Paper 259-2008

Revisiting DDE: An Updated Macro for Exporting SAS® Data into Custom-Formatted Excel® Spreadsheets

Part I - Usage and Examples

Nathaniel Derby, Statis Pro Data Analytics, Seattle, WA

ABSTRACT

There are a number of ways to export data from SAS® into Microsoft® Excel®. However, very few allow for exporting into custom-formatted spreadsheets such as those demanded for specific reports, in which explicit attributes such as font sizes and column widths are required. Traditionally, Dynamic Data Exchange (DDE) was the only way to do this. Now there are newer methods for this which are considered superior, and DDE is thought to be obsolete. Part I ("Usage and Examples") argues that DDE is in fact far from obsolete, that it outperforms the newer methods in some cases that often arise in practice. Furthermore, Parts I and II offer an accessible, flexible and modifiable macro which exports data from SAS into custom-formatted Excel using DDE. The macro extends the functionality of a previously published macro and addresses some criticisms of DDE in general. Using this macro requires no knowledge of DDE programming and thus brings the full power of DDE to any PC SAS user. Part I ("Usage and Examples") describes the usage and provides examples, while Part II ("Programming Details") explains some details of the macro program itself. This macro should work on all versions of PC Base SAS, Windows, and Excel.

Keywords: DDE, Excel, Export, X4ML.

This paper is an excerpt[1] from Derby (2007), which may be updated and can be downloaded for more information.

INTRODUCTION

EXPORTING SAS DATA INTO EXCEL

Many SAS users recognize the need for exporting their SAS output into Microsoft Excel. Currently there are many ways to export SAS data into Excel: The Export Wizard; `PROC EXPORT`; `PROC DBLOAD`; ODS; ODBC; OLE DB; or by writing the data into a CSV, DAT, TXT or HTML file that can be read by Excel. However, these methods do not allow for importing SAS data into a *custom formatted* Excel worksheet. Figure 1 shows two examples of the pre-set formats of these methods. Any other kind of format, such as for a customized report, must be done manually from within Excel, which can be time-consuming. The user can cut and paste the data into a custom-formatted Excel template, but this is tedious if there are many tables to be exported.

Dynamic Data Exchange (DDE), as explained in Vyverman (2001) and Watts (2005), is a practical, effective and reliable solution to this problem. The defining characteristic of DDE is that SAS operates the interface of a PC application directly, using commands written in the application native language within a `DATA _NULL_` step. While DDE can be used with any program (e.g., Word (Vyverman, 2003) and PowerPoint (Vyverman, 2005)), most of its use is with Excel, with commands written in X4ML (the predecessor to VBA). Watts (2005) lists many of the advantages of DDE with Excel, and Vyverman (2000, 2001) provides an easily implemented macro to perform such a DDE export[2]. However, because it relies on the unsupported X4ML language, or because of the awkward nature of SAS driving Excel, some users[3] consider DDE obsolete and look for newer, alternative attempts to export SAS data into custom-formatted Excel worksheets.

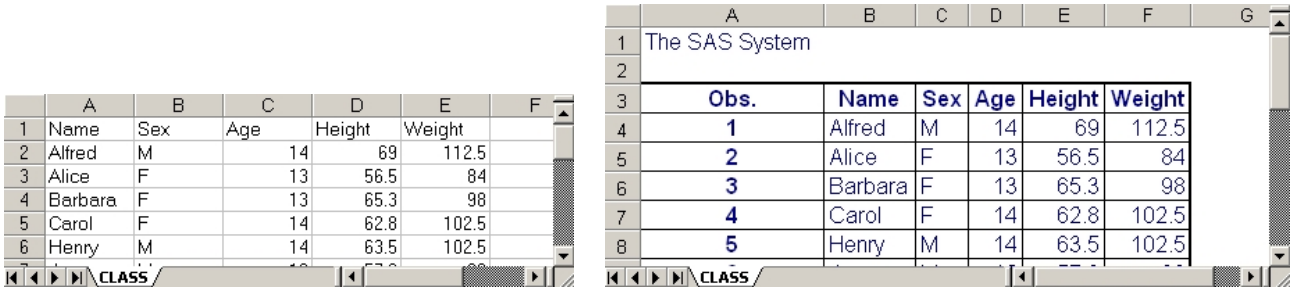
Attempts at alternative approaches include writing VBA code that will read SAS data sets and pour them into custom-formatted worksheets (Adlington (2005), Conway (2005)) or using XML and HTML code together to do this (Parker, 2003). However, as of this publication (1/29/08), no one has yet developed a simple method that implements either approach[4]. For users with access to the cost-prohibitive Enterprise Guide®, a *stored process* (Fecht and Bennett, 2006) can be used to generate custom-formatted Excel worksheets via the SAS Web Report Studio®, but because Excel doesn't inherently read from HTML and CSS files, problems can arise.

[1] Reprinted with permission of the author.

[2] Additionally, Poppe (2001) provides an ODS tagset that performs a DDE export into Excel, but it appears nonfunctional and/or underdeveloped. Beal (2004) provides many instrumental macros for DDE exports, but does not provide a stand-alone macro like Vyverman's.

[3] e.g., various coworkers of the author and SAS staff, as expressed in conversations with the author. Foster (2005) does not call it obsolete per se, but deplores the Excel interface and rejects it in favor of something with "a more seemless feel to the application". Gebhart (2007a) simply states "DDE is still a pain."

[4] Brown (2005) writes about one but does not distribute it, so it isn't very helpful. Furthermore, it appears to do less than the DDE solution presented in this paper, it appears more complicated to modify than our DDE solution, and it requires Excel version 2002 or later.



	A	B	C	D	E
1	Name	Sex	Age	Height	Weight
2	Alfred	M	14	69	112.5
3	Alice	F	13	56.5	84
4	Barbara	F	13	65.3	98
5	Carol	F	14	62.8	102.5
6	Henry	M	14	63.5	102.5

CLASS

	A	B	C	D	E	F	G
1	The SAS System						
2							
3	Obs.	Name	Sex	Age	Height	Weight	
4	1	Alfred	M	14	69	112.5	
5	2	Alice	F	13	56.5	84	
6	3	Barbara	F	13	65.3	98	
7	4	Carol	F	14	62.8	102.5	
8	5	Henry	M	14	63.5	102.5	

CLASS

Figure 1: The first five observations of the SAS data set `sashelp.class` exported into Excel via `PROC EXPORT` (left) and `ODS HTML` opened in Excel (right). Note their different formats, which can only be changed manually from within Excel.

Generally, the only reliable alternative to DDE is the *ExcelXP tagset* of ODS MARKUP (DelGobbo, 2006, 2007; Gebhart, 2005, 2006, 2007a,b). This is a well-designed routine for exporting into Excel. It can make custom formats with many options, it is quicker than DDE, and it works without having to open Excel. However, there are still limits to what ExcelXP can do, such as making graphics, creating side-by-side tables, using pre-formatted Excel worksheets, or various other issues (e.g., see Watts (2004, p. 5)) – not to mention that the newest version only works with newer versions of SAS (9.1.3+) and Excel (2002+). Some of these limitations will disappear with further development, but there will always be situations where ExcelXP works with difficulty or not at all. In these cases, *DDE is preferable and may be the only method that works*.

