

Generating Custom Excel Spreadsheets using ODS

Chevell Parker, SAS Institute, Cary, NC

ABSTRACT

This paper will demonstrate techniques on how to effectively generate files that can be read with Microsoft Excel using the Output Delivery System. This paper will further discuss a variety of methods that will allow customization of the every part of the Excel file from ODS. Some of the tips provided will work with Excel 97, 2000 and 2002. However, much of what is covered especially, the advanced techniques using XML and the special Microsoft Office style properties apply to Excel 2000 and 2002.

INTRODUCTION

As you will see, creating files with the Output Delivery System that can be read with Excel is very easy, however, some additional work may be required to customize the output as you like. Topics of discussion will include the following: Techniques for creating files with the ODS that can be read by Excel, General appearance issues and common task, Advanced techniques using XML and the ODS Markup destination to modify the Excel file, and Using Excel Macros with ODS.

GENERATING EXCEL FILES

There are several methods of generating files that can be read by Excel using the Output Delivery System. The methods discussed in this paper will be using the ODS HTML and CSV destinations to generate the Excel or spreadsheet files. Generic XML files can be read with Excel 2002 and can be generated with the XML engine on the LIBNAME statement.

When you specify a procedure or data step within the ODS HTML statement with the .XLS or .CSV extensions, Microsoft Excel is opened in the Results Viewer on the PC. Excel is not an ODS destination and the fact that the file is opened in Excel is not a product of ODS. Excel sees a file generated with the registered extension of .XLS, or .CSV and attempts to open this file within the registered program which is Excel on the PC.

```
ods html file="c:\temp.xls";
proc print data=sashelp.class;
run;
ods html close;
```

The new ODS CSV destination can also be used to create files that can be read by Microsoft Excel. The acronym CSV stands for Comma Separated Value. This new destination is experimental with Version 8.2 as part of the ODS Markup Language. The New CSV destination defaults can be changed by modifying the default tagset as we will see shortly. Excel has the ability to read CSV files, so specifying the ODS CSV destination with the extension .CSV will create a comma separated file that is opened in Excel by default. Also, the delimiter can be changed from a comma to any other delimiter by modifying the CSV tagset. Use the CSVALL destination to maintain the titles and footnotes and bylines.

```
ods csv file="c:\temp.csv";
proc print data=sashelp.class;
run;
ods csv close;
```

GENERAL APPEARANCE AND COMMON TASK

TITLES AND FOOTNOTES

Using the ODS HTML destination to create the .XLS or .CSV files will place the entire title or footnote in the first cell. The effect of this is that the first column will become the width of the title or footnote. This occurs because the ODS HTML destination uses the non-standard <Table> tag for the titles and footnotes and bylines which Excel does not expect for a header. The width of the title or footnote will extend as much as 4 cells before wrapping. To change this behavior, one of the HTML tagsets can be used. Most of the HTML tagsets use the header tags <h1> by default for titles, footnotes and bylines. This is the tag that Excel expects for its headers and footers.

The HTML tagsets shipped for 9.0 are HTML4, which is the default with the ODS HTML destination in 9.1, HTMLCSS, PHTML, CHTML and IMODE. The tagsets can be specified as a destination like the example below, or as a value of the TAGSET= option on the ODS MARKUP statement. The titles and footnotes can also be merged in Excel using the COLSPAN= attribute in the titles or footnotes to determine how many columns to span. In the first example below, the PHTML tagset is used to extend the titles beyond the first cell. The second example spans the titles over 4 columns in the table using the COLSPAN= HTML attribute.

```
ods phtml file='c:\temp.xls'
stylesheet="c:\temp.css";
proc print data=sashelp.class;
run;
ods phtml close;

ods html file="temp.xls";
title "<td align=center colspan=4><font
size=4><b>this is a test</b></font></td>";
proc print data=sashelp.class;
run;
ods html close;
```

STARTING OUTPUT IN ROW 1

HTML

Output generated with the ODS HTML destination begins in row 2 by default. This happens because of the non-breaking space character () in the anchor tag. The only way to get rid of this anchor tag in the HTML destination is to post process the HTML file. The HTML tagsets of the Markup destination can also be used to begin the output in row 1. The HTML tagsets of the ODS Markup destination do not have this non-breaking space character in the anchor tag. See the prior example for syntax.

CSV

The CSV destination generates output beginning in row 3 of the Excel file. This is the default of the ODS CSV destination. The defaults of the destination or tagset can be changed by modifying the tagset and overriding the defaults. The sample

code below modifies the CSV tagset and starts the data in row 1 by removing the first 2 empty rows.

```
proc template;
  define tagset tagsets.newcsv;
    parent = tagsets.csv;
    define event table;
      finish:
        put NL;
      end;
    define event row;
      finish:
        put NL;
      end ;
    end;
  end;
run;

ods tagsets.newcsv body='c:\test.csv' ;
proc print data=sashelp.class label; run;
ods tagsets.newcsv close;
```

REDUCING FILE SIZE

There are a few techniques that can be employed to reduce the size of Excel files and reduce the time it takes for the files to load. The first method involves creating a CSS style sheet with the ODS HTML destination. This allows you to separate the formatting instructions from the data and the need for each record to have formatting instructions. If you specify the STYLESHEET= option with a file, an external CSS file is generated. Excel 97 ignores this CSS style sheet.

The second method of reducing the size of the .XLS files created is to use one of the HTML tagsets of the ODS Markup destination. All of the HTML tagsets of the ODS Markup destination follow the HTML 4.0 standard which separates the formatting instructions from the data. All of the HTML tagsets except CHTML allow formatting with the use of a CSS style sheet. The CHTML tagset does not allow the use of a CSS file. These HTML tagsets all have minimal formatting such as the borders without the use of the CSS file.

