Paper 031-2010

ODS ExcelXP: Tag Attr Is It! Using and Understanding the TAGATTR= Style Attribute with the ExcelXP Tagset

Eric Gebhart, SAS Institute Inc., Cary, NC

ABSTRACT

The ODS ExcelXP tagset has many options to change its behavior. Many are set with the options in the ODS statement. There are others that must be set through the style. Most of these are set in the TAGATTR= style attribute, an unused, leftover attribute from SAS® Release 7.01. This paper will show how to use the TAGATTR= style attribute to control the ExcelXP tagset in order to change the type, format, formula rotation, hiding, merging, and wrapping of cells, rows, and columns. This paper will also go behind the scenes to show how the tagset handles these settings and manages the problems presented by the XML that the tagset must create.

INTRODUCTION

The ODS style attribute TAGATTR= was born in the early days of ODS when the only Web browsers were Mosaic, Netscape Navigator, Internet Explorer, and Opera. The browser wars were underway, and the HTML specification was changing rapidly. The releases of SAS were still spaced widely apart, and with the first release of ODS HTML we, the developers of ODS, wanted to be sure we could continue to take advantage of new developments in HTML between the SAS releases. TAGATTR= got its name from tag attribute, and our thinking was that it would enable new attributes to be added into various parts of the HTML, even if they were completely new attributes that someone had just invented. That never came true. TAGATTR= was never used. Then came tagsets, and TAGATTR= was suddenly useful. It could hold anything, and a tagset could do anything with it. The ExcelXP tagset has made the greatest use of the TAGATTR= style attribute. This paper will explore the various uses for TAGATTR= and also explain how it actually works. There are several reasons TAGATTR= might be used: the first is its ease of use for the SAS programmer, the second is the way ODS works and the XML it has to create. Last is the way Excel interacts with its own XML. The attribute must be user-friendly, but the other relationships are adversarial at best. This paper will give some insight into the way the ODS ExcelXP tagset uses TAGATTR=. TAGATTR= is it when it comes programmatically accessing the special Excel abilities within cells. But TAGATTR= passes the “it” to the tagset, which then has to pass it on to the XML, which then passes the tag on to Excel.

The ODS Tagsets.Excelxp statement has many options, but those options are usually general in nature, and apply to the entire spreadsheet. When it comes to adding special control to specific cells or rows those options do not work so well. That is when TAGATTR= comes into play. By specifying values on the TAGATTR= style attribute it is possible to precisely change just one cell or row within a spreadsheet. Within the XML these controls manifest in two different ways. They either cause a new style definition to be created, or they add an XML attribute to the cell or row being written at that moment.

Since TAGATTR= is an ODS style attribute, it is possible to specify its value with a style element definition or as a style override. There will be examples of both methods in this paper. How you decide to do it is your choice.

TAG: FORMAT IS IT!

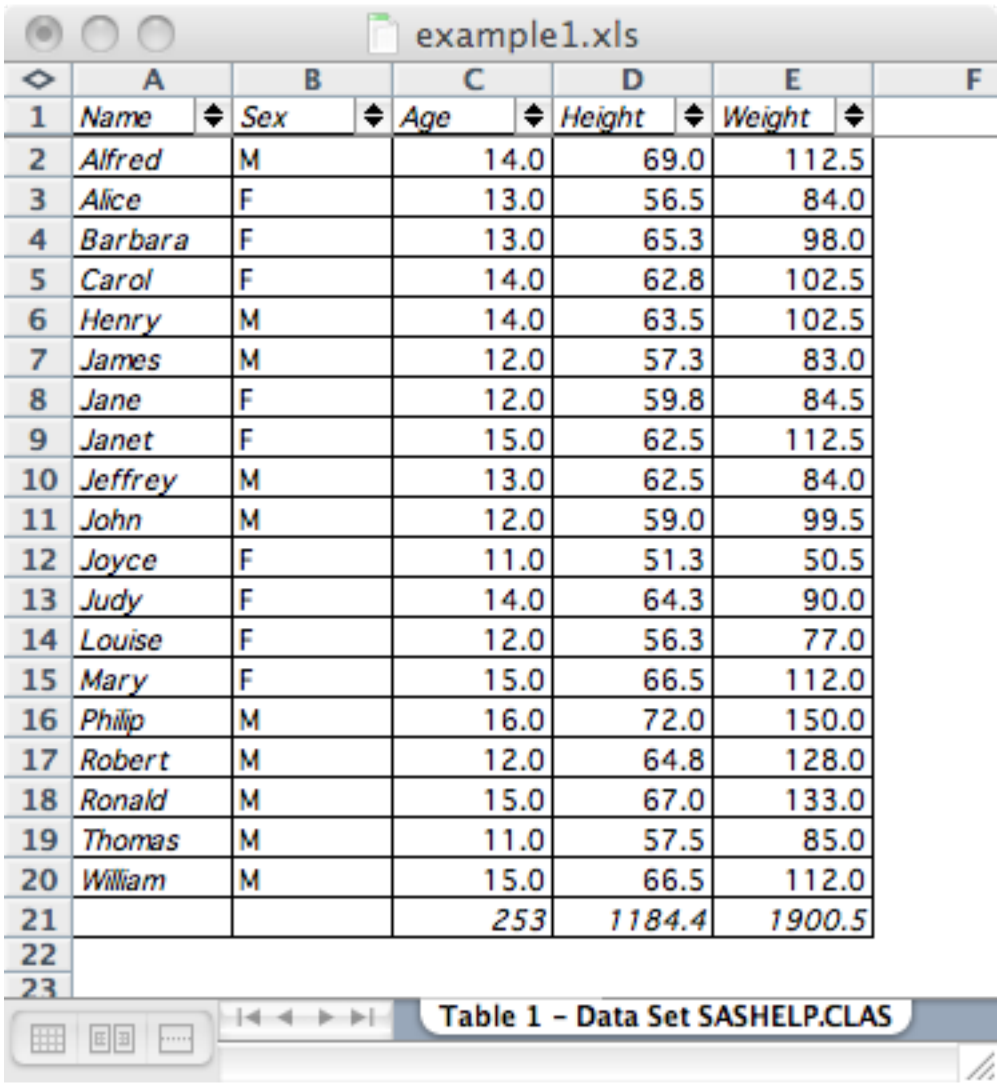
The first uses for TAGATTR= had to do with assigning Excel formats to various cells. Excel does not understand SAS formats, and there was no way to convert them internally, so if there was a special format that was desired beyond currency, percentage, string, or number then a format had to be provided. It was not long after that when we realized that type was also sometimes necessary. The original TAGATTR= processing took only a format. But when it became necessary to add type there had to be another way. That is when the new syntax was added. Each value had to be labeled so the tagset could pull it all apart and use it. Instead of just TAGATTR="###,###,###.00" the value was now TAGATTR="format:###,###,###.00 type:numeric".

```
ods tagsets.excelxp

    options(
    frozen_headers='Yes' autofilter='All'
    embedded_titles='No'
    )
```

```
      file="example1.xls" style=journal;


   ods noproctitle;
   proc print data=sashelp.class
      contents="sashelp.class";

      id Name;
      var Sex;
      var Age Height Weight / style={tagattr='format:##0.0'};
      sum Age Height Weight;
   run;

  ods tagsets.excelxp close;
```

The effect of the Excel format can be seen in that all the number

fields have preserved a decimal place with a 0.



	A	B	C	D	E	F
1	Name	Sex	Age	Height	Weight	
2	Alfred	M	14.0	69.0	112.5	
3	Alice	F	13.0	56.5	84.0	
4	Barbara	F	13.0	65.3	98.0	
5	Carol	F	14.0	62.8	102.5	
6	Henry	M	14.0	63.5	102.5	
7	James	M	12.0	57.3	83.0	
8	Jane	F	12.0	59.8	84.5	
9	Janet	F	15.0	62.5	112.5	
10	Jeffrey	M	13.0	62.5	84.0	
11	John	M	12.0	59.0	99.5	
12	Joyce	F	11.0	51.3	50.5	
13	Judy	F	14.0	64.3	90.0	
14	Louise	F	12.0	56.3	77.0	
15	Mary	F	15.0	66.5	112.0	
16	Philip	M	16.0	72.0	150.0	
17	Robert	M	12.0	64.8	128.0	
18	Ronald	M	15.0	67.0	133.0	
19	Thomas	M	11.0	57.5	85.0	
20	William	M	15.0	66.5	112.0	
21			253	1184.4	1900.5	
22						
23						

THE XML

This first example was the first TAGATTR= option that was added to the Excel tagset. It is not the most straightforward value to process, but it is one of the more common types. For the most part, anything specified by TAGATTR= either goes into a style definition or into the XML code that defines the cell. The options that go directly into the definition of the cell are the easiest, but the type that go into the style are not too awful. There is already processing in place for style attribute overrides of other types, so processing formats and other TAGATTR= attributes that go into the style was just a matter of taking advantage of the style attribute override processing that was already in place.

The XML that causes this format to be applied is in two places. First a new style definition must be created.

```
<Style ss:ID="data__r1" ss:Parent="data__r">
<Protection ss:Protected="1" />
<NumberFormat ss:Format="##0.0" />
</Style>
```

Then that style must be referenced by the cell that wants to use that format. In this case it is Age, Height and Weight.

The entire row of cells looks like this.

```
<Row ss:AutoFitHeight="1" ss:Height="15">
<Cell ss:StyleID="rowheader__l" ss:Index="1"><Data ss:Type="String">Alfred</Data></
Cell>
<Cell ss:StyleID="data__l" ss:Index="2"><Data ss:Type="String">M</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="3"><Data ss:Type="Number">14</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="4"><Data ss:Type="Number">69.0</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="5"><Data ss:Type="Number">112.5</Data></Cell>
</Row>
```

TAG: FORMAT AND TYPE, STUCK IN THE MUD!

