PharmaSUG2011 - Paper TT01

Don't Gamble with Your Output: How to Use Microsoft Formats with ODS

Cynthia L. Zender, SAS Institute, Inc., Cary, NC, USA

ABSTRACT

Are you frustrated when Excel does not use your SAS® formats for number cells? Do you lose leading zeros on ZIP codes or ID numbers? Does your character variable turn into a number in Excel? Don't gamble with your output! Learn how to use the HTMLSTYLE and TAGATTR style attributes to send Microsoft formats from SAS to Excel.

This paper provides an overview of how you can use the HTMLSTYLE attribute with HTML-based destinations and the TAGATTR attribute with the TAGSETS.EXCELXP destination to send Microsoft formats from SAS to Excel using the Output Delivery System (ODS) STYLE= overrides. Learn how to figure out what Microsoft format to use and how to apply the format appropriately with ODS. A job aid is included in the paper; it lists some of the most common Microsoft formats used for numeric data. The examples in this paper demonstrate PROC PRINT, PROC REPORT, and PROC TABULATE coding techniques. Other job aids are provided that list some of the most common style attributes used in STYLE= overrides and show how to investigate Microsoft formats.

INTRODUCTION

When you want to interface SAS with Excel, you first have to decide what you want to get into Excel: SAS data or SAS procedure output. If you want your SAS data, and only your SAS data in an Excel workbook, then your choices are to use PROC EXPORT or the SAS Excel LIBNAME engine. This method of sending SAS data to Excel actually "translates" your SAS proprietary format data file into proprietary Excel workbook/worksheet format.

If you want your SAS procedure output (such as from PROC REPORT, PROC GLM or PROC UNIVARIATE, etc.) placed into an Excel workbook, then you have two main choices:

1) Use the output option of your procedure to create an output data set, and then use PROC EXPORT or the SAS Excel LIBNAME engine; or
2) Use ODS to route your procedure output to an ODS destination file. Then, once you have your ODS destination file, use Excel to open and render the ODS destination file.

This paper is not about using the #1 method. This paper is about impacting Microsoft Excel's view of your output using ODS techniques. So, if you are going down the #2 road, you have to use an ODS destination that Excel knows how to open and render in workbook/worksheet format. Knowing what other file types Excel can open will point you to the ODS destination you should use, as shown in Table 1.

File Formats Excel Will Open	File Formats Excel Will Save	ODS Destination
All Web Pages (*.htm; *.html; *.mht; *.mhtml)	Web Page (*.htm; *.html) Single File Web Page (*.mht; *.mhtml)	HTML, HTML3 MSOFFICE2K, TAGSETS.MSOFFICE2K_X, TAGSETS.TABLEEDITOR PHTML, CHTML
XML Files (*.xml)	XML Spreadsheet 2003 (*.xml) Excel 2003 + higher XML Spreadsheet (*.xml) Excel 2002	TAGSETS.EXCELXP
Text Files (*.prn; *.txt; *.csv)	CSV (Comma delimited) (*.csv) Formatted Text (Space delimited) (*.prn)	CSV CSVALL TAGSETS.CSVBYLINE

Table 1. Other File Formats That Excel Can Open and Save

You might be saying to yourself, "Hey, I didn't know I could create an Excel file with ODS HTML. That's pretty cool." Yes, I agree, it is pretty cool. But now it's time for me to switch into curmudgeon mode for a nanosecond or two. I am duty bound to point out that you **never** create true, binary format Excel files with ODS. ODS creates ASCII text files—either HTML-based, XML-based or delimiter-based—that Excel knows how to open and render inside a worksheet. As far as I know, Excel has always been able to open comma-delimited files and tab-delimited files since the beginning of Microsoft Excel as a spreadsheet application. Excel has supported opening HTML files since Excel 97, and it has supported opening Spreadsheet Markup Language XML files since Excel 2002. Even if you name your

ODS result files with a file extension of '.XLS', you are merely fooling the Windows registry into launching Excel when the file icon is clicked. OK, curmudgeon time is over.

There actually are more than two choices to get your SAS data and procedure output into Excel, but they are more advanced methods and are beyond the scope of this paper. For example, there is the SAS Add-in for Microsoft Office, part of the SAS Platform for Business Analytics, which would allow you to directly open your data and stored process results into an Excel worksheet. Or, there are Microsoft technologies, like DDE, ODBC and OLE-DB, which you can use with SAS in order to transfer your data into Excel using Microsoft methods.

If you know how to use the basic "sandwich" technique with ODS, then you already know how to get your SAS procedure output into Excel. But this paper isn't about the basics of getting your output from SAS to Excel. This paper is about what happens after you open your output file with Excel. For example, as you're looking at your ODS results file in Excel, you notice that the leading zeros in your ID number are gone, that your percent signs do not show up or there are two decimal places, in spite of the fact that your SAS code used PERCENT8.1 as the format.. Then you notice that your dates are not in the format you wanted, with the separator you wanted, or even worse, that your date values appear to be about 60 years off. What happened?

What happened is that Excel has default ways to treat numeric variables and even if you have a variable that is a SAS character variable (like an ISBN, product ID or ZIP code), if the variable contains numbers, you might find that Excel still treats the column as though it is a numeric column, no matter what your SAS variable type is.

The good news is that you can influence the way that Excel treats your output by telling Excel what Microsoft-specific formats to use when Excel is rendering your output from HTML or XML format into workbook/worksheet format. Notice that I didn't say you could influence how Excel opened your CSV file. That's because a CSV file is the data and only the data—there's no way in a CSV file to send Microsoft formats to Excel. So the main focus of this paper will be to illustrate how to use Microsoft formats with HTML output from ODS and with SpreadsheetML XML output from ODS. For each topic, I will show you the data, and then show Excel's default treatment of output from SAS. Most of these programs use PROC REPORT to illustrate the default behavior, although you might experience the same issues with other SAS procedures. It doesn't matter which procedure you use, the "fix" will still be the same, using a STYLE= override.

MICROSOFT EXCEL DEFAULTS AND HOW TO CHANGE THEM

The examples in this gallery are pretty much self-explanatory. Each example shows the SAS data in PROC PRINT output first and then the default results (opened with Excel 2010) with the HTML output on the left (or sometimes first) and the XML output on the right (or sometimes second). Instead of showing all the possible HTML-based outputs from ODS, I chose to use ODS MSOFFICE2K because it is the ODS destination that conforms to the Microsoft HTML specification for HTML files. And, since ODS TAGSETS.EXCELXP is also a Microsoft XML specification, for an XML description of a workbook, we are assured that all our output files conform to Microsoft standards.

In the zip file of programs that you can download for this paper, you will discover that each demo program has a DATA step program that creates a small data file.

Excel Treats CHARACTER Variable as Numeric Column

In this scenario, we have the "Mom and Pop Bookstore," which has a small online business. Part of the information collected is information typed by customers, in which a book's ISBN information can be typed (by the customer) in a variety of formats, as shown in Display 1, which shows a PROC PRINT of the data for this example. Display 2 shows the default treatment of the ISBN column in an Excel 2010 workbook.

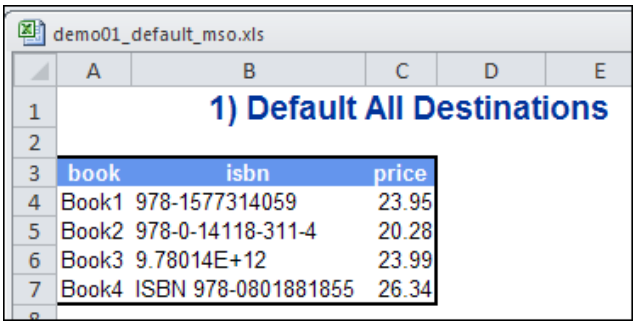
Obs	book	isbn	price
1	Book1	978-1577314059	23.95
2	Book2	978-0-14118-311-4	20.28
3	Book3	9780142000083	23.99
4	Book4	ISBN 978-0801881855	26.34

Display 1. ISBN Data Displayed in LISTING Destination

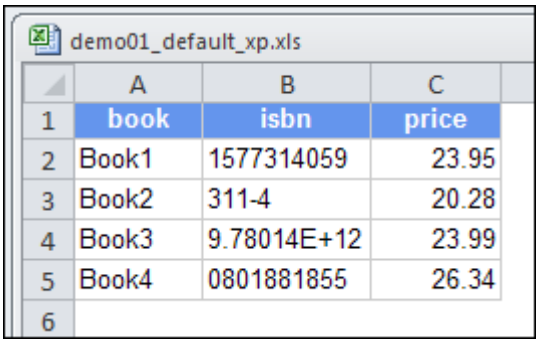
When you look at the default output, in Display 2, you will notice some difference between the HTML and the XML output. The title that shows in the HTML output is in the header area of the worksheet when the XML output is opened in Excel. The column width in the HTML output is OK, but the column width for the rendered XML output is not OK. Both outputs reveal the basic issue, that the ISBN number, for Book3, without any punctuation is displayed in scientific notation.

The way to fix this issue is the same for both destination outputs. Excel needs to be instructed to treat this mixed column as a TEXT column so that it doesn't use scientific notation for the large ISBN value. And, it would be nice if the HTML and XML output used a different column width.

HTML output



XML output



Display 2. Default Treatment of ISBN Column

The Excel column width used for the XML output file is not wide enough to display the whole ISBN value (Book2); and, although the HTML output is wide enough for the ISBN value, it might be useful to make that column wider as well. When you go into Excel and format a numeric column as text, the internal representation for a TEXT format is '@'. I'll talk more about how to figure that out after the code.

```
ods msoffice2k file='c:\temp\demo01_MSO.xls' style=sasweb;

proc report data=mom_and_pop_books nowd;
  title '2) HTMLSTYLE= Style Override';
  column book isbn price;
  define book / order;
  define isbn / display
    style(column)={htmlstyle="mso-number-format:'\@';width:'190pt'"} ;
  define price / sum;
run;

ods _all_ close;

ods tagsets.excelxp file='c:\temp\demo01_XP.xls' style=sasweb
    options(absolute_column_width='12,20,12');

proc report data=mom_and_pop_books nowd;
  title '3b) TAGATTR Style Override';
  column book isbn price;
  define book / order;
  define isbn / display
         style(column)={tagattr='Format:@'};
  define price / sum;
run;

ods _all_ close;
```

In the ODS MSOFFICE2K program, the HTMLSTYLE attribute sets the MSO-NUMBER-FORMAT style property to be an '@', which is the specification for a TEXT column. In addition, the width is set to 190pt, which makes the cell wider than it was in the original output. The ODS TAGSETS.EXCELXP output, on the other hand, uses either the Microsoft specification of 'Format:@' (or you could also use 'Format:Text ') for the TAGATTR value, while the suboption for width is specified in the ODS invocation statement. (A later example shows the use of the ODS style override for the WIDTH attribute instead of using the ABSOLUTE_COLUMN_WIDTH suboption.) The MSOFFICE2K output is shown

in Display 3, and the TAGSETS.EXCELXP output is shown in Display 4.

demo01_MSO.xls

	A	B	C
1		2) HTMLSTYLE= Style Override	
2			
3	book	isbn	price
4	Book1	978-1577314059	23.95
5	Book2	978-0-14118-311-4	20.28
6	Book3	9780142000083	23.99
7	Book4	ISBN 978-0801881855	26.34
8			

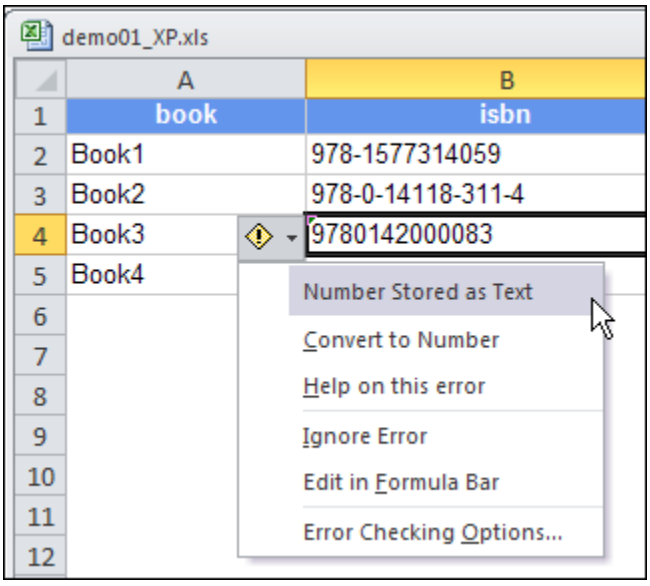
Display 3. ODS MSOFFICE2K Output

demo01_XP.xls

	A	B	C
1	book	isbn	price
2	Book1	978-1577314059	23.95
3	Book2	978-0-14118-311-4	20.28
4	Book3	9780142000083	23.99
5	Book4	ISBN 978-0801881855	26.34

Display 4. ODS TAGSETS.EXCELXP Output

The green triangle on the row for Book 3 means that Excel still feels the need to warn you that a number is being treated as TEXT. In fact, the popup window, as shown in Display 5, provides the options you have in Excel for how this cell should be treated.



