Paper 089-30

Using Single-Purpose SAS® Macros to Format EXCEL Spreadsheets with DDE

Perry Watts, Fox Chase Cancer Center, Philadelphia, PA

Abstract

While it is relatively easy to transfer data between SAS and EXCEL, it is a different story when it comes to formatting spreadsheets to meet project specifications. Many workbooks with multiple spreadsheets may be required to complete an assignment, and formatting requirements for each sheet may be idiosyncratic. Manual formatting is onerous and time-consuming. Errors frequently occur and frustration increases when they too have to be corrected. The single-purpose macros described in this paper enable the SAS user to accurately format spreadsheets with ease. More specifically, positional parameters in X4ML (EXCEL version 4 Macro Language) are replaced with keyword parameters in the SAS macro language. X4ML, a predecessor to VBA, (Visual Basic for Applications) supports DDE (Dynamic Data Exchange), a legacy communications protocol accessible in BASE SAS.

Besides the SAS macros, X4ML syntax is explained in detail, and instructions are provided for communicating with EXCEL via DDE. Additionally, an ODS-HTML Macro Library Descriptor is used for summarizing the collection of SAS macros stored in a single library. The goal of the paper is to enable SAS users to develop customized macro libraries for SAS-to-EXCEL communications that meet specific business needs.

Introduction: Why use DDE for Communicating with EXCEL

Dynamic Data Exchange (DDE) is defined as a "mechanism that permits two applications to talk to each other by continuously and automatically exchanging data" **[7]Microsoft, p.209**. In this paper, the communicators are Microsoft EXCEL playing the role of *server*, and SAS which operates as a *client*. SAS communicates with EXCEL by inserting EXCEL 4 macro language (X4ML) function calls into DDE commands. This way more than 400 EXCEL functions can be accessed from inside SAS to automate the production of customized workbooks.

Unfortunately for SAS users, Microsoft switched its support in the early 90's from X4ML to Visual Basic for Applications (VBA). VBA uses Object Linking and Embedding (OLE) instead of DDE for inter-application communications. Despite the switch, DDE-X4ML has continued to retain its appeal for the following reasons:

1) Only BASE SAS is needed, and DDE commands work equally well in any version of SAS. On the other hand, SAS/AF with SAS Component Language (SCL) is required for a SAS-to-EXCEL link via OLE, and the ODS-XML connection to EXCEL works best with Version 9 SAS software. Furthermore, SAS/Connect software is not required for DDE communications.
2) X4ML is modular. Each function completes a specific task. The programmer only needs to know X4ML function names, purposes and parameter definitions. Function internals are well hidden, and the familiar data _NULL_ step is used for function call management. In contrast, modularity and information-hiding are absent in ODS-HTML where modifications are applied directly to a template or template component, not to a template interface.
3) Bodt points out that direct data transfers between applications leads to greater control over spreadsheet appearance **[1],p.6**. In DDE, both SAS and EXCEL must be opened for communication to take place. The communication is serial in nature meaning that command *order* also has an impact on spreadsheet appearance. Tweaking final EXCEL output is easily accommodated in SAS by rearranging X4ML calls. Such tweaking is not possible in ODS-HTML, because communication is non-interactive. In ODS-HTML, EXCEL needs to be closed in order to receive all directives in a single file transfer from SAS.
4) More user group papers have been written about DDE than any other SAS-to-EXCEL communication method. A partial listing can be found in the reference section. If you are new to DDE, be sure to download Koen Vyverman's tutorial *Creating Custom Excel Workbooks from Base SAS with Dynamic Data Exchange: a Complete Walkthrough* available with test programs at http://www.sas-consultant.com/professional/papers.html.
5) While hard-copy publications about X4ML are out of print, the functions are fully described in MACROFUN.HLP available from http://www.microsoft.com/learning/ (search for MACROFUN.EXE). Installation and usage instructions are provided later in the paper.
6) Finally, X4ML commands can be grouped to form user-friendly single-purpose SAS macros. The macros are stored in a SASAUTOs library together with an ODS-HTML Descriptor that summarizes the macros and lists their parameters. Direct access to MACROFUN.HLP along with the ODS-HTML Descriptor facilitates the construction of customized EXCEL spreadsheets.

A Pictorial Guide for Converting a SAS data set into a Formatted EXCEL Spreadsheet

Two steps are required for converting the example data set, SASHELP.CLASS, into a formatted EXCEL spreadsheet. First, the data are transferred to EXCEL via DDE *triplets*, and then X4ML commands are applied in serial order with a DDE *doublet*, a term coined by Koen Vyverman **[13],p.6**. Labels have been added to SASHELP.CLASS to make the headers more informative in EXCEL, and the term *data* in the example encompasses not only the variables, but the headers and title as well. Figures 1 and 2 below portray the results of the data processing described in this paper.

Figure 1. Two basic steps must be completed to format an EXCEL spreadsheet containing SAS data.

First 7 observations of the labeled SASHELP.Class data set

SASHelp.Class to be Formatted in EXCEL with SAS Macros

Obs	First Name	Sex	Age	Height (inches)	Weight (pounds)
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5

Step #1: Title, labels, variable names and data points are transferred as *data* in DDE *triplets* to EXCEL.

	A	B	C	D	E	F
1	SASHelp.Class Formatted in EXCEL with SAS Macros					
2	Name	Sex	Age	Height	Weight	
3	First Name	Sex	Age	Height (inc	Weight (pounds)	
4	Alfred	M	14	69	112.5	
5	Alice	F	13	56.5	84	
6	Barbara	F	13	65.3	98	
7	Carol	F	14	62.8	102.5	
8	Henry	M	14	63.5	102.5	
9	James	M	12	57.3	83	
10	Jane	F	12	59.8	84.5	

Step #2: The EXCEL spreadsheet is formatted with X4ML commands embedded in DDE *doublets*.