A NEW LOOK AT DDE

DDE is far from obsolete – as long as the current version of Excel still accepts X4ML commands[5], DDE is very much still applicable and useful. Further advantages of DDE as detailed by Watts (2005) still hold now and bear summarizing here:

- Only Base SAS and Excel are needed (of any version).
- In contrast to VBA or XML, X4ML is *modular*: Each function completes a specific task, making it easy to use. These functions (embedded within the SAS code) can be grouped into user-friendly macros for further ease of use.
- There are vast amounts of accessible DDE resources – user groups, papers, macros, etc.

Vyverman (2000) presents a single, highly flexible macro, `%sastoxl`, which exports data from SAS to Excel using DDE. This macro, which was updated one year later[6], makes the power of DDE fully available to all PC SAS users, since knowledge of DDE is not required to use it. However, some weaknesses are evident in practice:

- The code is very long and can be confusing, making it difficult to modify.
- It can't be used with Excel in a language other than English.
- The method used to open SAS is (much) slower than the method by Roper (2000) and used in Watts (2005).
- It lacks the following useful options: Merging cells, excluding headers, adding Excel formulas, keeping Excel open after exporting, using one worksheet as a template for others, and pre-formatting the numeric cells in Excel (avoiding possible data corruption as described in Adlington (2005)).
- Other minor issues (e.g., doesn't allow an Excel file name with a period in it, as in `07.01-Output.xls`).

Vyverman (2002) offers DDE code to solve a few of these problems, but he never offers a version of `%sastoxl` updated to implement these solutions. Here we present a macro, `%exportToXL`, that updates and enhances `%sastoxl`, overcoming the limitations listed above, adding some new features, and utilizing some ideas from Vyverman (2002) and Watts (2004, 2005). A summary of comparisons to ExcelXP is shown in Table 1. Generally, `%exportToXL` is preferable to ExcelXP for the user using PC SAS and Excel who does any of the following:

- Uses SAS version $<$ 9.1.3 or Excel version $<$ 2002.
- Wants to export onto a pre-formatted worksheet.
- Wants to include graphical output.
- Wants to create side-by-side tables.
- Can't figure out how to do it using ExcelXP.

Overall, `%exportToXL` makes it very easy to get worksheets formatted a specific way, either onto a pre-formatted worksheet (the easiest way) or through X4ML commands[7].

Both of these situations will be illustrated with a few examples.

[5] Excel's backward compatibility should guarantee this. It works with Excel 2007, so we're fine at least until the next version.
[6] This update is available at `http://www.sas-consultant.com/professional/sastoxl-for-SUGI26.sas`.
[7] via the `%format_&wsformat` submacro – see Part II of this paper for details.

	ExcelXP	%exportToXL
Works with any version of PC SAS or Excel	No[a]	Yes
Can make side-by-side tables (example 3)	No	Yes
Can export onto a pre-formatted worksheet (examples 4 and 5)	No	Yes
Can do almost anything to the worksheet[b]	No[c]	Yes[d]
Works with graphical output within Excel (examples 4 and 5)	No	Depends[e]
Works quickly	Yes	Depends[f]
Works without opening Excel	Yes	No
Works on any platform	Yes	No – only PC SAS on Windows
Works with OpenOffice.org Calc	Yes	No – maybe in a later version
Code can be modified to export onto something else (RTF, HTML)	Yes[g]	No

[a] Requires SAS 9.1+ and Excel 2002+, and the updated version of ExcelXP requires SAS 9.1.3.
[b] i.e., make Excel able to do anything that is possible manually.
[c] e.g., examples 3, 4 and 5 cannot be done with ExcelXP.
[d] Almost anything manually done in Excel can also be done via an X4ML command, or via the `send.keys` X4ML command. These commands can be written to a `format_xxx` submacro – see Part II of this paper for details.
[e] Can't (yet) create a graph, but can feed data into a graph that's already been made within a template.
[f] Not considerably slower unless exporting many data sets.
[g] Can be done by choosing a different ODS destination.

Table 1: A basic comparison of the ExcelXP tagset (v1.70, from 6/5/07) and `%exportToXL` – with some references to examples in the **EXAMPLES** subsection. This list is not meant to be exhaustive – there will probably always be situations where one method works better than the other.

USAGE

INSTALLATION

The macro introduced here[8], `%exportToXL`, is actually a group of smaller macros, as explained in Part II of this paper. To use it within a SAS program, put these macros (all found in the *exportToXL* directory of the unzipped file) into a central directory – e.g., `c:\SAS\exportToXL`. If the Excel application is in a language other than English, open the file `exportToXL.sas` and change the language by changing the default input `lang=en` to `lang=xx`, where `xx` is the appropriate language code[9]. Then add the following[10] to the SAS program in question (usually at the heading) before calling `%exportToXL`:

```
%LET exroot = c:\SAS\exportToXL;

OPTIONS SASAUTOS=( "&exroot" ) MAUTOSOURCE MCOMPILENOTE=all NOTES SOURCE SOURCE2;
```

This tells SAS to look at the contents of the `&exroot` directory to find new macro definitions (in particular, `%exportToXL` and its component macros). We could avoid the `%let` statement altogether and just write `sasautos=( 'c:\SAS\exportToXL' )` in the `options` statement, but the above solution is preferable for program portability. For each example in this paper, we define an output root for the exported Excel files:

```
%LET outroot = c:\SAS\Example xx;
```

PARAMETERS

`%exportToXL` is based on the `%sastoxl` macro by Vyverman (2000) and has a similar structure. Here we explain its usage and provide examples – a discussion of the code itself can be found in Part II of this paper. Some of the parameters might best be understood via the examples that follow.

```
%exportToXL( libin = work,          tmplsheet = ,                    statvars = ,
             dsin = ,               deletetmplsheet = no,            weightvar = ,
             cell1row = 1,          savepath = c:\temp,              mergeacross = 1
             cell1col = 1,          savename = exportToXL Output,    mergedown = 1,
             nrows = ,              sheet = ,                        exporttmplifempty = no,
             ncols = ,              wsformat = default,              exportheaders = yes,
             tmplpath = ,           lang = en,                       exportvarfmts = yes,
             tmplname = ,           sumvars = ,                      endclose = yes          )
```