The final method for reducing the size of the Excel file is to use the Minimal style. The Minimal style is one of the default styles shipped with SAS. The Minimal style has very few formatting instructions which reduces the size of the file. Referenced are the statistics of the 5 variable, 19 observation SASHELP.CLASS data set. This was done in Version 8.2. As the observations grew, PHTML became more efficient than its HTMLCSS counterpart. Not listed, the CSV destination was the smallest of all at 1K.

8.2 Benchmark

HTML	HTML/ CSS	PHTML HTMLCSS	CHTML	MINIMAL
21K	5k	5k	4k	5k

CELL FORMATING

One of the most problematic areas that you will face when creating Excel files from ODS is with cell formatting. The problems are the same whether using the CSV or the HTML destinations. The problem occurs because Excel uses a General format to import cell values. The General format reads the cell values as they are typed, however, there are some common problems that you should be aware of.

- Both numeric and character variables will lose leading

and trailing zeroes when creating .XLS or .CSV files with the ODS HTML and CSV destinations. You will not realize the problem until the leading or trailing zeroes are omitted from an account number, an ID, or a zip code.

- Numbers with lengths greater than 11 characters are displayed in scientific notation.
- Unformatted dates in SAS will be totally different in Excel because their beginning date starts with January 1, 1900 by default.

NUMBER FORMATS

Importing the cells as text using the Text format for the cell values allow the cell values to come over without any interpretation and does not strip the leading or trailing zeroes. Using the **mso-number-format:\@** style property allows the cell values to be imported using the Text format for Excel 2000 and above. For Excel 97, the style property is **vnd.ms-excel.numberformat:@**. Below are examples of applying the Text format and the more common number formats.

```
/* Apply text format to all cells */

data one;
  input acc_no zipcode;
  cards;
  0111 023560
  0333 023334
  ;
run;

ods html file='temp.xls' headtext='<style>
  td {mso-number-format:\@}</style>';
proc print data=one;
  run;
ods html close;

/* Text format applied to a single column */

ods html file='temp.xls' headtext='<style>
.zero {mso-number-format:@}</style>';
proc print data=one;
  var acct_no / style={htmlclass="zero"};
  var zipcode;
  run;
ods html close;

/* Excel 97 solution */
ods html file='temp.xls';
proc print data=one;
  var acct_no / style={htmlstyle="vnd.ms-
  excel.numberformat:@"};

  var zipcode;
  run;
ods html close;
```

COMMON NUMBER FORMATS

mso-number-format:0	NO Decimals
mso-number-format:"0,000"	3 Decimals
mso-number-format:"#,##0\,000"	Comma w/3 dec
mso-number-format:"mmVddVyy"	Date7
mso-number-format:"mmmm\ d\,\, yyyy"	Date9
mso-number-format:"mVdVyy\ h:mm\ AMVPM"	D -T AMPM
mso-number-format:"Medium Date"	01-mar-98
mso-number-format:"d\-mmm\-yyyy"	01-mar-1998

mso-number-format:"Short Time"	5:16
mso-number-format:"Medium Time"	5:16 am
mso-number-format:"Long Time"	5:16:21:00
mso-number-format:Percent;	Percent
mso-number-format:0%	No percent
mso-number-format:"\0.E+00";	Fractions
mso-number-format:"\@"	Text

CELL FORMATING IN THE CSV DESTINATION

To prevent losing the leading zeroes when using the CSV destination, an "=" can be added in front of the character strings. This allows the fields to be read using the text format. This solution also works with the HTML destination. The CSV tagset can also be modified to add the "=" before the data values. To modify a specific field, add the "=" in front of the data value within the data step.

```
proc template;
  define tagset Tagsets.test;
    parent=tagsets.csv;
    define event data;
      put "," / if !cmp( COLSTART , "1" );
      put '=' "" "" / if cmp( TYPE ,"string" );
      put VALUE;
      put "" "" / if cmp( TYPE , "string" );
    end;
  end;
run;

ods markup file="c:\temp.csv"
tagset=tagsets.test;
proc print data=one; run;
ods markup close;
```

ROW HEIGHT AND COLUMN WIDTH

When the row height and column width are set with a style in ODS, they are ignored by Microsoft Excel. A special MSO CSS style property has to be set before Excel will recognize the row height and column width set. If numbers have widths greater than the column width, the number will be displayed as #####. For character values, they will appear truncated if the cell to its right is not empty. When applying the height or the width, the special MSO style property **mso-height-source:userset** and the **mso-width-source:userset** have to be set before specifying a width or a height. The example below applies the width to a single column by defining the class with the HEADTEXT= ODS HTML option.

```
ods html file='temp.xls' headtext=
'<style> .test {mso-width-source:
userset;width:200pt}</style>';
proc print data=sashelp.class;
  var age / style(column)={htmlclass="test"};
  var sex height weight;
run;
ods html close;
```