Display 5. Popup Window for Book 3 ISBN Value

Notice that Excel considers this situation (of a "Number Stored as Text") to be an error condition. Your only choice at this point, is to select the "Ignore Error" choice or to pre-process the data so that there are hyphens in the ISBN value or every value is preceded by the string "ISBN".

Let's talk for minute about how you figure out that @ is the right format to use for text strings. To figure this out, we

either have to 1) read the Microsoft documentation or 2) reverse engineer the format by looking in the type of HTML or XML file that Microsoft generates. (Table A.3 at the end of this paper illustrates these steps in detail.)

If you go into a new Excel workbook and type a long number and then format the number as text, you have to manually save the worksheet as an HTML Web Page file, and then look into one of the HTML files in order to see the <TD> tag:

```
<td class=xl65 width=182 style='width:137pt'>9780142000083</td>
```

You can see the size to use for the WIDTH= attribute in the <TD> tag. You can also see, by the CLASS= attribute, which CSS class to look at in order to discover the correct MSO-NUMBER-FORMAT. The nice thing about looking in the HTML file for the worksheet of interest is that the format you find there will work almost unchanged in the TAGSETS.EXCELXP syntax. The relevant CSS code snippet (from the CSS file generated by Excel) is shown below.

```
.xl65
    {mso-style-parent:style0;
    mso-number-format:"\@";}
```

Now you can cut and paste the entire style property into the HTMLSTYLE attribute. For the XML file, you can either try the same string that you use for the HTMLSTYLE attribute or you can save your Excel worksheet as XML and look in the XML file, where you will see something like the XML snippet shown below:

```
<Style ss:ID="s65">
 <NumberFormat ss:Format="@"/>
</Style>
```

This <Cell> tag for the ISBN number references the style ID value in the following manner:

```
<Cell ss:StyleID="s65"><Data ss:Type="String">9780142000083</Data></Cell>
```

The backslash (\) character which is needed for the HTML format is generally not needed for the TAGATTR format. Also, if you look at the internal documentation for the TAGATTR style attribute, you can find some other useful formats, as shown in Display 6.

```
TagAttr Style Element:   Default Value ''
     Values: <ExcelFormat> or
     <Type: dataType>
     <Format: ExcelFormat>
     <Formula: ExcelFormula>
     <Rotate: degrees of rotation.>
     <Hidden: Yes/No.>
     <mergeacross: yes/No/number>
     This is not a tagset option but a style attribute that the tagset will
     use to get formula's and column formats. One caveat, is that there should be
     no space between the : and the value for any of these settings.
     The format and formula's given must be valid to excel.
     The rotation must be a valid angle for text, 90 through -90.
     The only recognized value for Hidden is yes.  When set to yes on any cell, that
     row will be hidden.
     MergeAcross is to force a cell to merge across the current width of the worksheet.
     Using a number will cause the cell to merge across that many columns.
     The Type should be General, String, Number, or DateTime. Excel is case sensitive
     It should be unnecessary to specify type except when DateTime is being used.
     Even when doing numbers as text format

     A single value without a keyword is interpreted as a format.
     A formula, format and rotation can be specified together with keywords.
     There should be no spaces except for those between the two values
     The keyword and value must be separated by a ':'
     tagattr='format:###.## formula:SUM(R[-4]C:R[-1]C rotate:90').

      Text ---- @

      Type = DateTime
      Time - 0:00         ----  Short Time
      Time - 0:00:0       ----  h:mm:s
      Time - 00:00.0      ----  mm:ss.0
      Time - 00:00 AM     ----  Medium Time
      Time - 12:00:00 AM ----  Long Time
      Time - 24:00:00     ----  [h]:mm/:s
      Time - 3/14/01 1:30 PM ----  m/d/yy\ h:mm\ AM/P

      Percentage - 6 decimals  ----  0.00000%
      Special - zip code       ----  00000
      Special - zip code + 4   ----  00000\-0000

      Scientific                     ---- Scientific
      Scientific - 4 decimals        ---- 0.0000E+00
      Fraction - As sixteenths (8/16) ---- #\ ??/1
```

Display 6. Internal Documentation for the TAGATTR Style Attribute

Learning how to reverse engineer Microsoft formats is not the main focus of this paper, however. So let's look at another example

Excel Does Not Display Leading Zeros for NUMERIC Variables

You can always use the SAS supplied *Zw.d* format to show leading zeros in SAS output. Or you can define your "leading zero" number as a character variable. However, when you have data like the data shown in Display 7, it doesn't matter whether you have a numeric variable formatted with leading zeros (TRANSID) or a character variable (CHAR_TRANSID).

Obs	dest	trans ID	char_ transID	amt	profit
1	CHICAGO	00010	00010	100	10.0%
2	CHICAGO	00222	00222	200	15.0%
3	GENEVA	00330	00330	300	12.5%
4	GENEVA	00444	00444	-400	17.5%
5	LONDON	00550	00550	500	20.0%
6	LONDON	06000	06000	600	22.0%
7	LONDON	00777	00777	700	24.0%
8	PARIS	00088	00088	800	26.0%
9	PARIS	00990	00990	900	27.0%

Display 7. SAS Output from PROC PRINT

For either variable, Excel strips leading zeros, as shown in Display 8.

HTML output

demo02_default_mso.xls

1) Default for Leading Zero and Percent

dest	transID	char_transID	amt	Pct Profit
CHICAGO	10	10	100	0.1
CHICAGO	222	222	200	0.15
GENEVA	330	330	300	0.125
GENEVA	444	444	-400	0.175
LONDON	550	550	500	0.2
LONDON	6000	6000	600	0.22
LONDON	777	777	700	0.24
PARIS	88	88	800	0.26
PARIS	990	990	900	0.27

XML output

demo02_default_XP.xls

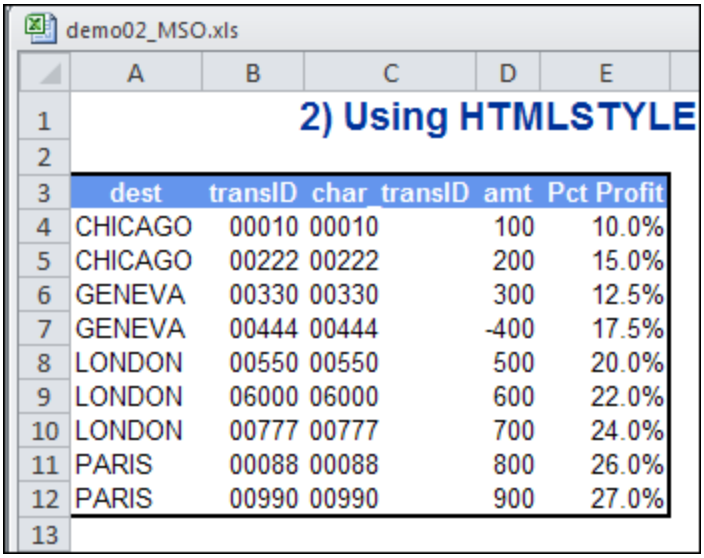
dest	transID	nsID	amt	Pct Profit
CHICAGO	10	10	100	0.1
CHICAGO	222	222	200	0.15
GENEVA	330	330	300	0.125
GENEVA	444	444	-400	0.175
LONDON	550	550	500	0.2
LONDON	6000	6000	600	0.22
LONDON	777	777	700	0.24
PARIS	88	88	800	0.26
PARIS	990	990	900	0.27

Display 8. Default Treatment of Leading Zeros

In addition, the SAS PERCENTw.d format for the PROFIT variable was not used by Excel. The code below shows how to change both of these defaults to the desirable formats.

```
ods msoffice2k file='demo02_MSO.xls' style=sasweb;

proc report data=ticket nowd;
  title '2) Using HTMLSTYLE';
  column dest transID char_transid amt profit;
  define transID /
         style(column)={htmlstyle="mso-number-format:'00000'"};
  define char_transid / display
         style(column)={htmlstyle="mso-number-format:'00000'"};
  define profit/ 'Pct Profit'
         style(column)={htmlstyle="mso-number-format:'###.0%';width='100pt'"};
run;

ods _all_ close;

ods tagsets.excelxp file='demo02_XP.xls' style=sasweb
    options(absolute_column_width='15');

proc report data=ticket nowd;
  title '3) Using TAGATTR';
  column dest transID char_transid amt profit;
  define transID / style(column)={tagattr='00000'};
  define char_transID / style(column)={tagattr='00000'};
  define profit/ 'Pct Profit'
         style(column)={tagattr='format:###.0%'};
run;

ods _all_ close;
```

The code shows the same type of format specification that has already been used. For the HTMLSTYLE= attribute, the MSO-NUMBER-FORMAT style property is used. For the TAGSETS.EXCELXP program, the string 'Format:' is not required in the syntax if the attribute value is a valid Microsoft format. In the above case, the '0' is a placeholder that instructs Excel to use a '0' if there is no number in the corresponding position of the number in the cell. The results of these programs are shown in Display 9 and Display 10.



demo02_MSO.xls

	A	B	C	D	E
1	2) Using HTMLSTYLE				
2					
3	dest	transID	char_transID	amt	Pct Profit
4	CHICAGO	00010	00010	100	10.0%
5	CHICAGO	00222	00222	200	15.0%
6	GENEVA	00330	00330	300	12.5%
7	GENEVA	00444	00444	-400	17.5%
8	LONDON	00550	00550	500	20.0%
9	LONDON	06000	06000	600	22.0%
10	LONDON	00777	00777	700	24.0%
11	PARIS	00088	00088	800	26.0%
12	PARIS	00990	00990	900	27.0%
13					