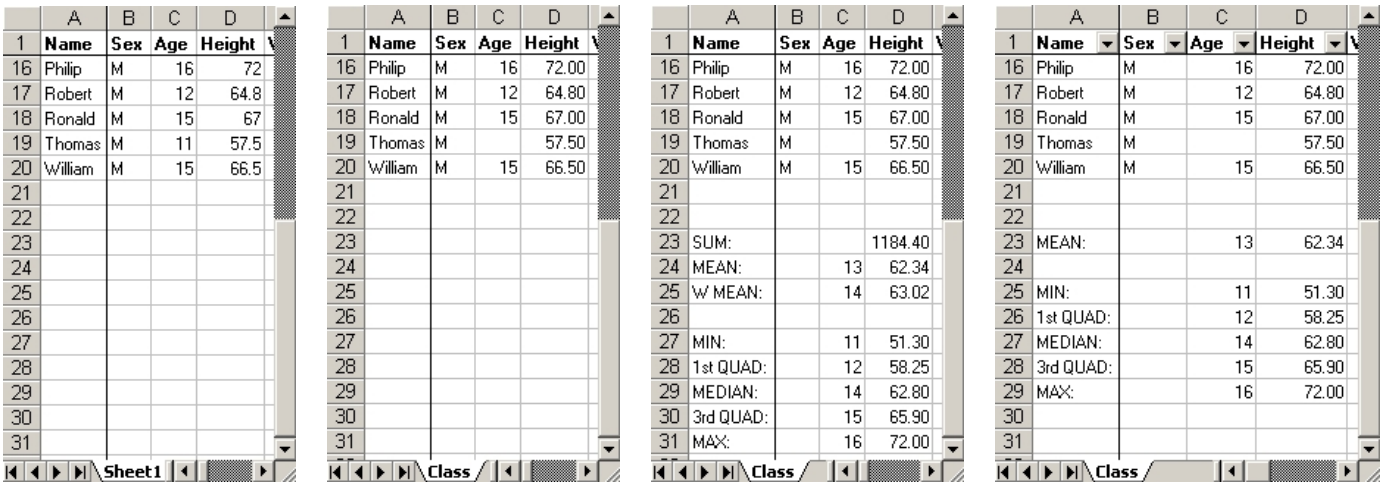
[8] This macro, the code for the following examples, and this user's guide can all be downloaded from `http://exporttoxl.sourceforge.net`.
[9] `da` for Danish, `de` for German, `es` for Spanish, `fi` for Finnish, `fr` for French, `it` for Italian, `nl` for Dutch, `no` for Norwegian, `pt` for Portuguese, `ru` for Russian, or `sv` for Swedish. **`%exportToXL` will not work if this parameter is set incorrectly**.
[10] For the reader who is new to SAS: The `%let` statement defines a *macro variable* (named *exroot*) which stores a collection of characters (*c:\SAS\Export*). To recall the macro variable later in the code, add an ampersand (&) to the name, as is used above, with `&exroot`. For more details, see Delwiche and Slaughter (2003, p. 200-213). Also, an alternative to this approach would be to save the `%exportToXL` macros in a data library and access them via a `libname` statement.

- `libin` (optional): The name of the SAS library where the input data set is located. Default value: *work*.
- `dsin` (required): The name of the input SAS data set.
- `cell1row`/`cell1col` (optional): The row/column number of the first cell of the worksheet where the data should be inserted – i.e., the upper left corner of the data block. Default value for each: *1*.
- `nrows`/`ncols` (optional): The first *n* rows/columns of the input data set that will be inserted into the worksheet. If no value is specified, an attempt will be made to insert all rows and columns (which must be smaller than the maximal number of rows/columns allowed by Excel – e.g., 256/65536 for Excel 2000 or 2003).
- `tmplpath`/`tmplname` (optional): The full directory path (`tmplpath`) and name (`tmplname`) of the Excel workbook into which the data will be written. If either value is omitted or the directory path does not exist, a standard new workbook will be created with one standard worksheet, named the value of `sheet`.
- `tmplsheet` (optional): The name of the specific worksheet within `tmplname` into which the data will be written. This is used only if `tmplpath` and `tmplname` are both given and exist. This is useful if one worksheet is used as a template for several worksheets within the same workbook. If a value is not given or does not exist, a standard new worksheet will be used.
- `deletetmplsheet` (optional): Indicates whether `tmplsheet` should be deleted from the workbook after it is used (i.e., copied and renamed to the value of `sheet`). Default value: *no*.
- `savepath`/`savename` (optional): The full directory path (`savepath`) and filename (`savename`) where the finalized Excel workbook needs to be saved (to be used independently of each other). Default values: *c:\temp* and *exportToXL Output*.
- `sheet` (optional): This has two meanings, depending on whether `tmplpath`, `tmplname` and `tmplsheet` are all given and exist:
  - If all are given and exist, then `tmplsheet` is renamed as `sheet` and placed at the end of the workbook (i.e., it becomes the last worksheet). If there is already a worksheet named `sheet` within `tmplname`, it is deleted and replaced.
  - Otherwise, `sheet` is the name of the specific worksheet within the Excel workbook (`tmplname` or a standard workbook) into which the data will be written. If there is already a worksheet named `sheet` within `tmplname`, it is *not* deleted, but is partially overwritten[11]. Otherwise, a standard worksheet is added to the end of the workbook.

  Default value = *SheetN*, where *Sheet* is in the language of the Excel application and *N* is the lowest positive integer not already used for a name of a worksheet in the given workbook (e.g., *Sheet4* if *Sheet1* - *Sheet3* already exist).
- `wsformat` (optional): The *worksheet format*, indicating how we want to custom-format the worksheet (fonts, margins, etc). A number of worksheet formats are available, which can be added to. Default value: *default* (Sans serif 8.5pt font, header in bold, best fit column width, 12pt row height, panes frozen), although *none* is also possible. For more details, see Derby (2007).
- `lang` (optional): The language of the Excel application[12]: *en* (English), *da* (Danish), *de* (German), *es* (Spanish), *fi* (Finnish), *fr* (French), *it* (Italian), *nl* (Dutch), *no* (Norwegian), *pt* (Portuguese), *ru* (Russian) or *sv* (Swedish). **The macro will not work if this parameter is set incorrectly.** Other languages are possible – see Derby (2007). Default value: *en* (English) – although it should be changed appropriately as mentioned in the **INSTALLATION** subsection.
- `sumvars`/`statvars` (optional): A list of variables in the data set (separated by spaces) for which a sum (for `sumvars`) or summary statistics (for `statvars`) should be listed at the bottom of the worksheet. Default value for each: (none).
- `weightvar`: The variable used as the weight variable for the weighted average in the summary statistics – ignored if `statvars` is empty. If no variable is given or exists, no weighted average is given. Default value: (none).
- `mergeacross`/`mergedown` (optional): Indicates how many cells should be merged across (`mergeacross`) or down (`mergedown`) over the range of the data. Default value for each: *1*.
- `exporttmplifempty`: Indicates whether `tmplsheet` should be exported (i.e., copied and renamed to the value of `sheet`) if `dsin` is empty – ignored unless `tmplpath`, `tmplname` and `tmplsheet` are all given and exist. Figure 6(a) is an example of this. Default value: *no*.
- `exportheaders`: Indicates whether the headers of the data set should be exported. Default value: *yes*.
- `exportvarfmts`: Indicates whether the variable formats (e.g., `w.d`, `percentw.d`) should be exported. Useful if `tmplsheet` or `sheet` is pre-formatted. Default value: *yes*.
- `endclose`: Indicates whether Excel should be closed after exporting the data and closing the file. This is useful for saving time when the macro is used repeatedly. Default value: *yes*.