BORDERS, ALIGNMENT AND PATTERNS

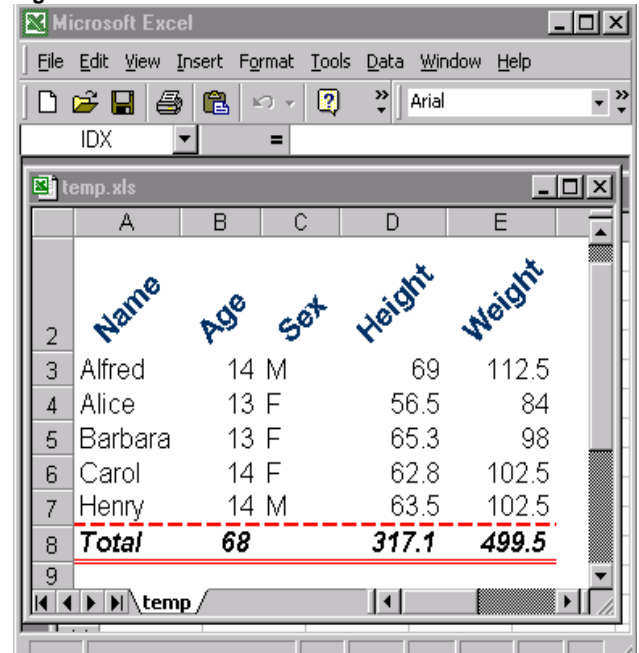
Generating customized borders can be done by using PROC TEMPLATE, procedures that support the STYLE= option, or with CSS style sheets. This section will demonstrate how to generate customized borders for a table. The first thing that is done is to turn off the borders at the table level so that the borders can be customized for individual cells. To do this, use the CSS style property Border. The Border style property has 3 separate values: weight, style, and color. The border style property can be used with the style attribute

HTMLSTYLE= to control the borders on an individual level.

The border style property will control the overall border, however, the **border-left**, **border-right**, **border-top** and **border-bottom** style properties control the various parts of the border. The **mso-pattern** style property can be used to specify the various patterns or the various shades of gray. The alignment is controlled with the JUST= attribute or the **text-align** style property. The text orientation can be modified by using the **layout-flow** style property which takes the value of vertical and horizontal and the **mso-rotate** style property which allows the rotation based on degrees. The **mso-text-control: shrinktofit** style attribute and value is used to force the value to fit in the cell by reducing the size. Other style attributes that affect how the text is rendered are the **white-space** style property with the values wrap, to wrap the text on the blank spaces and normal which is the default. The last style property that I will mention is the **text-indent**. This allows the indentation of the cell values. Below is an example that shows how this is done.

```
ods html file='temp.xls';
title;
proc report data=sashelp.class(obs=5) nowd
style(report)={rules=none }
style(column)={background=white
htmlstyle='border:none'}
style(header)={htmlstyle="mso-rotate:45;
height:50pt; border:none"
background=_undef_};
col name age sex height weight;
compute after;
  name="Total";
endcomp;
rbreak after / summarize
style={font_weight=bold htmlstyle="border-
bottom:5px double red;border-
left:none;border-right:none;border-
top:5px dashed red"};
run;
ods html close;
```

Figure 1. Customized Borders



&P	&N	&T	&D	&F	&B	&I
Page	# Pages	Time	Date	File	Bold	Italic

PAGE SETUP

Page setup options can be set with a combination of style properties and XML. In the page set up, we have the ability to modify all of the various items within the page set up such as the margins of the page, the margins of the header and footer, the page orientation, the DPI (data per inch) of the output, the paper size, the first page number and every other item. Many of these items can be set using the CSS style properties and the Microsoft Office specific style properties. The remainder can be set using XML.

MARGINS AND PAGE ORIENTATION

Margins can be set for the page to include the top, bottom, left and right margins. The margins can also be specified for the headers and footers and the justification of the page vertically and horizontally. Other items that can be specified such as the paper size, the page orientation, and headers and footers all can be set using the CSS @Page rule.

The margins for the page can be set using the style property **Margin**. The margins for the headers and footers can be specified using the Microsoft Office specific **mso-header-margin** and **mso-footer-margin** style properties. The alignment of the table horizontally and vertically on the page can be set using the **mso-horizontal-page-align** and the **mso-vertical-page-align** style properties. The paper size can be modified with the **size** style property with the appropriate paper size. This can also be set with XML which is shown in a later example. The page orientation can be modified with the **mso-page-orientation** style property, however, for Office 2000 at least, this has to be augmented with the XML <ValidPrinter> tag within the Print element .

```
ods html file='temp.xls' headtext=
'<style> @page{margin:1.0in .75in 1.0in
75in;
mso-header-margin:.5in;
mso-footer-margin:.5in;
mso-horizontal-page-align:center;
mso-vertical-page-align:center;
mso-page-number-start:1;}
</style>';
proc print data=sashelp.class;
run;
ods html close;
```

HEADERS AND FOOTERS

Headers and footers can also be defined within the @Page rule using the MSO style properties **mso-header-data** and **mso-footer-data**. This allows you to specify customized headers for the printed output. The headers and footers can be a generic page number, to the more sophisticated page X of Y, date time, a signature, very customized headers and footers with text on the left, right, top and bottom that include a variety of the fore-mentioned. The below example uses the Page X of Y header at the top of the page and some customized text at the left, center and right at the bottom of the page. &P is the current page number, &N is the total number of pages. In the footer &L, left justifies the text following and &C and &R center and right justify, respectively. The font name, style and size all can also be modified for the headers and footers as well. Also the CRLF character can be specified using the \000A to split text over multiple lines.