	A	B	C	D	E
1	SASHelp.Class Formatted in EXCEL with SAS Macros				
3	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
5	Alfred	M	14	69.0	112.5
6	Alice	F	13	56.5	084.0
7	Barbara	F	13	65.3	098.0
8	Carol	F	14	62.8	102.5
9	Henry	M	14	63.5	102.5
10	James	M	12	57.3	083.0
11	Jane	F	12	59.8	084.5

Step #2 in Figure 1 above formats the spreadsheet by invoking SAS macros containing data _NULL_ steps with multiple, related X4ML commands that accomplish a single task. First, the title is centered and emboldened. Next labels are displayed in bold Italic font. Column width then is set so that all text is visible, and data values are variously justified to reflect data *type*. WEIGHT appears with a leading zero for alignment purposes. Lastly variable NAMEs in row #2 are hidden, and a thin gray line is inserted to emphasize the headers. Hiding and inserting rows must succeed earlier formatting activities, or the cell SELECT statements in the macros will not work properly.

In Figure 2, all formatting is achieved with an application of a single X4ML command that contains an EXCEL auto-format. If an auto-format such as the one depicted in Figure 2 meets your needs, it is not necessary to read this paper in depth. Instead, read about the DDE *triplet* and *doublet* and then take a look at the complete set of X4ML callable auto-formats in an associated file attachment on the NESUG 2004 CD.

If, however, you require the customized output of Figure 1, you need to understand what is in the paper. Key to that understanding is an appreciation for the concept of *order* described in point #3 of the introduction. Order is what makes Step #2 in Figure 1 work as intended. Again, hiding the second row and inserting a thin gray line between the headers and data must come *after* other formatting tasks have been completed. LILO (**L**ast **I**n : **L**ast **O**ut), at work here, is perhaps more clearly illustrated in the overlapping concentric circles shown at the right. The black circle is partially hidden by the plum circle, but the only fully visible circle is the blue one because it is drawn last.

Figure 2. Applying an EXCEL *auto-format* to an unformatted spreadsheet.

Step #1: Title, labels, variable names and data points are transferred as *data* in DDE triplets to EXCEL.

	A	B	C	D	E	F
1	SASHelp.Class Formatted in EXCEL with SAS Macros					
2	Name	Sex	Age	Height	Weight	
3	First Name	Sex	Age	Height (inc	Weight (pounds)	
4	Alfred	M	14	69	112.5	
5	Alice	F	13	56.5	84	
6	Barbara	F	13	65.3	98	
7	Carol	F	14	62.8	102.5	
8	Henry	M	14	63.5	102.5	
9	James	M	12	57.3	83	
10	Jane	F	12	59.8	84.5	

Step #2: The EXCEL spreadsheet is quickly formatted with an EXCEL auto-format.

	A	B	C	D	E
1	*SASHelp.Class in EXCEL AUTO Classic 3 Format*				
2	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
3	Alfred	M	14	69	112.5
4	Alice	F	13	56.5	84
5	Barbara	F	13	65.3	98
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	James	M	12	57.3	83
9	Jane	F	12	59.8	84.5

Transferring SAS data to EXCEL with a DDE *Triplet*

The DDE *triplet* is used to identify a spreadsheet region that serves either as data source or target destination for read|write operations in SAS. The three pieces of the *triplet* arranged hierarchically are <Server Name> <Topic> and <Item> or specifically for EXCEL: <Excel> <Spreadsheet ID> <Rectangular Cell Range>. The DDE *triplet* is used in three FILENAME statements below to transfer SAS data to EXCEL:

```
/* WRITE TITLE */
filename title dde "excel|[xlsAF3.xls]sheet1!r1c1:r1c&nvars" notab; ❶

/* WRITE ROW HEADERS */
filename HEADERS dde "excel|[xlsAF3.xls]sheet1!r2c1:r2c&nvars" notab; ❷

/* WRITE BODY */
 filename ddedata dde
 "excel|[xlsAF3.xls]sheet1!r&startRow.c1:r&nrows.c&nvars" notab;
```

❶ <Server Name>, <Topic> and <Item> are color coded in yellow, green and cyan respectively. In regards to *topic* (green area), only enough information for identifying a specific worksheet is actually required. For example, If a single workbook is opened on the desktop, `filename title dde "excel|sheet1!..."` will suffice. However, full path names inside the square brackets are recommended for robust applications that are self-documenting. Reserve `excel|sheet1!` for test programs using the default workbook (Book1).

Item (cyan area) is specified by identifying upper-left and lower-right corners of a rectangular cell region. The SAS macro variable NVARS is easily accommodated. See also **[1]Bodt,p.19, [10]SAS Institute,cha.11, [11]Schreier,p.5,** and **[13]Vyverman,p.6** for additional details about the DDE triplet.

❷NOTAB prevents TAB delimiters from being automatically inserted between each word in an output string. Otherwise 'First Name' would occupy two adjacent cells. For additional details see **[13]Vyverman,p.4,** and **[11]Schreier,p.4-5.**

While the DDE triplet can be used to read the contents of an EXCEL spreadsheet into SAS, this paper focuses entirely on the WRITE operation. The actual data transfer takes place in a data _NULL_ step:

```
data _null_;
  set class;
  file ddedata; ❶
  put Name '09'x  Sex '09'x  Age '09'x  Height '09'x  Weight;❷
run;
```

❶ **ddedata** was defined above as a DDE triplet in the FILENAME statement.