Display 9. HTML Output with Leading Zeros and Percents

demo02_XP.xls

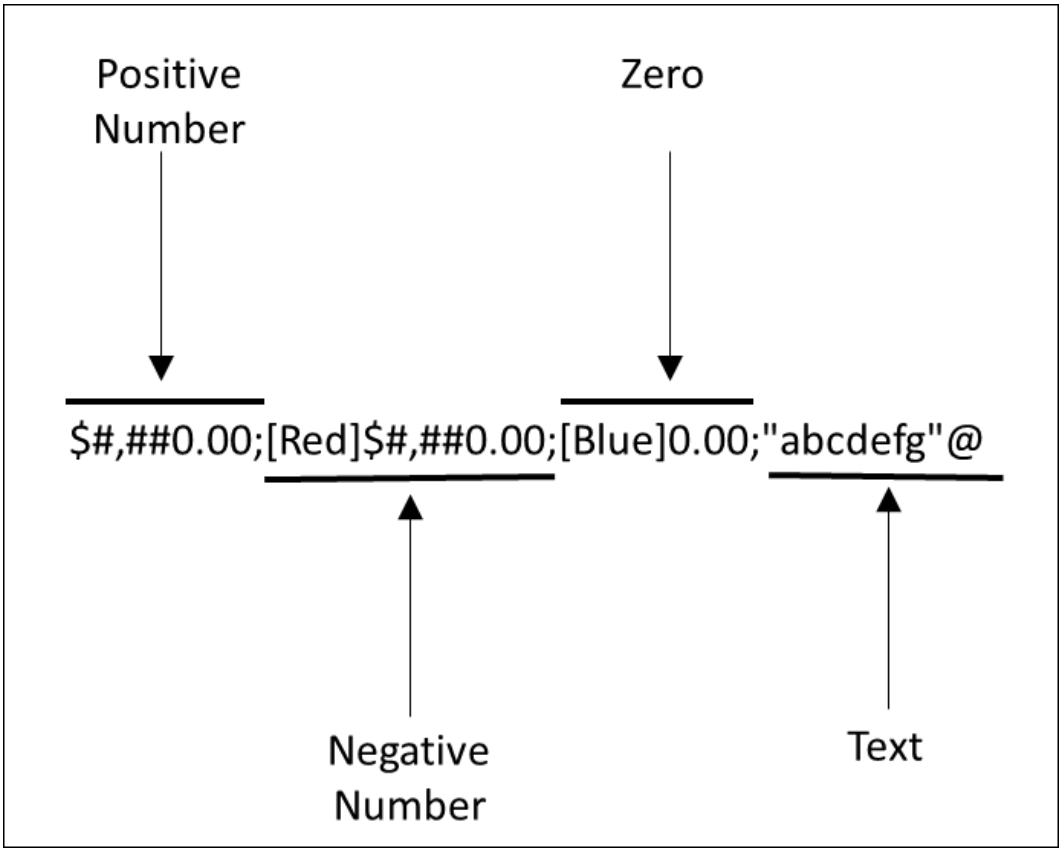
	A	B	C	D	E
1	dest	transID	char_transID	amt	Pct Profit
2	CHICAGO	00010	00010	100	10.0%
3	CHICAGO	00222	00222	200	15.0%
4	GENEVA	00330	00330	300	12.5%
5	GENEVA	00444	00444	-400	17.5%
6	LONDON	00550	00550	500	20.0%
7	LONDON	06000	06000	600	22.0%
8	LONDON	00777	00777	700	24.0%
9	PARIS	00088	00088	800	26.0%
10	PARIS	00990	00990	900	27.0%

Display 10. XML Output with Leading Zeros and Percents

The use of the ‘#’ in the Microsoft format for the percent plays a different role than the ‘0’ in the format specification. A "#" in an Excel format displays one or more numeric digits, excluding insignificant zeros, and the "0" instructs Excel to display a single numeric digit, including insignificant zeros. The SUGI 28 paper by Vincent DelGobbo, entitled “A Beginner's Guide to Incorporating SAS® Output in Microsoft® Office Applications” (http://www2.sas.com/proceedings/sugi28/052-28.pdf) contains a list of the common Microsoft formats and an explanation of how to specify Microsoft formats. (The list of common formats is in Table A.1.)

Excel Needs Custom Format for Negative Numbers

What if you wanted a different format for the negative number in the AMT column? Microsoft provides a way to format positive and negative numbers. The general model is shown below.



Display 11. General Model for Positive and Negative Format with Dollar Sign

A Microsoft format can have up to four sections of format specification. Each section of a format model is separated from the others by a semicolon. The four sections define the formats for positive numbers, negative numbers, zero values, and text, in that order. If you specify only two sections, the first is used for positive numbers and zeros, and the second is used for negative numbers. If you specify only one section, it is used for all numbers. If you skip a section, include the ending semicolon for that section. This technique may require some further investigation, since a SAS format doesn't operate in the same manner—where you can apply colors and formats in one shorthand notation system.

However, if the above format style is used with the HTMLSTYLE and the TAGATTR style attributes, you can achieve the desired results. Display 12 shows a variation of the TICKET data.

Obs	trans ID	dest	type	amt
1	1	CHICAGO	TEL	1100
2	2	CHICAGO	TEL	2200
3	3	GENEVA	WEB	3300
4	4	GENEVA	WEB	-4400
5	5	LONDON	TEL	0

Display 12. PROC PRINT Report of TICKET Data

HTML output

	dest	transID	amt
4	CHICAGO	1	1100
5	CHICAGO	2	2200
6	GENEVA	3	3300
7	GENEVA	4	-4400
8	LONDON	5	0

XML output

	dest	transID	amt
2	CHICAGO	1	1100
3	CHICAGO	2	2200
4	GENEVA	3	3300
5	GENEVA	4	-4400
6	LONDON	5	0

Display 13. Default Excel Treatment of Negative Numbers

Display 13 shows what we would normally expect to see when Excel uses the default format. But it would be nice to see the negative number in red and the zero in blue. Since you've seen a complete PROC REPORT step several times now, only the relevant DEFINE statement is shown for each destination in the code below.

Define Statement for MSOFFICE2K (HTML) Step:

```
    define amt /
      style(column)={htmlstyle="mso-number-format:
'\0022$\0022\#\,\#\#0\.00\;\[Red\]\(\0022$\0022\#\,\#\#0\.00\)\;\[Blue\]0\.00'"};
```

Notice how the HTML specification becomes quite lengthy, since every special character needs to be preceded with a backslash. This is because, in order to display both text and numbers in a cell, you need to enclose the text characters such as a "$" or even a ";" in double quotation marks (" ") or precede a single character with a backslash (\). In our example, the 0022 is the hex code for a double quotation mark. (This string for the HTMLSTYLE= attribute was modified only slightly from the CSS file that was created by Excel, after the custom format was manually created in Excel.)

Define Statement for TAGSETS.EXCELXP (XML) Step:

```
    define amt /
      style(column)={tagattr='format:$#,##0.00;[Red]($#,##0.00);[Blue]0.00;'};
```

The TAGATTR= value does not need the backslashes or the quotes around the dollar sign, so it is much easier to read. We can see this format in the <STYLE> section of a manually saved XML Spreadsheet 2003 file.

```
<NumberFormat
ss:Format="&quot;$&quot;#,##0.00;[Red]\(&quot;$&quot;#,##0.00\);[Blue]0.00"/>
```

The entity " is used in the XML file because it makes quoting easier in the CSS file, but as you can see in the TAGSETS.EXCELXP code, you can use a simple dollar sign without a double quotation mark in your TAGATTR= attribute value. That really simplifies the quoting issues. The final HTML output using the HTMLSTYLE= attribute is shown in Display 14, and the XML output using the TAGATTR= attribute is shown in Display 15.

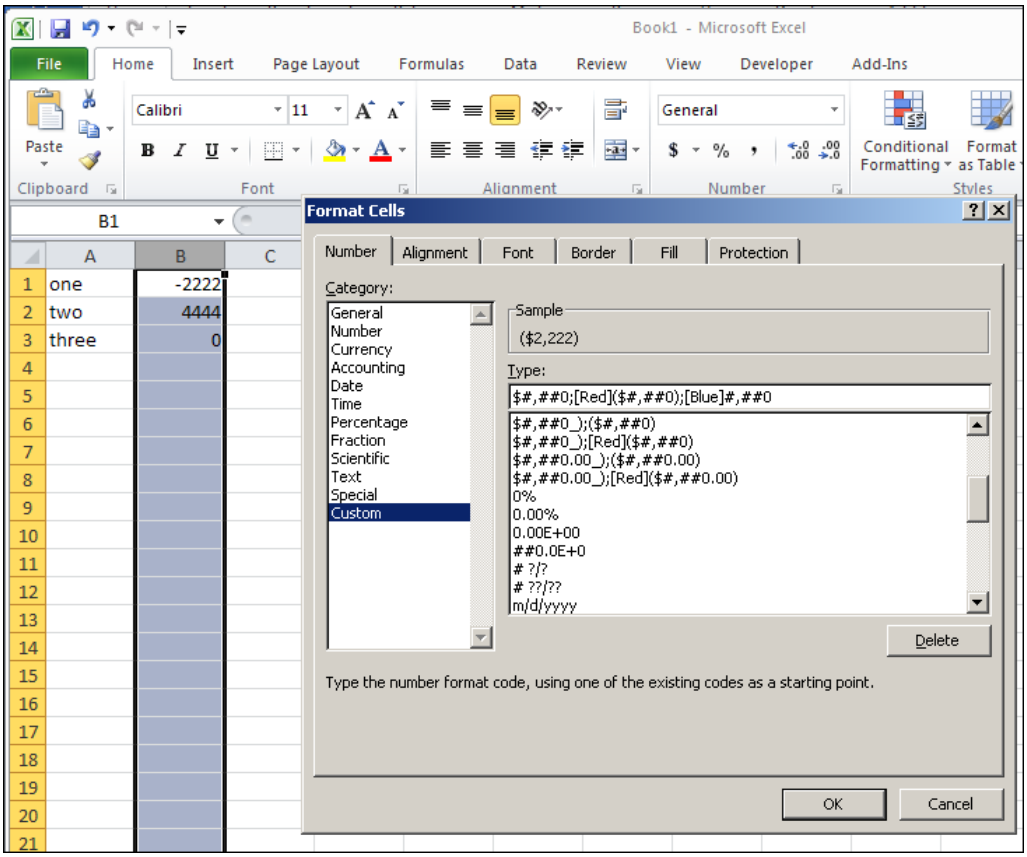
	dest	transID	amt
4	CHICAGO	00001	$1,100.00
5	CHICAGO	00002	$2,200.00
6	GENEVA	00003	$3,300.00
7	GENEVA	00004	($4,400.00)
8	LONDON	00005	0.00

Display 14. HTML Output Using HTMLSTYLE

	A	B	C
1	dest	transID	amt
2	CHICAGO	00001	$1,100.00
3	CHICAGO	00002	$2,200.00
4	GENEVA	00003	$3,300.00
5	GENEVA	00004	($4,400.00)
6	LONDON	00005	0.00

Display 15. XML Output Using TAGATTR

The way you create a custom format is with the Format Cells icon on the Excel ribbon, as shown in Display 16.



Display 16. Custom Format Dialog Box in Excel

Notice that when you build the format in the dialog box, you do not need to use any of the backslashes for the non-numeric characters. Excel puts those characters into the HTML or XML file when you save as those file types. There's not a lot more to say about custom formats. You have to figure out what you want by reading the documentation or "reverse engineering" as shown in Table A.3 at the end of this paper. This site is the original Microsoft Office HTML and XML reference, which contains the MSO-NUMBER-FORMAT information: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoffxml/html/ofxml2k.asp

I also found these sites to be very useful:

http://support.microsoft.com/kb/264372

http://support.microsoft.com/kb/214233

http://office.microsoft.com/en-gb/excel-help/create-or-delete-a-custom-number-format-HP005199500.aspx

http://cosicimiento.blogspot.com/2008/11/styling-excel-cells-with-mso-number.html

Excel Might Not Honor Date Values from SAS

For Excel, there are NO valid dates that occurred before January 1, 1900. Excel's date system starts at January 1, 1900, whereas SAS dates span the whole range of the Gregorian calendar, starting in 1582 to AD 20,000 (yes, that is a 5-digit year). These Tech Support notes explain how the adoption of the Gregorian calendar might impact dates and date calculations, such as for age:

http://support.sas.com/kb/2/429.html Date values will be truncated if outside the range of valid SAS Dates

http://support.sas.com/kb/24/808.html Calculating Age with Only One Line of Code

So, although SAS can deal with a wide range of dates easily, using a "0" date of January 1, 1960, Excel does not have a "0" date. The serial number 1 in Excel on Windows represents the date January 1, 1900 (12:00:00 am). In Excel for the Macintosh platform, on the other hand, the serial number 1 represents January 2, 1904 (12:00:00 am).