[11] If `tmplpath`, `tmplname` and a sheet named *xx* are all given and exist, `sheet=xx` is equivalent to `tmplsheet=xx, sheet=xx, deletetmplsheet=yes`, except that in the latter case the worksheet is moved to the end of the workbook.

[12] `%exportToXL` has only been tested in English and German at this time (1/29/08) – there may be problems with the other languages. See Derby (2007) or the project website listed under **CONTACT INFORMATION** for updates and details.



	Name	Sex	Age	Height
16	Philip	M	16	72
17	Robert	M	12	64.8
18	Ronald	M	15	67
19	Thomas	M	11	57.5
20	William	M	15	66.5

Sheet1

(a)

	Name	Sex	Age	Height
16	Philip	M	16	72.00
17	Robert	M	12	64.80
18	Ronald	M	15	67.00
19	Thomas	M		57.50
20	William	M	15	66.50

Class

(b)

	Name	Sex	Age	Height
16	Philip	M	16	72.00
17	Robert	M	12	64.80
18	Ronald	M	15	67.00
19	Thomas	M		57.50
20	William	M	15	66.50
23	SUM:			1184.40
24	MEAN:		13	62.34
25	W MEAN:		14	63.02
27	MIN:		11	51.30
28	1st QUAD:		12	58.25
29	MEDIAN:		14	62.80
30	3rd QUAD:		15	65.90
31	MAX:		16	72.00

Class

(c)

	Name	Sex	Age	Height
16	Philip	M	16	72.00
17	Robert	M	12	64.80
18	Ronald	M	15	67.00
19	Thomas	M		57.50
20	William	M	15	66.50
23	MEAN:		13	62.34
25	MIN:		11	51.30
26	1st QUAD:		12	58.25
27	MEDIAN:		14	62.80
28	3rd QUAD:		15	65.90
29	MAX:		16	72.00

Class

(d)

Figure 2: The first four variables and last five observations of the SAS data set `sashelp.class` (or `work.class`) exported into Excel via `%exportToXL` with different options, as explained in Example 1.

EXAMPLES

In deference to Watts (2005) and Gebhart (2005, 2006, 2007a,b), the examples are based on the `sashelp.class` data set.

Example 1: Exporting onto a Basic Worksheet

Let's begin with a basic export of `sashelp.class`:

```
%exportToXL( libin=sashelp, dsin=class, savepath=&outroot, savename=Example 1-1 );
```

Here, `libin` and `dsin` define the library and name, respectively, of the SAS data set we wish to export, and `savepath` and `savename` define the directory and name of the outputted Excel file. We see in the output in Figure 2(a) that the worksheet has the default name (*Sheet1*) and that the variable *height* is unformatted – sometimes with a decimal place, sometimes without. This is because the variables of `sashelp.class` are unformatted, which we rectify via a simple `data` statement (not shown) to create `work.class`, where we also set Thomas' age as unknown, for illustrative purposes. We export this, now naming the worksheet as *Class*. These two commands are identical, since *work* is the default value of `libin`:

```
%exportToXL( libin=work, dsin=class, savepath=&outroot, savename=Example 1-2, sheet=Class );
%exportToXL( dsin=class, savepath=&outroot, savename=Example 1-2, sheet=Class );
```

In the output, in Figure 2(b), we now have all *height* values with two decimal places, as well as a new name on our worksheet. Note also that Thomas' missing age is denoted with a blank space rather than the SAS notation of a period. If we want to export only the first 10 rows and 3 columns of this data set, we can write the following (output not shown):

```
%exportToXL( dsin=class, savepath=&outroot, savename=Example 1-3, sheet=Class, nrows=10, ncols=3 );
```

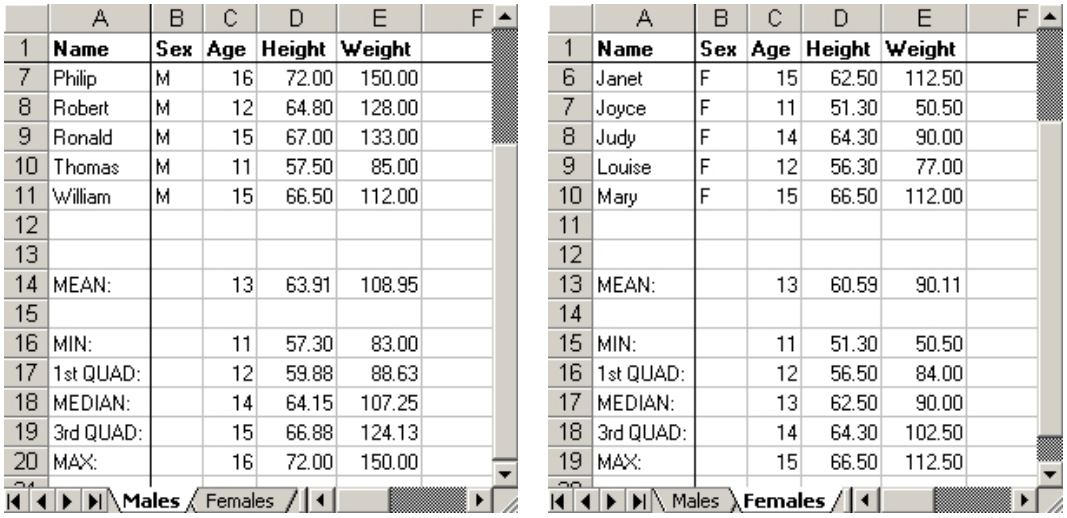
Suppose we would like some summary statistics on *age*, *height* and *weight*, with *age* being the weight variable for the weighted average. Furthermore, suppose we would also like a sum for *height*:

```
%exportToXL( dsin=class, savepath=&outroot, savename=Example 1-6, sheet=Class,
  statvars=age height weight, weightvar=age, sumvars=height );
```

This gives us Figure 2(c), where we can see the summary statistics at the bottom. We might like to add an auto filter: We do this via the `wsformat=default_afilter`, which means "the default, with an autofilter":

```
%exportToXL( dsin=class, savepath=&outroot, savename=Example 1-7, sheet=Class,
  statvars=Age Height Weight, wsformat=default_afilter );
```

In the output in Figure 2(d), we note that we no longer have the sum or weighted average statistics, since we included no `sumstat` or `weightvar` options.