❷ '09'x is the ASCII hexadecimal code for the invisible TAB character. TABs here are being explicitly re-inserted into the source code so that variable values will occupy adjacent cells in the spreadsheet. This way, a new student named "Mary Anne", won't be misaligned as she is in Figure 3:

Figure 3. The entry for Mary Anne is misaligned when **NOTAB** and TAB delimiters (**'09'x**) are removed from the source code.

```
data test;
 length name $10 sex $1;
 input name & sex age height weight;
 cards;
  Alfred     M  14   69.0  112.5
  Mary Anne  F  12   66.5   94.0
run;
…
/* WRITE BODY */
filename ddedata dde "excel|sheet1!r3c1:r4c5";
data _null_;
 set test;
 file ddedata;
 put name sex age height weight;
run;
```

	A	B	C	D	E
1	Mary Anne goes to CLASS				
2	*name*	*sex*	*age*	*height*	*weight*
3	Alfred	M	14	69	112.5
4	Mary	Anne	F	12	66.5

The PUT statement in Figure 3 can be replaced with an invocation of the SAS macro **WriteDStoXLS** (write-data set-to-EXCEL). The correction is shown in Figure 4.

Figure 4. Cell mapping is corrected in the single-purpose macro function **WriteDStoXLS**. The macro invocation and its return value are highlighted in the two side-by-side panels. **WriteDStoXLS** is listed in the file attachment associated with this paper.

```
/* WRITE BODY */
filename ddedata dde "excel|sheet1!r3c1:r4c5"
  NOTAB;

data _null_;
 set test;
 file ddedata;
 put %WriteDStoXLS(dsName=test)
run;
```

```
/* WRITE BODY */
filename ddedata dde "excel|sheet1!r3c1:r4c5"
  NOTAB;

data _null_;
 set test;
 file ddedata;
 put NAME +(-1) '09'x SEX +(-1) '09'x AGE +(-1)
'09'x  HEIGHT +(-1) '09'x WEIGHT;
run;
```

NOTAB is assumed to be in effect here, with variable names and tab delimiters `('09'x)` being automatically supplied by the macro. Trailing blanks generated by using list-style output in a data _NULL_ step are also removed. Inserting `+(-1)` as a pointer control *before* the tab delimiter in the PUT statement gets rid of them. It is important to get rid of trailing blanks in spreadsheet entries, because'Jane' and 'Jane●' have different values in EXCEL. Koen Vyverman mentions that EXCEL functions do not discount blanks, so their presence can produce unwanted results for the user who works with SAS data in EXCEL **[12],p.3**.

Because communication is ongoing between SAS and EXCEL in a DDE protocol, spreadsheets will be updated each time a SAS program is modified and re-executed. LILO, still in effect here, is responsible for the erroneous duplications shown in Figure 5. A larger data set has been replaced with a smaller one leaving the last two records untouched in subsequent runs. Think of the analogy to the concentric circles. Figuratively speaking, James and Jane would remain untouched at the edge of the largest circle. To correct overlay problems, simply reinitialize spreadsheets between program runs with the CLEAR X4ML command.

Figure 5. Side-effects are possible when DDE is used to update existing EXCEL spreadsheets with data sets having different dimensions. Spreadsheet cells need to be CLEARED between program runs.

Run #1: The original class with 7 students.

	A	B	C	D	E
1	BigClass contains 7 students				
2	***name***	***sex***	***age***	***height***	***weight***
3	Alfred	M	14	69	112.5
4	Alice	F	13	56.5	84
5	Barbara	F	13	65.3	98
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	James	M	12	57.3	83
9	Jane	F	12	59.8	84.5

Run #2: Alfred and Alice dropped out, but James and Jane are now listed twice.

	A	B	C	D	E
1	SmallClass should contain 5 students				
2	***name***	***sex***	***age***	***height***	***weight***
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	James	M	12	57.3	83
9	Jane	F	12	59.8	84.5

Formatting EXCEL spreadsheets with the DDE *Doublet*

The DDE *doublet* retains the <Server Name> and <Topic> components. The meaning for *topic* however changes from the spreadsheet ID, which varies, to the fixed assignment of SYSTEM. Schreier points out that the SYSTEM *topic* lets the client application send commands that are carried out in EXCEL **[11],p.6**. Therefore, communication is unidirectional: from SAS to EXCEL. Also missing from the *doublet* definition is the *item* that defines target cell ranges. This component is transferred to an X4ML function such as SELECT. Given these modifications, the *doublet* only has to be defined once in a SAS session:

```
/* DEFINE SYSTEM CONNECTION */
filename DDEcmds dde 'excel|system';
```

Data _NULL_ also works differently. With the *doublet*, X4ML commands are immediately passed through to EXCEL whereas data buffering is part of the processing associated with the *triplet* **[13]Vyverman,p.6.** In other words, the *triplet* is compiled and the *doublet* is interpreted. Later on, you will see how interpreted code is easier to debug.

X4ML: EXCEL version 4 Macro Language

X4ML operates like any macro in that it generates language instructions as output. The output from a SAS macro, for example, is SAS source code, and a keyboard macro in the SAS Enhanced Editor stores entered keystrokes. Similarly, X4ML works behind the scenes to mimic user selections inside the EXCEL workspace. This connection would be much more obvious if EXCEL looked the same way it did 12 years ago. Figure 6 shows the correspondence between the Format Border dialog box and the X4ML BORDER command.

Accessing Macrofun.hlp

Because manuals are out of print, access to MACROFUN.HLP is essential for working with X4ML. Otherwise, you are restricted to the command sets referenced in SAS User group papers. As mentioned previously, just go to http://www.microsoft.com/learning/ and search for MACROFUN.EXE. Download the executable and write MACROFUN.HLP to any subdirectory other than where EXCEL is stored. Do not associate the HELP file with EXCEL. Access is improved by placing a shortcut to MACROFUN.HLP on the desktop:

Navigating in Macrofun.hlp

Commands are arranged both alphabetically and by category in the *Contents* window of MACROFUN.HLP. If you cannot find what you are looking for, go into EXCEL to capture keywords by manually performing the operation you are trying to automate. Then return to MACROFUN.HLP and perform a *search* on the keywords.