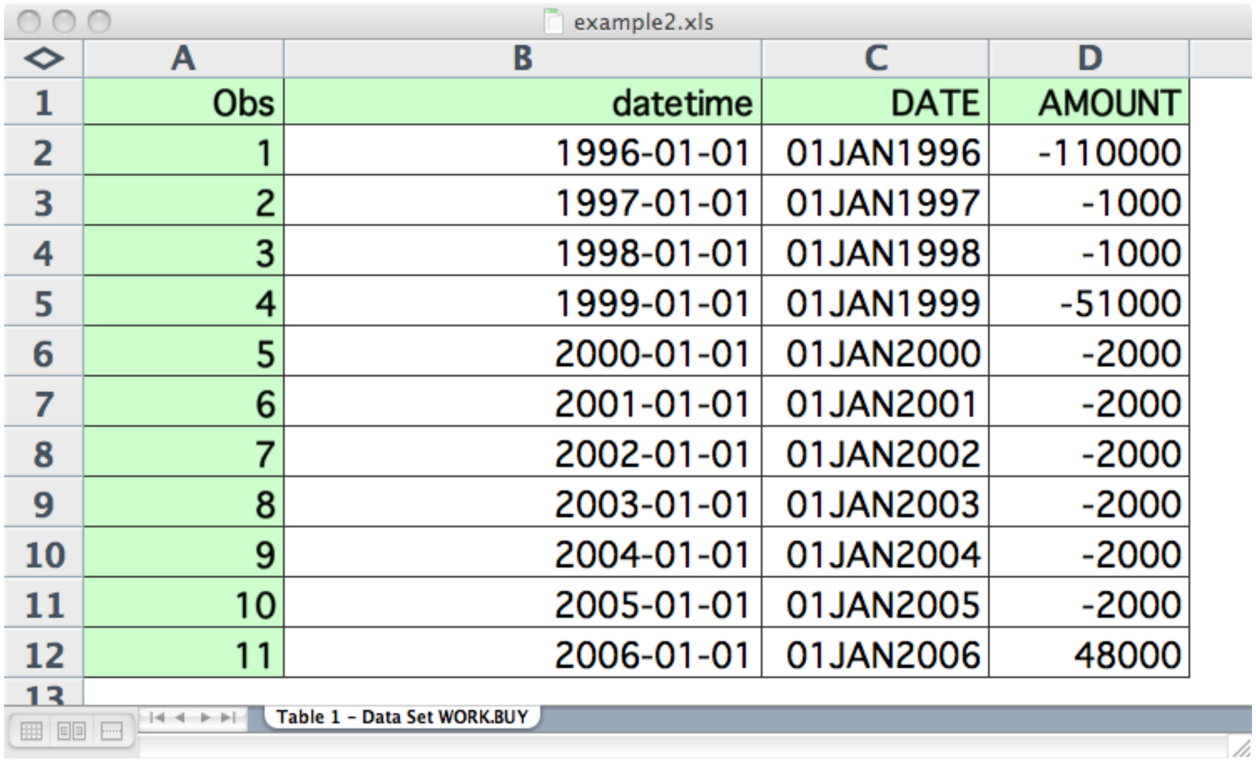
“Stuck in the mud”—it's like tag, but once tagged, you have stay put until a free person frees you by crawling between your legs.

The type is rarely used by its self. This example uses a SAS format and an Excel format and type to get the proper results for a date.

Setting the type on a value is not required most of the time, It is sometimes good for fields that look like numbers but you would rather have them as strings. The other most common use is to set up a datetime field. This example shows how that is done. The first part is to make sure the date is actually a date time and to multiply that value by the number of seconds in a day, 86400. The second part is tell SAS to use the ISO Date format, e8601dt, which is the format Excel expects the datetime to be in. This is the way that Excel stores dates. After that it is necessary to tell Excel that this column is a datetime and to also give it the format that it should use for the datetime.

```
ods tagsets.excelxp file='example2.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;
```

This example creates a table with two date columns: one is an actual Excel datetime and the other is a SAS date represented as a string in Excel. The datetime field can be used in formulas and displayed in various formats, while the SAS date column is just text that cannot do much of anything.



example2.xls

	A	B	C	D
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

THE XML

Here the format goes into the style, and the date and time is readily visible within the cell. In this row, notice the difference between the two date fields. One is actually an Excel datetime, and the other is just a string.

```
<Style ss:ID="data__r1" ss:Parent="data__r">
<Protection ss:Protected="1" />
<NumberFormat ss:Format="YYYY-MM-DD" />
</Style>


<Row ss:AutoFitHeight="1" ss:Height="15">
<Cell ss:StyleID="rowheader__r" ss:Index="1"><Data ss:Type="Number">1</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="2"><Data
ss:Type="DateTime">1996-01-01T00:00:00</Data></Cell>
<Cell ss:StyleID="data__r" ss:Index="3"><Data ss:Type="String">01JAN1996</Data></
Cell>
<Cell ss:StyleID="data__r" ss:Index="4"><Data ss:Type="Number">-110000</Data></
Cell>
</Row>
```

HIDE & SEEK, FORMULAS

Formulas are nice to have, but you can't necessarily see them until you change something or go looking for them.

This example shows some simple examples of formulas.

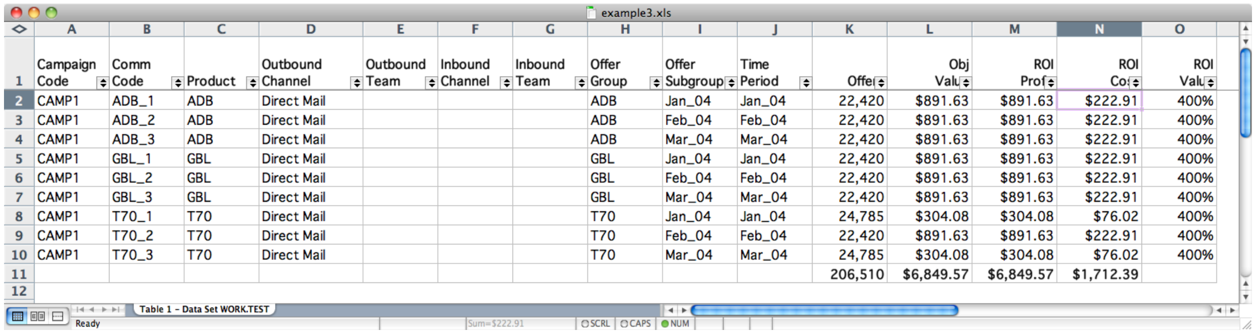
The need for formulas followed quickly on the heels of formats, so another possible attribute was added to the TAGATTR= processing. Formulas are quite useful, but they must be specified using the relative cell syntax rather than the absolute syntax commonly used by Excel. SpreadSheetML does not provide for absolute cell addressing like A1 or B4.

```
ods listing close;


data test;
length camp_code  comm_code $5
       product $3
       outbound_channel $11
       outbound_team
       inbound_channel  inbound_team $1
       offer_group $3
       offer_subgroup  time_period $6
       offers obj_value  roi_profit  roi_cost  roi_value 8;
input @1  camp_code
      @9  comm_code
      @17 product
      @23 outbound_channel $char11.
      @37 offer_group
      @43 offer_subgroup
      @52 time_period
      @61 offers
      @69 obj_value;
roi_profit = obj_value;
roi_cost   = roi_profit*0.25;
roi_value  = roi_profit/roi_cost;
label camp_code        = 'Campaign*Code'
      comm_code        = 'Comm*Code'
      product          = '*Product'
      outbound_channel = 'Outbound*Channel'
      outbound_team    = 'Outbound*Team'
      inbound_channel  = 'Inbound*Channel'
      inbound_team     = 'Inbound*Team'
      offer_group      = 'Offer*Group'
      offer_subgroup   = 'Offer*Subgroup'
      time_period      = 'Time*Period'
      offers           = '*Offers'
      obj_value        = 'Obj*Value'
      roi_profit       = 'ROI*Profit'
      roi_cost         = 'ROI*Cost'
      roi_value        = 'ROI*Value';
format offers comma8. obj_value roi_profit roi_cost
       dollar9.2 roi_value percent.;
cards;
CAMP1   ADB_1   ADB   Direct Mail   ADB   Jan_04   Jan_04   22420   891.634432
CAMP1   ADB_2   ADB   Direct Mail   ADB   Feb_04   Feb_04   22420   891.634432
CAMP1   ADB_3   ADB   Direct Mail   ADB   Mar_04   Mar_04   22420   891.634432
CAMP1   GBL_1   GBL   Direct Mail   GBL   Jan_04   Jan_04   22420   891.634432
CAMP1   GBL_2   GBL   Direct Mail   GBL   Feb_04   Feb_04   22420   891.634432
CAMP1   GBL_3   GBL   Direct Mail   GBL   Mar_04   Mar_04   22420   891.634432
CAMP1   T70_1   T70   Direct Mail   T70   Jan_04   Jan_04   24785   304.082208
CAMP1   T70_2   T70   Direct Mail   T70   Feb_04   Feb_04   22420   891.634432
CAMP1   T70_3   T70   Direct Mail   T70   Mar_04   Mar_04   24785   304.082208
;
```

```
run;


  /*-- Modify the Statistical style to fix --*/
  /*-- missing border lines.               --*/
  /*---------------------------------------*/
proc template;
  define style styles.XLStatistical;
    parent = styles.Statistical;
    style Header from Header /
      borderwidth=2;
    style RowHeader from RowHeader /
      borderwidth=2;
    style Data from Data /
      borderwidth=2;
  end;
run;
quit;


ods tagsets.excelxp file="example3.xls" style=xlstatistical
    options(autofilter='all' frozen_headers='yes' auto_subtotals='yes');


  /*-- Use Excel formulas to represent computed cells, and use an --*/
  /*-- Excel format (0%) to force Excel to report the percentage  --*/
  /*-- as 400% instead of 400.00%.  In the formulas 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 formulas used here match the computations performed in --*/
  /*-- the DATA step that created the columns.                    --*/
  /*--------------------------------------------------------------*/

title2 'Proc print of data using style statement with tagattr';
proc print data=test noobs label split='*';
  var camp_code comm_code product outbound_channel outbound_team
      inbound_channel inbound_team offer_group offer_subgroup
      time_period offers obj_value roi_profit;
  var roi_cost / style={tagattr='formula:RC[-1]*0.25'};
  var roi_value / style={tagattr='format:0% formula:RC[-2]/RC[-1]'};
  sum offers / style={tagattr='format:#,###'};
  sum obj_value  roi_profit roi_cost;
run;



ods tagsets.excelxp close;
```