And, Excel has other issues besides not being able to deal with dates prior to 1900. For example, Excel treated 1900 as a leap year, but it was not a leap year, so some dates in 1900 might be a day off from the SAS date. Or you may

discover that your dates appear to be about 60 years off. If this is the case, then you may need to add or subtract 21,916 to make the date a valid SAS date value. For more information about dates, as represented in Excel, refer to these Microsoft Knowledge Base articles:

http://support.microsoft.com/kb/214330

http://support.microsoft.com/kb/214058

http://support.microsoft.com/kb/214094

Consider the following data in Display 17 and PROC CONTENTS information about the data in Display 18.

transID	textdate	SAS_internal	MS_date_val	date	SAS_date9
1	November 15, 1950	Internal Number: -3334	18582	-3334	15NOV1950
2	January 1, 1960	Internal Number: 0	21916	0	01JAN1960
3	November 29, 1984	Internal Number: 9099	31015	9099	29NOV1984
4	December 5, 2005	Internal Number: 16775	38691	16775	05DEC2005
5	January 20, 2011	Internal Number: 18647	40563	18647	20JAN2011

Display 17. SAS Data Set WORK.DATES

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
4	MS_date_val	Num	8	
6	SAS_date9	Num	8	DATE9.
3	SAS_internal	Char	25	
5	date	Num	8	
2	textdate	Char	30	
1	transID	Num	8	

Display 18. PROC CONTENTS Information for WORK.DATES

As you can see, TEXTDATE and SAS_INTERNAL are character variables that represent a "word date" and a text string that shows the internal SAS date value, respectively. SAS_DATE9 is a numeric variable that is formatted with the DATE9. format. Except for TRANSID, MS_DATE_VAL and DATE are numeric variables that represent date values, as created by the following code:

```
data dates;
  length transID 8 textdate $30. SAS_internal $25 MS_date_val 8;
  input transID $ date : mmddyy10.;
  textdate = put(date,worddate30.);
  SAS_internal = catx(' ','Internal Number:',put(date,best8.));
  MS_date_val = date + 21916;
  SAS_date9 = date;
  format SAS_date9 date9.;
return;
datalines;
1 11/15/1950
2 01/01/1960
3 11/29/1984
4 12/05/2005
5 01/20/2011
;
run;
```

Now let's look at the default treatment of those variables in HTML output and XML output, as shown in Display 19. Remember that the output in Display 19 shows the default treatment for the variables given the data creation above. In this sample data, TEXTDATE is a character variable in the SAS dataset; and, although DATE is read with an MMDDYY10. Informat, the DATE variable is not formatted with a SAS-supplied format. On the other hand, the SAS_DATE9 variable is numeric and is formatted with the DATE9. SAS format.

HTML output

	A	B	C	D	E	F
1	1) Default for Dates					
2						
3	transID	Character Worddate	Stored in SAS	Microsoft Internal Value SAS val + 21916	NO SAS Fmt for Date Excel shows internal number	SAS Date9. Format Used Explicitly
5	1	15-Nov-50	Internal Number: -3334	18582	-3334	15-Nov-50
6	2	1-Jan-60	Internal Number: 0	21916	0	1-Jan-60
7	3	29-Nov-84	Internal Number: 9099	31015	9099	29-Nov-84
8	4	5-Dec-05	Internal Number: 16775	38691	16775	5-Dec-05
9	5	20-Jan-11	Internal Number: 18647	40563	18647	20-Jan-11

XML output

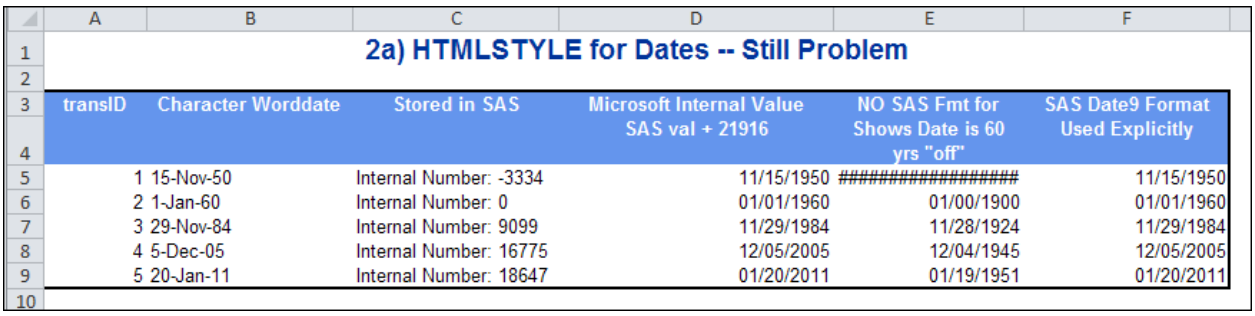
	A	B	C	D	E	F
1	transID	Character Worddate	Stored in SAS	Microsoft Internal Value SAS val + 21916	NO SAS Fmt for Date Excel shows internal number	SAS Date9. Format Used Explicitly
2	1	November 15, 1950	Internal Number: -3334	18582	-3334	15NOV1950
3	2	January 1, 1960	Internal Number: 0	21916	0	01JAN1960
4	3	November 29, 1984	Internal Number: 9099	31015	9099	29NOV1984
5	4	December 5, 2005	Internal Number: 16775	38691	16775	05DEC2005
6	5	January 20, 2011	Internal Number: 18647	40563	18647	20JAN2011

Display 19. Default Excel Treatment of Formatted and Unformatted Date Variable Values

There are more issues with the HTML output (created by ODS MSOFFICE2K) than with the XML output (created by TAGSETS.EXCELXP), when both outputs are opened with Excel, as shown in Display 19. As you can see, Excel respected the DATE9. format for the SAS_DATE9 variable, and it respected the character variable value for the TEXTDATE variable in the XML output. But Excel did not respect the character variable TEXTDATE in the HTML output.

The good news is that using Microsoft date formats, you can control how Excel treats the dates in the HTML file that you create, as shown in the following code. (Side note: The regular slash (/) must be preceded by a backslash (\), which results in \/ in the code. This is not a capital letter 'V'. Rather, it is two characters: '\/' .

```
ods tagsets.msoffice2k file='demo04a_dates_mso.xls' style=sasweb;
proc report data=dates nowd
  style(column)={width=1.5in};
  title '2a) HTMLSTYLE for Dates -- Still Problem';
  column transID textdate SAS_internal MS_date_val
         date SAS_date9;
  define transID /display style(column)={width=.75in};
  define textdate / 'Character Worddate';
  define SAS_internal / display 'Stored in SAS';
  define MS_date_val / 'Microsoft Internal Value/SAS val + 21916'
         style(column)={width=2in
                        htmlstyle="mso-number-format:mm\/dd\/yyyy"};
  define date / display 'NO SAS Fmt for Date/Shows Date is 60 yrs "off"'
         style(column)={htmlstyle="mso-number-format:mm\/dd\/yyyy"};
  define SAS_date9 /display  f=date9. 'SAS Date9 Format/Used Explicitly'
         style(column)={htmlstyle="mso-number-format:mm\/dd\/yyyy"};
run;
ods _all_ close;
```



2a) HTMLSTYLE for Dates -- Still Problem

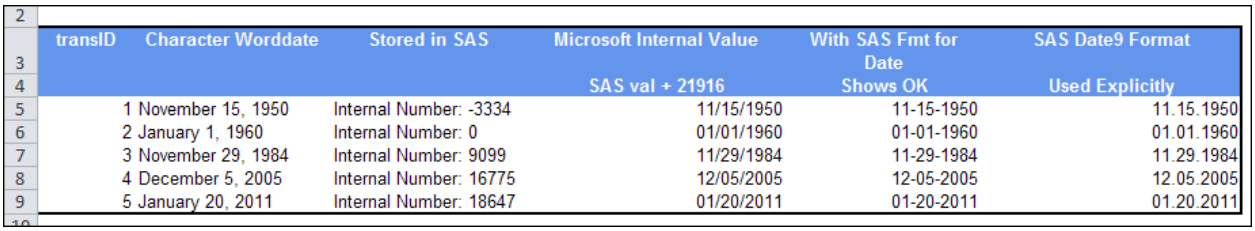
transID	Character Worddate	Stored in SAS	Microsoft Internal Value SAS val + 21916	NO SAS Fmt for Shows Date is 60 yrs "off"	SAS Date9 Format Used Explicitly
1	15-Nov-50	Internal Number: -3334	11/15/1950	################	11/15/1950
2	1-Jan-60	Internal Number: 0	01/01/1960	01/00/1900	01/01/1960
3	29-Nov-84	Internal Number: 9099	11/29/1984	11/28/1924	11/29/1984
4	5-Dec-05	Internal Number: 16775	12/05/2005	12/04/1945	12/05/2005
5	20-Jan-11	Internal Number: 18647	01/20/2011	01/19/1951	01/20/2011

Display 20. Microsoft Formats Used to Create HTML Output

Notice how all the other dates do correctly use the mm/dd/yyyy format that was specified, but look at the actual values displayed for the DATE variable. They are 60 years "off" from what is expected. That's because Excel just saw a number, and without any information from SAS about what that number meant (such as a DATE9 format), Excel displayed the number as though it was offset from Jan 1, 1900 (Excel's start date). And there was an issue with the number value of -3334 for DATE on row 1, because Excel doesn't allow negative numbers for date values. So, to fix the HTML output, the above code will be revised slightly, only in these two DEFINE statements:

```
define textdate / 'Character Worddate'
       style(column)={htmlstyle="mso-number-format:\@"};
define date /display f=date9. 'With SAS Fmt for Date/Shows OK'
       style(column)={htmlstyle="mso-number-format:mm\-dd\-yyyy"};
define SAS_date9 /display f=date9. 'SAS Date9 Format/Used Explicitly'
       style(column)={width=2in htmlstyle="mso-number-format:mm\.dd\.yyyy"};
```

As already demonstrated, defining the TEXTDATE variable to use an MSO-NUMBER-FORMAT of '\@' will ensure that it is treated as a text string, which is desirable because it is a character variable. Using a SAS format of DATE9. for the variable SAS_DATE9 as well as DATE, gives Excel the clue that it needs to use MSO-NUMBER-FORMAT appropriately. Note that in Display 21, we see the different formats used for each of the numeric variables, as coded. Note that slightly different formats were used for DATE and SAS_DATE9 in order to show how to use different date delimiters with the MSO-NUMBER-FORMAT specification.



transID	Character Worddate	Stored in SAS	Microsoft Internal Value SAS val + 21916	With SAS Fmt for Date Shows OK	SAS Date9 Format Used Explicitly
1	November 15, 1950	Internal Number: -3334	11/15/1950	11-15-1950	11.15.1950
2	January 1, 1960	Internal Number: 0	01/01/1960	01-01-1960	01.01.1960
3	November 29, 1984	Internal Number: 9099	11/29/1984	11-29-1984	11.29.1984
4	December 5, 2005	Internal Number: 16775	12/05/2005	12-05-2005	12.05.2005
5	January 20, 2011	Internal Number: 18647	01/20/2011	01-20-2011	01.20.2011

Display 21. Microsoft Formats Used Appropriately by All Variables in HTML Output

The code used with TAGSETS.EXCELXP specifies SAS formats for all the numeric date variables except for the Microsoft date value stored in the MS_DATE_VAL variable. The MSO-NUMBER-FORMAT used for that variable shows that Microsoft interprets those variable values as the dates represented in the 1900 date system that is used by Excel for Windows. (As shown in Display 17, the values for the MS_DATE_VAL variable were created by adding 21,916 to the SAS date value.)

```
ods tagsets.excelxp file='demo04b_dates_XP.xls' style=sasweb;

proc report data=dates nowd style(column)={width=1.5in};
  column transID textdate SAS_internal MS_date_val date SAS_date9;
  define transID /display style(column)={width=.75in};
  define textdate / 'Character Worddate';
  define SAS_internal / display 'Stored in SAS';
  define MS_date_val / 'Microsoft Internal Value/SAS val + 21916'
         style(column)={width=2in
                        tagattr="format:mm/dd/yyyy"};
  define date / f=mmddyyd10. display 'Variable use MMDDYYD10 SAS Format';
  define SAS_date9 /display f=mmddyyp10. 'SAS_DATE9 Variable Uses MMDDYYP10 SAS
Format';
run;

ods _all_ close;
```

As seen in Display 22, with TAGSETS.EXCELXP, if you use a SAS format with a date variable, such as the DATE and SAS_DATE9 variable, when Excel opens and renders the XML file, the SAS formats are respected.