X4ML Syntax Explained by Example

As with the DDE *triplet*, a data _NULL_ step is used to communicate with EXCEL. This time FILE points to the *doublet fileref.* Individual X4ML commands are embedded SAS PUT statements. The source code and results for exercising the BORDER command described in Figure 6 on select cells in a spreadsheet is displayed in Figure 7.

Figure 6. The X4ML Border command copied from MACROFUN.HLP reflects the structure of the Border dialog box. However, there is no consistency between line type number (0-7) and the placement of the style inserts in the dialog box.

Border

Macro Sheets Only

Equivalent to choosing the Border tab in the Format Cells dialog box, which appears when you choose the Cells command from the Format menu. Adds a border to the selected cell or range of cells.

Syntax

BORDER(outline, left, right, top, bottom, shade, outline_color, left_color, right_color, top_color, bottom_color)

BORDER?(outline, left, right, top, bottom, shade, outline_color, left_color, right_color, top_color, bottom_color)

Outline, left, right, top, and bottom are numbers from 0 to 7 corresponding to the line styles in the Border dialog box, as shown in the following table.

Argument	Line type
0	No border
1	Thin line
2	Medium line
3	Dashed line
4	Dotted line
5	Thick line
6	Double line
7	Hairline

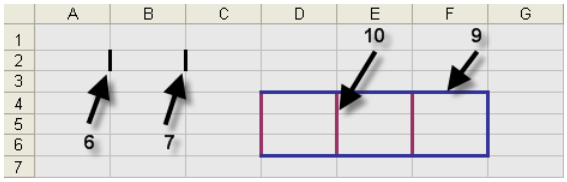
Shade corresponds to the Shade check box in the Border dialog box of Microsoft Excel version 4.0. This argument is included for compatibility only.

Figure 7. The BORDER command in Figure 6 is exercised below. SAS code is indigo; X4ML is plum. X4ML uses positional parameters in function calls. See numeric annotations below for contextual interpretations of consecutive commas.

●: SAS ●: X4ML

```
 1  filename DDEcmds dde ' excel|system ';
 2  data _null_;
 3   file DDECmds;
 4   put '[workbook.activate("sheet1")]'; ▲
 5   put '[select("r2c2")]';
 6   put '[border(,5)]'; ❻
 7   put '[border(,,5)]'; ❼
 8   put '[select("r4c4:r6c6")]';
 9   put '[border(5,,,,,,5)]'; ❾
10   put '[border(,5,,,,,,3)]'; ❿
11  run;
```

Numeric annotations reference line numbers in the source code at the left.



▲Lines 4 and 5 serve the same purpose as *item* in the DDE triplet.

❻ A thick individual *left* border line is placed at r2c2. The preceding comma is a place holder for *outline* which is turned off. Parameters following *left* are set to their defaults, meaning that *left_color* is set to black and no additional lines are drawn. *Shade* is ignored.

❼ A thick individual *right* border line is placed at r2c2. Again, DDE-X4ML overwrites. It doesn't re-write. Therefore, the left border that was assigned in the previous command remains intact.

❾ A thick indigo outline is placed at r4c4:r6c6. Note that *left, right, top, bottom,* and *shade* are set to missing. Trailing commas do not have to be displayed.

❿ Thick plum left border lines are placed at r4c4:r6c6. Since *left* comes after *outline* in the command order, plum replaces indigo in the outside border.

A combined review of Figures 6 and 7 demonstrates that the meaning of consecutive commas in the parameter list for the BORDER command cannot be taken out of context. They are not merely placeholders, but send various signals to EXCEL such as "keep the current setting", "use the default", or "disregard this obsolete parameter". Additional language conventions include:

1) Spaces are not permitted in an X4ML command.

2) Target worksheets and cells must be explicitly SELECTED, or the command will fail to generate any output.

3) SELECT statements must use row-column notation ("r1c1:r2c2"). Column-row (!$A$1:$B$2) only works within the confines of an EXCEL macro, not in DDE X4ML communications.

4) If MACROFUN.HLP stipulates that a parameter is TEXT, then the parameter value must be surrounded by double quotes. WORKBOOK.ACTIVATE and SELECT in Figure 7 contain TEXT parameters.

Automation in EXCEL relies upon Looping in SAS

While X4ML is not really needed for inserting a few borders into a spreadsheet, it is essential when repetitive processing is required for formatting a workbook. In this section, three methods are described for highlighting every other student's entry in the SASHELP.CLASS data set. The first two methods use a DO-LOOP to handle repetitive processing whereas an EXCEL auto-format followed by several corrective X4ML commands is featured in the third method. The output from all three methods is displayed in Figure 8, and the syntax for the PATTERNS function that supplies background colors is taken directly out of MACROFUN.HLP:

PATTERNS(apattern, afore, aback, newui)

Apattern	is a number corresponding to the area patterns in the Patterns tab of the Format Cells or Format Object dialog box. (1=default=solid)
Afore	is a number from 1 to 56 corresponding to the 56 area foreground colors in the Patterns tab of the Format Cells dialog box. (blank=no foreground color)
Aback	is a number from 1 to 56 corresponding to the 56 area background colors in the Patterns tab of the Format Cells dialog box.
Newui	is a logical value that specifies whether to use the foreground, background, and patterns of Microsoft Excel 5.0. If TRUE or omitted, the colors and patterns of Microsoft Excel 5.0 will be used. If FALSE, the colors and patterns of Microsoft Excel 4.0 will be used.