The only way to really see what is going on here is to highlight one of the calculated cells and look at the formula window.

```
=M2*0.25
```



example3.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Campaign Code	Comm Code	Product	Outbound Channel	Outbound Team	Inbound Channel	Inbound Team	Offer Group	Offer Subgroup	Time Period	Offe	Obj Valu	ROI Prof	ROI Co	ROI Valu
2	CAMP1	ADB_1	ADB	Direct Mail				ADB	Jan_04	Jan_04	22,420	$891.63	$891.63	$222.91	400%
3	CAMP1	ADB_2	ADB	Direct Mail				ADB	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
4	CAMP1	ADB_3	ADB	Direct Mail				ADB	Mar_04	Mar_04	22,420	$891.63	$891.63	$222.91	400%
5	CAMP1	GBL_1	GBL	Direct Mail				GBL	Jan_04	Jan_04	22,420	$891.63	$891.63	$222.91	400%
6	CAMP1	GBL_2	GBL	Direct Mail				GBL	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
7	CAMP1	GBL_3	GBL	Direct Mail				GBL	Mar_04	Mar_04	22,420	$891.63	$891.63	$222.91	400%
8	CAMP1	T70_1	T70	Direct Mail				T70	Jan_04	Jan_04	24,785	$304.08	$304.08	$76.02	400%
9	CAMP1	T70_2	T70	Direct Mail				T70	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
10	CAMP1	T70_3	T70	Direct Mail				T70	Mar_04	Mar_04	24,785	$304.08	$304.08	$76.02	400%
11											206,510	$6,849.57	$6,849.57	$1,712.39	
12															

Table 1 – Data Set WORK.TEST

Ready Sum=$222.91 SCRL CAPS NUM

THE XML

Since formulas are more of an interactive feature it is hard to show them on paper or slides. I can show the XML. In this case, unlike most of the other features, formulas are actually specified as part of the XML that defines the cell. This is the easiest type of TAGATTR= value to process. If it goes directly in the cell the tagset just has to print it along with everything else when the cell is written out.

```
<Cell ss:StyleID="data__r_1" ss:Formula="rc[-1]*0.25" ss:Index="14"><Data
ss:Type="Number">222.91</Data></Cell>
<Cell ss:StyleID="data__r_2" ss:Formula="RC[-2]/RC[-1]" ss:Index="15"><Data
ss:Type="Number">4</Data></Cell>
```

TAG, ROTATION IS IT!

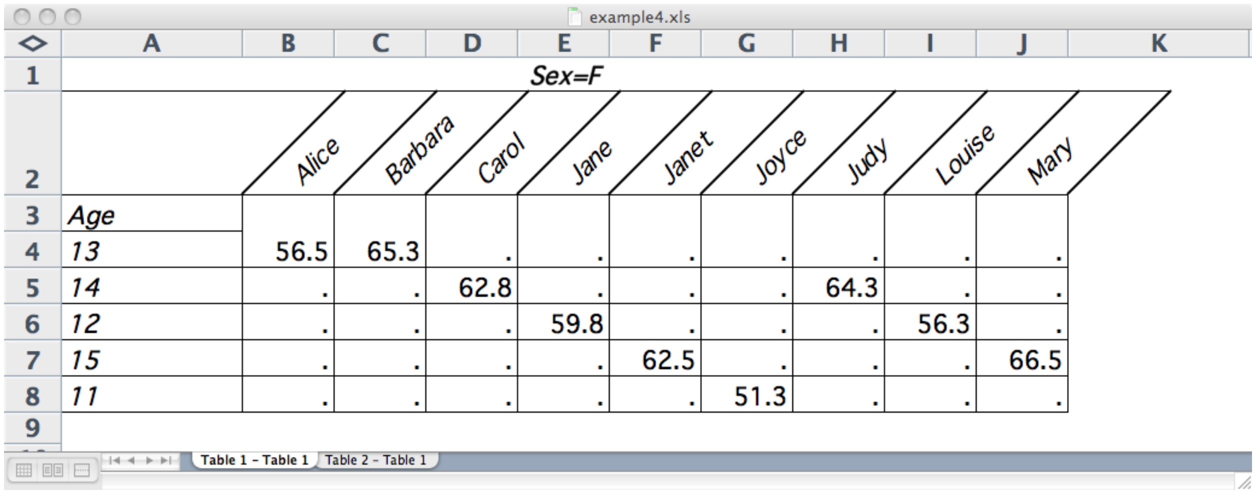
Rotation came about not because we absolutely needed it, but because it would be nice to have and it can make tables look quite nice. It is a value that still applies to a single cell and, like format and formula, also goes into the style definition used by that cell. So the internal processing for rotation was very much the same as that for format and formula.

```
proc template;
   define style styles.myjournal;
     parent = styles.Journal;
       style angle_header from header /
             tagattr = 'rotate:45'
         ;
   end;
 run;

 ods tagsets.ExcelXP file="example4.xls"
              style=myjournal;

                 proc sort data=sashelp.class out=foo;
                 by sex;

             proc tabulate data=foo
                  order=data missing format=8.0 noseps
                  formchar=',           ';
                by sex;
                class age sex name;
                classlev name /s=angle_header;
```

```
            var height weight;
            title;
            table age,  name=' '*(height=' '*median=' '*F=5.3);

        run;


    ods tagsets.ExcelXP close;
```

Rotation is one of the TAGATTR= values that adds to the style definition. This could be difficult, but because the tagset already has to accommodate all the normal style overrides like font, color, font weight, and so on, the processing required is not so difficult. Especially in this case, the processing is brought to a minimum because this is not even a style attribute override—it is just a new style, and the name of the style is the override. There is no special handling required like there would be if TAGATTR="Rotate:45" were specified down in the PROC Tabulate code where we set the style equal to angle_header.

The output from this program looks quite nice with angled headers.



	A	B	C	D	E	F	G	H	I	J	K
1	Sex=F										
2		Alice	Barbara	Carol	Jane	Janet	Joyce	Judy	Louise	Mary	
3	Age										
4	13	56.5	65.3	.	.	.	.	.	.	.	
5	14	.	.	62.8	.	.	.	64.3	.	.	
6	12	.	.	.	59.8	.	.	.	56.3	.	
7	15	.	.	.	.	62.5	.	.	.	66.5	
8	11	.	.	.	.	.	51.3	.	.	.	
9											

THE XML

The XML for rotation is part of the style definition.

```
<Style ss:ID="angle_header" ss:Parent="table">
<Alignment ss:WrapText="1" ss:Rotate="45"/>
<Font ss:FontName="Arial, Helvetica, Helv" ss:Size="10" ss:Italic="1"
ss:Color="#000000" />
<Interior ss:Color="#FFFFFF" ss:Pattern="Solid" />
<Protection ss:Protected="1" />
</Style>
```

5,10,15,20,25,30,35,40,... HIDDEN

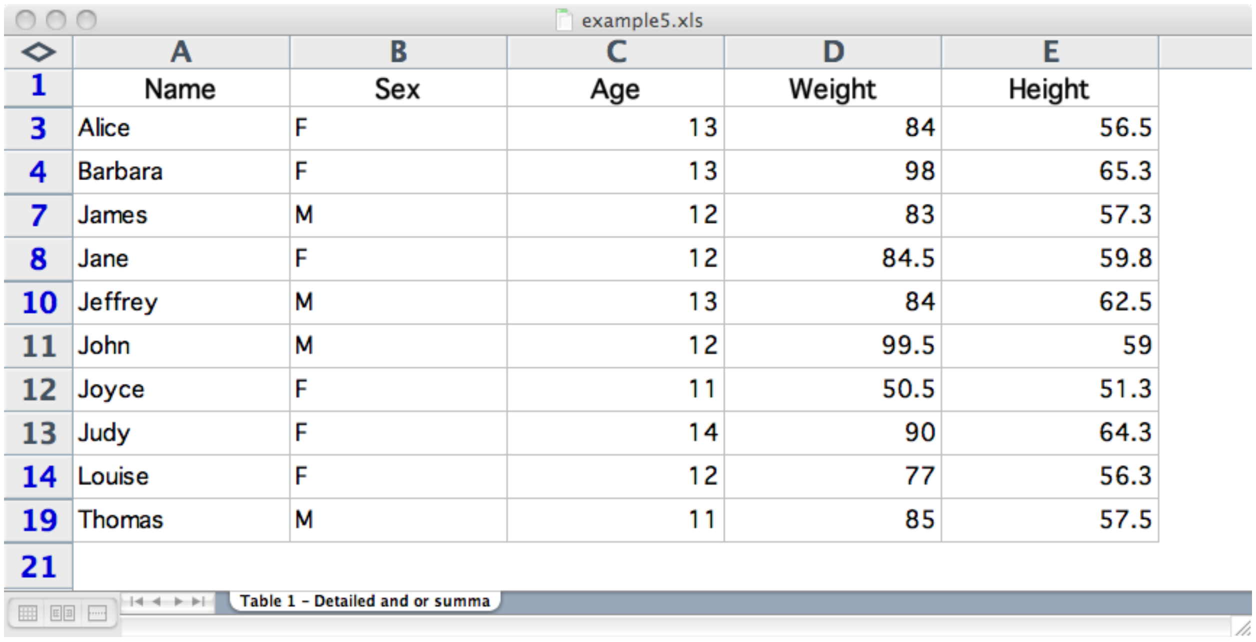
As a kid, home base for hide and seek was a telephone pole, and we usually counted by fives to 500.

