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

Nathaniel Derby, Washington Mutual, Seattle, WA

ABSTRACT

There are several ways to export data from SAS® into Microsoft® Excel® but very few allow for exporting into custom-formatted spreadsheets such as those demanded for specific reports. 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. This paper argues that DDE is actually far from obsolete and outperforms the newer methods in some cases that often arise in practice. Furthermore, this paper offers 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 general criticisms of DDE. Using it requires no DDE programming knowledge and thus brings the power of DDE to any PC SAS user. This macro works 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, 2003b) 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 (7/25/07), 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.

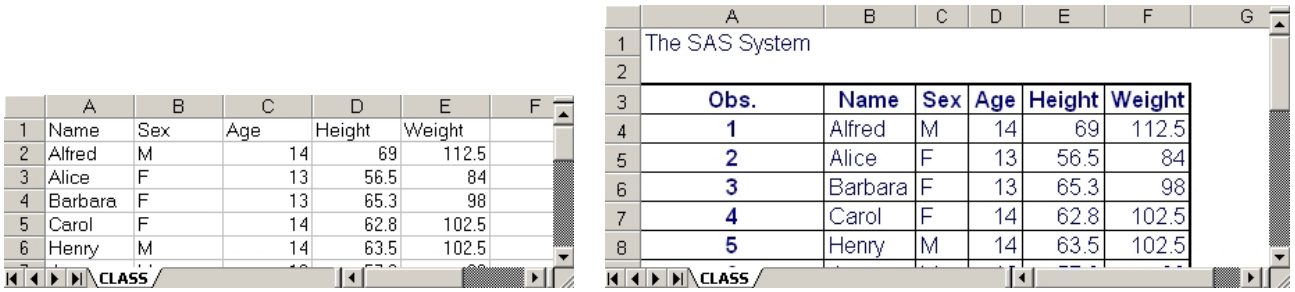
Generally, the only reliable alternative to DDE is the *ExcelXP tagset* of ODS MARKUP (DelGobbo, 2006, 2007; Gebhart, 2005, 2006, 2007a,b). This is a well-designed routine for exporting into Excel. It can make custom formats with many options, it is quicker than DDE, and it works without having to open 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+).

[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	F
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.

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` macro – see the **`%format_&wsformat`** and **MAKING A NEW WORKSHEET FORMAT** subsections 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` macro – see the **MAKING A NEW WORKSHEET FORMAT** subsection.
[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 the **PROGRAMMING DETAILS** section. 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          )
```

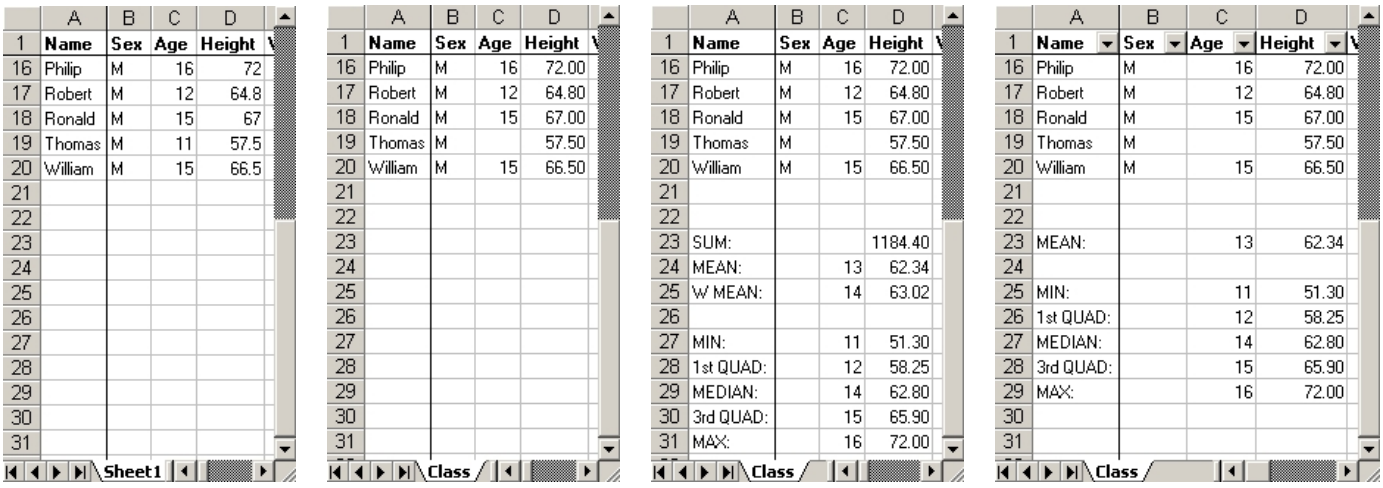
[8] This macro, the code for the following examples, and this user's guide can all be downloaded from `http://exporttoxl.sourceforge.net`.
[9] `da` for Danish, `de` for German, `es` for Spanish, `fi` for Finnish, `fr` for French, `it` for Italian, `nl` for Dutch, `no` for Norwegian, `pt` for Portuguese, `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 the **CUSTOM FORMATTING** subsection.
- `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 (7/25/07) – there may be problems with the other languages. See the **`%setVariables`** subsection for details.



(a) (b) (c) (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 );
```

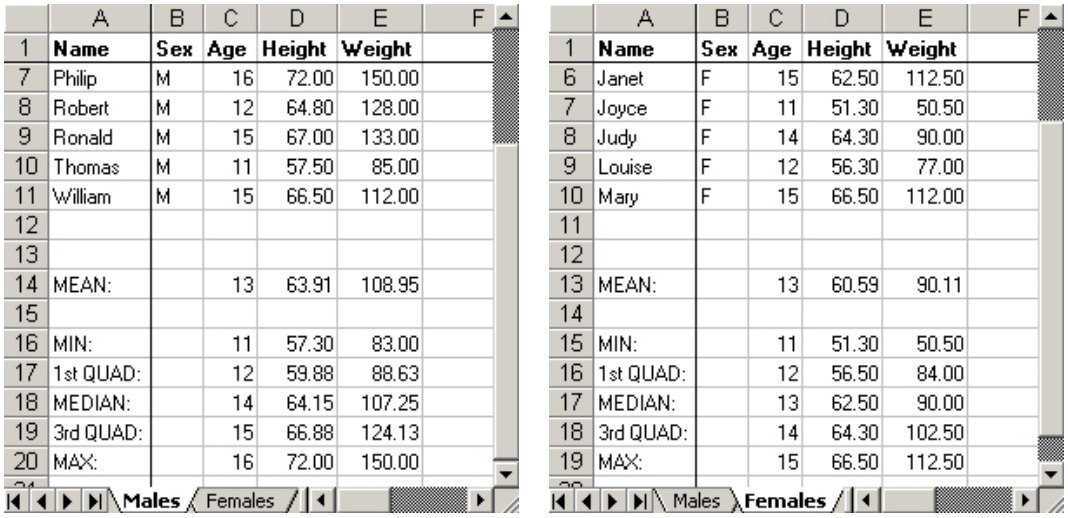
Suppose we would like some summary statistics on *age*, *height* and *weight*, with *age* being the weight variable for the weighted average. Furthermore, suppose we would also like a sum for *height*:

```
%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 );
```