Regardless of the setting for *newui*, there is no correspondence between color numbers (1-56) and the order of colors displayed in the PATTERNS tab of the FORMAT CELLS dialog box in recent versions of EXCEL. A correct mapping for the X4ML color numbers can be found in **ExcelColorMap.xls** in the file attachment on the NESUG CD.

When an explicit DO-LOOP or an implicit reiteration with a SET statement is used in a data _NULL_ step, the boundaries between the SAS and X4ML programming languages become blurred. The blurring is especially pronounced in method #1 where *words* (demarcated by spaces) contain both SAS and X4ML components. The language mixture can be observed in the mixed-color underlines.

- **Method #1: Do-Loop generated in SAS**

```
data _null_; ❶
 file DDECmds;
 put '[workbook.activate("sheet1")]';
 DO row = 3 to 9 by 2;
   put '[select("R' row +(-1) 'C1:R' row +(-1) 'C5")]'; ❷
   put '[patterns(1,,15)]'; ❸
 END;
run;
```

❶ Again indigo (●) represents SAS code and plum (●) is reserved for the X4ML portion of the line command.

❷ Constant text and variables are intermingled. The pointer control +(-1) is required here, since spaces are not allowed in X4ML. Underlines identify *words* that separate constant text from variables.

❸ 15 is light gray.

- **Method #2: Do-Loop generated in the SAS Macro Language**

```
data _null_;
 file DDECmds;
 put %unquote(%bquote('[workbook.activate("&sheetName")]')); ❶
 %DO row = 3 %to 9 %by 2;
   put %unquote(%bquote('[select("r&row.c1:r&row.c5")]')); ❷
   put '[patterns(1,,15)]';
 %END;
run;
```

❶ Sheet name can be assigned at run time. The single quotes need to be QUOTED (hidden) so that the macro variable SHEETNAME will resolve. After resolution, the single quotes then need to be UNQUOTED (revealed) so that SAS knows that the text that follows is a character string. See **[16]Whitlock** for a discussion about macro quoting.

❷ With macro variables, the string is not parsed in any way. There is no need for +(-1) here.

- **Method #3: Using EXCEL auto-format #10 for Alternate Row Highlighting**

```
data _null_;
  file ddecmds;
  put '[workbook.activate("sheet3")]';
  put '[select("r1c1:r9c5")]'; ❶
  put "[format.auto(10)]";
run;
```

❶ The SELECT statement for the auto-format eliminates the need for a LOOP. However, the title is positioned incorrectly over Column #1. Additional X4ML commands shown below correct the problem.

```
data _null_;
  file ddecmds;
  put '[workbook.activate("sheet4")]';
  put '[select("r1c1:r9c5")]';
  put "[format.auto(10)]";
  put '[select("r1c1:r1c5")]'; ❶
  put '[format.font("Arial",10,TRUE,TRUE,false,false,9)]';
  put '[alignment(7)]';                        *-- 7=CENTER ACROSS SELECTION;
  put '[column.width(0,"c1:C5",false,3)]'; *-- AUTO FIT COLUMN WIDTH;
run;
```

❶ The modifications to the title colored in plum prevail, because they are applied *after* the auto-format. The syntax for FORMAT.FONT, ALIGNMENT, and COLUMN.WIDTH can be found in MACROFUN.HLP. Again, customization requires knowledge of X4ML.

Figure 8. Three methods for Alternate Row Highlighting.

Methods #1,#2: Using a SAS DO-LOOP or a SAS macro language %DO-LOOP produces the same output.

	A	B	C	D	E
1	Alternate Row Highlighting for SASHELP.CLASS				
2	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
3	Alfred	M	14	69.0	112.5
4	Alice	F	13	56.5	084.0
5	Barbara	F	13	65.3	098.0
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	James	M	12	57.3	083.0
9	Jane	F	12	59.8	084.5

Method #3: Using EXCEL's auto-format #10 incorrectly positions the title over column #1.

	A	B	C	D	E
1	*Alternate Row Highlighting for SASHELP.CLASS*				
2	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
3	Alfred	M	14	69	112.5
4	Alice	F	13	56.5	84
5	Barbara	F	13	65.3	98
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	James	M	12	57.3	83
9	Jane	F	12	59.8	84.5

Method #3 Corrected: Corrective X4ML commands are added to the DATA _NULL_ step to fix the title.

	A	B	C	D	E
1	*Alternate Row Highlighting for SASHELP.CLASS*				
2	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
3	Alfred	M	14	69	112.5
4	Alice	F	13	56.5	84
5	Barbara	F	13	65.3	98
6	Carol	F	14	62.8	102.5
7	Henry	M	14	63.5	102.5
8	James	M	12	57.3	83
9	Jane	F	12	59.8	84.5

Looping can also be used to create non-rectangular EXCEL spreadsheets. In *Give Your Clients What They Want: Fill Report Tables with the SAS® System and DDE*, William C. Murphy shows how to insert values into a non-rectangular spreadsheet template **[8]**. Essentially, he assigns data values to unordered, individual cells by looping through a SAS data set in a data _NULL_ step. A similar approach is taken in *Highlighting Inconsistent Record Entries in EXCEL: Possible with SAS® ODS, Optimized in Microsoft® DDE* **[14],Watts**. The optimized source code for color assignment along with the corresponding output is shown in Figure 9. No row or column-derived patterns from the BGCOLORS data set are evident in the output. Hence a rectangular table is superimposed over non-rectangular background colors.

Figure 9. Background colors in the Mouse Data are non-rectangular. Colors are assigned one cell at a time by processing BGCOLORS in an implicit LOOP.