When looking at usability, hiding columns seemed easiest with the ODS statement options. But hiding a row is not so easy because it's frequently data driven, and who knows which rows should be hidden ahead of time?

There were many requests for the ability to hide rows or columns, but it took a while before these features were actually added to the tagset. The difficulty came from how hiding works. Hiding a column requires that the column definition know that it is hidden, and that definition is the first thing in the table, long before any cells come along. A row must be defined as hidden, and that definition comes before any of the cells that it contains. The solution was to add column hiding as an option on the ODS statement and add row hiding to TAGATTR=. The only real problem is that then the row definition and all the cells the row contains had to be delayed until all the cells in the row were done. This required different processing than any of the other attribute on TAGATTR, so it took longer to implement.

```
proc template;
  define style styles.mymeadow;
    parent=styles.meadow;

      style hidden from data/
        tagattr="hidden:yes"
    ;
end;
run;

ods tagsets.excelxp file="example5.xls" style=mymeadow;

proc report data=sashelp.class nowd;
   column Name Sex Age Weight Height;
   define Weight / display;

   compute Weight;
      if (Weight > 100) then do;
         CALL DEFINE(_ROW_, "STYLE",
         "STYLE=hidden");
      end;
   endcomp;
run;

proc report data=sashelp.class nowd;
   column Name Sex Age Weight Height;
   define Weight / display;

run;

ods _all_ close;
```

The output created by this program looks fine, but notice there are no rows where the weight is over 100. The clue lies in the row numbers. Rows 2, 5, 9 and 15-18 are all hidden.


example5.xls

	A	B	C	D	E
1	Name	Sex	Age	Weight	Height
3	Alice	F	13	84	56.5
4	Barbara	F	13	98	65.3
7	James	M	12	83	57.3
8	Jane	F	12	84.5	59.8
10	Jeffrey	M	13	84	62.5
11	John	M	12	99.5	59
12	Joyce	F	11	50.5	51.3
13	Judy	F	14	90	64.3
14	Louise	F	12	77	56.3
19	Thomas	M	11	85	57.5
21					

THE XML

This is a little bit sneaky—TAGATTR= is given the hidden option in a style definition. But hidden is not part of Excel style definition. The goal is to hide a row where there is a cell that uses the hidden style. The hidden attribute actually goes on the row tag.

Here is what a hidden row looks like.

```
<Row ss:AutoFitHeight="1" ss:Hidden="1" ss:Height="15">
<Cell ss:StyleID="hidden" ss:Index="1"><Data ss:Type="String">William</Data></Cell>
<Cell ss:StyleID="hidden" ss:Index="2"><Data ss:Type="String">M</Data></Cell>
<Cell ss:StyleID="hidden" ss:Index="3"><Data ss:Type="Number">15</Data></Cell>
<Cell ss:StyleID="hidden" ss:Index="4"><Data ss:Type="Number">112</Data></Cell>
<Cell ss:StyleID="hidden" ss:Index="5"><Data ss:Type="Number">66.5</Data></Cell>
</Row>
```

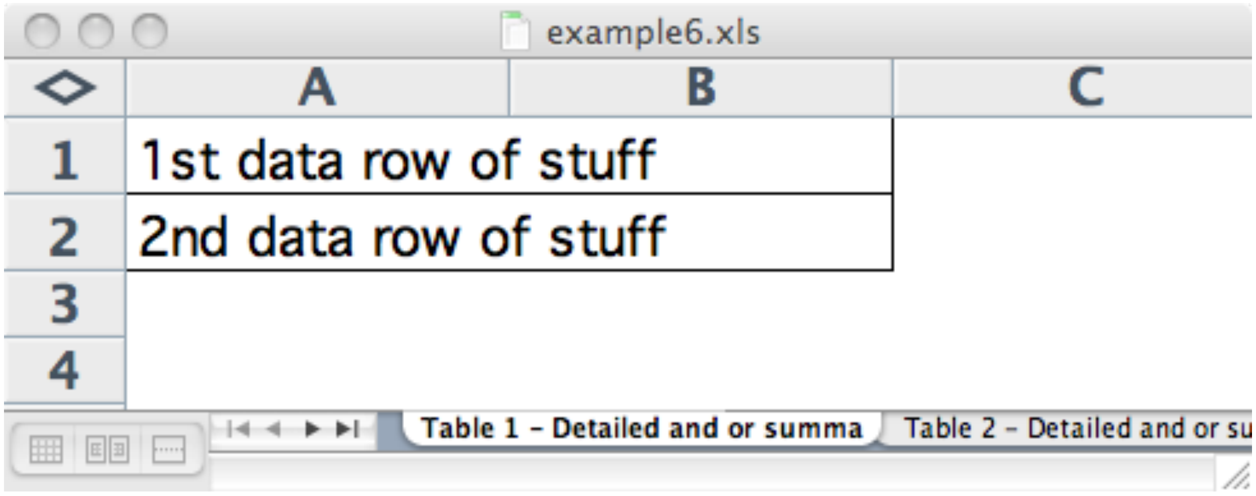
The processing for TAGATTR= in this case is a bit tricky. The row has to be stashed away until all the cells have been processed, and if any of those cells have a TAGATTR= with hidden:yes then the row has to specify ss:Hidden="1".

TAG, MERGING IS IT

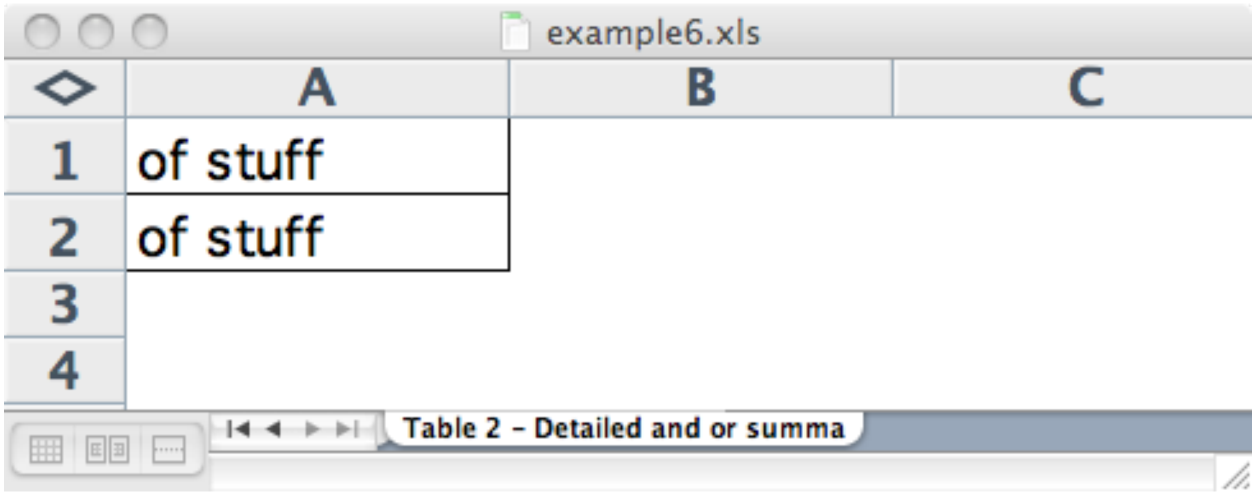
The MERGEACROSS option becomes useful only rarely, it enables longer data elements to span across the width of the current table or even the title width if that was given on the ODS statement. This example is a very simplistic example of how this option works. The most common use for this is when there is string data, like comments, that can vary greatly in length. This feature can be used to enable the data to span as far as it needs to past the edge of the table.

```
ods listing close;

data test;
  d1 = '1st data row of stuff'; output;
  d1 = '2nd data row of stuff'; output;
run;

ods tagsets.excelxp file="example6.xls" style=journal ;

title ;

proc report data=test nowd;
  columns d1;
  define d1 / '' style={tagattr="mergeAcross:yes"};
run;

proc report data=test nowd;
  columns d1;
  define d1 / '';
run;


ods tagsets.excelxp close;
ods listing;
```

The easiest way to see the effects of this is to look at the row numbers in the spreadsheet and to compare it against a spreadsheet that has no hidden rows. The first PRINT procedure creates a table with merged cells that show their entire contents. The second PRINT procedure creates a table without merged cells where their contents are partially hidden.



1	1st data row of stuff
2	2nd data row of stuff



1	of stuff
2	of stuff

THE XML

The MERGEACROSS attribute appears in the cell tag, and is therefore one of the easier options to process. Everything is right there when the cell is created, and nothing has to go anywhere else.

