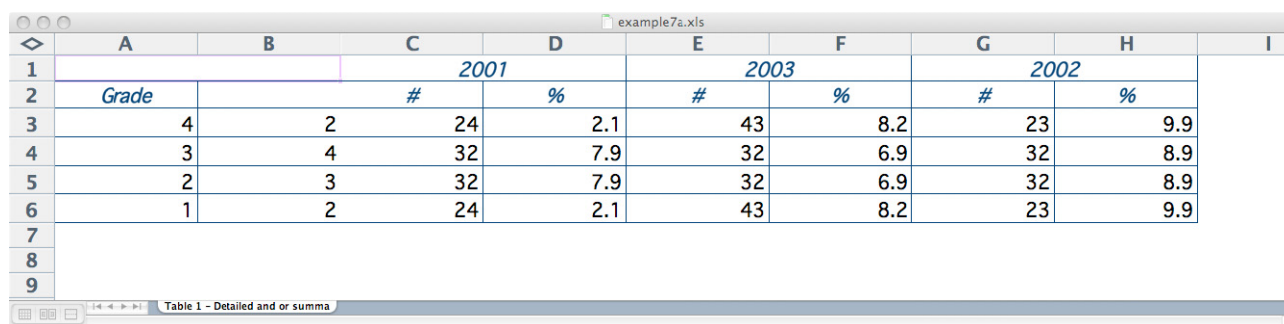Table 1 – Data Set WORK.PRDSALE   Table 2 – Data Set WORK.PRDSALE

This example could be made even more beautiful with the addition of some well placed borders and the removal of other borders, but that is for the next example.

13

BORDERS AND FINISHING TOUCHES

With Excel, cell borders can make all the difference in readability. Unfortunately borders are difficult to deal with because Excel doesn't have the concept of a table like HTML or RTF or even LaTeX. In Excel the only thing that can have borders is an individual cell. This complicates things quite a bit. At one time the ExcelXP tagset used the table style for the entire worksheet. That worked okay, but frequently gave the wrong background colors and caused the cell outlines to show through out the entire worksheet. The last few versions of the Excel tagset (since version 1.52 was released on 03/05/07) has changed that behavior. Now the body style is used for the worksheet and the only thing the table style is used for is to look for possible border settings that are automatically applied to the cell styles. Any style with 'data' or 'header' as part of its name will get this treatment.

That behavior makes things a little more complicated, but if you create a custom data or header style element, just make sure that data or header is part of the name and everything will work just fine.

This example uses the journal style as a parent. Here is what the output for this basic SAS program looks like in the journal style.

	A	B	C	D	E	F	G	H	I
			2001		*2003*		*2002*		
1			*2001*		*2003*		*2002*		
2	*Grade*		*#*	*%*	*#*	*%*	*#*	*%*	
3	4	2	24	2.1	43	8.2	23	9.9	
4	3	4	32	7.9	32	6.9	32	8.9	
5	2	3	32	7.9	32	6.9	32	8.9	
6	1	2	24	2.1	43	8.2	23	9.9	
7									
8									
9									

Table 1 – Detailed and or summa

The journal style already has some borders applied, they are thin, black lines with a width of 1. Valid widths in Excel are 1-4. To make this look nicer it just needs a few style elements for the headers and data cells that have a little thicker border lines in just the right places. This style uses the same dark cyan that the headers use as the color for the thicker lines.

This isn't particularly difficult, but it is a bit like a jigsaw puzzle. A new style needs to be created for each type of border on the cells. Some headers need a thick border on the right, some on the bottom and right. A few of the data cells need a border on the right. It is sometimes easier to build them one at a time and let the later style elements inherit from the earlier ones. If the style is specified for the borders in a generic 'do all the borders' way then the thin borders need to be added back in to the new style definitions. This can be seen in the data_bottom style, which is then used as a parent for the data_bottom_right style. The bottom border has a width of 1 while the right border has a width of 2. That gives the desired effect of looking just like the other data cells but with a thicker, colored border on the right. Here is the style definition that is used to tweak the borders.

```
proc template;
   define style styles.journal_borders;
      parent = styles.journal;

      style header from header/
          font_weight = bold
          foreground = cx006666
      ;


      style header_right from header /
          borderrightstyle=solid
          borderrightcolor=cx006666
          borderrightwidth=2
      ;

      style header_bottom_right from header_right /
          borderbottomstyle=solid
```

14

```
               borderbottomcolor=cx006666
               borderbottomwidth=2
           ;

       style header_thinbottom_right from header_right /
               borderbottomstyle=solid
               borderbottomcolor=cx006666
               borderbottomwidth=1
           ;
       style header_bottom from header /
               borderbottomstyle=solid
               borderbottomcolor=cx006666
               borderbottomwidth=2
           ;

       style data_bottom from data /
               borderbottomstyle=solid
               borderbottomcolor=black
               borderbottomwidth=1
           ;
       style data_bottom_right from data_bottom /
               borderrightstyle=solid
               borderrightcolor=cx006666
               borderrightwidth=2
           ;
     end;
  run;
```

