Paper 260-2008

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

Part II - Programming Details

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

This paper is intended for advanced users who wish to understand and/or modify the `%exportToXL` code described in Part I of this paper. 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 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 Part I of this paper, 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 1. 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 paper – 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 1 are explained in Part I of this paper. 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.

[1] Reprinted with permission of the author.

```
%MACRO 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 1: 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`[2]. 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

[2]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[3] 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) 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 1 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.

Which of the two is chosen is determined by the value of `&exporttmplifempty`, explained in Part I of this paper:

- 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.

[3] `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.

- 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"[4]:

```
%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 (2003).
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.
   (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`.

[4] `%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.

   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, in Figure 2(d) of Part I of this paper, all partitions have columns 1-5. 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`).
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.

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 2: A few SAS formats and their Excel equivalents.

%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 2 – 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[5].
- 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);_($* "-"??_);_(@_)")],
```

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 Part I of this paper, 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.

[5] 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).

There are nine kinds of *general worksheet formats*, as described in Derby (2007). The code should be relatively straightforward, reflecting the modular structure of the X4ML commands. For example, the default format, `%format_default`, is defined as

```
%* 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;
    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.

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[6], 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;
```

[6] This is similar to the `color1` and `color2` worksheet formats of Example 4 of Part I of this paper.

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 of Part I of this paper), 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 Part I of this paper, 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`.

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[7]) 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`[8]:

```
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

[7] The two types of worksheet formats, *general* and *specific*, are explained in the **`%format_&wsformat`** subsection.
[8] Watts (2004, 2005) has an alternative way of doing this, using `%unquote` and `%bquote`.

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[9] 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 (1/29/08), 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**.

REFERENCES

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

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. (2003), 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`

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`

[9] or something very much like it, as in the *Accounting* format string as discussed in the **`%makeExcelFormats`** subsection.

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`