In this test, the cells in the first table have an additional MERGEACROSS attribute, whereas the cells in the second table do not.

```
<Cell ss:StyleID="data__l" ss:MergeAcross="1" ss:Index="1"><Data
ss:Type="String">1st data row of stuff</Data></Cell>

<Cell ss:StyleID="data__l" ss:Index="1"><Data ss:Type="String">1st data row of
stuff</Data></Cell>
```

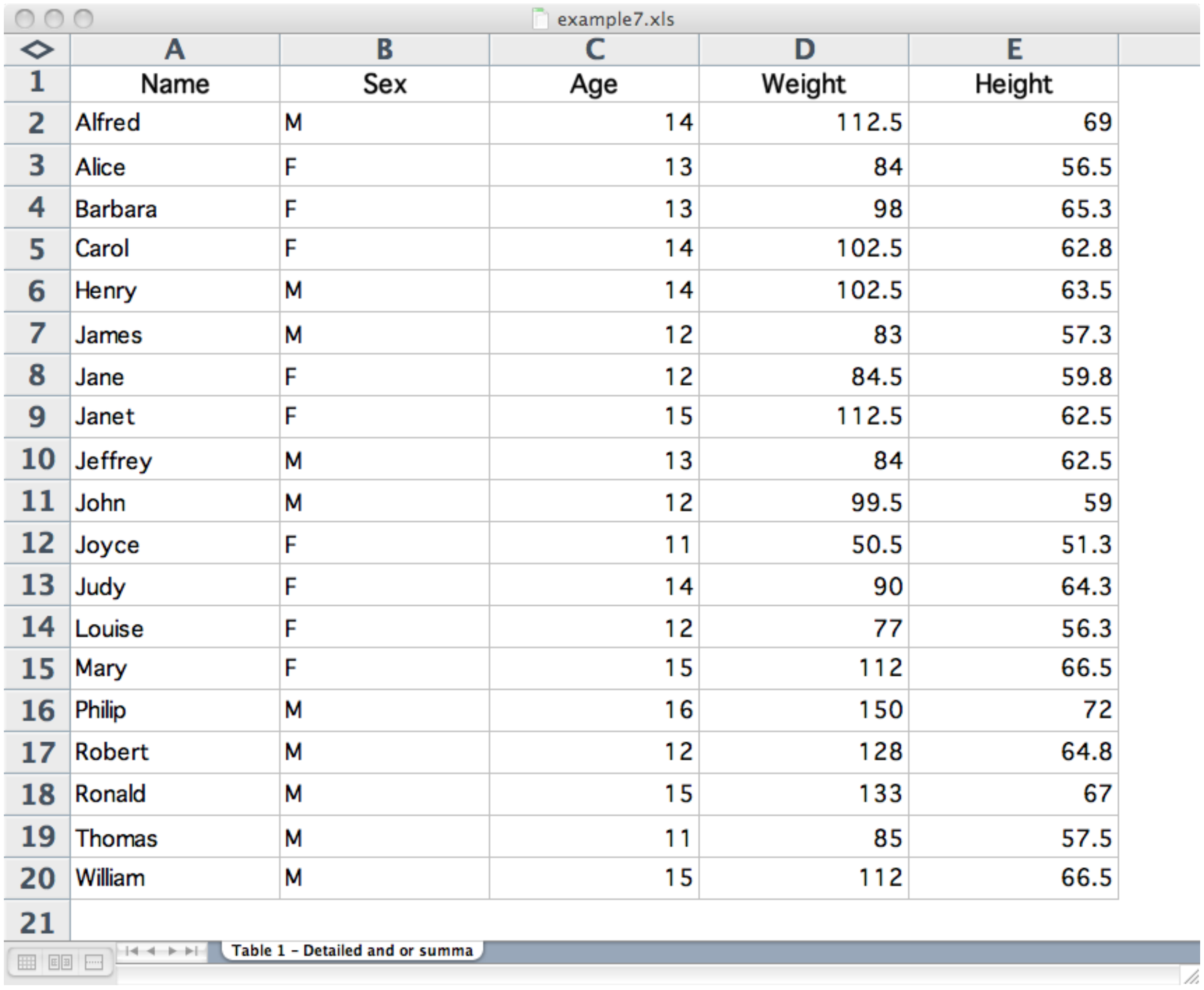
The difference between them is easily visible.

HIDE & SEEK, WRAP IS QUIETLY HIDDEN UNTIL IT IS UNCOVERED

Wrapping text is a real problem in Excel. Whether wrapping is specified or not, Excel behaves the same way until you try to manipulate the worksheet. OpenOffice works the way any of us would expect, and is therefore better in its rendering of the spreadsheet. Nonetheless, wrapping has an effect in Excel once you try to change the worksheet after it is created. There are two interactive options that control wrapping. Wrapping is on by default but it can be turned off with the WRAPTEXT option. There have been many requests for the ability to turn off wrapping. `wraptext='no'` does that. Setting wrap on TAGATTR= can counter either behavior for a given cell. In this example the default behavior is being turned off but wrapping is being turned on for a specific cell.

```
proc template;
  define style styles.mymeadow;
    parent=styles.meadow;

      style hidden from data/
        tagattr="wrap:no"
    ;
end;
run;

ods tagsets.excelxp file="example7.xls" style=mymeadow options(wraptext='no'
doc='help');

proc report data=sashelp.class nowd;
   column Name Sex Age Weight Height;
   define Weight / display;

   compute Weight;
      if (Weight > 100) then do;
         CALL DEFINE(_ROW_, "STYLE",
         "STYLE=data[tagattr='wrap:yes']");
      end;
   endcomp;
run;



ods _all_ close
```

The output for this test case does not show much. It really looks quite normal. But a quick look at the XML shows a different story. And manipulating the spreadsheet by hand will reveal different behavior as well.


example7.xls

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

THE XML

Since this attribute mostly controls how the worksheet behaves after the fact, it is easiest to see the results of these options by looking at the XML. The only place WRAPTEXT appears is in the style definitions. Normally, it will be in every style that is defined. But setting `wraptext=no` turns this off. The data style definition is affected by this, and is therefore missing the WRAPTEXT attribute. Because this feature goes in the style definition, the tagset treats it as a style override. It is a sort of exception, but the tagset has processing to handle overrides, and this TAGATTR= value adds to that.

```
<Style ss:ID="data" ss:Parent="table">
<ss:Borders>
<ss:Border ss:Position="Right" ss:Color="#CCD6BE" ss:Weight="1"
ss:LineStyle="Continuous" />
<ss:Border ss:Position="Bottom" ss:Color="#CCD6BE" ss:Weight="1"
ss:LineStyle="Continuous" />
</ss:Borders>
<Font ss:FontName="Arial, Albany AMT, Helvetica" ss:Size="8" ss:Color="#000000" />
<Interior ss:Color="#FFFFFF" ss:Pattern="Solid" />
<Protection ss:Protected="1" />
</Style>
```

Later, the REPORT procedure specifies `wrap:yes` on TAGATTR=. This results in two new style definitions being created for use by the cells in any row with a weight greater than 100. There is one left-justified data style and one right-justified data style.

```
<Style ss:ID="data__l1" ss:Parent="data__l">
<Alignment ss:WrapText="1" ss:Horizontal="Left"/>
<Protection ss:Protected="1" />
<NumberFormat ss:Format="General" />
</Style>
<Style ss:ID="data__r1" ss:Parent="data__l">
<Alignment ss:WrapText="1" ss:Horizontal="Left"/>
<Protection ss:Protected="1" />
<NumberFormat ss:Format="General" />
</Style>
```

The row that uses these styles looks like this.

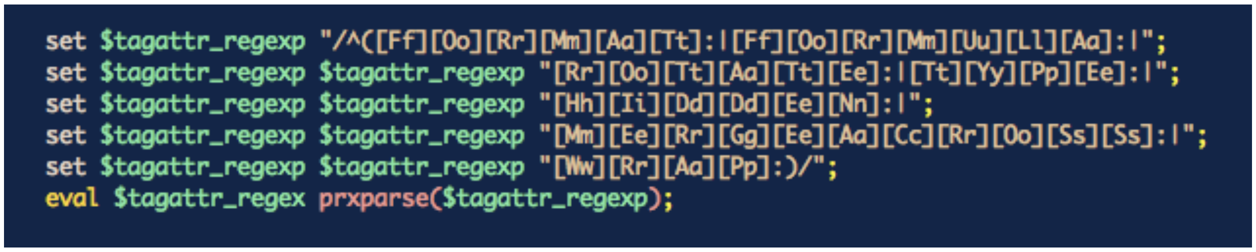
```
<Row ss:AutoFitHeight="1" ss:Height="15">
<Cell ss:StyleID="data__l1" ss:Index="1"><Data ss:Type="String">Alfred</Data></
Cell>
<Cell ss:StyleID="data__l1" ss:Index="2"><Data ss:Type="String">M</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="3"><Data ss:Type="Number">14</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="4"><Data ss:Type="Number">112.5</Data></Cell>
<Cell ss:StyleID="data__r1" ss:Index="5"><Data ss:Type="Number">69</Data></Cell>
</Row>
```

Now, for more geeky tagset programming stuff! Feel free to cover your eyes and count to 1000 until this part is over...

THE INTERNAL DETAILS.

These examples have shown both how to use the options on TAGATTR, and the snippets of XML that are affected by each option. But there is more to it than that. The tagset has to parse the TAGATTR= value for the options and their values, it then has to save them in a way that they can be easily used when the time comes. There are three different scenarios: the first is that the value will directly affect the attributes for the cell, the second is that it will go into a new style definition that the cell will then reference, and the third is the one used by hidden where the value of TAGATTR= changes the XML for the row but not the cell.