One last minor point is the addition of the zoom option. Zoom causes the worksheet to open at the desired magnification. This is very nice to use if you have a large table that you want to be scaled down for easier viewing. Scale is a related option which controls the size of the printed worksheet. It and every other print option are available through the tagset options, so you can set up your workbook to print just the way you want no matter who you send it to. Here is the rest of the program:

```
data prof;
 input grade rpass year count percent;
cards;
1 2 2002 23 9.9
1 2 2003 43 8.2
1 2 2001 24 2.1
2 3 2002 32 8.9
2 3 2001 32 7.9
2 3 2003 32 6.9
3 4 2002 32 8.9
3 4 2001 32 7.9
3 4 2003 32 6.9
4 2 2002 23 9.9
4 2 2003 43 8.2
4 2 2001 24 2.1
;

ods tagsets.ExcelXP file='example7.xls' style=journal_borders options(zoom="200");

proc report data = prof nowd;
    column grade grade=grade2 rpass year,(count percent);
    define grade / group order=data descending 'Grade'
                            style(column)=header
```

15

```
                                style(header) = header_bottom;
     define grade2 / noprint;
     define rpass / group ' ' order=internal
                              style(column) = header_thinbottom_right
                              style(header) = header_bottom_right;
     define year / across order=data descending ' '
                              style(header)=header_right;
     define count / '#'   style(header) = header_bottom;
     define percent / '%' style(header) = header_bottom_right
                              style(column) =
 data_bottom_right{tagattr='format:0.0%'};

  run;


  ods tagsets.ExcelXP close;
```
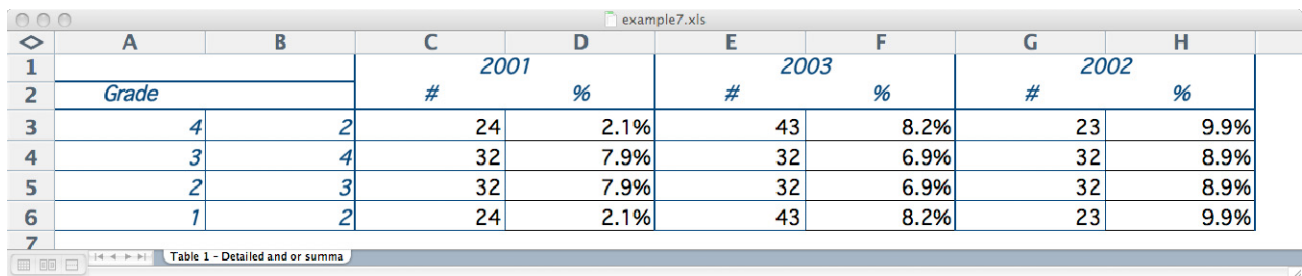
Here is what the new output looks like.

	Grade	2001		2003		2002	
		#	%	#	%	#	%
4	2	24	2.1%	43	8.2%	23	9.9%
3	4	32	7.9%	32	6.9%	32	8.9%
2	3	32	7.9%	32	6.9%	32	8.9%
1	2	24	2.1%	43	8.2%	23	9.9%

Table 1 – Detailed and or summa

SOME FINAL DETAILS

There are still some more things that can be done to make this a little nicer. Alternating row colors could make this report easier to read, and a better worksheet name would also be nice. Alternating colors is not as easy as you might expect in this case. The problem is all those special borders. What would be nice is if the tagset would handle that for us, but currently that is not the case so it has to be done with the REPORT procedure. This is only really possible with SAS 9.2. PROC PRINT introduced a new feature called STYLE/MERGE attribute name option in SAS 9.2. Unfortunately it only knows how to merge overrides, not actual styles, so it can be a bit painful, especially for this case.

The solution is to create macro variables that match the styles that we've already created.

```
  %let data_bottom = borderbottomstyle=solid borderbottomcolor=black
                    borderbottomwidth=1;

  %let data_bottom_right = &data_bottom borderrightstyle=solid
                        borderrightcolor=cx006666 borderrightwidth=2;

  %let header = foreground = cx006666 font_style = italic;

  %let header_thin_bottom_right = &header
                              borderrightstyle=solid borderrightcolor=cx006666
                              borderrightwidth=2
                              borderbottomstyle=solid borderbottomcolor=cx006666
                              borderbottomwidth=1 ;
```

These macro variables can then be used in REPORT procedure's CALL DEFINE statement to make sure that the cells get the borders and style we want, on top of any row style that might be applied to the entire table rows. The first step is to create a compute block to change the style every other time. This example uses a MOD function for the grade variable to set the background color to light blue.