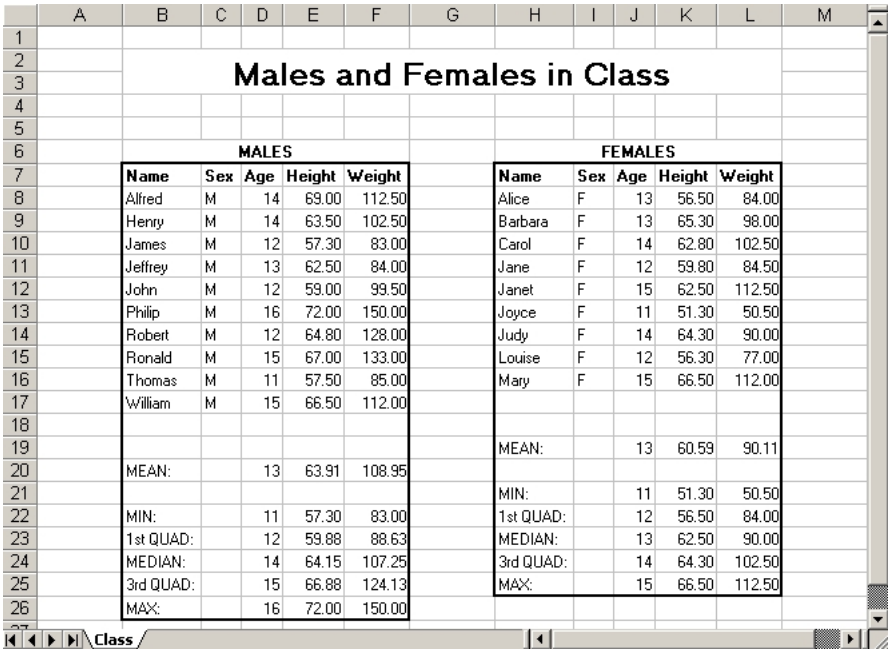
Example 3: Exporting onto Different Regions of One Worksheet

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:

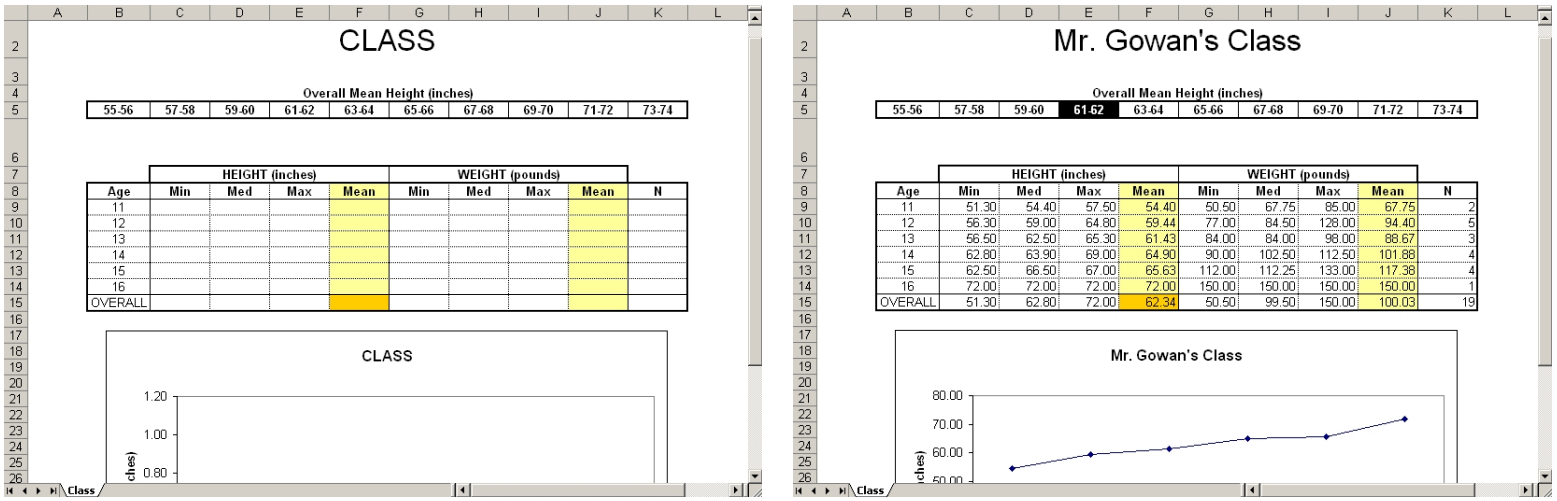
```
%exportToXL( dsin=males, savepath=&outroot, savename=Example 3, sheet=Class, wsformat=bordered,
  cell1row=7, cell1col=2, statvars=age height weight, endclose=no );
```

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,xtmplname=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 );
```

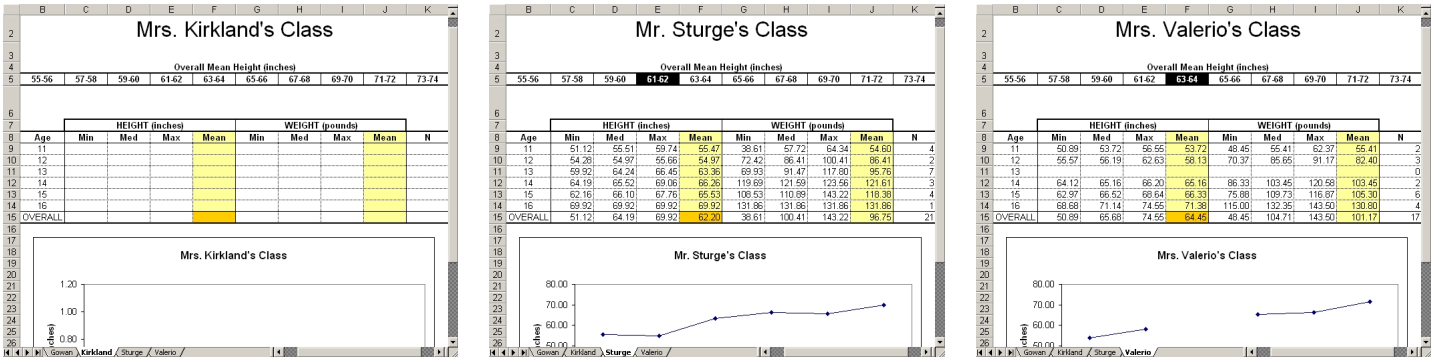
There are two ways we might improve on the output from the above code (not shown). Firstly, running the code in Excel 2003 or later will produce flags in the upper left of the cells in the *Age* column. This indicates that there is a numeric value in a cell designated for text – since `age` is a text 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 the **`%format_&wsformat`** subsection.