Header and Footer codes

```
ods htmlcss file='temp.xls'
stylesheet="temp.css" headtext=
'<style> @Page {mso-header-data:"Page &P of
&N"; mso-footer-data:"&Lleft text &Cpage
&P&R&D&T"};
</style>';
proc print data=sashelp.class;
run;
ods htmlcss close;
```

USING XML TO MODIFY EXCEL APPLICATIONS

XML can be used to modify Excel applications created with ODS. With the use of XML and the CSS style properties, virtually every part of the Excel file can be modified from ODS. The XML included is added between the <head> and </head> tags of the HTML file. The various XML elements control the different actions or options within Excel. A complete list of all of the XML elements and style properties that can be used to modify your Excel applications can be found at the URL located in the references at the end of the paper. The ODS MARKUP destination is used in the below examples to supply the XML. The reason the Markup destination was chosen was because of its flexibility. With the Markup destination, you have the ability to control the flow of the HTML generated. With the **doc** event, the Microsoft Office and the Excel namespace are added to the opening <HTML> tag. The XML is added to the event **doc_head** which is structured by adding new line characters at the end of each statement. Unlike the HEADTEXT= option which has a 256 character limit, adding the values to this event has no physical limit. The data step can also be used to append the header. We will just touch on the power that XML plays in modifying your Excel applications.

With the use of XML, we can perform such functions as generating multiple worksheets per workbook, naming worksheets within the workbook, activating and selecting cells, hiding worksheets, supply worksheet options, add formulas, name formulas, modify the resolution of the printed output, selecting the number of copies to print, scaling the printed output, generating backups, splitting windows, modify the window size, data validation, sorting, conditional formatting, set filters, supply and remove gridlines, protect cells, supply or remove scroll bars, generate charts, define macros, and so on. I will show some examples of using XML to modify your Excel applications.

The first example demonstrates generating multiple worksheets for a workbook. Within the Worksheet element, I have named 3 separate worksheets named Claims, Approved, and Paid by specifying the names in the Name tag. Within the WorksheetSource tag, the URL is specified for the sheet. In this example, the HTML files were generated in a prior step. This creates a workbook with the name temp and 3 worksheets: Claim, Approved, and Paid. The ActiveSheet tag is used to select the active worksheet.

```
proc template;
define tagset tagsets.test;
parent=tagsets.phtml;
define event doc;
start:
put '<html xmlns:o="urn:schemas-
microsoft-com:office:office" NL;
put 'xmlns:x="urn:schemas-
```

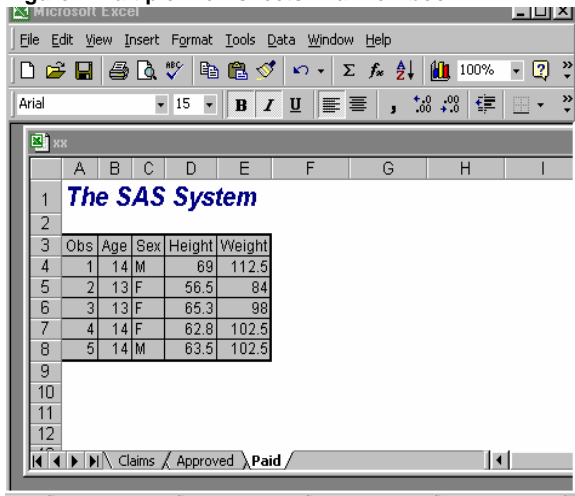
```

        microsoft-com:office:excel" ' NL;
    finish:
        put "</html>" NL;
    end;
define event doc_head;
start:
    put "<head>" NL;
    put '<meta name="Excel Workbook
        Frameset">';
finish:
    put "<!--[if gte mso 9]><xml>" NL;
    put "<x:ExcelWorkbook>" NL;
    put " <x:ExcelWorksheets>" NL;
    put " <x:ExcelWorksheet>" NL;
    put " <x:Name>Claims</x:Name>" NL;
    put " <x:WorksheetSource
        HRef='c:\T1.html' />" NL;
    put " </x:ExcelWorksheet>" NL;
    put " <x:ExcelWorksheet>" NL;
    put " <x:Name>Approved</x:Name>" NL;
    put " <x:WorksheetSource
        HRef='c:\T2.html' />" NL;
    put " </x:ExcelWorksheet>" NL;
    put " <x:ExcelWorksheet>" NL;
    put " <x:Name>Paid</x:Name>" NL;
    put " <x:WorksheetSource
        HRef='c:\T3.html' />" NL;
    put " </x:ExcelWorksheet>" NL;
    put " </x:ExcelWorksheets>" NL;
    put "<x:WindowHeight>5000
        </x:WindowHeight>" NL;
    put " <x:WindowWidth>10380
        </x:WindowWidth>" NL;
    put " <x:WindowTopX>480</x:WindowTopX>" NL;
    put " <x:WindowTopY>45</x:WindowTopY>" NL;
    put " <x:ActiveSheet>3</x:ActiveSheet>" NL;
    put "</x:ExcelWorkbook>" NL;
    put "</xml><![endif]-->" NL;
    put "</head>" NL;
end;
end;
run;

ods markup file="c:\temp.xls"
    tagset=tagsets.test;
data _null _;
    file print;
        put "testing";
    run;
ods markup close;