	A	B	C	D	E	F
1	transID	Character Worddate	Stored in SAS	Microsoft Internal Value SAS val + 21916	Variable use MMDDYYD10 SAS Format	SAS_DATE9 Variable Uses MMDDYYP10 SAS Format
2	1	November 15, 1950	Internal Number: -3334	11/15/1950	11-15-1950	11.15.1950
3	2	January 1, 1960	Internal Number: 0	01/01/1960	01-01-1960	01.01.1960
4	3	November 29, 1984	Internal Number: 9099	11/29/1984	11-29-1984	11.29.1984
5	4	December 5, 2005	Internal Number: 16775	12/05/2005	12-05-2005	12.05.2005
6	5	January 20, 2011	Internal Number: 18647	01/20/2011	01-20-2011	01.20.2011

Display 22. SAS Formats Used Appropriately with TAGSETS.EXCELXP Output

The bottom line is that as long as you understand whether your date is a SAS internal value for a date (an offset from January 1, 1960) or a Microsoft date (an offset from January 1, 1900), you now know how to get the formats you want when Excel opens your HTML or XML output files.

Perhaps the best advice to follow (for TAGSETS.EXCELXP output) is the advice provided by Vincent DelGobbo in his paper, "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). In the paper, DelGobbo recommends: "One way to correct this behavior [dates getting messed up in Excel] is to write SAS code to convert SAS date values to Excel date values, but this approach is problematic because you must alter your original SAS data. While you can create a new view or SAS 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 using a FORMAT statement, 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."

Excel Uses a Default Format for Percents

Display 10 and the code that created Display 10 showed how to specify a percent as an MSO-NUMBER-FORMAT for HTMLSTYLE= and as a TAGATTR= value for TAGSETS.EXCELXP output. So this example is just reinforcing that earlier example. Consider the following data.

Obs	grade	rpass	year	count	percent
1	1	2	2002	23	0.099
2	1	2	2003	43	0.082
3	1	2	2001	24	0.021
4	2	3	2002	32	0.089
5	2	3	2001	32	0.079
6	2	3	2003	32	0.069
7	3	4	2002	32	0.089
8	3	4	2001	32	0.079
9	3	4	2003	32	0.069
10	4	2	2002	23	0.099
11	4	2	2003	43	0.082
12	4	2	2001	24	0.021

Display 23. Data Shows a Percent Variable

Even if you use a SAS PERCENT6.1 format, when the HTML and XML output is opened in Excel, the percents show two decimal places, as shown in Display 24.

HTML output

			2001		2003		2002	
Grade		#	%		#	%	#	%
4	2	24	2.10%		43	8.20%	23	9.90%
3	4	32	7.90%		32	6.90%	32	8.90%
2	3	32	7.90%		32	6.90%	32	8.90%
1	2	24	2.10%		43	8.20%	23	9.90%

XML output

			2001		2003		2002	
Grade		#	%		#	%	#	%
4	2	24	2.10%		43	8.20%	23	9.90%
3	4	32	7.90%		32	6.90%	32	8.90%
2	3	32	7.90%		32	6.90%	32	8.90%
1	2	24	2.10%		43	8.20%	23	9.90%

Display 24. Percent Shows Two Decimal Places

Again, using the guidelines already discussed, the fix for this is as simple as specifying a Microsoft format for the percent value, as shown in the code below.

```
ods msoffice2k file='demo5_percent_mso.xls' style=sasweb;
  proc report data = prof nowd;
  column grade rpass year,(count percent);
  define grade / group order=data descending 'Grade';
  define rpass / group ' ' order=internal;
  define year / across order=data descending ' ';
  define count / '#';
  define percent / '%'
    style(column)={htmlstyle='mso-number-format:##0.0%'};
run;
ods _all_ close;

ods tagsets.ExcelXP file='demo5_percent_XP.xls' style=sasweb
    options(sheet_Name="Grade Passrate");
  proc report data = prof nowd;
  column grade rpass year,(count percent);
  define grade / group order=data descending 'Grade';

  define rpass / group ' ' order=internal;
  define year / across order=data descending ' ';
  define count / '#';
  define percent / '%' style(column)={tagattr='format:##0.0%'};
run;
ods _all_ close;
```

Display 25 shows the HTML output, and Display 26 shows the XML output when opened and rendered with Excel.

	2001		2003		2002	
Grade	#	%	#	%	#	%
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%

Display 25. HTML Output Shows Percent Value with One Decimal Place

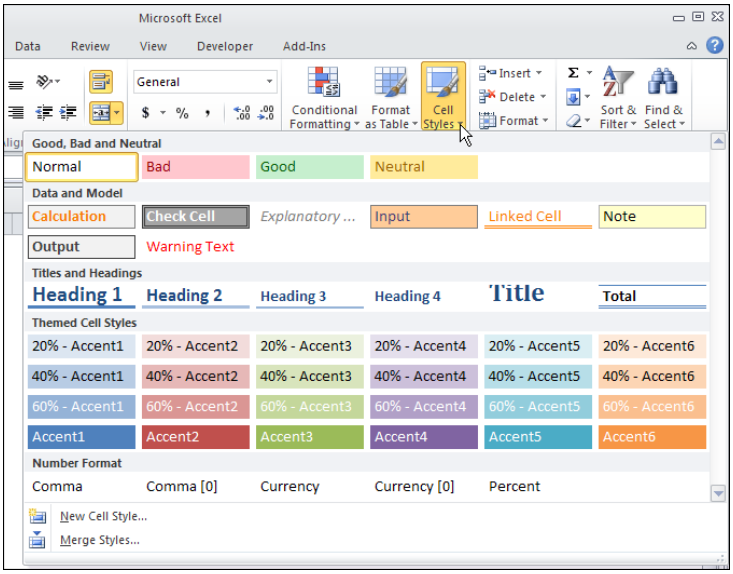
		2001		2003		2002	
Grade		#	%	#	%	#	%
4	2	24	2.1%	43	8.2%	23	9.9%
3	4	32	7.9%	32	6.9%	32	8.9%
2	3	32	7.9%	32	6.9%	32	8.9%
1	2	24	2.1%	43	8.2%	23	9.9%

Display 26. XML Output Shows Percent Value with One Decimal Place

Formatting Single Cells or Formatting the Output Table in Excel

The last topic is a slightly different topic. Instead of focusing on individual cell values, this topic is about the overall style of your cells and table. Note that when I say "style" I mean colors or fonts – not numeric formatting. These next examples show ODS STYLE= color formatting, but ODS STYLE= font formatting will use similar techniques. Excel has three point-and-click methods for applying style to a cell or to a table. One method is the Format Cells tab from the Excel ribbon. We saw this option in Display 16. The two other methods are shown in Display 27 (Cell Styles icon) and Display 31 (Format as Table icon).

Display 27 shows the Cell Styles dialog box on the Excel ribbon. When you choose a cell or range of cells and then choose one of the formatting styles, Excel uses your choice for the cell style.



Display 27. Cell Styles Formatting Method on Excel Ribbon

ODS does give you a way to achieve this type of cell level formatting by simply specifying a STYLE= option on your ODS invocation statement (such as STYLE=SASWEB or STYLE=EGDEFAULT or STYLE=OCEAN). But there is another way to impact the style of your output. However, this method does not use the HTMLSTYLE= or TAGATTR= style attribute methods that were discussed earlier. To achieve these types of results, instead, you will use other style attributes, such as BACKGROUND and FOREGROUND (or BACKGROUNDCOLOR and COLOR, which are the attribute names in SAS 9.2) in your ODS statement level STYLE= option.

Consider the following code:

```
ods msoffice2k file='demo06_change_header_mso.xls' style=sasweb;
ods tagsets.ExcelXP file='demo06_change_header_xp.xls' style=sasweb;
  proc report data = prof nowd
       style(report)={background=white}
       style(header)={background=cx8064A2};
  column grade rpass year,(count percent);
  define grade / group order=data descending 'Grade';
  define rpass / group ' ' order=internal;
  define year / across order=data descending ' ';
  define count / '#' ;
  define percent / '%' f=percent6.1;
run;

ods _all_ close;
```

HTML output

	A	B	C	D	E	F	G	H
1				2001		2003		2002
2	Grade		#	%	#	%	#	%
3	4	2	24	2.10%	43	8.20%	23	9.90%
4	3	4	32	7.90%	32	6.90%	32	8.90%
5	2	3	32	7.90%	32	6.90%	32	8.90%
6	1	2	24	2.10%	43	8.20%	23	9.90%

Partial XML output

	A	B	C	D
1			2001	
2	Grade		#	%
3	4	2	24	2.10%
4	3	4	32	7.90%
5	2	3	32	7.90%
6	1	2	24	2.10%

Display 28. HTML and XML Output Using New Background Color for Header Cells

By adding the instructions for the BACKGROUND color change in the PROC REPORT statement, all the header cells at the top of the table (using the previous data) are changed to a lovely shade of purple. The SASWEB style has been used previously in other examples, and you may remember that the default color for header cells is a light blue shade with the SASWEB style. This means that the BACKGROUND attribute in PROC REPORT was used to override the default color in the SASWEB style definition. (Note: Only partial output is shown for the TAGSETS.EXCELXP output, due to space considerations.)

The change for the STYLE(REPORT) background as WHITE just ensures that the HTML output has a white background for any unused cells that are to the right of the table cells in the worksheet. You can use the style attribute of BACKGROUND or BACKGROUNDCOLOR (SAS 9.2) in your STYLE= code or attributes of FOREGROUND or COLOR (SAS 9.2), to mention just a few of the most frequently used attributes.

What if you want to change the COUNT column to be in a different color based on the number of students in each grade? That type of conditional highlighting is called trafficlighting, and you can also achieve that with STYLE= overrides. A simple example creates a user-defined format for the desired colors (in this case, yellow, light green and light red), such as shown for the HTML output in Display 29. (The XML output will look essentially the same, so it is not shown here.)

	A	B	C	D	E	F	G	H
1			2001		2003		2002	
2	Grade		#	%	#	%	#	%
3	4	2	24	2.10%	43	8.20%	23	9.90%
4	3	4	32	7.90%	32	6.90%	32	8.90%
5	2	3	32	7.90%	32	6.90%	32	8.90%
6	1	2	24	2.10%	43	8.20%	23	9.90%

Display 29. Trafficlighting the COUNT Cells Based on Cell Value (HTML Output)

The above example (and the XML output, too) was created with the following code. The key to understanding this approach is to know how user-defined formats work and how a user-defined format can be specified as a style attribute value. So, in the following code, the user-defined format must be set up first, before it is used. Consider the following code:

```
proc format;
  value cback low-25='light red'
              26-32 = 'yellow'
              33-high = 'light green';
run;

ods msoffice2k file='demo06_tlite_data_mso.xls' style=sasweb;
ods tagsets.ExcelXP file='demo06_tlite_data_xp.xls' style=sasweb;
  proc report data = prof nowd
       style(report)={background=white}
       style(header)={background=cx8064A2};
  column grade rpass year,(count percent);
  define grade / group order=data descending 'Grade';
  define rpass / group ' ' order=internal;
  define year / across order=data descending ' ';
  define count / '#'
         style(column)={background=cback.};
  define percent / '%' f=percent6.1;
run;

ods _all_ close;
```

Note that only the desired color value is specified in the user-defined format. Then the format is used in the STYLE(COLUMN) override in the PROC REPORT DEFINE statement for the COUNT variable. This (or similar) syntax would work with PROC PRINT, PROC TABULATE and with Table templates. Note that the colors applied only to the COUNT values because the STYLE override was added to the DEFINE statement. (A full discussion of user-defined formats is outside the scope of this paper, but there is a way to create user-defined formats from a SAS data set, so it would be possible to automate the creation of formats when you have a long list of possible values.)