```
%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

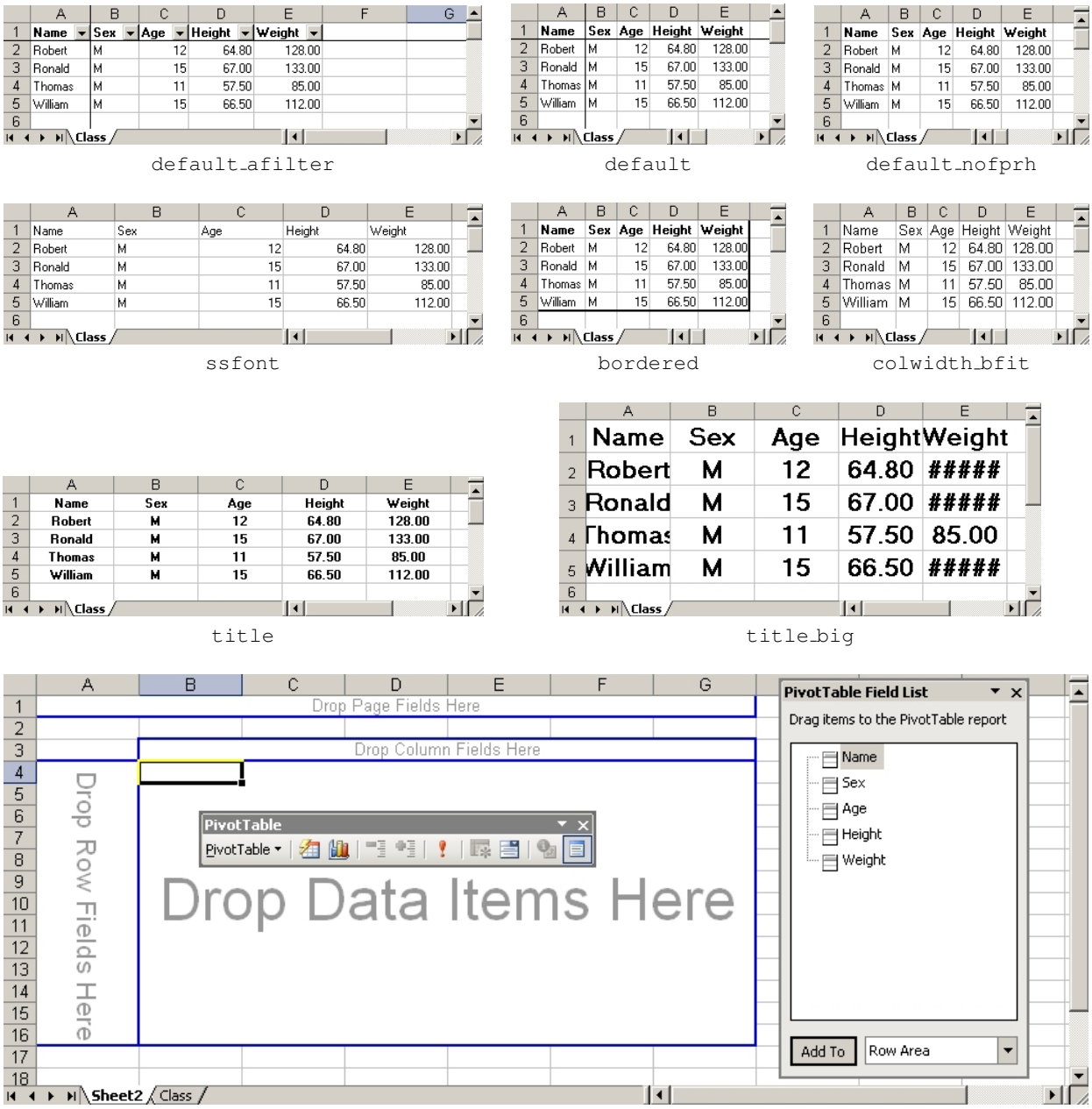
In Example 4, we used the worksheet in Figure 5(a) as a template for the one in Figure 5(b) – we just poured data onto the template worksheet. Now we would like to use the worksheet in Figure 5(a) as a template for several worksheets, producing not only the worksheet in Figure 5(b), but also those in 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*. Furthermore, we set `exporttmplifempty=yes` to say that 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 );
```



`default_pivot` (The *Class* worksheet has the `default` format)

Figure 7: The last four observations of `sashelp.class` shown with each available worksheet format (values for `wsformat`).

CUSTOM FORMATTING

As explained in the introduction, custom formatting is what differentiates DDE from most other methods for exporting data from SAS into Excel. In `%exportToXL`, there are two ways to export data onto a custom-formatted worksheet:

- Format the Excel worksheet manually, then export the data into the pre-formatted worksheet, using the `wsformat=none` and possibly the `exportvarfmts=no` options. `wsformat=none` tells SAS to leave all aspects of the worksheet format alone, except possibly for the variable formats – e.g., the value and format of the date "08-15-1977" is to be exported. `exportvarfmts=no` tells SAS to leave the variable formats alone as well – e.g., the value of "08-15-1977" is to be exported into a cell preformatted as `m/d/yy`[14], thus getting the formatted value of "8/15/77". This option is unavailable in any other method described in the introduction[15].

[14] This is an example of exporting into a Excel variable format not available in SAS. For an explanation of this and other Excel formats, see the **`%makeExcelFormats`** subsection.

[15] A stored process can be used to export a SAS data set onto a pre-formatted Excel worksheet, but it is done manually – exporting more than one data set can be quite time-consuming and onerous.

- Use a *worksheet format* – code telling SAS how to format the destination Excel worksheet, treated as a component of `%exportToXL`.

Currently there are nine different worksheet formats, as shown in Figure 7:

- `default`: MS Sans Serif, 8.5pt font with a bold header, best fit column width, 12.00 row height[16] and frozen panes.
- `default_afilter`: The same as `default`, but with an autofilter added in the headings.
- `default_nofprh`: The same as `default`, but without the frozen panes or the row height set to a particular value.
- `ssfont`: MS Sans Serif, 8.5pt font.
- `bordered`: MS Sans Serif, 8.5pt font with a bold header, best fit column width, 12.00 row height and with the entire data set bordered with a thick border. The effect isn't shown very well in Figure 7, since two borders are hidden – a better example is shown in Figure 4.
- `colwidth_bfit`: The column width is set to best fit – everything else is unspecified.
- `title`: MS Sans Serif, 8.5pt bold font, centered. Mostly appropriate for data sets corresponding to titles (data sets with one variable and one observation, as with the `title` data set in Example 3). This is used for the **MALES** and **FEMALES** titles in Figure 4.
- `title_big`: MS Sans Serif, 18pt bold font, centered. Mostly appropriate for data sets corresponding to titles (data sets with one variable and one observation, as with the `title` data set in Example 3). This is used for the **Males and Females in Class** title in Figure 4.
- `default_pivot`: the same as `default`, but with an unspecified pivot table shown on another worksheet.

Each format above is coded in the macro `%format_xxx` (where `xxx` is the format name) and stored in `format_xxx.sas` in the `&exroot` directory as described in the **INSTALLATION** subsection.

The above list was not intended by any means to be exhaustive! Rather, they are just illustrative of the possibilities – in fact, they were created solely as the need arose for various projects done during the testing phase of this macro. Many other possibilities exist, and many common ones can be formed simply by editing the code for the `default` format – see the **MAKING A NEW WORKSHEET FORMAT** subsection for details.

PROGRAMMING DETAILS

This section is intended for advanced users who wish to understand and/or modify the code. Any PC SAS user with moderate programming skills should have no problems using `%exportToXL` without learning these details.

This macro is based on Vyverman's (2000) macro `%sastoxl`, but with added functionality (e.g., allowing for template sheets) and more robustness (e.g., not crashing when the input data set is empty or nonexistent). As such, this macro's structure, shown in Figure 8, is similar to that of `%sastoxl`, but with one important difference: Following suggestions of Watts (2005), it is broken up into a set of smaller macros. This makes it easier to understand and modify various components of the code.

As explained in the **INSTALLATION** subsection, the code itself is found in the downloaded *exportToXL* directory, where there are many SAS files. The main file, `exportToXL.sas`, is shown in Figure 8. Following conventional use, each macro is defined in the SAS program file of the same name – e.g., `%setVariables` is defined in `setVariables.sas`. Some of these macros use other macros. Each macro code is liberally supplied with commented explanations of what is happening, which covers more than what is explained in this section – The interested reader should refer to the macros themselves for more information. Here we just present a summary of some of the main ideas behind each macro program.

The parameters in Figure 8 are explained in the **PARAMETERS** subsection. The variables local to `%exportToXL` are all used in the various component macros – most notably, in `%setVariables`, as explained below. `&misspar` and `&missparextmpl` are 0-1 variables set to 1 only if there are errors in the inputs (e.g., the input SAS data set does not exist), thus causing SAS to go to `%mquit` and bypass the main parts. Various details about each macro are detailed below.

`%info`

This contains only commented instructions on the use of `%exportToXL`, analogous to those in the header of `%sas2xl`. This macro is completely optional, as it has no effect on the rest of the code.

[16] This is actually the default for Excel, and could have been left out for the same result. It is put into the code for ease of modifying to accomodate a different row height.