```

Figure2. Multiple Worksheets in a Workbook



The example below uses XML to validate data that is passed to Excel. For the cell B4, the value has to be a whole number and has to be less than 0. The Type and Qualifier tags within the DataValidation element determine this. When the field B4 is selected, the input title will be displayed along with the input message. If you attempt to change this value and the data is not validated, the error message is displayed along with the error title. In the below example, the cell B4 is selected automatically when the .XLS file is opened. This is done by adding the Activerow and Activecol tags within the WorksheetOptions element. Warning, the active row and active column is 1 less than it needs to be to select the cell correctly. We could have easily checked an entire range rather than a single cell using the RangeSelection tag with the appropriate ranges.

```

proc template;
define tagset tagsets.test;
    parent=tagsets.htmlcss;
    define event doc;
start:
    put '<html xmlns:o="urn:schemas-
        microsoft-com:office:office" ' NL;
    put '  xmlns:x="urn:schemas-microsoft-
        com:office:excel" ' NL;
finish:
    put "</html>" NL;
end;
define event doc_head;
start:
    put "<head>" NL;
    put VALUE NL;
    put "<style>" NL ;
    put "<!--" NL;
    trigger alignstyle;
    put "-->" NL;
    put "</style>" NL;
finish:
    put "<!--[if gte mso 9]><xml>" NL;
    put '<x:ExcelWorkbook>' NL;
    put ' <x:ExcelWorksheets>' NL;
    put ' <x:ExcelWorksheet>' NL;
    put ' <x:Name>testing1</x:Name>' NL;
    put ' <x:WorksheetOptions>' NL;
    put ' <x:Selected/>' NL;
    put ' <x:DoNotDisplayGridlines/>' NL;
    put ' <x:Panes>' NL;
    put ' <x:Pane>' NL;
    put ' <x:Number>3</x:Number>' NL;
    put ' <x:ActiveRow>3
        </x:ActiveRow>' NL;
    put ' <x:ActiveCol>1
        </x:ActiveCol>' NL;
    put ' </x:Pane>' NL;
    put ' </x:Panes>' NL;
    put ' </x:WorksheetOptions>' NL;
    put ' <x:DataValidation>' NL;
    put ' <x:Range>B4</x:Range>' NL;
    put ' <x:Type>Whole</x:Type>' NL;
    put ' <x:Qualifier>Less
        </x:Qualifier>' NL;
    put ' <x:Value>0</x:Value>' NL;
    put ' <x:InputTitle>Tip
        </x:InputTitle>' NL
    put ' <x:InputMessage>Verify number
        </x:InputMessage>' NL;
    put ' <x:ErrorMessage>incorrect
        number </x:ErrorMessage>' NL;
    put ' <x:ErrorTitle>stop
        </x:ErrorTitle>' NL;
    put ' </x:DataValidation>' NL;
    put ' </x:ExcelWorksheet>' NL;
    put ' </x:ExcelWorksheets>' NL;
    put '</x:ExcelWorkbook>' NL;
    put "</xml><![endif]-->" NL;
    put "</head>" NL;
end;

```

```

end;
run;

ods markup file="c:\test1.xls"
tagset=tagsets.test stylesheet='c:\temp.css';
proc print data=sashelp.class(obs=5);
var age sex height weight;
run;
ods markup close;

```

Figure 3. Data validation

Obs	Age	Sex	Height	Weight
1	14	M	69	112.5
2	56.5		84	
3	65.3		98	
4	14	F	62.8	102.5
5	14	M	63.5	102.5

The example below writes information to the Summary tab of the document properties. Values that can be supplied are the title, subject, author, manager, company, category, keywords, comments, and hyperlink base. The values can all be supplied with the below like named tags within the DocumentProperties element. The title will get its value from the <title> HTML tag if it's present, therefore we add the Title= ODS HTML sub-option to supply a value to this tag. Otherwise, the value defaults to "SAS Output". If the title tag were not present, then it would use the value specified within the Title XML tag. The hyperlink base specifies the defaults for all unqualified files. To populate this value use the <BASE > HTML tag with the HREF= attribute and the appropriate location of where Excel should look for these files.

```

proc template;
define tagset tagsets.test;
parent=tagsets.htmlcss;
define event doc;
start:
put '<html xmlns:o="urn:schemas-
microsoft-com:office:office"' NL;
finish:
put "</html>" NL;
end;
define event doc_head;
start:
put "<head>" NL;
put VALUE NL;
put "<style>" NL;
put "<!--" NL;
trigger alignstyle;
put "-->" NL;
put "</style>" NL;
finish:
put "<!--[if gte mso 9]><xml>" NL;
put "<o:DocumentProperties>" NL;
put "<o:Title>Sugi 28</o:Title>" NL;

```

```

put "<o:Author>B.Smith</o:Author>" NL;
put "<o:Subject>Demo</o:Subject>" NL;
put "<o:Company>SAS</o:Company>" NL;
put "<o:Manager>J.Bloe</o:Manager>" NL;
put "<o:Category>A</o:Category>" NL;
put "<o:Keywords>Test</o:Keywords>" NL;
put "<o:Description>Monthly Report
</o:Description>" NL;
put "</o:DocumentProperties>" NL;
put "</xml><![endif]-->" NL;
put "</head>" NL;
end;
end;
run;

ods markup
file="c:\temp.xls"(title="sugi28")
tagset=tagsets.test
stylesheet='c:\temp.css'
headtext='<base href="c:\sugi28">';
proc print data=sashelp.class;
run;
ods markup close;

```

Figure 4. Summary Tab of the Document Properties

This example is a continuation of the page set up options that can be specified within ODS. The example shows how to set the remaining options for a worksheet within page set up. Within the Print element, we specify that the output is printed in black and white, draft quality, legal paper size, scaled to 85%, gridlines are printed, row and column headers are printed, and that the horizontal resolution is 300 DPI. I don't think we would want this along with draft quality, but wanted to show that this can be used. The column headers on row 3 are repeated for each page. This is done by adding the value Print_Titles in the Name tag within the ExcelName element. The page orientation is also landscape because the **mso-page-orientation :landscape** style is specified in conjunction with the ValidPrintInfo XML tag of the Print element. To print only a specified area, the Print_Area value

can be added to the Name node within the ExcelName element. The PaperSizeIndex tag is used to control the paper size. This is specified within the Print element and can have the following values.