	A	B	C	D	E	F
1	**Name**	**Sex**	**Age**	**Height**	**Weight**	
7	Philip	M	16	72.00	150.00	
8	Robert	M	12	64.80	128.00	
9	Ronald	M	15	67.00	133.00	
10	Thomas	M	11	57.50	85.00	
11	William	M	15	66.50	112.00	
12						
13						
14	MEAN:		13	63.91	108.95	
15						
16	MIN:		11	57.30	83.00	
17	1st QUAD:		12	59.88	88.63	
18	MEDIAN:		14	64.15	107.25	
19	3rd QUAD:		15	66.88	124.13	
20	MAX:		16	72.00	150.00	

Males Females

(a)

	A	B	C	D	E	F
1	**Name**	**Sex**	**Age**	**Height**	**Weight**	
6	Janet	F	15	62.50	112.50	
7	Joyce	F	11	51.30	50.50	
8	Judy	F	14	64.30	90.00	
9	Louise	F	12	56.30	77.00	
10	Mary	F	15	66.50	112.00	
11						
12						
13	MEAN:		13	60.59	90.11	
14						
15	MIN:		11	51.30	50.50	
16	1st QUAD:		12	56.50	84.00	
17	MEDIAN:		13	62.50	90.00	
18	3rd QUAD:		14	64.30	102.50	
19	MAX:		15	66.50	112.50	

Males Females

(b)

Figure 3: The last five observations of two worksheets within the same workbook as explained in Example 2.

Example 2: Exporting onto Different Worksheets of One Workbook

Now let's split up `sashelp.class` into data sets of males (`males`) and females (`females`), with their variables formatted as in Example 1 (code not shown here). We would like to output the males and females onto separate worksheets of the same workbook. This is done in two steps – indeed, we need to call `%exportToXL` once for each data set exported. First we export `males` as in Example 1, requesting summary statistics for *age*, *height* and *weight*. This time, though, we also set `endclose=no` to not close Excel after the export is complete – to save a little run time, since we'll need it again for exporting `females`:

```
%exportToXL( dsin=males, savepath=&outroot, savename=Example 2, sheet=Males, statvars=Age Height Weight,
  endclose=no );
```

Now we will do the same for `females`, except that we want it to be exported into the workbook we just created in the first step. We do this by setting `tmplpath` and `tmplname` (which we can think of as *template path* and *template name*) equal to the values we gave `savepath` and `savename` in the last step. This time we do not include `endclose=no`, since this time we want Excel to close at the end (since we're only exporting two data sets). In this step we give `savepath` and `savename` the same values as before, getting the outcome shown in Figure 3(a) and (b):

```
%exportToXL( dsin=females, tmplpath=&outroot, tmplname=Example 2, savepath=&outroot, savename=Example 2,
  sheet=Females, statvars=Age Height Weight );
```

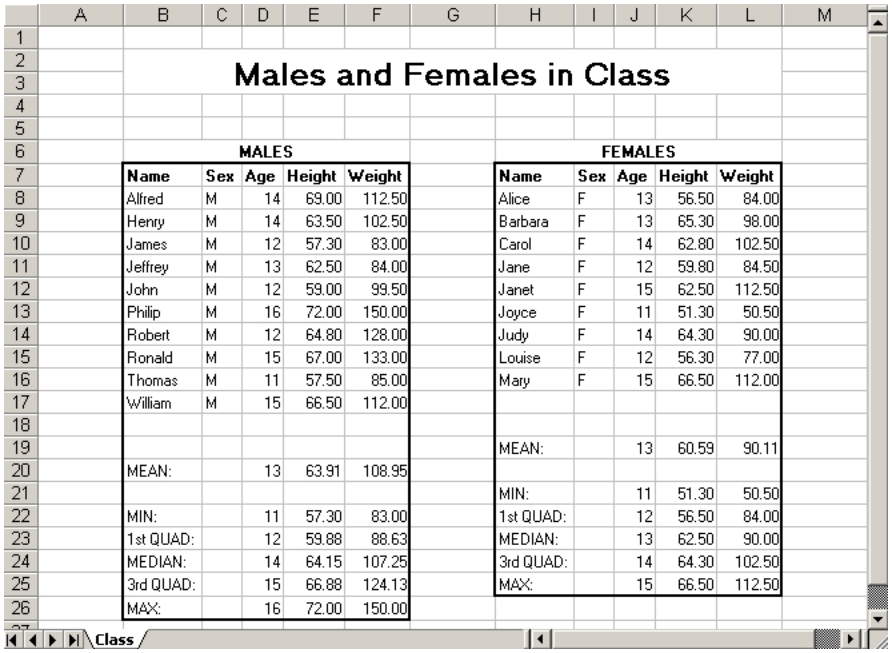
Example 3: Exporting onto Different Regions of One Worksheet

Now suppose we want to take the two data sets made in the previous example – `males` and `females` – and export them onto side-by-side regions of the same worksheet, along with a few titles. Up to now, we have not done anything that the ExcelXP tagset (explained in the introduction) cannot do. However, ExcelXP (v1.70, from 6/5/07) does not allow for side-by-side tables, as in Figure 4. `%exportToXL` does this easily – as well as any other nonstandard configuration.

First, we need to make a data set for each title we would like to export, as shown in Figure 4. We would like a title over each data set, as well as an overall title (in a larger font) over both data sets. We can do this by making a very simple data set for each title – with one variable formatted as text and one observation (the title itself). For the main title, we make the data set `title`:

```
DATA title;
    FORMAT text $30.;
    text = 'Males and Females in Class';
RUN;
```

Similarly, we make the data set `mheader` and `fheader` for the headers for the data sets `males` and `females`, respectively.



Males and Females in Class

MALES

Name	Sex	Age	Height	Weight
Alfred	M	14	69.00	112.50
Henry	M	14	63.50	102.50
James	M	12	57.30	83.00
Jeffrey	M	13	62.50	84.00
John	M	12	59.00	99.50
Philip	M	16	72.00	150.00
Robert	M	12	64.80	128.00
Ronald	M	15	67.00	133.00
Thomas	M	11	57.50	85.00
William	M	15	66.50	112.00
MEAN:		13	63.91	108.95
MIN:		11	57.30	83.00
1st QUAD:		12	59.88	88.63
MEDIAN:		14	64.15	107.25
3rd QUAD:		15	66.88	124.13
MAX:		16	72.00	150.00

FEMALES

Name	Sex	Age	Height	Weight
Alice	F	13	56.50	84.00
Barbara	F	13	65.30	98.00
Carol	F	14	62.80	102.50
Jane	F	12	59.80	84.50
Janet	F	15	62.50	112.50
Joyce	F	11	51.30	50.50
Judy	F	14	64.30	90.00
Louise	F	12	56.30	77.00
Mary	F	15	66.50	112.00
MEAN:		13	60.59	90.11
MIN:		11	51.30	50.50
1st QUAD:		12	56.50	84.00
MEDIAN:		13	62.50	90.00
3rd QUAD:		14	64.30	102.50
MAX:		15	66.50	112.50

Figure 4: The data sets `males` and `females` exported onto the same worksheet, formatted, and given formatted titles, as explained in Example 3.