```
%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          );

  %local misspar missparexptmpl cnotes csource csource2 cmlogic csymbolg cmprint tab ulrowlab ulcollab
    lrrowlab lrcollab ulrowdat ulcoldat lrrowdat lrcoldat ulrowstat ulcolstat lrrowstat lrcolstat lrecl
    types vars i colind  closeExcel weightvar crash maxmrow printHeaders macrosheet sasnote saswarning
    saserror c r alignment appactivate appmaximize average border clear columnwidth copy editcolor error
    false fileclose filter fontproperties formatfont formatnumber formulareplace freezepanes getdocument
    getworkbook halt max median min new open pastespecial patterns percentile quit rowheight run saveas
    select selection sendkeys sendkeycmd setname setvalue sheetname sum sumproduct true windowmaximize
    workbookactivate workbookcopy workbookdelete workbookinsert workbookmove workbookname workbooknext;

  %info;
    %* Gives basic information about how to use EXPORTTOXL.                          ;

  %setVariables;
    %* Initializes many of the local macro variables listed above.                   ;

  %checkParms;
    %* Checks the parameters.                                                        ;

  %if &misspar %then %goto mquit;

  %openDDE;
    %* Opens Excel and sets up a DDE dialogue with it.                               ;

  %setTemplate;
    %* Sets up the template and gathers information about it.                        ;

  %if &missparexptmpl %then %goto mquit;

  %inputData;
    %* Pours the data in.                                                            ;

  %if &wsformat ne none %then %format_&wsformat;
    %* Formats the Excel worksheet if formatting is desired.                         ;

  %mquit:

  %closeDDE;
    %* Closes the file and the DDE connection.                                       ;

%mend exportToXL;
```

Figure 8: The `%exportToXL` macro. Semicolons are included after each macro call for debugging purposes (i.e., each macro can be commented out via a preceding `%*`).

`%setVariables`

Here most of the macro variables local to `%exportToXL` are initialized – most notably, the translations of the X4ML commands into the native language of the Excel application. SAS uses these commands to tell Excel what to do, as explained in Vyverman (2000, 2001, 2002) and Watts (2004, 2005). These commands are fully detailed in a help file from Microsoft, `Macrofun.hlp`, which can be downloaded from this project's website at `http://exporttoxl.sourceforge.net` or from Microsoft at `http://www.microsoft.com/learning`[17]. However, these only work for the English version of Excel – if Excel is in another language, the X4ML commands in that language are needed. For example, the English Excel cell formula "`=SUM(A1:A5)`" would need to be "`=SUMME(A1:A5)`" for the German version of Excel.

`%exportToXL` gets around this language problem by issuing each X4ML command as a macro variable, defined in `%lang_xx`, where `xx` is the language code. For example, for English (`%lang_en`) and German (`%lang_de`) we have

[17] Search for and run the file `Macrofun.exe`.