Method #1 looping from Figure 8 works best with a SET statement.

```
filename ddeCMDS dde 'excel|system';
data _null_;
  file ddeCmds;
  set BGColors end=last;❶
  if _n_ eq 1 then do;
   put '[workbook.activate("sheet1")]';
   put %unquote(%bquote(
    '[select("r&stRow.c1:r&nrows.c&nvar")]'));❷
   put '[clear(1)]';
   put '[border(,1,1,1,1,,,15,15,15,15)]';
  end;
  put '[select("' rowCol +(-1) '")]';❸
  put '[patterns(1,0,' color +(-1) ')]';❹
run;
```

Additional details about the output below can be found in *Highlighting Inconsistent Record Entries in EXCEL: Possible with SAS® ODS, Optimized in Microsoft® DDE.*

	A	B	C	D
1	Edit Sheet for Mouse Study			
2	Green=Missing	Yellow=Dates Out of Order	Blue=Outlier	Red=Error
3	*MouseID*	*Age(Days)*	*RxDate #1*	*RxDate #2*
5	1	25	1/7/2004	1/14/2004
6	2	9	1/20/2004	1/24/2004
7	3	-2		1/14/2004
8	4	85	1/20/2004	1/14/2004
9	5	24	1/1/2004	1/7/2004
10	6	19	1/8/2003	1/22/2004
11	7	29	1/7/2004	1/7/2004
12	8	5	1/2/2004	

❶The data set BGCOLORS contains two variables: ROWCOL (row-column or cell address) and COLOR (background color).

❷ From the right panel, **r4c1:r11c4** would be defined as the parameter in the SELECT function with the gray dividing line between headers and data being added later. In the next PUT statement cell contents and formats from previous runs are CLEARED to eliminate the possibility of unwanted side-effects caused by layering output, and finally in the last PUT statement associated with the IF condition, thin gray cell borders are drawn around cells in the data region so that highlighted cells in the region will be in conformance with all cells in the spreadsheet. Otherwise, cell borders would automatically be erased when background colors are inserted.

❸ One cell is selected at a time from the BGCOLORS data set. Note that the command line is separated into *words* representing constant text or SAS variables.

❹ Here is where the background color is actually assigned. There are no LILO conflicts between background color and text displays. Text is always placed over the background color, even when it is assigned earlier in the program.

Simplify X4ML with Single-Purpose SAS Macros

In *Code Complete*, Steve McConnell asserts that the "single most important reason to create a routine is to reduce a program's complexity" **[5],p.74**. Routines with a few well-named parameters can function as "black boxes" where "you know what goes in and you know what comes out, but you don't know what happens inside" **[5],p.116**. The maximum number of parameters is set to about seven, a number that corresponds to the limits of human comprehension according to a reference in McConnell **[5],p.108**. Unfortunately, it is difficult to work with over 400 X4ML functions where the magic number 7 for the maximum number of parameters goes unheeded. The single-purpose SAS macros described in this section reduce complexity by:

1) Replacing positional parameters in X4ML with self-documenting keyword parameters in the SAS macro language.

Parameter values for FORMAT.FONT inside the SAS **FormatTitle** macro

```
put '[format.font("Verdana",12,TRUE,false,false,false,1,false,false)]';
```

are specified by stipulating fontSize in the macro call

```
%formatTitle(fontSize=12, maxCol=5)
```

2) Reducing the number of macro parameters by setting values internally or by assigning defaults to the keyword parameters.

Only MAXCOL has to be specified in **FormatTitle**:

```
%macro formatTitle(cmdFileName=ddeCmds, fontSize=11, sheetName=sheet1, rowHeight=20,
    row=1,minCol=1, maxCol=);
```

All other parameters have default values. Internally, the font face has been set to VERDANA, and the font color is always black (1).

3) Combining X4ML commands inside a SAS macro to accomplish the single task reflected in the macro's name.

This makes for more modular programming. The SAS programmer can arrange macro invocations in the calling program to complete specific tasks. LILO can also be accommodated when the focus of the macro is relatively narrow. Nevertheless, abstraction is present as seen by the fact that **FormatTitle** invokes five X4ML commands:

```
WORKBOOK.ACTIVATE
SELECT
FORMAT.FONT
ALIGNMENT
ROW.HEIGHT
```

4) *Documenting the macros fully and storing them in a SASAUTOS library.*

In addition to the macro's name, purpose and parameter specifications, descriptions for each of the X4ML commands invoked inside the macro are included in the header comments. To learn how to build and use macro libraries see Art Carpenter citations listed in the reference section **[2],[3]**. The macros along with the calling program that uses them are included in the associated file attachment on the NESUG CD.

5) *Listing members of the macro library in an ODS-HTML Macro Library Descriptor. Users can reference the Descriptor as they write programs that call the macros.*

The first page in the ODS-HTML Macro Library Descriptor is shown in Figure 10 below. It lists the macros that are described in the HTML pages that follow. A complete copy of the ODS-HTML Macro Library Descriptor can be found in the associated file attachment on the NESUG CD.

Figure 10. Listing of the 11 Single-Purpose SAS Macros stored in the DDE SASAutos library.

List of Macros in Library: c:\N04\DDEwSASMac\SASauto

Macro	Description
alignWorkSheet	Center adjust and auto-fit selected cells in a spreadsheet.
formatBody	Format body text in a spreadsheet.
formatHeaders	Format Row Headers in a spreadsheet by defining font size, justification, color, and column width.
formatNumber	Format numbers in a column with a "custom" EXCEL format.
formatTitle	Format a spreadsheet title in black Verdana and center it across a selection of multiple columns.
HdrWVblLABEL	Create the text for a header line containing variable LABELS.
HdrWVblNAME	Create the text for a header line containing variable NAMES.
InsertLineDivider	Insert a thin colored line below the headers extending from column #1 to the number of columns in a data set.
OpenExcel	Open EXCEL from inside SAS.
SaveANDClose	Allow a conditional EXCEL file SAVE and CLOSE based on the parameter setting.
WriteDStoXLS	Create a customized PUT statement for writing a SAS data set out to an EXCEL file.