With the CALL DEFINE statement in PROC REPORT (and with the CELLSTYLE-AS statement in Table templates), it is possible to apply trafficlighting to an entire row based on a variable value or to trafficlight one cell based on the value of another cell.

For example, what if, in the above output, you did not want to use trafficlighting for the COUNT values for Grade 1? And what if you also wanted to highlight the entire row for Grade 2 (for some reason) and you plan to explain this information in a note? Using CALL DEFINE, you can modify the above code with this COMPUTE BLOCK syntax. The new code creates the output shown in Display 30.

```
compute count;
  if grade=1 then do;
    call define(_col_,'style','style={background=white}');
  end;
  else if grade = 2 then do;
    call define(_row_,'style','style={background=cxE4DFEC}');
  end;
endcomp;
```

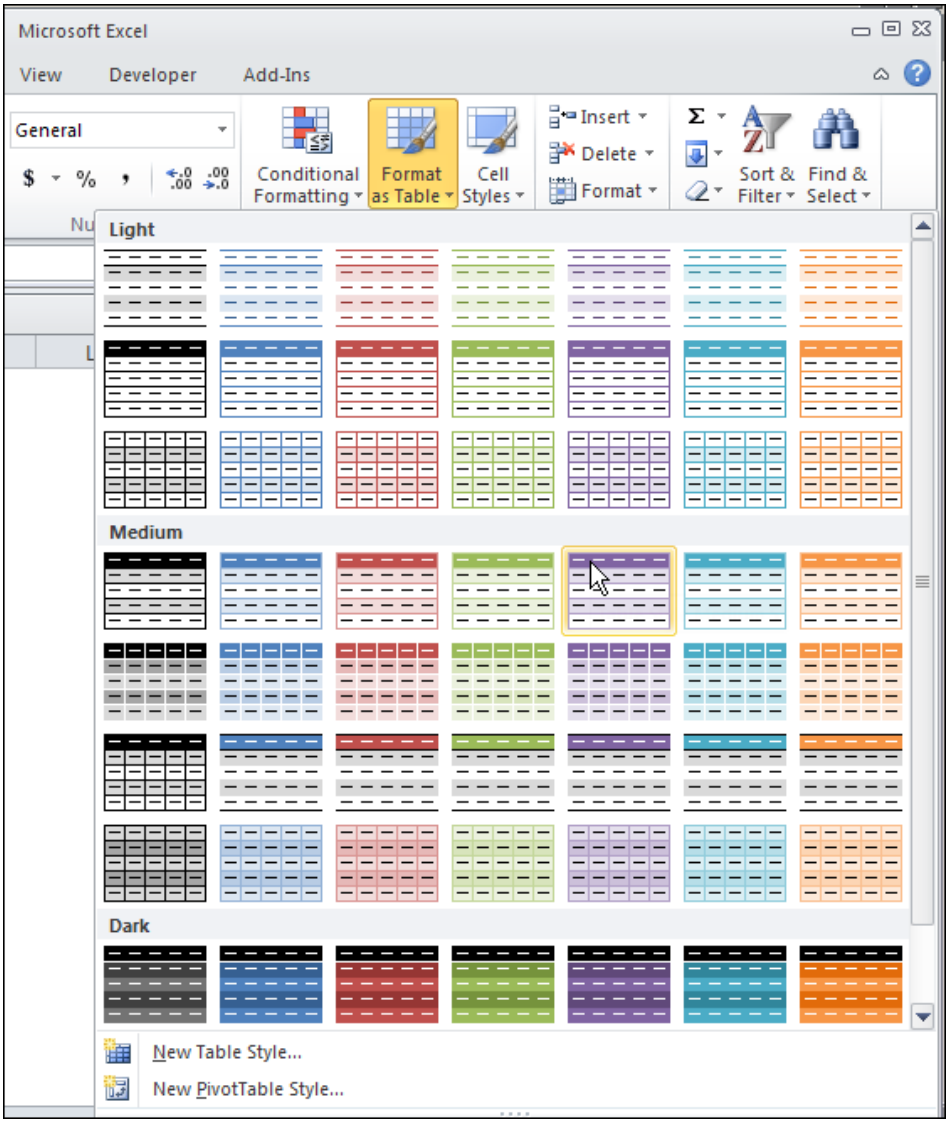
	A	B	C	D	E	F	G	H
1			2001		2003		2002	
2	Grade		#	%	#	%	#	%
3	4	2	24	2.10%	43	8.20%	23	9.90%
4	3	4	32	7.90%	32	6.90%	32	8.90%
5	2	3	32	7.90%	32	6.90%	32	8.90%
6	1	2	24	2.10%	43	8.20%	23	9.90%

Display 30. Altering Cell Values or Row Style Based on GRADE Variable (XML Output)

Display 30 shows the TAGSETS.EXCELXP output, and the test for GRADE=1 gives an opportunity for the CALL DEFINE statement to change the style of the background color for COUNT back to white (instead of using the previously defined colors in the CBACK. format). Since it is only the COUNT column that needs to change, the _COL_ argument is used in the CALL DEFINE statement. Then the entire row for GRADE=2 was changed to a light shade of lavender to call attention to the entire row for the defined report needs (using _ROW_ in the CALL DEFINE statement). Since the HTML output will look essentially the same, it is not shown here.

A full discussion of STYLE= overrides, COMPUTE blocks and CALL DEFINE syntax is outside the scope of this paper. PROC REPORT is the only report procedure (outside of creating a custom table template) that provides this level of style control in the procedure syntax. And, although PROC PRINT and PROC TABULATE do allow you to use STYLE= statement level overrides, neither procedure has the equivalent of the COMPUTE block for row or column level alteration. For more examples of what you can do with PROC REPORT, refer to Allison Booth's 2010 SAS Global Forum paper entitled, "Evolve from a Carpenter's Apprentice to a Master Woodworker: Creating a Plan for Your Reports and Avoiding Common Pitfalls in REPORT Procedure Coding".

Let's build on the previous example. What if you just wanted every other row of your SAS output to be that same shade of lavender used for Grade 2 in the previous example? This is a common table-level formatting choice in Excel, if you look at the third formatting option on the Excel ribbon—the "Format as Table" icon, as shown in Display 31.



Display 31. Table Level Formatting Method on Excel Ribbon

This type of style, with rows of alternating colors, comes to us from old mainframe days, when the printers on the mainframe most commonly used "green bar" paper, where the paper was printed with alternating green lines on white paper. The green bar paper could be purchased with wider green lines or narrower green lines, depending on whether you wanted every other line to have a green background or every three or four lines to have a green background.

Excel considers coloring alternating rows in a table to be a table-level formatting change. As far as SAS and ODS are concerned, you could change the cells in a row one by one or you could use row-level formatting such as that shown using PROC REPORT in the previous example. There is also a way for most SAS procedures to implement table-level alternating colors for procedure output, but that is a more advanced topic and is described well in the SAS TEMPLATE procedure documentation topic entitled "Example 6: Creating a Master Template."

However, let's look at the previous data, slightly changed to contain a variable called ROWVAR, as shown in Display 32.

Obs	rowvar	grade	rpass	year	count	percent
1	1	1	2	2002	23	0.099
2	1	1	2	2003	43	0.082
3	1	1	2	2001	24	0.021
4	2	2	3	2002	32	0.089
5	2	2	3	2001	32	0.079
6	2	2	3	2003	32	0.069
7	3	3	4	2002	32	0.089
8	3	3	4	2001	32	0.079
9	3	3	4	2003	32	0.069
10	4	4	2	2002	23	0.099
11	4	4	2	2003	43	0.082
12	4	4	2	2001	24	0.021

Display 32. PROC PRINT Display of WORK.PROF with the ROWVAR Variable

You can see that the ROWVAR variable is just a number from 1 to 4. (Because GRADE also is a value from 1 to 4, for this example, GRADE was used to create ROWVAR.) Since the YEAR variable will be used as an ACROSS report item, the final table will have only 4 rows, and so we will use ROWVAR to achieve trafficlighting, by dividing the ROWVAR value by 2. When the remainder is 0, that means the number was evenly divisible by 2. When the remainder is anything other than 0, that means the number was NOT evenly divisible by 2. So now, the logic in PROC REPORT can merely test for "even" or "odd" by actually testing the result of the MOD function. For this example, we will show you the results first and then show all the code. But, what if you did not want to alternate every other row, but instead wanted to alternate the color change for every five (5) rows?

Look at a modified version of SASHELP.CLASS, as shown in Display 33, where you can see how the ROWVAR variable value increases every 5 rows.

Obs	rowvar	Name	Age	Sex	Height	Weight
1	1	Alfred	14	M	69.0	112.5
2	1	Alice	13	F	56.5	84.0
3	1	Barbara	13	F	65.3	98.0
4	1	Carol	14	F	62.8	102.5
5	1	Henry	14	M	63.5	102.5
6	2	James	12	M	57.3	83.0
7	2	Jane	12	F	59.8	84.5
8	2	Janet	15	F	62.5	112.5
9	2	Jeffrey	13	M	62.5	84.0
10	2	John	12	M	59.0	99.5
11	3	Joyce	11	F	51.3	50.5
12	3	Judy	14	F	64.3	90.0
13	3	Louise	12	F	56.3	77.0
14	3	Mary	15	F	66.5	112.0
15	3	Philip	16	M	72.0	150.0
16	4	Robert	12	M	64.8	128.0
17	4	Ronald	15	M	67.0	133.0
18	4	Thomas	11	M	57.5	85.0
19	4	William	15	M	66.5	112.0

Display 33. PROC PRINT Display of WORK.CLASS with the ROWVAR Variable

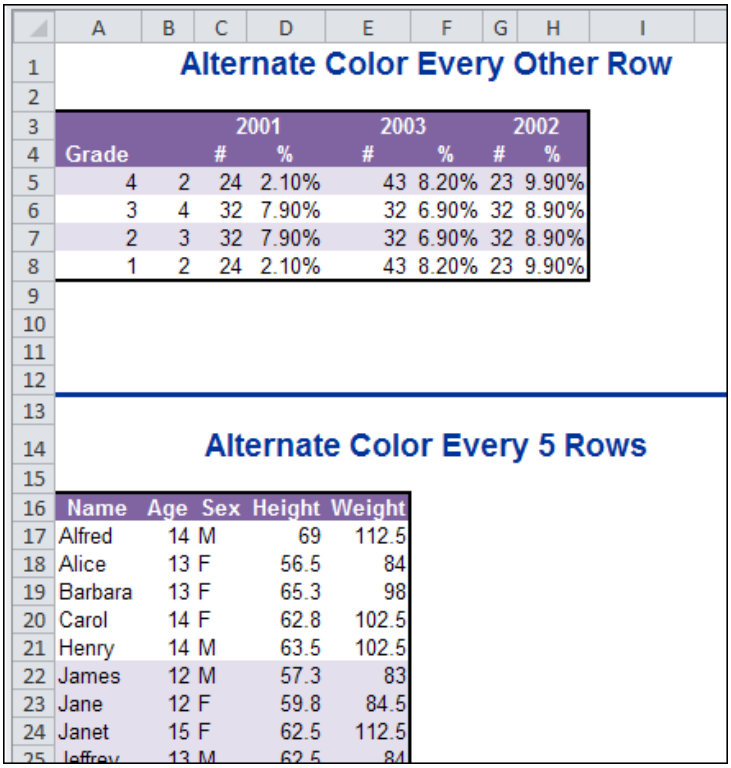
For the data in Display 33, the value of the ROWVAR variable was created using this statement:

```
rowvar = ceil(divide(_n_,5));
```

This caused the value for ROWVAR to change for every 5 observations, because the CEIL function takes the result of the division and returns the smallest integer number that is greater than or equal to the argument. For example, the CEIL function would return 1 if the result of the division was 0.2 or 0.8, and it would also return a value of 1 if the result of the division was 1. This means that the value for rowvar changes on a regular and predictable basis, and we can use the same MOD technique in PROC REPORT to perform the row-level highlighting. Once you understand how the ROWVAR variable is created and used and you get more comfortable with PROC REPORT, you will realize that ROWVAR could be calculated inside PROC REPORT without altering or amending the original data. No matter whether you create ROWVAR outside of the PROC REPORT step or inside the PROC REPORT step, the basic

MOD function technique will remain the same. The output for the above two data sets is shown in Display 34 and is self-explanatory. When the HTML file and the XML file are opened with Excel, the alternating color scheme is used for the output.