```
%macro lang_en

  %let c = c;
  %let r = r;
  %let appactivate = app.activate;
  %let appmaximize = app.maximize;
  %let average = average;
  %let border = border;
  ...

%mend lang_en;
```

```
%macro lang_de;

  %let c = s;
  %let r = z;
  %let appactivate = anw.aktivieren;
  %let appmaximize = anw.vollbild;
  %let average = mittelwert;
  %let border = rahmenart;
  ...

%mend lang_de;
```

Later in the `%exportToXL` code, when we want to use an X4ML command such as `app.maximize` (which maximizes the application window), instead of using it directly, we use `&appmaximize` and thus provide the appropriate translation from within `%lang_xx`. Doing this for each X4ML command allows an easy switch from one Excel native language to another, or to add another language not currently supported.

For good measure, we also assume that the SAS installation is in the same language as the Excel installation and translate the SAS log keywords NOTE, WARNING and ERROR, respectively stored as `&sasnote`, `&saswarning` and `&saserror`. These will be used for the notes, warning and error log notes generated by `%exportToXL`.

`%setVariables` chooses the appropriate language (defined by `&lang`) by cycling through the list of available languages:

```
%let langs = da de en es fi fr it nl no pt ru sv;
  %* The available languages.                                                    ;
%let ii = 1;
%let nullid = ;

%do %until( %scan( &langs, &ii ) = &nullid );
  %if %scan( &langs, &ii ) = &lang and %sysfunc( fileexist( &exroot\lang_&lang..sas ) ) = 1 %then %do;
    %* Double check to make sure the language file exists.                      ;
    %lang_&lang;
    %goto quit;
    %end;
  %let ii = %eval( &ii + 1 );
  %end;

  %put &saserror: The language code chosen for the Excel application, &lang, is not supported!;
  %let misspar = 1;

  %quit:
```

An error message appears if a language is chosen for which there is no translation file available. A few further notes:

- While most X4ML commands need to be translated, some of them (like `QUIT`) appear to work untranslated in at least two languages (English and German). To be safe, translations are provided for all commands.
- The code listed here has only been tested for English and German versions of Excel – for any other language, the input may have been misspelled, which would cause it to fail. The macro `%lang_xx` would be the place to rectify this.
- In particular, one X4ML command[18] was not listed in the Excel translation file, so it is a guess at this time for languages other than English and German. If the reader is using this with another supported native Excel language (i.e., Danish, Dutch, Finnish, French, Italian, Norwegian, Portuguese, Russian, Spanish, or Swedish), please contact the author about this.
- Any user who wishes to add another language or more X4ML commands would do so here – see Derby (2007) or the **MAKING A NEW WORKSHEET FORMAT** subsections for details.

%checkParms

Basic parameter checks are performed here. The macro variables `&misspar` and `&missparexptmpl` (local to `%exportToXL`) are 0-1 variables, set to 1 within this macro if there are problems with the `libin` or `dsin` parameters:

- If `&misspar` is set to 1, the main code in Figure 8 skips to `%mquit` after `%checkParms`, so that nothing is exported.
- If `&missparexptmpl` is set to 1, the main code skips to `%mquit` after `%setTemplate`, so that only the template sheet (without any data places into it) is exported.

[18] `sendkeycmd` – not actually an X4ML command, but rather the key commands to merge cells: ALT+O → E → A → ALT+M → RETURN in English. Since the early versions of Excel did not allow for merging cells, X4ML does not have a command for this. This is a way to work around this limitation.

Which of the two is chosen is determined by the value of &exporttmplifempty, explained in the **PARAMETERS** subsection:

- If either libin or dsin is not given, misspar is set to 1 and nothing is exported. This is done regardless of the value of &exporttmplifempty – if we don't even have the names of the library or data set, we shouldn't go any further. In practice, since the default value of libin is *work*, this only occurs if dsin is not given.
- If both libin and dsin are given but the data set is empty or nonexistent, &missparexptmpl is set to 1 and the template will be exported if exporttmplifempty = 1. Otherwise, &misspar is set to 1 and nothing is outputted.

Further parameter checks include the following:

- If cell1row or cell1col is not given, the missing values are each assigned to their default values of 1.
- If savepath or tmplpath has a backslash at the end, the backslash is deleted (required for the rest of the code).
- If savepath or savename is not given, the missing value is assigned to its default values.
- The values of endclose or exportheaders are translated to the 0-1 macro variables &closeExcel and &printHeaders, both local to %exportToXL.
- If nrows or ncols is either not given or larger than the actual number of rows or columns of the data set, it is set to the actual numbers of rows/columns.
- If either of tmplpath or tmplname is not given or nonexistent, a standard template is used.

Note that we translate the parameters endclose and exportheaders into 0-1 macro variables. We do this for robustness and flexibility – so that we allow endclose and exportheaders to take on values such as "true", "1" or "YES"[19]:

```
%if %among( %upcase( %substr( &endclose, 1, 1 ) ), Y T 1 ) %then %let closeExcel = 1;
%if %among( %upcase( %substr( &exportheaders, 1, 1 ) ), Y T 1 ) %then %let printHeaders = 1;
```

Lastly, in this macro many SAS system options (e.g., NOTES, SOURCE, SYMBOLGEN) are turned off, to minimize the amount of extraneous information shown in the log.

%openDDE

This is where a DDE dialogue with Excel is established, using the method introduced by Roper (2000), described by Vyverman (2001, 2002) and used by Watts (2005).

%setTemplate

This is where the *template* is set up – the target workbook and worksheet that the data is to be poured into. This involves collecting information about the target workbook (e.g., the names of the existing worksheets), naming the template worksheet, and setting up a DDE dialogue between SAS and this worksheet. This macro runs as follows:

1. If tmplpath and tmplname are given (and exist, as verified in %checkParms), we open it. Otherwise, we open a standard workbook.
2. Save the above file (our template workbook) as savename, in the savepath directory.
3. A macro worksheet must be established to carry out the X4ML instructions. Specifically, at each group of steps, a set of X4ML commands will be listed on this sheet, then executed. This worksheet will be named *MacroN*, where *Macro* is in the language of the Excel application (e.g., *Macro* in English, *Makro* in German) and *N* is the lowest positive integer not already used for the name of a macro worksheet. Rather than provide a translation of *Macro* in %lang_xx, we can simply look at the name of the worksheets before and after adding the macro worksheet, then look at the name of the added worksheet, assigning it to the local macro variable &macrosheet. We do this following the method of Vyverman (2003a).
4. We move the macro sheet (&macrosheet) to the first position of the workbook, so that we will always know where it is. This will be important in subsequent macros.
5. A DDE connection to the macro sheet is established.
6. To prepare for the actual writing of the data, some worksheet logic is implemented:
   (a) If &tmplname and &tmplpath are both blank, this is a standard template with one worksheet, whose name can be found via a PROC SQL statement.
      i. If the sheet parameter is given, this worksheet is renamed.
      ii. Otherwise, sheet equals to the name found in the PROC SQL statement.

[19] %among is a temporary replacement for the macro in operator, which was introduced in SAS 9, disabled in SAS 9.1.3, and will be back in a future SAS release. This code for %among is from SAS Institute (2006, p. A-10) and is included as an %exportToXL component macro.

(b) Otherwise the path and name of the template workbook are given and exists, so `%loadNames` (described below) is used to collect the names of the worksheets in it. If `&tmplsheet` is given, we check to see if it exists among these worksheets:

i. If the `sheet` parameter is left blank, we remember the names of the worksheets from before. Then we add another worksheet and note its name (via `%loadNames` and a `PROC SQL` statement, as above), dropping that value into `sheet`.

ii. Otherwise, the code checks to see if `sheet` already exists in the workbook. If it does not exist, we add a new sheet and rename it as `sheet`. Otherwise, it exists, so there is no new sheet to create.

7. If the template sheet exists (checked above), we delete the new worksheet just created and copy the template worksheet instead, renaming it `sheet`. If `deletetmplsheet` is indicated, the template worksheet is deleted (unless `sheet` has the same value as `tmplsheet`, in which case we merely moved `tmplsheet` to the end of the workbook).

   It may seem inefficient to use this process to incorporate a template sheet, but it always get the right value of `sheet`, notably when it is not given and thus given the value of *SheetN*.

Note: One problem with `%sas2xl` is that it will crash if the resulting Excel file name has a period in it (e.g., `file.01.xls`). This is avoided by adding the `.xls` suffix explicitly in various parts of the code, as in

```
ddecmd = "[&saveas("||'"'||"&savepath"||'\'||"&savename"||'.xls")]';
put ddecmd;
```

rather than

```
ddecmd = "[&saveas("||'"'||"&savepath"||'\'||"&savename)]";
put ddecmd;
```

which works as long as there is not a period in the resulting file name. This is done wherever possible in various macros.

`%inputData`

This is where `%exportToXL` pours the SAS data into our desired Excel workbook. It is the most important component macro, as well as the most complex. This macro follows the following pseudocode:

1. We compute the number of rows to add to the range of cells where data will be exported. This is based on whether the fields `sumvars`, `statvars` and `weightvar` are empty.
2. This range of cells is partitioned into rows corresponding to the headers, the data, and the sum/summary statistics. Each of these partitions is given a range of cells. For instance, for the output in Figure 2(d), all partitions have columns 1-5. For rows, the header is row 1, the data is rows 2-20, and the sum/summary statistics are rows 21-29.
3. Metadata for the variable formats and names are gathered on the input data set `dsin` from `PROC CONTENTS` output. These are separated by spaces and put into local macro variables `&types` (`TYPE`), `&fmts` (`FORMAT`), `&fmtls` (`FORMATL`), `&fmtds` (`FORMATD`), `&vars` (`NAME`), and `&lengs` (`LENGTH`). For example, for `sashelp.class`, we have `&vars = Name Sex Age Height Weight` and `&types = 1 1 2 2 1`.
4. We define the local macro variables `&sumnumlist` and `&statnumlist` to give the numbers (ordered, separated by a space) of columns corresponding to columns of variables indicated by `sumvars` or `statvars`. For instance, if we export `sashelp.class` with `sumvars = Weight` and `statvars = Weight Height`, we have `&sumnumlist = 5` and `&statnumlist = 4 5`.
5. If `exportvarfmts` is indicated, we format the cells before pouring data into them:
   (a) We make the Excel formats from `%makeExcelFormats` (described below). This gives us a local macro variable, `&xlfmts`, which contain the Excel version of the formats, separated by exclamation points (since that is a character not used in any Excel format). If no format is listed, the entry is `NONE`. For instance, for `sashelp.class`, we have `&xlfmts = @!@!NONE!NONE!NONE` (since the numeric variables are unformatted).
   (b) If `printHeaders`=1, we format the first row as text. Then we format the rest of the rows as the formats dictated by `&xlfmts` (left unformatted if `NONE`).
6. We then define DDE links to the ranges of cells for the header, data, and summary data that we defined in step 2 above.
7. If `&printHeaders`, we pour the variable labels into the first row, skipping columns and rows if `mergeacross` or `mergedown` > 1. (The cells will be merged later – for now, they are merely skipped)
8. We use `%makeVarList` (explained in Derby (2007)) to make an array of variable names separated by an appropriate number of tabs (as dictated by `mergeacross` – again, we skip them for now, to be merged later), resulting in `&varlist`.
9. Finally, we pour the data in, using `&varlist`. We make sure not to insert an extra space in any cell (which was a problem with `%sastoxl`).

SAS format	Excel format string	Excel format name
$8.	@	Text
8.2	0.00	Number, 2 decimal places
z8.2	00000.00	(none)
percent8.2	0.00%	Percentage, 2 decimal places
mmddyy8.	mm/dd/yy	Date, type "03/14/01"
comma12.2	#,##0.00	Number, 2 decimal places, with comma separator
dollar12.2	_($* #,##0.00_);_($* (#,##0.00);_($* -??_);_(@_)	Accounting, 2 decimal places

Figure 9: A few SAS formats and their Excel equivalents.

10. If `sumvars` or `statvars` is given, we export the formulas for sum/summary statistics:
    (a) We first define the local macro variable `&wvarnum` as the column number of the weight variable `weightvar`. For example, if we set `weightvar = Age` for `sashelp.class`, we have `&wvarnum = 3`. If none is found, an error message shows in the SAS log, and `weightvar` is set to a blank.
    (b) We write the sum and stat summary data, indexing through the variable number `&varnum` – if it equals a number on `&sumnumlist`, we put in the formula. The same is done for `&statnumlist`.

    Note that SAS outputs the row labels for the sum/summary data whenever `sumvars` or `statvars` is given, whether or not these variables exist in the data set (e.g., the misspelling `statvars = Waight` for `sashelp.class`). Also, the column indices starts at 2 – no sum/summary statistics can be shown for the first variable.
11. Lastly, we merge the cells, across and and down, using the `&sendkeycmd`.

Note that only those variables with a SAS format are formatted on the worksheet.

`%makeExcelFormats`

Given a column number of the input SAS data set `dsin`, this macro makes the Excel version of the SAS format for the corresponding variable. This Excel format will be used within `%inputData` to properly format the variable – so that each entry of the resulting Excel worksheet will have roughly the same format as the corresponding entry of `dsin`.

Here, "Excel format" refers to the options available in the *Number* tab of *Format* → *Cells* within Excel. Each Excel format name (i.e., the name under *Category* and the various options) has a corresponding format string, which can be found under the *Custom* category. However, some Excel format strings do not have a corresponding name. Examples are shown in Figure 9 – a more complete list can be found under `Microsoft Excel XLS Files: ACCESS Procedure: XLS Specifics` in the SAS Help files (search for it).

Note that there is not a one-to-one relationship between SAS and Excel formats:

- The Excel format *m/d/yy* deletes leading zeroes for the month and day values, returning values like "9/4/07" and "10/17/07". There is no standard SAS format for this[20].
- The `dollarw.d` SAS format has many matches in Excel:
  - The *Currency* category places the dollar sign next to the number (like $5.00) and allows negative numbers to be displayed with a negative sign, in red, in parentheses, or in red *and* in parentheses.
  - The *Accounting* category places the dollar sign to the left of the cell (like $ 5.00).

Overall, the user must pick and choose. For example, because of the author's personal preference, the accounting Excel format is used for the `dollarw.d` SAS format – but any user can change this. See the **MAKING A NEW VARIABLE FORMAT** subsection for details. If no Excel format is found (or SAS provides no format), `NONE` is returned.

The outcome is the macro variable `&xlfmts`, which is a string of formats separated by exclamation points. For example, `@!@!mm/dd/yy!#,##0.00!0.00%` would indicate that the first two variables would be text, followed by a date, followed by comma, followed by a percent. An exclamation point is used because (unlike a space) it is not used in any Excel formats.

One final note: Strictly speaking, the equivalent to the Excel accounting format is