Paper Size values

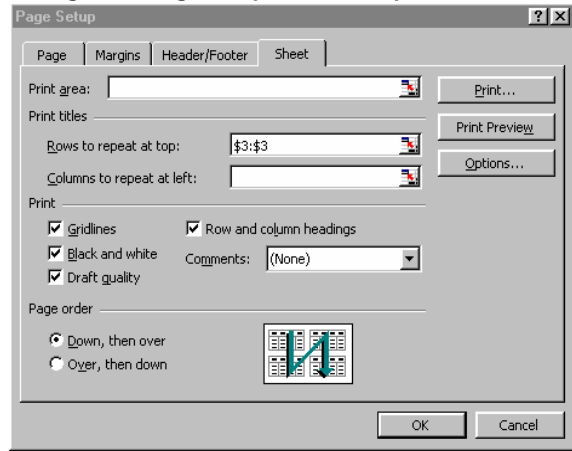
Legal	Executive	A4	A5	B5	No.10	A2.	DL	C6
5	7	9	11	13	15	17	19	21

```
proc template;
define tagset tagsets.test;
parent=tagsets.htmlcss;
define event doc;
start:
put '<html xmlns:o="urn:schemas-
microsoft-com:office:
office"' NL;
put 'xmlns:x="urn:schemas-microsoft-
com:office:excel">' NL;
finish:
put "</html>" NL;
end;
define event doc_head;
start:
put "<head>" NL;
put VALUE NL;
put "<style>" NL;
put "<!--" NL;
trigger alignstyle;
put "-->" NL;
put "</style>" NL;
finish:
put "<!--[if gte mso 9]><xml>" NL;
put "<x:ExcelWorkbook>" NL;
put "<x:ExcelWorksheets>" NL;
put " <x:ExcelWorksheet>" NL;
put " <x:Name>Sheet1</x:Name>" NL;
put " <x:WorksheetOptions>" NL;
put " <x:DisplayPageBreak/>" NL;
put " <x:Print>" NL;
put " <x:BlackAndWhite/>" NL;
put " <x:DraftQuality/>" NL;
put " <x:ValidPrinterInfo/>" NL;
put " <x:PaperSizeIndex>5
</x:PaperSizeIndex>" NL;
put " <x:Scale>85</x:Scale>" NL;
put " <x:HorizontalResolution>300
</x:HorizontalResolution>" NL;
put " <x:Gridlines/>" NL;
put " <x:RowColHeadings/>" NL;
put " </x:Print>" NL;
put " </x:WorksheetOptions>" NL;
put " </x:ExcelWorksheet>" NL;
put " </x:ExcelWorksheets>" NL;
put "</x:ExcelWorkbook>" NL;
put "<x:ExcelName>" NL;
put "<x:Name>Print_Titles</x:Name>" NL;
put "<x:SheetIndex>1
</x:SheetIndex>" NL;
put "<x:Formula>=Sheet1!$3:$3
</x:Formula>" NL;
put "</x:ExcelName>" NL;
put "</xml><![endif]-->" NL;
put "</head>" NL;
end;
end;
run;

ods markup file="c:\temp.xls"
tagset=tagsets.test
```

```
stylesheet='c:\temp.css'
headtext="<style> @page {mso-page-
orientation:landscape} </style>" ;
proc print data=sashelp.class;
title;
run;
ods markup close;
```

Figure 6. Page Setup and Sheet options



Window options can be specified for the Excel file using the WorksheetOptions element. The below example changes all of the window options. The gridlines are removed, zeroes are not displayed, the column headers are not displayed, and the outline is not specified. The Workbook element is responsible for removing the horizontal and vertical scroll bars, and hiding the workbook tabs.

```
proc template;
define tagset tagsets.test;
parent=tagsets.htmlcss;
define event doc;
start:
put '<html xmlns:o="urn:schemas-
microsoft-com:office:office"' NL;
put 'xmlns:x="urn:schemas-microsoft-
com:office:excel">' NL;
finish:
put "</html>" NL;
end;
define event doc_head;
start:
put "<head>" NL;
put VALUE NL;
put "<style>" NL ;
put "<!--" NL;
trigger alignstyle;
put "-->" NL;
put "</style>" NL;
finish:
put "<!--[if gte mso 9]><xml>" NL;
put "<x:ExcelWorkbook>" NL;
put "<x:ExcelWorksheets>" NL;
put " <x:ExcelWorksheet>" NL;
put " <x:Name>Sheet1</x:Name>" NL;
put " <x:WorksheetOptions>" NL;
put " <x:DisplayPageBreak/>" NL;
put " <x:Selected/>" NL;
```