Now we have five data sets to export – three for the titles, and two for the original data sets. Let's begin with the data – we export `males` onto an area starting at the 7th row and 2nd column, designated by `cell1row` and `cell1col`. As before, we name the new worksheet *Class*, get summary statistics for the numerical variables, and specify that Excel not be closed after the export. We introduce a new formatting option, `bordered`, which places a thick border around the data set:

```
%exportToXL( dsin=males, savepath=&outroot, savename=Example 3, sheet=Class, wsformat=bordered,
  cell1row=7, cell1col=2, statvars=age height weight, endclose=no );
```

For the `females` data set, we set `tmplpath` and `tmplname` the same as `savepath` and `savename` as before, to export onto the same workbook. By specifying `sheet=Class`, we export onto the same worksheet we previously exported onto – this time, exporting onto the 7th row, 8th column:

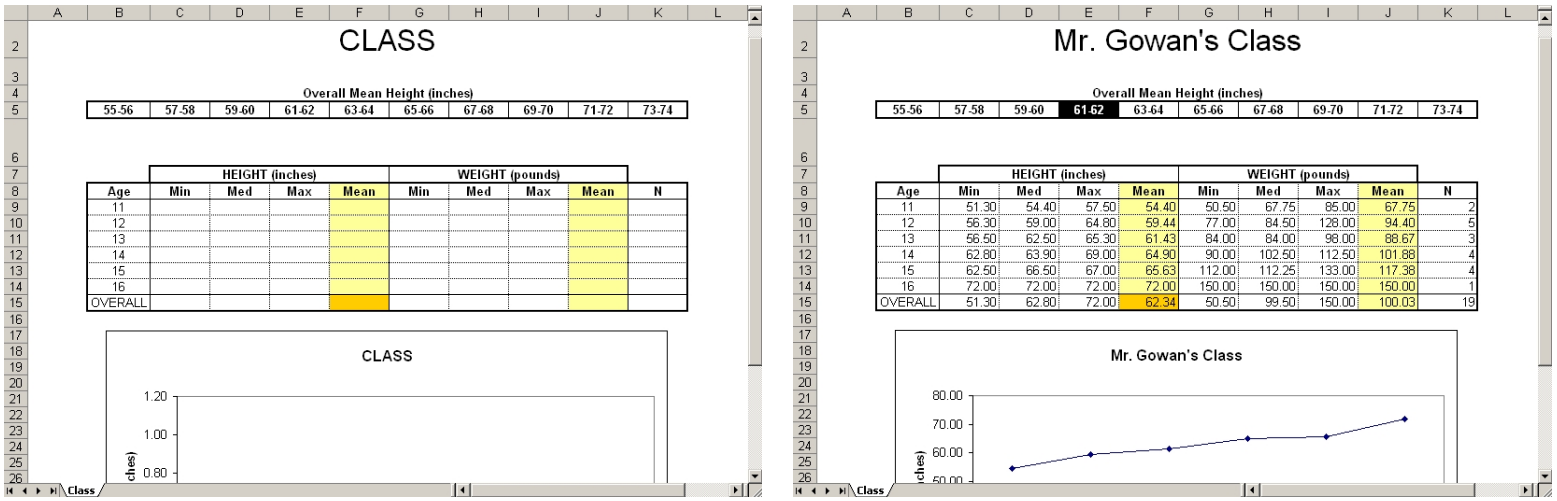
```
%exportToXL( dsin=females, savepath=&outroot, savename=Example 3, tmplpath=&outroot,
  tmplname=Example 3, sheet=Class, wsformat=bordered, cell1row=7, cell1col=8, statvars=Age Height Weight,
  endclose=no );
```

For the data sets for the titles, `fheader`, `mheader` and `title`, we proceed as above, except that we use two new formatting options, `title` and `title_big`, which make the fonts boldface and (for the second one) large. We also wish to exclude the headers of the data sets, which we do by setting `exportheaders=no`. We would like the titles to be on cells merged `x` down and/or `y` across, which we set by `mergedown=x` and `mergeacross=y`. As before, we delete `endclose=no` on the last step, since we then want Excel to close. The final product is shown in Figure 4:

```
%exportToXL( dsin=mheader, savepath=&outroot, savename=Example 3, tmplpath=&outroot, tmplname=Example 3,
  sheet=Class, wsformat=title, cell1row=6, cell1col=2, mergeacross=5, exportheaders=no, endclose=no );

%exportToXL( dsin=fheader, savepath=&outroot, savename=Example 3, tmplpath=&outroot, tmplname=Example 3,
  sheet=Class, wsformat=title, cell1row=6, cell1col=8, mergeacross=5, exportheaders=no, endclose=no );

%exportToXL( dsin=title, savepath=&outroot, savename=Example 3, tmplpath=&outroot, tmplname=Example 3,
  sheet=Class, wsformat=title_big, cell1row=2, cell1col=2, mergeacross=11, mergedown=2, exportheaders=no );
```



(a) (b)

Figure 5: The template (a) and the output (b). The title of the graph is linked to the merged cell in row 2.

Example 4: Exporting onto a Worksheet of a Template Workbook

This is a situation where we are given an Excel template – the destination worksheet is already pre-formatted, and SAS just needs to place the data into the correct places. ExcelXP cannot do this.

Let's assume the entries of `sashelp.class` are students in Mr. Gowan's class. We then make the data set `gowanstats` with the min, median, max, and mean values of both height and weight, as well as the number of students, by age (code not shown here). We also make a data set for the header, entitled `gowantitle`. We would like to export them into the Excel template[13] on Figure 5(a), called `Class Template.xls`. We do this using what we have done before – using two calls to `%exportToXL` (one for each data set) – except that in the first call, `tmplpath` and `tmplname` refer to the directory path and name of our template file. For both calls, we set `wsformat=none`, since our template is formatted already:

```
%exportToXL( dsin=gowanstats, savepath=&outroot, savename=Example 4-1, tmplpath=&outroot, sheet=Class,
  tmplname=Class Template, wsformat=none, cell1row=9, cell1col=2, exportheaders=no, endclose=no );

%exportToXL( dsin=gowantitle, savepath=&outroot, savename=Example 4-1, tmplpath=&outroot, sheet=Class,
  tmplname=Example 4-1, wsformat=none, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no );
```

There are two ways we might improve on the output from the above code (not shown). Firstly, running the code in Excel 2003 or later will produce flags in the upper left of the cells in the *Age* column. This indicates that there is a numeric value in a cell designated for text – since `age` is a text variable (done to accomodate the value of `OVERALL` in the last observation). This is not a huge problem. However, it underscores a problem that can arise when `%exportToXL` formats the cells the same way that the variable is formatted within SAS, which is done by default. This can be avoided by setting `exportvarfmts=no`, which keeps the formats from being exported – in this case, keeping the cells unformatted and thereby keeping the flags away.