```
_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_);_(@_)
```

rather than what is shown in the table (the difference being `"-"??` rather than `-??`), but then that would result in the X4ML command

```
[format.number("_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_);_(@_)")],
```

[20] However, one can be created, as in `PROC FORMAT; picture date8_ ( default=8 ) low-hight='%m/%d/%0y' ( datatype=date ); RUN;` (suggested by SAS support).

which gives an error because of the two sets of quotation marks. The solution here is a little different from the accounting format, but gives the same result.

%format_&wsformat

Worksheet formats, as described in the **Custom Formatting** subsection, can be classified into two types:

- *General*: Can be applied to any SAS data set or Excel worksheet. No specific data set is accessed, and no specific range of cells is called. Since it is general, the code file should be included in the same directory as the rest of the `%exportToXL` macro files.
- *Specific*: Can be applied to one specific SAS data set or Excel worksheet. A specific data set or range of cells is involved, and using it for another data set or worksheet would result in an error or unintended consequences. The code file should be in a directory with other files pertaining to this specific project, rather than with the other `%exportToXL` macros.

There are nine kinds of *general worksheet formats*, as described in the **Custom Formatting** subsection. The code should be relatively straightforward, reflecting the modular structure of the X4ML commands. For example, the default format, `%format_default`, is defined as such:

```
%* Formats the Excel worksheet if formatting is desired.                                    ;
%*                                                                                         ;
%* FONT: MS Sans Serif, 8.5 pt                                                            ;
%* HEADER: Bold                                                                           ;
%* COLUMN WIDTH: Best fit                                                                 ;
%* ROW HEIGHT: 12.00 (The default for Excel)                                              ;
%* FREEZE PANES                                                                           ;

%macro format_default;

  data _null_;
    length ddecmd $200.;
    file sas2xl;
    put "[&error(&false)]";
    ddecmd = "[&workbookactivate("||'"'||"&sheet"||'"'||",&false)]";
    put ddecmd;
    ddecmd = "[&select("||'"'||"&r&ulrowlab&c&ulcollab:&r&lrrowstat&c&lrcolstat"||'")]';
    put ddecmd;
    ddecmd = "[&formatfont"||'("MS Sans Serif"'||",8.5,&false,&false,&false,&false,0,&false,&false)]";
    put ddecmd;
    ddecmd = "[&select("||'"'||"&r&ulrowlab&c&ulcollab:&r&lrrowlab&c&lrcollab"||'")]';
    put ddecmd;
    ddecmd = "[&formatfont"||'("MS Sans Serif"'||",8.5,&true,&false,&false,&false,0,&false,&false)]";
    put ddecmd;
    ddecmd = "[&columnwidth(0,"||'"'||"&c&ulcollab:&c&lrcollab"||'"'||",&false,3)]";
    put ddecmd;
    ddecmd = "[&rowheight(12.75,"||'"'||"&r&ulrowdat:&r&lrrowstat"||'"'||",&false)]";
    put ddecmd;
      %* Must be in points corresponding to a whole number of pixels -- e.g., 12.00 or   ;
      %* 12.75, but not 12.50.                                                           ;
    ddecmd = "[&freezepanes(&true,"||%eval(&cell1col+&mergeacross-1)||","||%eval(&cell1row+&mergedown-1)||")]";
    put ddecmd;
  run;

%mend format_default;
```

Each DDE command (`ddecmd`) is made indirectly, accessing the translation from the `%setVariables` macro, using the macro variable defined in one of the language macros (`%lang_xx`). This allows SAS to translate the commands to that of the Excel application – if, e.g., `row.height` were used rather than `&rowheight`, this would not work in a non-English Excel installation. Details are explained in the **%setVariables** subsection.

The macros for the other general worksheet formats are a variation of this, and all have the same summary note at the top, for ease of use.

Specific worksheet formats may or may not have a different structure from the above code. As an example, the code for the `color` worksheet format of Example 5[21], found in the *Macros* directory for that example, is as follows:

```
%macro format_color;

    %local colnum name;

    %let name = %sysfunc( lowcase( &sheet ) );
    %if %sysfunc( exist( &name.stats ) ) = 0 %then %goto endhere;

    data _null_;
        set &name.stats;
        if agec = 'OVERALL' then call symput( 'colnum', trim( left( round( ( meanh - 55.5 ) / 2 ) + 2 ) ) );
    run;

    data _null_;
        length ddecmd $200.;
        file sas2xl;
        put "[&error(&false)]";
        ddecmd = "[&workbookactivate("||'"'||"&sheet"||'"'||",&false)]";
        put ddecmd;
        ddecmd = "[&select("||'"'||"&r.5&c.&colnum"||'")]';
        put ddecmd;
        put "[&patterns(1,0,1)]";
        put "[&fontproperties(,,,,,,,,,2)]";
    run;