The first challenge is to parse the value of TAGATTR= for values. This is done with a regular expression, the expression must be case insensitive. The setup for the regular expression looks like this.



```
set $tagattr_regexp "/^([Ff][Oo][Rr][Mm][Aa][Tt]:|[Ff][Oo][Rr][Mm][Uu][Ll][Aa]:|";
set $tagattr_regexp $tagattr_regexp "[Rr][Oo][Tt][Aa][Tt][Ee]:|[Tt][Yy][Pp][Ee]:|";
set $tagattr_regexp $tagattr_regexp "[Hh][Ii][Dd][Dd][Ee][Nn]:|";
set $tagattr_regexp $tagattr_regexp "[Mm][Ee][Rr][Gg][Ee][Aa][Cc][Rr][Oo][Ss][Ss]:|";
set $tagattr_regexp $tagattr_regexp "[Ww][Rr][Aa][Pp]:)/";
eval $tagattr_regex prxparse($tagattr_regexp);
```

More simply, this expression is looking for any of the following: `format:, formula:, Rotate:, Type:, Hidden:, MergeAcross:,` or `Wrap:`.

At this point the regular expression is ready for use. Here is where the values get saved away into an array named `$attrs`. The entire process can be watched from the log by adding options(debug_level='-5') to the ODS statement. The first thing to check for is a `':'`. If that is not present than any value that is in TAGATTR= must be a format. If there is a `':'` then the value of TAGATTR= has to be parsed for any values it might have.

```
/*----------------------------------------------------eric-*/
/*-- If there is a : then we need to parse for format     --*/
/*-- and/or formula.  To add new attributes change the    --*/
/*-- tagattr_regexp above.                                --*/
/*-------------------------------------------------17Dec04-*/
do /if $debug_level = -5;
    putlog "Event: " event_name "tagattr: " tagattr;
done;
```

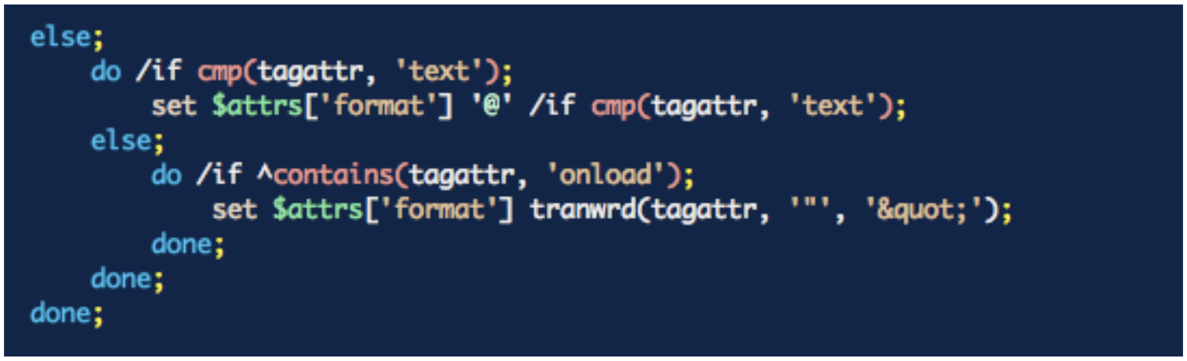
This section loops through the values listed in TAGATTR= with the regular expression. Each value is added to an array called `$attrs` with the option name as the key.

```
do /if index(tagattr, ":") > 0;

    eval $index 1;
    /* get the first section to look at */
    eval $tmp scan(tagattr, $index, ' ');

    do /while !cmp($tmp, ' ');

        /* look for an attribute */
        do /if prxmatch($tagattr_regex, $tmp);

            /* get the attribute name */
            eval $attr lowcase(scan($tmp, 1, ':'));
            /* get what is left */
            eval $len index($tmp, ':')+1;
            eval $tmp2 substr($tmp, $len);
            eval $attrs[$attr] strip(scan($tmp2, 1, " "));

        else;
            /* it didn't start with a name so add it on */
            set $attrs[$attr] $attrs[$attr] ' ' $tmp;
        done;

        eval $index $index + 1;
        /* get the next section */
        set $tmp scan(tagattr, $index, ' ');
    done;
```

Check to see this entry is a format and if it is `'text'` change it to `'@'` so Excel will be happy.

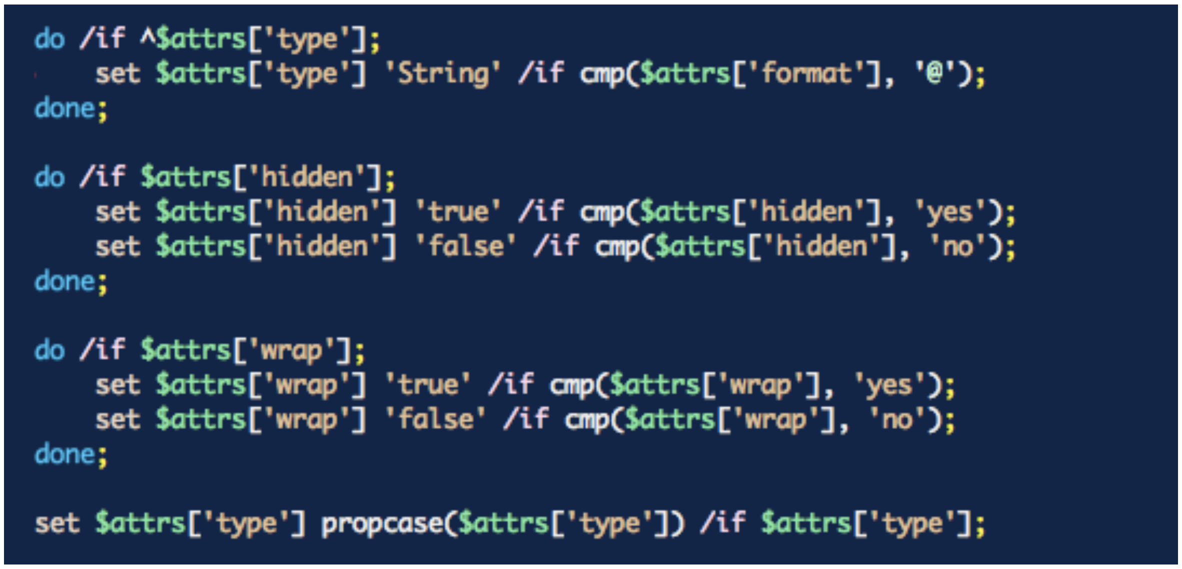
```
    do /if $attrs['format'];
        set $attrs['format'] '@' /if cmp($attrs['format'], 'text');
    done;
```

It has to be a format if there is not a name: in front of the value. There are some styles that have `'onload'` as the TAGATTR= value—those will cause problems, so get rid of them.



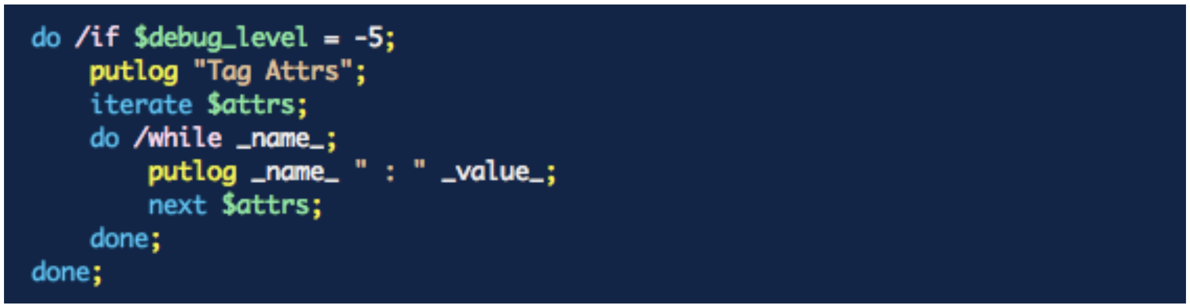
```
else;
    do /if cmp(tagattr, 'text');
        set $attrs['format'] '@' /if cmp(tagattr, 'text');
    else;
        do /if ^contains(tagattr, 'onload');
            set $attrs['format'] tranwrd(tagattr, '"', '&quot;');
        done;
    done;
done;
```

Now, post-process all the entries that need it. Type needs to be `'String'` if the format is `'@'`. Normalize yes/no, true/false to true/false so the values are easier to test. Type needs to be properly cased if it was set. The tagset cannot rely on the programmer to always properly case the type the way Excel wants.



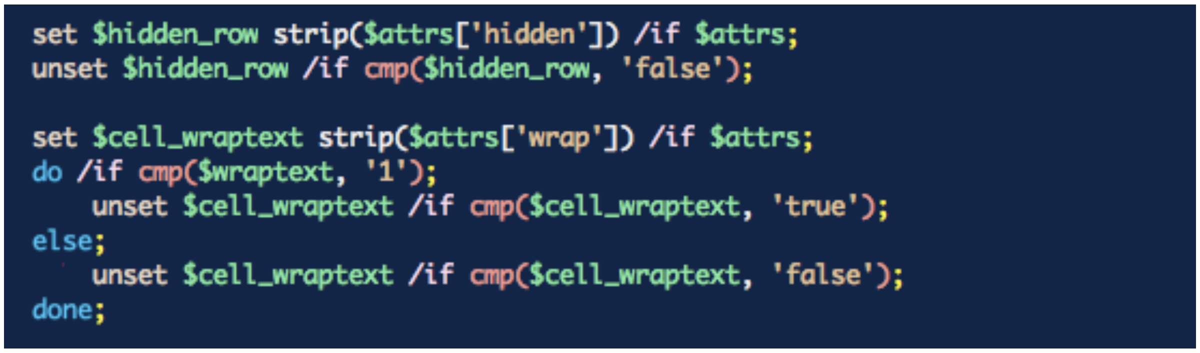
```
do /if ^$attrs['type'];
    set $attrs['type'] 'String' /if cmp($attrs['format'], '@');
done;

do /if $attrs['hidden'];
    set $attrs['hidden'] 'true' /if cmp($attrs['hidden'], 'yes');
    set $attrs['hidden'] 'false' /if cmp($attrs['hidden'], 'no');
done;

do /if $attrs['wrap'];
    set $attrs['wrap'] 'true' /if cmp($attrs['wrap'], 'yes');
    set $attrs['wrap'] 'false' /if cmp($attrs['wrap'], 'no');
done;

set $attrs['type'] propcase($attrs['type']) /if $attrs['type'];
```