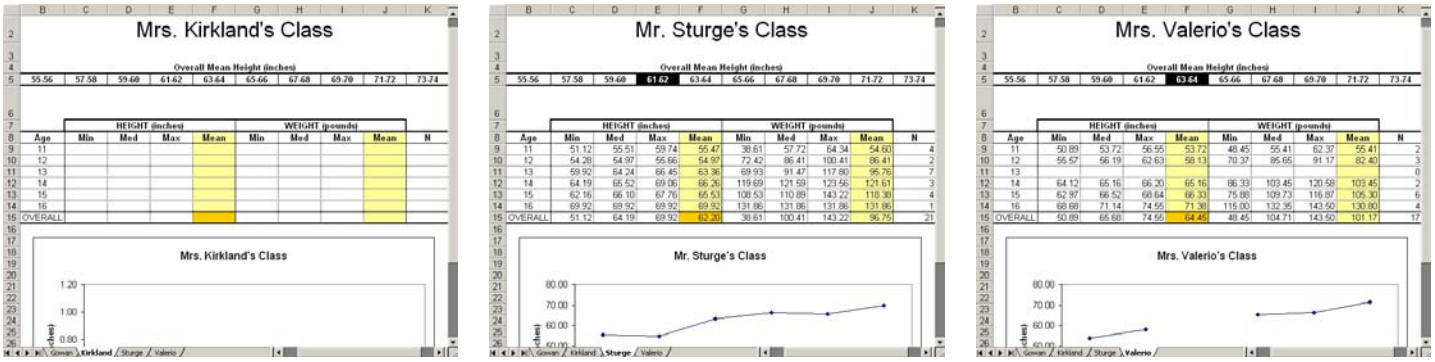
Secondly, we have a row of boxes on the top of the Excel template. Perhaps for a quick reference, we would like to darken the box that corresponds to the overall mean height of the class (shown in the orange cell). We can do this by setting `wsformat=color1`, which applies to this template only – for more details, see Part II of this paper.

```
%exportToXL( dsin=gowanstats, savepath=&outroot, savename=Example 4-2, tmplpath=&outroot, sheet=Class,
  tmplname=Class Template, wsformat=none, cell1row=9, cell1col=2, exportheaders=no, endclose=no,
  exportvarfmts=no );

%exportToXL( dsin=gowantitle, savepath=&outroot, savename=Example 4-2, tmplpath=&outroot, sheet=Class,
  tmplname=Example 4-2, wsformat=color1, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no,
  exportvarfmts=no );
```

The output is shown on Figure 5(b).

[13] Someone looking through the example code may note that we listed the entries in the *Age* column in the template, then exported the exact same entries over them when exporting `class`. We could have left out the entries in the template – we do this to make sure the rows in the data set match up with the rows in the template.



(a) (b) (c)

Figure 6: Output for Mrs. Kirland's (a), Mr. Sturge's (b) and Mrs. Valerio's (c) classes.

Example 5: Exporting onto a Template Worksheet of a Template Workbook

In Example 4, we used the worksheet in Figure 5(a) as a template for the one in Figure 5(b) – we just poured data onto the template worksheet. Now we would like to use the worksheet in Figure 5(a) as a template for several worksheets, producing not only the worksheet in Figure 5(b), but also those in Figures 6(a), (b) and (c). Moreover, we would like to name each of these worksheets after the respective teachers (e.g., *Gowan*, *Kirkland*, *Sturge* and *Valerio*). Note that Mrs. Kirkland's class is empty – let's assume she is on a leave of absence this year. However, for consistency, we would like to have a worksheet for her, even though it is empty. `%exportToXL` can be made to export the template worksheet even if `dsin` is empty or nonexistent.

First we export `gowanstats`, but this time we set `tmplsheet=Class` and `sheet=Gowan`, indicating that we want to use the worksheet *Class* as a template for a new worksheet, named *Gowan*. We set `exporttmplifempty=yes` to say we would like to export the template sheet even if the data set `gowanstats` is empty or nonexistent. We then do the same for the rest:

```
%exportToXL( dsin=gowanstats, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Gowan,
  tmplname=Class Template, tmplsheet=Class, wsformat=none, cell1row=9, cell1col=2, exportheaders=no,
  endclose=no, exportvarfmts=no, exporttmplifempty=yes );

%exportToXL( dsin=kirklandstats, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Kirkland,
  tmplname=Example 5-4, tmplsheet=Class, wsformat=none, cell1row=9, cell1col=2, exportheaders=no, endclose=no,
  exportvarfmts=no, exporttmplifempty=yes );

%exportToXL( dsin=kirklandtitle, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Kirkland,
  tmplname=Example 5-4, wsformat=color, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no, endclose=no );

%exportToXL( dsin=sturgestats, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Sturge,
  tmplname=Example 5-4, tmplsheet=Class, wsformat=none, cell1row=9, cell1col=2, exportheaders=no, endclose=no,
  exportvarfmts=no, exporttmplifempty=yes );

%exportToXL( dsin=sturgetitle, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Sturge,
  tmplname=Example 5-4, wsformat=color, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no, endclose=no );
```

In the last steps, we do a couple things differently. Firstly, when exporting `valeriostats`, it is the last time we need our template worksheet (*Class*), so we delete it, since we do not want it in our final output. We do this via `deletetmplsheet=yes`. Secondly, we export `gowantitle` last because we would like out final Excel file to be open to this worksheet first, since *Gowan* is the first worksheet alphabetically. And as before, since we want Excel to be closed at the end, we delete `endclose=no`:

```
%exportToXL( dsin=valeriostats, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Valerio,
  tmplname=Example 5-4, tmplsheet=Class, wsformat=none, cell1row=9, cell1col=2, exportheaders=no, endclose=no,
  exportvarfmts=no, deletetmplsheet=yes, exporttmplifempty=yes );

%exportToXL( dsin=valeriotitle, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Valerio,
  tmplname=Example 5-4, wsformat=color, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no, endclose=no );

%exportToXL( dsin=gowantitle, savepath=&outroot, savename=Example 5-4, tmplpath=&outroot, sheet=Gowan,
  tmplname=Example 5-4, wsformat=color, cell1row=2, cell1col=2, mergeacross=10, exportheaders=no );
```

CONCLUSIONS

This is just an introduction to `%exportToXL` – for more information, see Part II of this paper, or Derby (2007), or the project website listed under **CONTACT INFORMATION**.