    %endhere:

%mend format_color;
```

Like the `default` worksheet format, all X4ML commands are issued indirectly via a translation, but now we have a specific data set and worksheet range:

- *Specific data set*: The `&name.stats` is a specific SAS data set that is being accessed. In this case, we first check to see if this data set exists, and if so, extracts a specific value from it – the variable *meanh* from the observation where *agec*=`OVERALL`. If this data set does not exist, SAS skips to the `%endhere` entry and no worksheet formatting is done.
- *Specific range*: Assuming the `&name.stats` data set exists, we are working with the fifth row, as shown in the sixth line of the `data _null_` statement. This macro is thus designed for a worksheet with something special in the fifth row (i.e., the bar shown at the top of each worksheet in Figure 6), and wouldn't work for another worksheet.

For either reason above, this is a *specific* (rather than *generalized*) worksheet format.

Note that this macro is designed to not give any errors – if the data set `&name.stats` does not exist, the formatting code is skipped completely. No error is given if a different template is chosen – but most likely it would be inappropriate for a certain cell in the fifth row to be colored black.

`%closeDDE`

This closes the DDE connection, as well as Excel itself if `endclose` is indicated.

MAKING MODIFICATIONS

Through the use of macro components such as `%wsformat_xxx` and `%lang_xx`, `%exportToXL` can easily be modified to accommodate new worksheet formats, languages, and Excel formats. Here we present only a couple aspects of making modifications – for more information, see Derby (2007).

MAKING A NEW WORKSHEET FORMAT

As explained in the **CUSTOM FORMATTING** subsection, the *worksheet format* refers to anything involved with changing the appearance of the resulting worksheet – e.g., changing a font, adding colors to some cells, or adding a pivot table.

One of the features of `%exportToXL` is that making a new worksheet format is actually quite simple. First of all, for a worksheet format that is equivalent to the `default` one but with just a couple changes (say, without the frozen panes and with a row height of 12.00), all that is needed is to modify the `format_default.sas` code (including the header) appropriately. For consistency, rather than change the code directly, it would be better to leave that file as is and to name the modified format as something else, such as `format_default_rh12.00nofp.sas`.

[21] This is similar to the `color1` and `color2` worksheet formats of Example 4.

For an entirely new format, called `xxx`, create the SAS code file `format_xxx.sas`, to be saved in either the same directory as the other `%exportToXL` macros (for a general worksheet format[22]) or in the corresponding project-specific directory (for a specific one). The code in this file should have the following structure:

```
%macro format_xxx;

    data _null_;
        length ddecmd $200.;
        file sas2xl;
        ...
    run;

%mend format_xxx;
```

The body of the above macro will contain the X4ML commands. Some of them just need to be quotes – e.g., the command `[error(false)]` is entered into the macro sheet if the line `put "[&error&false)]";` is entered into the macro above.

However, most X4ML commands have an argument that must be enclosed in double quotes. For example, `[select("r5c6")]` is the X4ML command to select the cell in the fifth row, 6th column. This cannot be entered in as `put "["&r.5&c.6"]";` because of the two sets of double quotes. The solution is to resolve it via concatenation operators (`||`) with `ddecmd`[23]:

```
ddecmd = "[&select("||'"'||"&r.5&c.&colnum"||'")]';
put ddecmd;
```

When writing new worksheet formats, it is not necessary to use the macro variables representing the translations if the macro in question is always going to be used with Excel in one language. For example, it is perfectly acceptable to use the command `put "[error(false)]"` above rather than `put "[&error(&false)]"` if the worksheet format macro is always going to be used with the English version of Excel. However, if a new worksheet format is intended to be multilingual, a translation of each X4ML command must be found within each translation macro (`%lang_xx`). Although the translation macros included here include many of the most popular X4ML commands, translations of new commands may be required. In that case, the file `Excel Functions Translated.xls` should be used.

MAKING A NEW VARIABLE FORMAT

Making a new variable format, or changing an existing one (since there is not a one-to-one relationship between SAS and Excel formats, as discussed in the **`%makeExcelFormats`** subsection) follows the following process:

1. Determine the Excel format string for the desired SAS format. One way to do this is to find it in the menu of Excel formats and then click on *Custom*, thus showing the code of what was just entered. Another source of information is under `Microsoft Excel XLS Files: ACCESS Procedure: XLS Specifics` in the SAS Help files (search for it).
2. Determine how to build this format string[24] using `&types`, `&lengs`, `&fmts`, `&fmtls` and `&fmtds`, as explained in step 3 of the **`%inputData`** subsection.
3. Using step 2 above, add or modify code in `%makeExcelFormats` that maps onto this format string from `&types`, `&lengs`, `&fmts`, `&fmtls` and `&fmtds`.

Steps 2 and 3 above may be made easier by looking at the existing code for `%makeExcelFormats`.

CONCLUSIONS

Currently (7/25/07), it is not known whether `%exportToXL` will be further developed – it will depend on user interest, as well as time and energy of either the author or other developers. There are ideas for further development, such as implementing customized graphical output, components for Excel formulas, and compatibility with OpenOffice.org Calc – further details are documented in Derby (2007). For updates, see the project website listed under **CONTACT INFORMATION**.

[22] The two types of worksheet formats, *general* and *specific*, are explained in the **`%format_&wsformat`** subsection.
[23] Watts (2004, 2005) has an alternative way of doing this, using `%unquote` and `%bquote`.
[24] or something very much like it, as in the *Accounting* format string as discussed in the **`%makeExcelFormats`** subsection.

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

SAS Institute (2006), *SAS Macro Programming: Advanced Topics*, SAS Institute, Inc., Cary, NC.

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. (2003a), Excel exposed: Using Dynamic Data Exchange to extract metadata from MS Excel workbooks, *Proceedings of the Tenth Southeastern SAS Users Group Conference*.
http://www8.sas.com/scholars/05/PREVIOUS/2001_200.4/2004_MOR/Proceed/_2003/Tutorials/TU15-Vyverman.pdf

Vyverman, K. (2003b), 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 also 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
Washington Mutual
1301 Second Ave.
Seattle, WA 98101
206-500-1096
nderby@users.sourceforge.net
http://exporttoxl.sourceforge.net