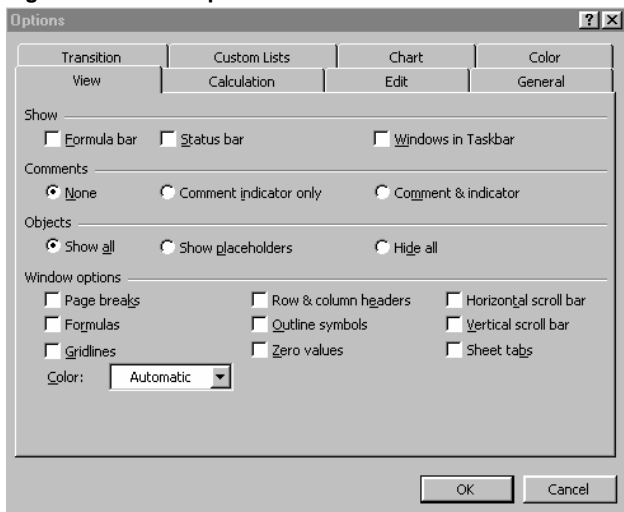
```

put "      <x:DoNotDisplayGridlines/>" NL;
put "      <x:DoNotDisplayZeros/>" NL;
put "      <x:DoNotDisplayHeadings/>" NL;
put "      <x:DoNotDisplayOutline/>" NL;
put "    </x:ExcelWorksheet>" NL;
put "  </x:ExcelWorksheets>" NL;
put " <x:HideHorizontalScrollBar/>" NL;
put " <x:HideVerticalScrollBar/>" NL;
put " <x:HideWorkbookTabs/>" NL;
put " <x:DisplayFormulas/>" NL;
put " </x:ExcelWorkbook>" NL;
put " </xml><![endif]-->" NL;
put " </head>" NL;
end;
end;
run;

ods markup file="c:\temp.xls"
tagset=tagsets.test
stylesheet='c:\temp.css';
proc print data=sashelp.class;
run;
ods markup close;

```

Figure 7. Window Options with View Tab



The Excel files can be sorted based on the field names in the output. The sort is done for the Excel output only. The data set is not sorted for this example. In order for Excel to treat the output as a database, the column headers are specified in row 1. The Sort element is specified within the ExcelWorksheet element. In the below example, the Name and Sex fields are in descending order with the Age field appearing in ascending order. A null title statement is specified so that the headers begin in row 1.

```

proc template;
define tagset tagsets.test;
parent=tagsets.htmlcss;
define event doc;
start;
put '<html xmlns:o="urn:schemas-
microsoft-com:office:office" NL;
put 'xmlns:x="urn:schemas-microsoft-

```

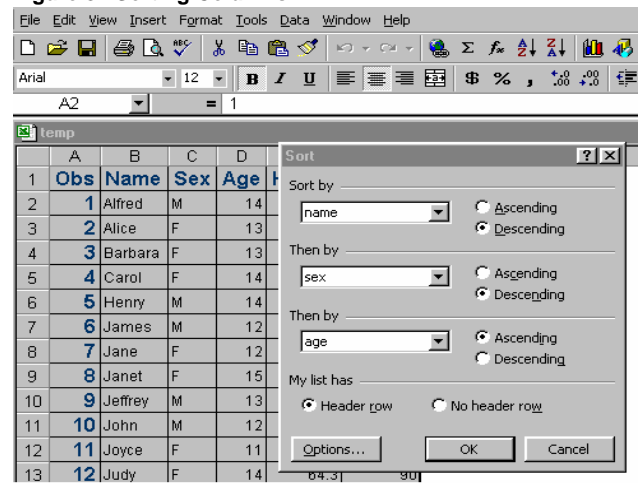
```

com:office:excel">' NL;
finish;
put "</html>" NL;
end;
define event doc_head;
start;
put "<head>" NL;
put VALUE NL;
put "<style>" NL;
put "<!--" NL;
trigger alignstyle;
put "-->" NL;
put "</style>" NL;
finish;
put '<!--[if gte mso 9]><xml>' NL;
put ' <x:ExcelWorkbook>' NL;
put ' <x:ExcelWorksheets>' NL;
put ' <x:ExcelWorksheet>' NL;
put ' <x:Name>Sheet1</x:Name>' NL;
put ' <x:Sorting>' NL;
put ' <x:Sort>name</x:Sort>' NL;
put ' <x:Descending/>' NL;
put ' <x:Sort>sex</x:Sort>' NL;
put ' <x:Descending/>' NL;
put ' <x:Sort>age</x:Sort>' NL;
put ' </x:Sorting>' NL;
put ' </x:ExcelWorksheet>' NL;
put ' </x:ExcelWorkbook>' NL;
put ' </xml><![endif]-->' NL;
put ' </head>' NL;
end;
end;
run;

ods markup file="c:\temp.xls"
tagset=tagsets.test stylesheet='c:\temp.css';
proc print data=sashelp.class;
run;
ods markup close;

```

Figure 8. Sorting Columns



USING EXCEL MACROS WITH ODS

A macro is a program that contains a list of instructions. Macros in Excel can be used to automate various tasks that are commonly used. Visual Basic for Applications (VBA) is the programming language used to drive macros with the

Microsoft Office products. VBA is now the standard programming language within Microsoft Office products as well as the ADOBE products. The use of macro in Excel is a very powerful and dynamic feature that I cannot cover sufficiently here, but will briefly discuss this and how to implement macro with files generated with ODS.

The type of macro that will be discussed in this section will be the command macro, or more commonly known as the "sub procedures" for obvious reasons. You might say that this is well, and good, but you are not interested in learning a new programming language. The best part is that you do not have to learn this programming language to develop great macro code. Excel allows you to cheat by turning on the macro recorder. This is done by going to **Tools->Macro->Recorder** and turning on the macro recorder. This will place a little icon on your worksheet. Until you turn the recorder off, it will record every action that is taken and translate this into VBA code. As you see, this reduces the need for you to be a real expert in the language. However, to modify these macros, you will need to know the basics of the language.

Unlike the old WordBasic or Excel 4.0 macro language, VBA 6.0 allows you to access almost every feature within the Excel application. Macros can be executed by defining a keystroke for the macro, going to: **Tools->Macro** and selecting Run, or run when the workbook is opened by naming the macro auto_open. There are various other ways to do this such as adding it to the tool bar, or as an add-in, but I will focus on the fore-mentioned three. Macros that are commonly used can also be placed in the personal.xls workbook, which is referred to as the "personal macro workbook". The macros located in this personal.xls file will be available to all workbooks opened. You can think of this as an autoexec file. This is done by placing the personal.xls workbook in the XLStart folder, which is located by default in C:\Program Files\Microsoft Office\Office\XLSTART. After the macros are stored in this personal.xls workbook, the workbook is hidden so that it is not displayed. This is done by going to: **Window-> Hide**. After saving this file, every workbook opened will have access to these macros.