Three Macros Described in Detail

Three macros have been selected for a detailed review. They are **OpenExcel**, **HdrWvblLabel**, and **FormatTitle**. Only the source code for the macros is listed in this section. Complete listings that include header comments can be found in an associated file attachment on the NESUG 2004 CD.

OpenExcel has been selected, because EXCEL must be opened for DDE communications to take place. The macro has been directly copied from Christopher Roper's paper *Intelligently Launching Microsoft Excel from SAS, using SCL functions ported to Base SAS* cited in the reference section **[9]**.

The next macro, **HdrWVblLabel**, passes labels for SAS data set variables to a header row in EXCEL via a DDE *triplet.* This macro is actually a SAS macro *function*. SCL BASE SAS I/O functions are used to poll a SAS data set inside the macro with SYSFUNC. The final macro, **FormatTitle** introduced earlier, is shown in its entirety as a representative of the six macros in the Library that format EXCEL output.

- **Macro: OpenExcel**

```
%macro OpenExcel(cmdfile=ddeCmds); ❶
   data _null_;
    length fid rc start stop time 8;
    fid=fopen("&cmdFile",'s'); ❷
    if (fid le 0) then do;
      rc=system('start excel'); ❸
      start=datetime();
      stop=start+10;
      do while (fid le 0); ❹
        fid=fopen("&cmdFile",'s');
        time=datetime();
        if (time ge stop) then fid=1;
      end;
    end;
    rc=fclose(fid);
   run;
 %mend OpenExcel;
```

❶ Roper's code has been inserted into a SAS macro essentially to reduce the number of lines of code in the SAS calling program **[9]**. There are no X4ML commands in **OpenExcel**, because EXCEL hasn't been opened to receive them. Instead, only SAS or DOS commands are used in the macro.

❷ FOPEN is a SAS SCL function ported into BASE SAS. 'S' stands for sequential-read mode. When EXCEL is fired up, FID switches from 0 to 1. What this means is EXCEL won't be restarted if the calling program is re-executed.

❸ SYSTEM is a SAS function that issues an Operating System command; in this case the DOS START command that finds EXCEL in the Windows registry and opens it.

❹ The DOWHILE loop gives WINDOWS additional time to open EXCEL.

- **Macro: HdrWvblLabel**

```
%macro HdrWVblLABEL(dsName=); ❶
  %local rc nVars MaxLess1 i; ❷
  %let rc=%sysfunc(open(&dsName));
  %let nVars = %sysfunc(attrn(&rc,NVARS));
  %let maxCol = %sysfunc(min(256,&nVars));    ❸
  %let MaxLess1=%eval(&maxCol - 1);
  %do i= 1 %to &MaxLess1; "%sysfunc(varLABEL(&rc,&i))" '09'x %end;
      "%sysfunc(varLABEL(&rc,&maxCol))"; ❹
  %let rc=%sysfunc(close(&rc));
%mend HdrWVblLABEL;
```

❶ **HdrWVblLabel** is actually a macro function. Art Carpenter lists the following characteristics of macro functions in his SUGI 27 paper entitled *Macro Functions: How to Make Them - How to Use Them* **[4]**:

- All statements in the macro must be macro statements.
- The macro should create NO macro variables other than those that are local to that macro.
- The macro should resolve to the value that is to be returned.

Note that every statement in the macro begins with a percent (%).

❷Local macro declarations are declared in the macro function.

❸The macro variable NVARS, obtained by exercising the SCL BASE SAS I/O function ATTRN, is adjusted to fit the dimensions of an EXCEL spreadsheet. If NVARS is greater than 256, only the first 256 columns will be labeled.

❹ Here is where the macro resolves to the value that is returned. A CR/LF has been inserted for readability, but the actual command must be confined to a single line so that the generated SAS PUT statement is displayed as intended. Here is a brief example that shows how **HdrWvblLabel** is invoked in a calling program:

```
filename HEADERS dde "excel|[&WkBook]sheet1!r3c1:r3c&nvar" notab;
data _null_;
  file HEADERS;
  put %HdrWvblLabel(dsName=MyData)
run;
```

The return value is placed directly over the function call.

- **Macro: FormatTitle**

```
%macro formatTitle(cmdFileName=ddeCmds, fontSize=11, sheetName=sheet1,
                   rowHeight=20, row=1, minCol=1, maxCol=); ❶
  data _null_;
    file &CmdFileName;
    put %unquote(%bquote('[workbook.activate("&sheetName")]'));
    put %unquote(%bquote('[select("r&row.c&minCol.:r&row.c&maxCol")]'));
    put %unquote(%bquote('[format.font("Verdana",&fontSize.,TRUE,false,false,
                          false,1,false,false)]'));
    put '[alignment(7)]';   ❷
    put '[alignment(,,2)]'; ❸
    put %unquote(%bquote('[row.height(&rowHeight.,"r&row",false)]')); ❹
  run;
%mend formatTitle;
```

❶ This macro motivated the development of the ODS-HTML Macro Library Descriptor. While the number of parameters has been reduced in the single-purpose macros, it is easy to forget the default values. Title formatting intended for *Sheet4* could go to *Sheet1*, for example. Or it is possible that a workbook would not even have a sheet named *Sheet1*.

❷ '7' centers the title across the column selection (from MINCOL to MAXCOL).

❸ '2' in the second position vertically centers the title.

❹ Row height is fixed in points.