HTML output



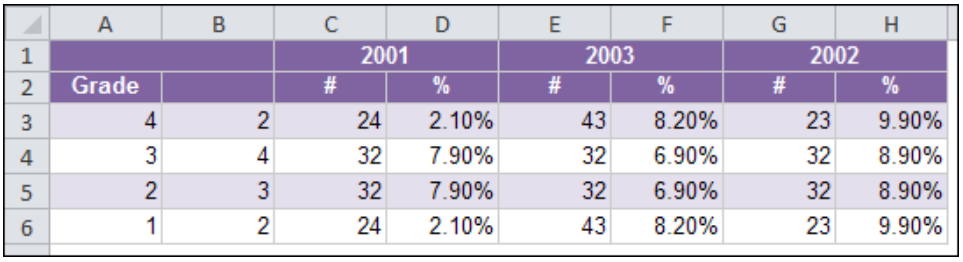
Alternate Color Every Other Row

Grade		2001 #	2001 %	2003 #	2003 %	2002 #	2002 %
4	2	24	2.10%	43	8.20%	23	9.90%
3	4	32	7.90%	32	6.90%	32	8.90%
2	3	32	7.90%	32	6.90%	32	8.90%
1	2	24	2.10%	43	8.20%	23	9.90%

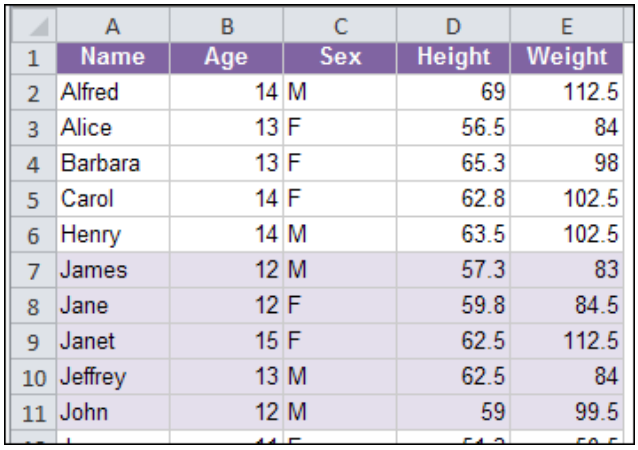
Alternate Color Every 5 Rows

Name	Age	Sex	Height	Weight
Alfred	14	M	69	112.5
Alice	13	F	56.5	84
Barbara	13	F	65.3	98
Carol	14	F	62.8	102.5
Henry	14	M	63.5	102.5
James	12	M	57.3	83
Jane	12	F	59.8	84.5
Janet	15	F	62.5	112.5
Jeffrey	13	M	62.5	84

XML output



Grade		2001 #	2001 %	2003 #	2003 %	2002 #	2002 %
4	2	24	2.10%	43	8.20%	23	9.90%
3	4	32	7.90%	32	6.90%	32	8.90%
2	3	32	7.90%	32	6.90%	32	8.90%
1	2	24	2.10%	43	8.20%	23	9.90%



Name	Age	Sex	Height	Weight
Alfred	14	M	69	112.5
Alice	13	F	56.5	84
Barbara	13	F	65.3	98
Carol	14	F	62.8	102.5
Henry	14	M	63.5	102.5
James	12	M	57.3	83
Jane	12	F	59.8	84.5
Janet	15	F	62.5	112.5
Jeffrey	13	M	62.5	84
John	12	M	59	99.5

Display 34. Excel View of HTML Results and XML Results

The program that produced the above results is shown below. Note that it is not too different from the previous program, where the highlighting was done for an entire row, based on the value of the GRADE variable.

```
data prof;
  input grade rpass year count percent;
  rowvar = grade;
return;
cards;
<. . .data as shown above . . .>
run;

data class;
  set sashelp.class;
  rowvar = ceil(divide(_n_,5));
run;

ods msoffice2k file='demo06_alt_row_mso.xls' style=sasweb;
ods tagsets.ExcelXP file='demo06_alt_row_xp.xls' style=sasweb;
  proc report data = prof nowd
       style(report)={background=white}
       style(header)={background=cx8064A2}
       style(column)={background=white};
  title 'Alternate Color Every Other Row';
  column grade rowvar rpass year,(count percent);
  define grade / group order=data descending 'Grade';
  define rowvar / group noprint;
  define rpass / group ' ' order=internal;
  define year / across order=data descending ' ';
  define count / '#' ;
  define percent / '%' f=percent6.1;
  compute rowvar;
    if mod(rowvar, 2) = 0 then do;
        call define (_row_, "style",
                    "style = {background = CXE4DFEC}");
    end;
  endcomp;
run;

  proc report data = class nowd
       style(report)={background=white}
       style(header)={background=cx8064A2}
       style(column)={background=white};
  title 'Alternate Color Every 5 Rows';
  column rowvar name age sex height weight;
  define rowvar / display noprint;
  define name / order;
  define age / display;
  define sex /display;
  define height / mean;
  define weight / mean;
  compute rowvar;
    if mod(rowvar, 2) = 0 then do;
        call define (_row_, "style",
                    "style = {background = CXE4DFEC}");
    end;
  endcomp;
run;

ods _all_ close;
```

There is so much more that you can do with STYLE= overrides in PROC REPORT, PROC TABULATE and PROC PRINT, whether you're sending your output to the primary ODS destinations (ODS HTML, ODS RTF, ODS PDF) or sending your ODS output to Excel. However, this is a good example on which to end this paper. This final example in

this paper isn't going to show code, but is going to show you more of what you can do with ODS and STYLE= options and SAS Style templates. This example is available in the download of programs for this paper and was slightly modified from an example, using TAGSETS.EXCELXP, in Eric Gebhart's SAS Global Forum 2008 paper, "The Devil Is in the Details: Styles, Tips, and Tricks That Make Your Microsoft Excel Output Look Great!" (available at http://www2.sas.com/proceedings/forum2008/036-2008.pdf). This example uses a custom style template and STYLE= overrides with PROC REPORT to generate output as shown in Display 35. Notice how Excel's interior table lines for the header areas have been modified from the type of interior lines shown in Display 34 (for the XML output file):

	A	B	C	D	E	F	G	H
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%

Display 35. XML Output with Changed Interior Borders

CONCLUSION

There is a lot to learn beyond the basic ODS "sandwich" technique, especially when you want to get your SAS report output opened and rendered in Excel. But what you have to learn is not as arcane as the strategy rules for blackjack or as random as playing roulette. The way that Microsoft formats work in Excel is well documented and is worth investigation. The way that you use Microsoft formats with ODS is slightly different for HTML destinations (using HTMLSTYLE=) versus the TAGSETS.EXCELXP destination (using TAGATTR=), but the key to providing values for either HTMLSTYLE or TAGATTR style attributes is understanding what Microsoft format you need to specify.

Beyond formatting numbers and text, using ODS STYLE= statement level style attribute overrides will also enable you to change colors, fonts and other default attributes. Depending on whether you use PROC REPORT, PROC TABULATE, PROC PRINT or other procedures to create your tabular output, you may find that you can achieve the look you want with simple style overrides in the procedure syntax or through the use of a custom style template.

The job aids at the end of this paper show some of the commonly used Microsoft formats and their SAS equivalents (Table A.1). There is also a job aid (Table A.2) that shows other style attributes that can be used to alter the color and fonts of cells within your ODS output. To help you figure out Microsoft formats, Table A.3 shows how to look inside HTML and XML files created with Excel to find the cell format that you need.

So now, the odds are in your favor, as far as achieving the formatting and style that you want in your ODS output.

REFERENCES

Booth, Allison. 2010. "Evolve from a Carpenter's Apprentice to a Master Woodworker: Creating a Plan for Your Reports and Avoiding Common Pitfalls in REPORT Procedure Coding". *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc.

DelGobbo, Vincent. 2003. "A Beginner's Guide to Incorporating SAS® Output in Microsoft® Office Applications" *Proceedings of the SAS Users Group International 28 Conference.* Cary, NC: SAS Institute Inc.

DelGobbo, Vincent. 2009. "More Tips and Tricks for Creating Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®" *Proceedings of the SAS Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc.

DelGobbo, Vincent. 2010. "Traffic Lighting Your Multi-Sheet Microsoft Excel Workbooks the Easy Way with SAS®" *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc.

Gebhart, Eric. 2007. "ODS Markup, Tagsets, and Styles! Taming ODS Styles and Tagsets." *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc.

Gebhart, Eric. 2008. "The Devil Is in the Details: Styles, Tips, and Tricks That Make Your Microsoft Excel Output Look Great!" ." *Proceedings of the SAS Global Forum 2008 Conference*. Cary, NC: SAS Institute Inc.

Parker, Chevell. "Generating Custom Excel Spreadsheets using ODS." *Proceedings of the SAS Users Group International 28 Conference.* Cary, NC: SAS Institute Inc.

Smith, Kevin D. 2006. "The TEMPLATE Procedure Styles: Evolution and Revolution." *Proceedings of the SAS Users Group International 31 Conference.* Cary, NC: SAS Institute Inc.

Zender, C. 2005. "The Power of Table Templates and DATA _NULL_". *Proceedings of the Thirtieth Annual SAS Users Group International Conference, 30*. CD-ROM. Cary, NC: SAS Institute Inc. (Paper 88-30)

Zender, Cynthia L. 2009. “Tiptoe through the Templates.” *Proceedings of the SAS Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc. (Paper 227-2009)

Zender, Cynthia L. 2010. “SAS® Style Templates: Always in Fashion” *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC: SAS Institute Inc. (Paper 033-2010)

ACKNOWLEDGMENTS

This paper would not be possible without the ODS developers who designed ODS in the first place. This paper has been made better because of Vincent DelGobbo, Michele Ensor, Bari Lawhorn, Chevell Parker, Lorilyn Russell, and Sue Rakes, who reviewed it, and John Kohl who edited it.

CONTACT INFORMATION

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

Cynthia Zender
SAS Institute, Inc
Cary, NC 27513
919-531-9012 (Mountain Time Zone)
Cynthia.Zender@sas.com

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.

APPENDIX: JOB AIDS

Excel Format String	SAS Format Specification
@	$w.
#,##0	COMMAw.d
#,##0.00	COMMAw.d
#,##0.00_);[Red](#,##0.00)	NEGPARENw.d
$#,##0_);($#,##0)	DOLLARw.d
$#,##0_);[Red]($#,##0)	DOLLARw.d
_(*#,##0_);_(*(#,##0);_(*"-"_);_(@_)	NEGPARENw.d
0%	PERCENTw.d
##0.00%	PERCENTw.d
0.00E+00	Ew.d
m/d/yy	MMDDYYw.
mmm-yy	MONYYw.
dd/mm/yy	DDMMYYw.
mm/dd/yy	MMDDYYw.
mmmyy	MONYYw.
mmmyyyy	MONYYw.

Table A.1. Common Microsoft Formats and Their SAS Format Equivalent

For a longer list of SAS formats compared to XLS Format strings, refer to the table entitled "SAS Formats for XLS File Data," which is contained in the documentation topic entitled, "ACCESS Procedure: XLS Files". When you specify Microsoft formats for the HTMLSTYLE= value, you must use a backslash to "escape" any special text characters.