REFERENCES

Adlington, T. (2005), Using VBA and Base SAS to get data from SAS to Excel without data integrity issues, *Proceedings of the 2005 Pharmaceutical users Software Exchange Conference*, paper AS11.
http://www.lexjansen.com/phuse/2005/as/as11.pdf

Beal, D. (2004), Using Dynamic Data Exchange to customize formatted reports in Microsoft Excel, *Proceedings of the Twelfth Southeast SAS Users Group Conference*, paper DP03.
http://www8.sas.com/scholars/05/PREVIOUS/2001_200.4/2004_MOR/Proceed/_2004/DataPresentation/DP03-Beal.pdf

Brown, D. (2005), %sas2xl: A flexible SAS macro that uses tagsets to produce complex, multi-tab Excel spreadsheets with custom formatting, *Proceedings of the Thirtieth SAS Users Group International Conference*, paper 092-30.
http://www2.sas.com/proceedings/sugi30/092-30.pdf

Conway, T. (2005), Making Bill Gates and Dr. Goodnight run your SAS code: Using VBA, ADO and IOM to make Excel and SAS play nice, *Proceedings of the Thirtieth SAS Users Group International Conference*, paper 157-30.
http://www2.sas.com/proceedings/sugi30/157-30.pdf

DelGobbo, V. (2006), Creating and importing multi-sheet Excel workbooks the easy way with SAS, *Proceedings of the Thirty-First SAS Users Group International Conference*, paper 115-31.
http://www2.sas.com/proceedings/sugi31/115-31.pdf

DelGobbo, V. (2007), Creating multi-sheet Excel workbooks the easy way with SAS, *Proceedings of the 2007 SAS Global Forum*, paper 229-2007.
http://www2.sas.com/proceedings/forum2007/229-2007.pdf

Delwiche, L. D. and Slaughter, S. J. (2003), *The Little SAS Book*, third edn, SAS Institute, Inc., Cary, NC.

Derby, N. (2007), User's guide to `%exportToXL`, version 1.0.
http://exporttoxl.sourceforge.net/docs/exporttoxlv1.0-ug-cur.pdf

Fecht, M. and Bennett, P. (2006), SAS Enterprise Guide Stored Processes, part 1: The information consumer's view; part 2: Creation and deployment, *Proceedings of the Thirty-First SAS Users Group International Conference*, paper 257-31.
http://www2.sas.com/proceedings/sugi31/257-31.pdf

Foster, E. (2005), SAS/AF and software prototyping - having your PIE and eating it!, *Proceedings of the 2005 Pharmaceutical users Software Exchange Conference*, paper AS08.
http://www.lexjansen.com/phuse/2005/as/as08.pdf

Gebhart, E. (2005), ODS MARKUP: The SAS reports you've always dreamed of, *Proceedings of the Thirtieth SAS Users Group International Conference*, paper 085-30.
http://www2.sas.com/proceedings/sugi30/085-30.pdf

Gebhart, E. (2006), The beginner's guide to ODS MARKUP: Don't panic!, *Proceedings of the Thirty-First SAS Users Group International Conference*, paper 263-31.
http://www2.sas.com/proceedings/sugi31/263-31.pdf

Gebhart, E. (2007a), ODS and Office integration, *Proceedings of the 2007 SAS Global Forum*, paper 227-2007.
http://www2.sas.com/proceedings/forum2007/227-2007.pdf

Gebhart, E. (2007b), ODS Markup, tagsets, and styles! taming ODS styles and tagsets, *Proceedings of the 2007 SAS Global Forum*, paper 225-2007.
http://www2.sas.com/proceedings/forum2007/225-2007.pdf

Parker, C. (2003), Generating custom Excel spreadsheets using ODS, *Proceedings of the Twenty-Eighth SAS Users Group International Conference*, paper 012-28.
http://www2.sas.com/proceedings/sugi28/012-28.pdf

Poppe, F. (2001), ExcelDDE tagset.
http://support.sas.com/rnd/base/topics/odsmarkup/customer.html

Roper, C. A. (2000), Intelligently launching Microsoft Excel from SAS, using SCL functions ported to Base SAS, *Proceedings of the Twenty-Fifth SAS Users Group International Conference*, paper 97-25.
http://www2.sas.com/proceedings/sugi25/25/cc/25p097.pdf

Vyverman, K. (2000), Using dynamic data exchange to pour SAS data into Microsoft Excel, *Proceedings of the Eighteenth SAS European Users Group International Conference*.
http://www.sas-consultant.com/professional/SEUGI18-Using-DDE-to-Pour-S.pdf

Vyverman, K. (2001), Using dynamic data exchange to export your SAS data to MS Excel - Against all ODS, Part I, *Proceedings of the Twenty-Sixth SAS Users Group International Conference*, paper 011-26.
`http://www2.sas.com/proceedings/sugi26/p011-26.pdf`

Vyverman, K. (2002), Creating custom Excel workbooks from Base SAS with Dynamic Data Exchange: A complete walkthrough, *Proceedings of the Twenty-Seventh SAS Users Group International Conference*, paper 190-27.
`http://www2.sas.com/proceedings/sugi27/p190-27.pdf`

Vyverman, K. (2003), Fancy MS Word reports made easy: Harnessing the power of Dynamic Data Exchange - Against all ODS, Part II, *Proceedings of the Twenty-Eighth SAS Users Group International Conference*, paper 016-28.
`http://www2.sas.com/proceedings/sugi28/016-28.pdf`

Vyverman, K. (2005), A matter of presentation: Generating PowerPoint slides from Base SAS using Dynamic Data Exchange, *Proceedings of the Thirtieth SAS Users Group International Conference*, paper 045-30.
`http://www2.sas.com/proceedings/sugi30/045-30.pdf`

Watts, P. (2004), Highlighting inconsistent record entries in Excel: Possible with SAS ODS, optimized in Microsoft DDE, *Proceedings of the Seventeenth Northeast SAS Users Group Conference*.
`http://www.nesug.info/Proceedings/nesug04/io/io01.pdf`

Watts, P. (2005), Using single-purpose SAS macros to format Excel spreadsheets with DDE, *Proceedings of the Thirtieth SAS Users Group International Conference*, paper 089-30.
`http://www2.sas.com/proceedings/sugi30/089-30.pdf`

ACKNOWLEDGMENTS

I am deeply indebted to Koen Vyverman and Perry Watts for their earlier works on this subject – in particular, for Vyverman's work on `%sastoxl`, which is the basis for `%exportToXL`. I merely filled in the details to their big ideas.

Furthermore, I thank many of the good people at SAS technical support, who kept me going when I got stuck – especially Peter Ruzsa (who made me realize that X4ML commands are language-specific), Russ Tyndall (who helped me with macro variables and made `%makeExcelFormats` functional) and Jennifer B (who answered my ODBC and OLE questions).

At SAS I also thank Eric Gebhart for checking my facts on the ExcelXP tagset, and Jim Simon for making my basic version of `%makeExcelFormats` much cleaner.

I thank Ron Fehd for providing me with a LaTeX template for SAS conference papers (used here).

I thank whomever first composed the list of Excel function translations (original source unknown).

Lastly, and most importantly, I thank Charles for his patience and support.

CONTACT INFORMATION

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

Nathaniel Derby
Statis Pro LLC
815 First Ave., Suite 287
Seattle, WA 98104-1404
206-973-2403
`nderby@users.sourceforge.net`
`http://exporttoxl.sourceforge.net`