16

```
    compute grade2;
        if mod(grade2, 2) = 0 then do;
            call define (_row_, "style/merge", "style = [background = CXCCFFFF]");
        end;
    endcomp;
```

This is the tricky part. If there had been no border changes this part wouldn't be necessary, but what happens is that the CALL DEFINE statement for the row ignores the actual styles applied to each cell and assumes a style of data. The result is that all the customizations are lost. This can be fixed by creating a compute block for each column with special borders. By using the macro variables the STYLE/MERGE option can merge the _row_ style with the over rides for each column.

```
    compute grade;
        call define(_col_, "style/merge", "style=[&header]");
    endcomp;

    compute rpass;
        call define(_col_, "style/merge", "style=[&header_thin_bottom_right]");
    endcomp;

    compute percent;
        call define(_col_, "style/merge", "style=[&data_bottom_right
                                            tagattr='format:0.0%']]");
    endcomp;
```

Here is the entire program:
```
    proc template;
        define style styles.journal_borders;
            parent = styles.journal;

            style header from header/
                font_weight = bold
                foreground = cx006666
            ;


            style header_right from header /
                borderrightstyle=solid
                borderrightcolor=cx006666
                borderrightwidth=2
            ;

            style header_bottom_right from header_right /
                borderbottomstyle=solid
                borderbottomcolor=cx006666
                borderbottomwidth=2
            ;

            style header_thinbottom_right from header_right /
                borderbottomstyle=solid
                borderbottomcolor=cx006666
                borderbottomwidth=1
            ;
            style header_bottom from header /
                borderbottomstyle=solid
                borderbottomcolor=cx006666
                borderbottomwidth=2
            ;
```

17

```
        style data_bottom from data /
               borderbottomstyle=solid
               borderbottomcolor=black
               borderbottomwidth=1
        ;
        style data_bottom_right from data_bottom /
               borderrightstyle=solid
               borderrightcolor=cx006666
               borderrightwidth=2
        ;
    end;
run;


%let data_bottom = borderbottomstyle=solid borderbottomcolor=black
borderbottomwidth=1;

%let data_bottom_right = &data_bottom borderrightstyle=solid
borderrightcolor=cx006666 borderrightwidth=2;

%let header = foreground = cx006666 font_style = italic;

%let header_thin_bottom_right = &header
                               borderrightstyle=solid borderrightcolor=cx006666
borderrightwidth=2
                               borderbottomstyle=solid borderbottomcolor=cx006666
borderbottomwidth=1 ;



data prof;
 input grade rpass year count percent;
cards;
1 2 2002 23 9.9
1 2 2003 43 8.2
1 2 2001 24 2.1
2 3 2002 32 8.9
2 3 2001 32 7.9
2 3 2003 32 6.9
3 4 2002 32 8.9
3 4 2001 32 7.9
3 4 2003 32 6.9
4 2 2002 23 9.9
4 2 2003 43 8.2
4 2 2001 24 2.1
;

ods tagsets.ExcelXP file='example8.xls' style=journal_borders options(sheet Name=
"Grade Passrate" :zoom="200");

proc report data = prof nowd;
     column grade grade=grade2 rpass year,(count percent);
     define grade / group order=data descending 'Grade'
                    style(column)=header
                    style(header) = header_bottom;
     define grade2 / noprint;
     define rpass / group ' ' order=internal
                    style(header) = header_bottom_right;
     define year / across order=data descending ' '
                    style(header)=header_right;
     define count / '#'    style(header) = header_bottom;
```

```
        define percent / '%' format=percent. style(header) = header_bottom_right ;

    compute grade2;
        if mod(grade2, 2) = 0 then do;
            call define (_row_, "style/merge", "style = [background = CXCCFFFF]");
        end;

    endcomp;

    compute grade;
        call define(_col_, "style/merge", "style=[&header]");
    endcomp;


    compute rpass;
        call define(_col_, "style/merge", "style=[&header_thin_bottom_right]");
    endcomp;

    compute percent;
        call define(_col_, "style/merge", "style=[&data_bottom_right
                                           tagattr='format:0.0%']");
    endcomp;

run;

ods tagsets.ExcelXP close;
```

The result is much easier to read.

A	B	C	D	E	F	G	H
		2001		2003		2002	
Grade		#	%	#	%	#	%
4	2	24	2.0%	43	8.0%	23	10.0%
3	4	32	8.0%	32	7.0%	32	9.0%
2	3	32	8.0%	32	7.0%	32	9.0%
1	2	24	2.0%	43	8.0%	23	10.0%

CONCLUSION

The ODS ExcelXP destination creates reasonably good output that doesn't need too much hand editing afterward, but to really get exceptional reports that don't require some Excel work requires a few tricks that can get your output out of purgatory and into heaven. Picking a nice style, setting the sheet names, applying a few formats, and adding a little extra style go a long way towards making your ExcelXP output a push-button process. Excel might represent purgatory but your Excel reports and the processes that create them can reside in Heaven.

CONTACT INFORMATION

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

Eric Gebhart
SAS Institute
SAS Campus Drive
Cary, NC 27513
Work Phone: 919-677-8000
E-mail: Eric.Gebhart@sas.com