Debug sections can show how everything looks. These are all over the tagset. This one will show the results of all this TAGATTR= processing.



```
do /if $debug_level = -5;
    putlog "Tag Attrs";
    iterate $attrs;
    do /while _name_;
        putlog _name_ " : " _value_;
        next $attrs;
    done;
done;
```

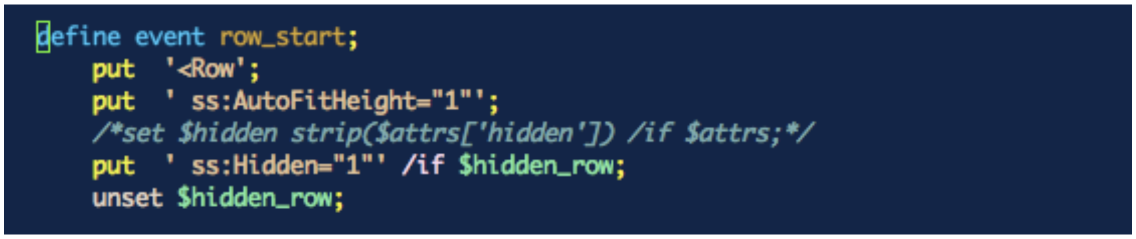
Finally, here are the two special cases, if hidden was set, the `$hidden_row` variable is now set so the row can easily hide itself independently of any fluctuations that might occur in other cells in the row. Wrapping text has interactions with the WRAPTEXT option in the ODS statement. All of that is handled here, so that the XML can be optimized to do the right thing.



```
set $hidden_row strip($attrs['hidden']) /if $attrs;
unset $hidden_row /if cmp($hidden_row, 'false');

set $cell_wraptext strip($attrs['wrap']) /if $attrs;
do /if cmp($wraptext, '1');
    unset $cell_wraptext /if cmp($cell_wraptext, 'true');
else;
    unset $cell_wraptext /if cmp($cell_wraptext, 'false');
done;
```

That's it for the processing of TAGATTR. `$attrs`, `$hidden_row`, and `$cell_wraptext` will be used later on as needed. Looking for those variables reveals their final use.

`$hidden_row` is one of the easiest to see. When the row starts, `ss:Hidden="1"` is printed if `$hidden_row` is true. `$hidden_row` is then unset for the next row.



```
define event row_start;
    put  '<Row';
    put  ' ss:AutoFitHeight="1"';
    /*set $hidden strip($attrs['hidden']) /if $attrs;*/
    put  ' ss:Hidden="1"' /if $hidden_row;
    unset $hidden_row;
```

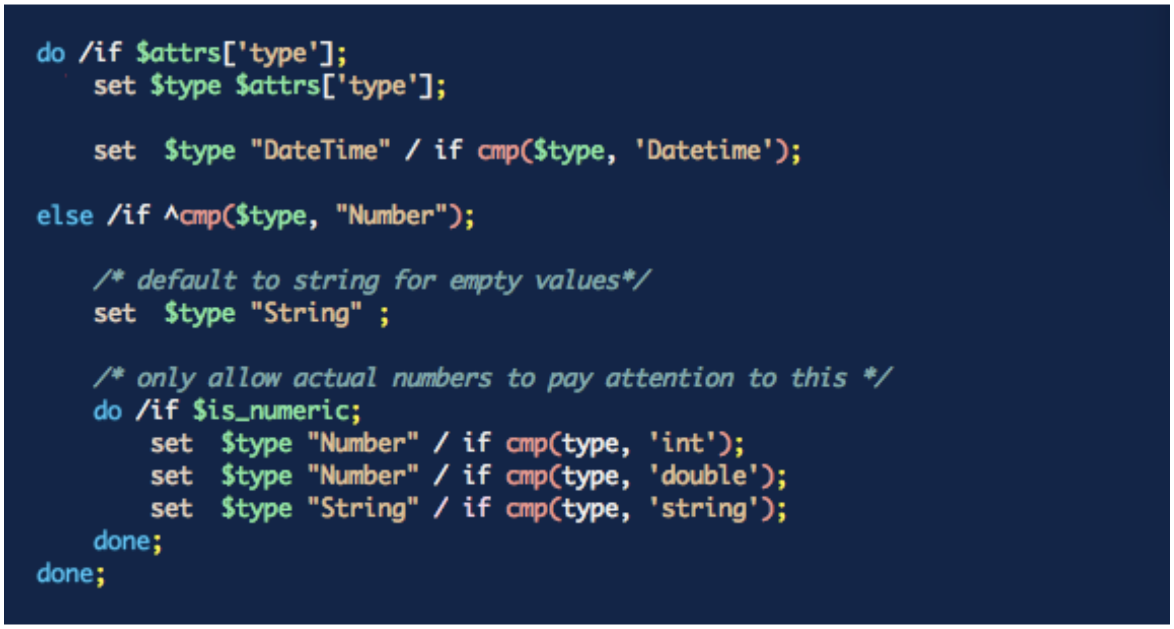
`$cell_wraptext` is a little more complicated. The alignment tag is part of the style definition. The align_tag event is used to create that section of the style. First, if `$cell_wraptext` is set, `just` and `vjust` must be redefined. If they are not, then Excel loses them even though they are defined in the parent style. I, if `$cell_wraptext` is not set, then it is changed to true, which is the default behavior. Finally, `ss:WrapText="1"` is printed if `$cell_wraptext` is true.

```
define event align_tag;
    start:
        break /if $align_tag;

        set $align_tag "True";

        do /if $cell_wraptext;
            set $just just;
            set $vjust vjust;
        done;

        do /if ^$cell_wraptext;
            set $cell_wraptext 'true' /if $wraptext;
        done;

        put '<Alignment';

        do /if contains($htmlclass, 'system')  ;
            do /if contains($just, 'l')  ;
                break /if !$merge_titles;
            done;
        done;

        do /if cmp($cell_wraptext, 'true');
            put ' ss:WrapText="1"';
        done;
        set $align_tag "True";

    finish:
        break /if ^$align_tag;
        putl '/>';
        unset $align_tag;
end;
```

Rotate is another piece of the style definition. After the first part of the align tag is created, this code evaluates `$attrs` to see if rotate should be added.

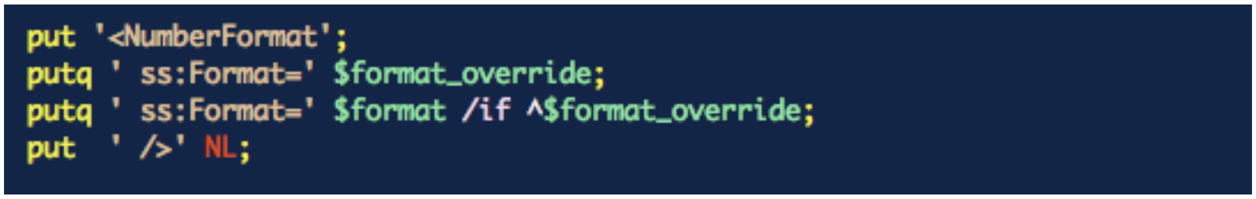
```
unset $rotate;
set $rotate strip($attrs['rotate']) /if $attrs;
do /if $rotate;
    trigger align_tag start;
    putq ' ss:Rotate=' $rotate;
done;
```

The type is handled along with the automatic types. If a type is set with TAGATTR, then that is the type that is used. If it happens to be a datetime, then it is set explicitly so that the uppercasing is correct. Otherwise, the type is set according to the type of value that ODS thinks it is.



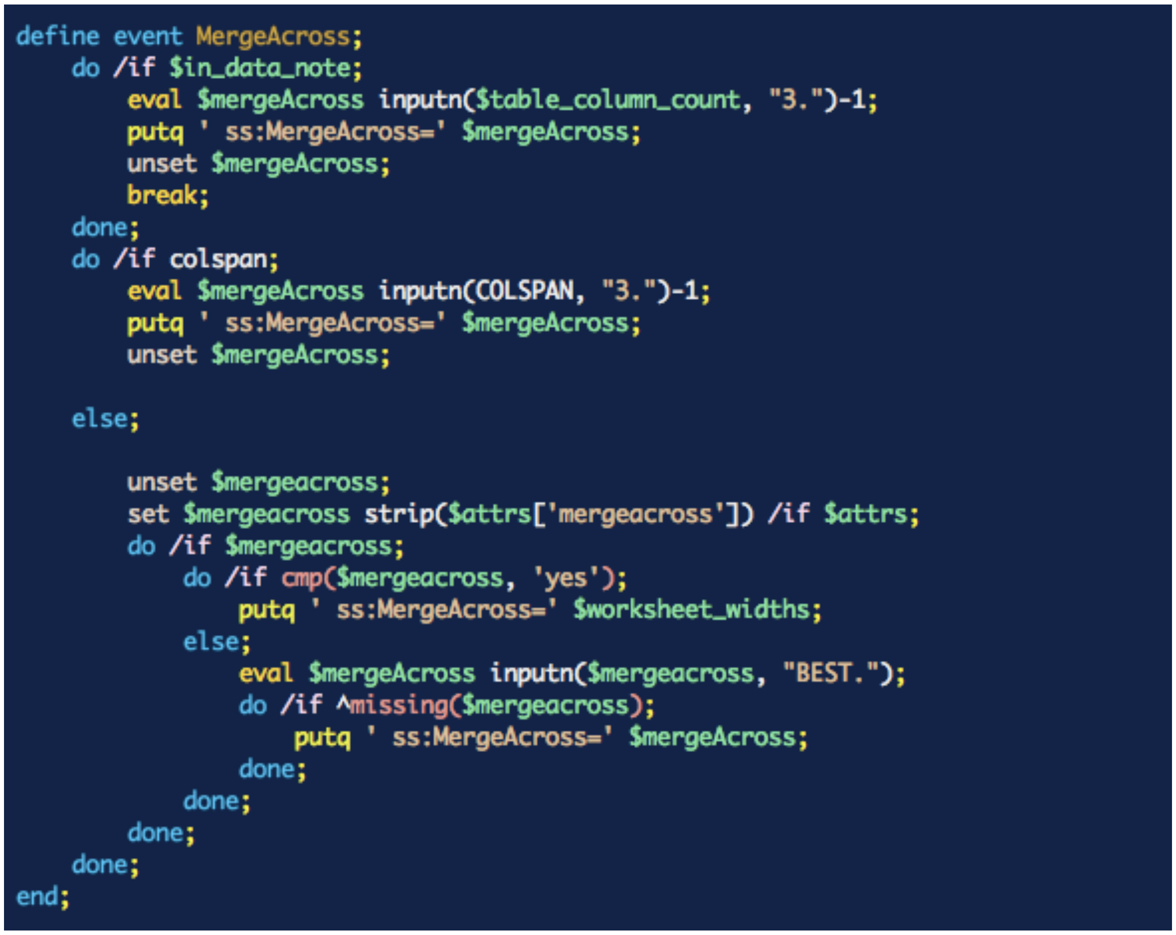
```
do /if $attrs['type'];
    set $type $attrs['type'];

    set  $type "DateTime" / if cmp($type, 'Datetime');

else /if ^cmp($type, "Number");

    /* default to string for empty values*/
    set  $type "String" ;

    /* only allow actual numbers to pay attention to this */
    do /if $is_numeric;
        set  $type "Number" / if cmp(type, 'int');
        set  $type "Number" / if cmp(type, 'double');
        set  $type "String" / if cmp(type, 'string');
    done;
done;
```