Excel files generated with ODS will have access to all of the macros defined in the personal.xls workbook when the .XLS files are opened which causes very little overhead. What we can do with VBA is endless. I will only touch this subject and present a few examples to show how this can be used effectively from ODS.

SAMPLE SYNTAX

```

/* Displays user created form */
Sub myform()
    userform2.show
End Sub

/* Changes window options */
Sub options()
    With ActiveWindow
        .DisplayGridlines = False
        .DisplayHeadings = False
        .DisplayOutline = False
        .DisplayZeros = False
        .DisplayHorizontalScrollBar = False
        .DisplayVerticalScrollBar = False
        .DisplayWorkbookTabs = False
    End With
    With Application
        .DisplayFormulaBar = False
        .DisplayStatusBar = False
        .DisplayCommentIndicator = 0
    End With

```

End Sub

While the personal.xls file is hidden, the macros in this file cannot be edited without un-hiding the workbook. To edit the macros in the personal.xls file without un-hiding this workbook, the XML element ExcelName can be specified with the name of how we want to address the macros in current workbook. The Formula tag specifies how the macros are addressed. In the Formula tag, the personal.xls workbook is specified with the "!" preceding the name of the macro. Keystrokes can also be specified for the macro specifying the Keystroke tag. The macros in the personal.xls file can be run when a new workbook is opened without any intervention. When the name of the macro in the current workbook is named auto_open, the macro in the personal macro workbook that we point to is executed automatically when a new workbook is opened. The reserved macro name auto_close can be specified to execute macros when the current workbook is closed. The name auto_activate can be specified to run when the workbook is activated. We are not limited to running macros stored in the personal.xls file. We can point to macros located in any workbook as long as the location and the name of the workbook are fully qualified in the formula tag with quotes. Only the path and the name of the workbook are quoted.

Below is an example of running a macro when the workbook is opened using XML with the Markup destination to define a macro with the name auto_open. Because we use this reserved name for the macro, the workbook will attempt to execute this macro when the workbook is opened. The current workbook is pointing to a macro in the personal.xls file by the name myform. When this workbook is opened, the workbook will bring up a form which I defined in the personal.xls file as userform2. The form contains buttons that allow the selection of the various corporate styles.

```

proc template;
define tagset tagsets.test;
parent=tagsets.phtml;
define event doc;
start:
    put '<html xmlns:o="urn:schemas-
        microsoft-com:office: office"' NL;
    put 'xmlns:x="urn:schemas-microsoft-
        com:office:excel"' NL;
finish:
    put "</html>" NL;
end;
define event doc_head;
start:
    put "<head>" NL;
    put VALUE NL;
    put "<style>" NL;
    put "<!--" NL;
    trigger alignstyle;
    put "-->" NL;
    put "</style>" NL;
finish:
    put "<!--[if gte mso 9]><xml>" nL;
    put " <x:ExcelName>" NL;
    put " <x>Name>auto_open</x>Name>" NL;
    put " <x:Macro>Command</x:Macro>" NL;
    put " <x:Formula>=personal.xls!myform
        </x:Formula>" NL;
    put "</x:ExcelName>" NL;
    put "</xml><![endif]-->" NL;
    put "</head>" NL;
end;
end;
run;

```

```
ods markup file="c:\temp.xls"
tagset=tagsets.test;
proc print data=sashelp.class;
title;
run;
ods markup close;
```

CONTACT INFORMATION

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

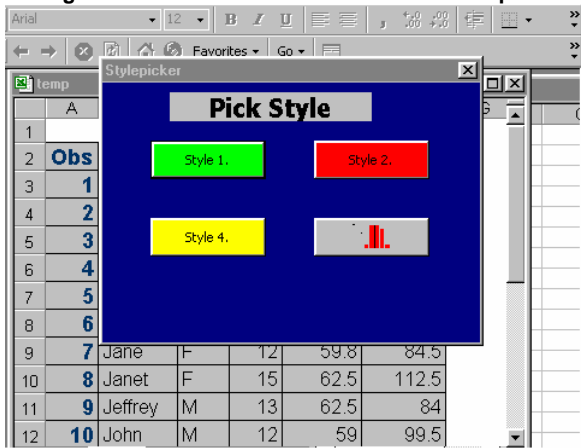
Chevell Parker
SAS
SAS Campus Drive
Cary, NC 27513

Email: Chevell.Parker@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.

Figure 9. Macro Executed when Workbook Opened



CONCLUSION

As you can see, generating files that can be read with Excel is very easy when using ODS. When you need more than what you are getting from the defaults, you can use some of the techniques mentioned in this document to fully customize your Excel output. Also mentioned were common issues that you should be aware of when generating Excel files from ODS.

REFERENCES

"Microsoft Office HTML and XML Reference"

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoffxml/html/ofxml2k.asp>

Parker, Chevell. "Tips for Creating Excel files with ODS".

<http://www.sas.com/rnd/base/topics/templateFAQ/Excel1.pdf>

"ODS FAQs" <http://www.sas.com/rnd/base/index-faq.html>

"Using ODS to Export Output in a Markup Language"

<http://www.sas.com/rnd/base/topics/odsmarkup/>