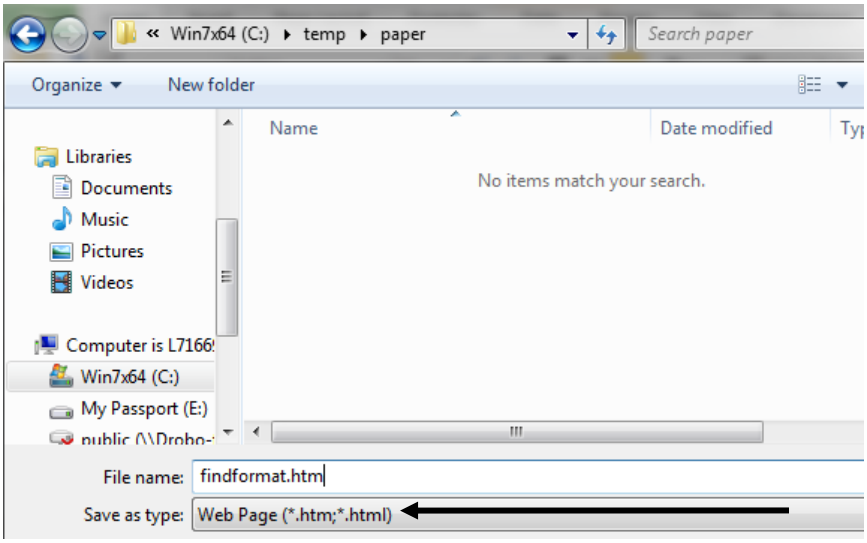
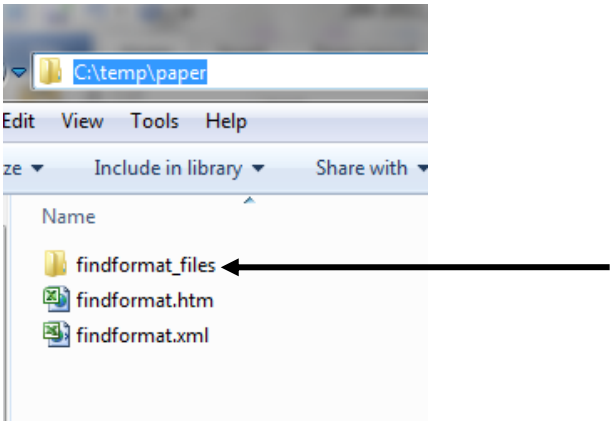
SAS Style Attribute	Possible Attribute Values
BACKGROUND or BACKGROUNDCOLOR	RED, YELLOW, WHITE, CXFFFFFF, **CXrrggbb**
FOREGROUND or COLOR	RED, YELLOW, **CXrrggbb**
FONT	Arial, 'Courier New', Times, 'Times New Roman'
FONT_SIZE or FONTSIZE	12pt, 14pt
FONT_WEIGHT or FONTWEIGHT	Medium, Bold
FONT_STYLE or FONTSTYLE	Italic, Slant, Roman
JUST	Left, Right, Center, L, R, C
VJUST	Top, Middle, Bottom, T, M, B

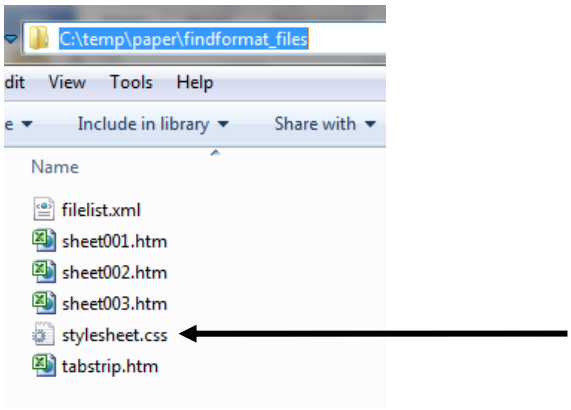
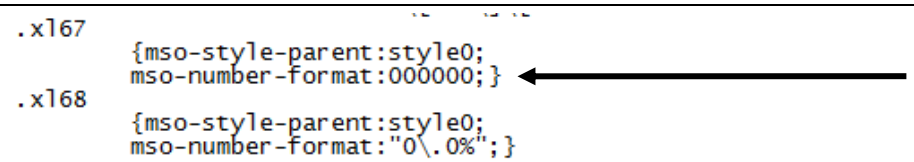
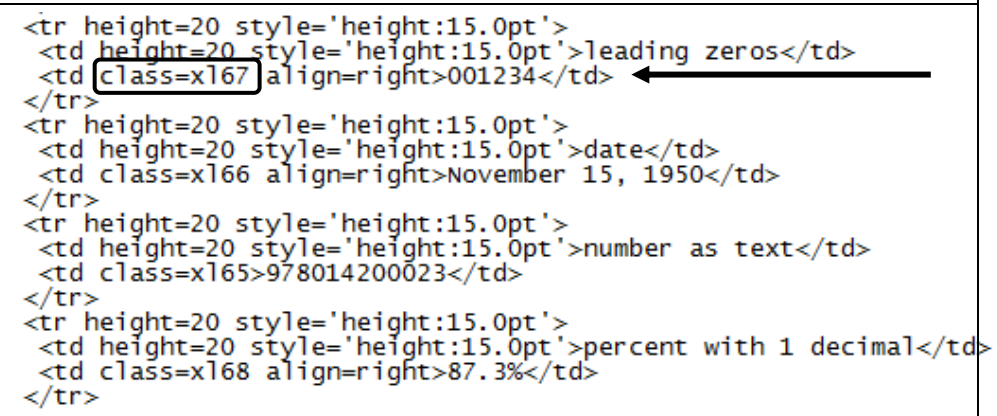
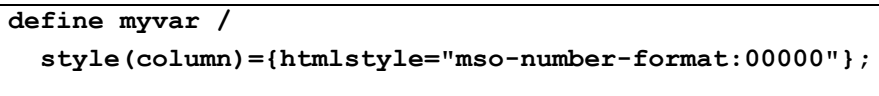
Table A.2. Common SAS Style Attributes

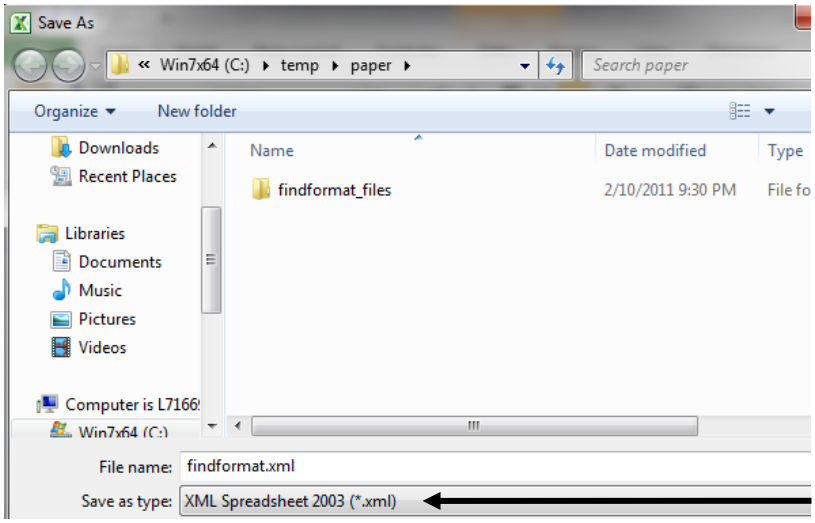
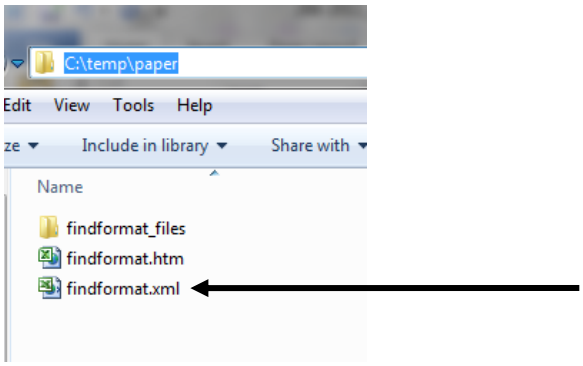
For a longer list of all the style attributes that you can specify with ODS, refer to the topic entitled "Style Attributes and Their Values" in the documentation for the TEMPLATE Procedure.

HOW TO FIND MICROSOFT FORMATS

If you need to find or "reverse engineer" Microsoft formats, one quick way to figure out the correct format is to create a small spreadsheet with all the formats you need and then save the Excel spreadsheet as an HTML Web page format or an XML Spreadsheet 2003 format. The steps and displays in Table A.3 do not show you every keystroke or mouse click. However, Table A.3 provides the general steps you can use to determine the correct format for your HTMLSTYLE= or TAGATTR= attribute values.

<table>
<tr><th>Steps</th><th>Notes</th></tr>
<tr><td>Open an Excel worksheet and manually type and format numbers as you want them to appear in your ODS output.</td><td>
<table>
<tr><th></th><th>A</th><th>B</th></tr>
<tr><td>1</td><td>type of format wanted</td><td>number</td></tr>
<tr><td>2</td><td>leading zeros</td><td>001234</td></tr>
<tr><td>3</td><td>date</td><td>November 15, 1950</td></tr>
<tr><td>4</td><td>number as text</td><td>978014200023</td></tr>
<tr><td>5</td><td>percent with 1 decimal</td><td>87.3%</td></tr>
<tr><td>6</td><td></td><td></td></tr>
</table>
</td></tr>
<tr><td colspan="2">Find MSO-NUMBER-FORMAT value for HTMLSTYLE= Style Attribute</td></tr>
<tr><td>Save the file as HTML. The Save as type choice should be Web Page.</td><td> 
</td></tr>
<tr><td>Find the files created by Excel. Excel creates one HTM or HTML file and a subdirectory of related files. Excel appends '_files' to the filename that you choose for your file to use as the directory name. You need to look inside this subdirectory.</td><td> 
</td></tr>
</table>

Open the CSS file in the subdirectory created by Excel. The file extension of this file will be .CSS and you can edit the file with Notepad or any text editor. If you did not use an external CSS file, then **stylesheet.css** will contain the CSS values used for your data.	
Search for "mso-number-format" and find the format of interest In the display the CSS class XL67 is the leading zero Microsoft format.	
If you only made one sheet, then **sheet001.htm** will contain the HTML tags for your data.You can verify the format setting by opening the **sheet001.htm** file and looking for the CLASS= attribute in the <td> tag. Here, you can see that XL67 is the CSS class used for the leading zero format.	
Cut and paste the format string into your HTMLSTYLE= code. Generally, if Excel has inserted backslash characters into the CSS value, then you should use the backslashes unchanged. The entire HTMLSTYLE= value must be quoted.	

Find XML FORMAT: value for TAGATTR= Style Attribute	
Save the file with the Save as type of XML 2003 Spreadsheet (In Excel 2002, this Save as type was just XML Spreadsheet.)	
Find the file created by Excel. It will have a file extension of .XML and you can open the file with Notepad or any text editor.	

Search for the format of interest. Note the ID= value in the format specification. For example, ID="s65" is the leading zero format.	`<Style ss:ID="s63">`
	`<NumberFormat ss:Format="@"/>`
	`</Style>`
	`<Style ss:ID="s64">`
	`<NumberFormat ss:Format="[ENG][$-409]mmmm\ d\,\ yyyy;@"/>`
	`</Style>`
	`<Style ss:ID="s65">`
	`<NumberFormat ss:Format="000000"/>`
	`</Style>`
	`<Style ss:ID="s66">`
	`<NumberFormat ss:Format="0.0%"/>`
	`</Style>`

Look for the <Cell> tag to verify the ID associated with your format of interest.	`<Row>`
	`<Cell><Data ss:Type="String">leading zeros</Data></Cell>`
	`<Cell ss:StyleID="s65"><Data ss:Type="Number">1234</Data></Cell>`
	`</Row>`

Cut and paste the format string into your TAGATTR= code. The entire TAGATTR value must be quoted.	`define myvar/ style(column)={tagattr='000000'};` `define myvar/ style(column)={tagattr='Format:00000'};`

Table A.3. How To Determine Microsoft Formats