Formats are simpler. The value of `$format_override` is set to `$attrs['format']`, and when the style is created that value is used. The value of `$format` could be currency or percentage if the tagset had detected a currency symbol or percentage sign.



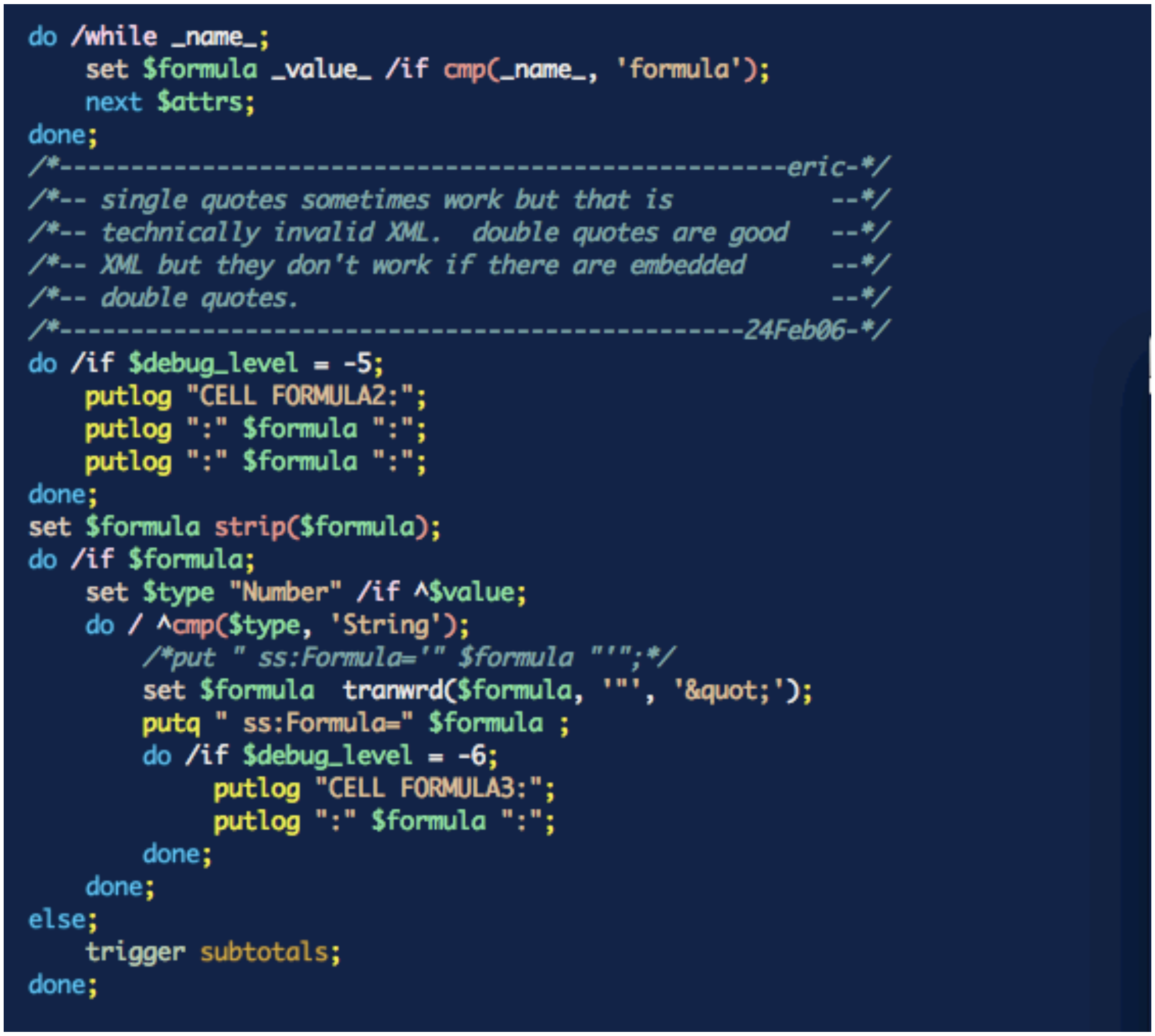
```
put '<NumberFormat';
putq ' ss:Format=' $format_override;
putq ' ss:Format=' $format /if ^$format_override;
put  ' />' NL;
```

MERGEACROSS has its own event to handle all the possibilities. The first section is for data notes, which only occur in the REPORT procedure. The second section is for normally spanning cells, which occur all the time in various places like titles and headers. The last section is the special case where the TAGATTR= MERGEACROSS option is taking control. If MERGEACROSS is `'yes'` then the current width of the sheet is used, but if it is a number, then the cell will merge across that many cells.



```
define event MergeAcross;
    do /if $in_data_note;
        eval $mergeAcross inputn($table_column_count, "3.")-1;
        putq ' ss:MergeAcross=' $mergeAcross;
        unset $mergeAcross;
        break;
    done;
    do /if colspan;
        eval $mergeAcross inputn(COLSPAN, "3.")-1;
        putq ' ss:MergeAcross=' $mergeAcross;
        unset $mergeAcross;

    else;

        unset $mergeacross;
        set $mergeacross strip($attrs['mergeacross']) /if $attrs;
        do /if $mergeacross;
            do /if cmp($mergeacross, 'yes');
                putq ' ss:MergeAcross=' $worksheet_widths;
            else;
                eval $mergeAcross inputn($mergeacross, "BEST.");
                do /if ^missing($mergeacross);
                    putq ' ss:MergeAcross=' $mergeAcross;
                done;
            done;
        done;
    done;
end;
```

Formulas are a little bit complicated as well. First `$formula` is set from `$attrs`. This is done in a strange way—by looping through the values of `$attr`. I'm not sure why it does it that way, but I'm not going to change it now. The next thing is that if there is a `$formula`, the type of the cell has to be a number if there is no value in the cell otherwise Excel does not like it. Then, as long as the type is not `'String'`, the formula gets processed for double-quotes and then printed.



```
do /while _name_;
    set $formula _value_ /if cmp(_name_, 'formula');
    next $attrs;
done;
/*-----------------------------------------------eric-*/
/*-- single quotes sometimes work but that is       --*/
/*-- technically invalid XML.  double quotes are good --*/
/*-- XML but they don't work if there are embedded   --*/
/*-- double quotes.                                  --*/
/*--------------------------------------------24Feb06-*/
do /if $debug_level = -5;
    putlog "CELL FORMULA2:";
    putlog ":" $formula ":";
    putlog ":" $formula ":";
done;
set $formula strip($formula);
do /if $formula;
    set $type "Number" /if ^$value;
    do / ^cmp($type, 'String');
        /*put " ss:Formula='" $formula "'";*/
        set $formula  tranwrd($formula, '"', '&quot;');
        putq " ss:Formula=" $formula ;
        do /if $debug_level = -6;
            putlog "CELL FORMULA3:";
            putlog ":" $formula ":";
        done;
    done;
else;
    trigger subtotals;
done;
```

Congratulations on reading this far! This is everything you never wanted to know about TAGATTR= and its use in the ExcelXP tagset. If all you take away from this is the capabilities of TAGATTR= and how to use them with the ExcelXP tagset, then you are doing well. You might have a little better idea of why TAGATTR= is used and how it interacts with the tagset, the XML it creates, and possibly even a little bit about how Excel works. Maybe you are ready to play with the tagset and add your own TAGATTR= options. Whatever you have taken away from this, you are it! Take this knowledge and run with it, play with it, and see how it might help you get better Excel reports from ODS.

CONTACT INFORMATION

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

Eric Gebhart
SAS Institute Inc.
SAS Campus Drive
Cary, NC, 27513
Work Phone: 919-632-1742
E-mail: Eric.Gebhart@sas.com
Web: http://support.sas.com/rnd/base/ods/odsmarkup/