The ODS-HTML Macro Library Descriptor

While default values assigned to parameters can save the SAS programmer keystrokes, if forgotten they can also be the source of nasty bugs. Therefore, the ODS-HTML Macro Library Descriptor has been developed to provide the user with a convenient way to obtain information about all the macros stored in a single library. In Figure 11 a standardized SAS header derived from a keyboard macro in the SAS Enhanced Editor is filled in with details about **InsertLineDivider.sas**. Information from the header then transferred to the ODS-HTML Macro Library Descriptor shown in Figure 12.

Figure 11. Standardized program headers provide details about the macros listed in the ODS-HTML Library Descriptor.

```
/*  ------------------------------------------------------------------------------------
    Program  :  InsertLineDivider.sas

    Author   :  Perry Watts
    Date     :  30May2004 16:12

    Path     :  c:\N04\DDEwSASmac\SASAuto

    Purpose  :  Insert a thin colored line below the headers extending from
                 column #1 to the number of columns in a data set.

    Parms    :  Name             Description                                    Default
                ---------------  ---------------------------------------------  -------
                CMDFILENAME      DDE Doublet file name                          ddeCmds
                SHEETNAME        Sheet Name                                     Sheet1
                COLOR            Background color.                              15(gray)
                ROW              Row where line appears                         NONE
                MAXCOL           Max Column#                                    NONE

    Notes    :  *Fixed Features:
                --Line Height is set to 5 points.
                *Invoke this macro at end of calling program after the
                 sheets have been loaded and formatted.

    DDE Cmds :  WORKBOOK.ACTIVATE(sheet_name)
                SELECT(selection, active_cell)
                INSERT(shift_num){shift_num ignored when whole row is being
                 inserted.}
                PATTERNS(apattern, afore, aback, newui)
                ROW.HEIGHT(height_num, reference, standard_height, type_num)

    Example  :  %insertLineDivider(row=4, sheetName=sheet1, maxCol=9)
    ------------------------------------------------------------------------------------  */
```

Figure 12. A subset of the information from the program header is used to populate the ODS-HTML Macro Library Descriptor. Note that the *name*, *purpose* and *parameter list* are prominently featured in the Descriptor. *Author*, *date*, and *path* have been removed, since they provide no guidelines for working with the macros.

Macro: InsertLineDivider

Purpose	Insert a thin colored line below the headers extending from column #1 to the number of columns in a data set.	
Parms	***Description***	***Default***
CMDFILENAME	DDE Doublet file name	ddeCmds
SHEETNAME	Sheet Name	Sheet1
COLOR	Background color.	15(gray)
ROW	Row where line appears	NONE
MAXCOL	Max Column#	NONE
Notes	*Fixed Features:	
	--Line Height is set to 5 points.	
	*Invoke this macro at end of calling program after the sheets have been loaded and formatted.	
DDE Cmds	WORKBOOK.ACTIVATE(sheet name)	
	SELECT(selection, active cell)	
	INSERT(shift num){shift num ignored when whole row is being inserted.}	
	PATTERNS(apattern, afore, aback, newui)	
	ROW.HEIGHT(height num, reference, standard height, type num)	
Example	%insertLineDivider(row=4, sheetName=sheet1, maxCol=9)	

Detailed instructions for constructing the ODS-HTML Descriptor will be covered in a future publication. For now, the complete listing of the Descriptor for the single-purpose macros can be found in the associated attachment on the NESUG 2004 CD.

Calling the Macros from an Application

Selected macro calls from the calling program, **N04CallingPgm.sas,** are listed in Figure 13. The complete program can also be found in the associated attachment on the NESUG 2004 CD. Note that the source code becomes compact, ordered, modular and self-documenting when the single-purpose SAS macros are added to the mix. Given the macro NAMES, some of the comments are even redundant! Also with the formatted spreadsheet, a manager can look at labeled columns, whereas a programmer can readily access variable NAMES by simply un-hiding row #2.

Figure 13. The calling program, **N04CallingPgm.sas**, uses single-purpose macros (●) to format a spreadsheet. Note how the X4ML customizations follow the macro invocations. The formatted spreadsheet is inserted lower down in the figure.

```
/* FOR ACCESS TO THE SINGLE PURPOSE MACROS */
options sasautos=(sasautos 'c:\N04\DDEwSASMac\sasauto');

%OpenExcel

/* WRITE BODY */
filename ddedata dde
"excel|[SASHelpClass.xls]sheet1!r&startRow.c1:r&nrows.c&nvars" notab;
data _null_;
  set class;
  file ddedata;
  put %WriteDStoXLS(dsName=class)
run;

/* WRITE TITLE */
filename title dde "excel|[SASHelpClass.xls]sheet1!r1c1" notab;
data _null_;
  file title;
  put "SASHelp.Class Formatted in EXCEL with SAS Macros";
run;

/* WRITE HEADERS: VARIABLE NAMES AND LABELS */
filename HEADERS dde "excel|[SASHelpClass.xls]sheet1!r2c1:r2c&nvars" notab;
data _null_;
  file HEADERS;
  put %HdrWvblName(dsName=class)
run;
filename HEADERS dde "excel|[SASHelpClass.xls]sheet1!r3c1:r3c&nvars" notab;
data _null_;
  file HEADERS;
  put %HdrWvblLabel(dsName=class)
run;

/* FORMAT TITLE*/
%formatTitle(MaxCol=&nVars, Fontsize=10, Row=1, RowHeight=17)

/* FORMAT HEADERS */
%formatHeaders(fontSize=9, Row=3, MaxCol=&nVars)

/* FORMAT BODY -- VERDANNA, REGULAR FONT*/
%formatBody(CmdFileName=ddeCmds, fontSize=9, MinRow=&startRow,
        Maxrow=&nrows, MinCol=1, MaxCol=&nVars)

/* CENTER WORKSHEET CELL TEXT VERTICALLY AND HORIZONTALLY*/
%alignWorkSheet(maxrow=&nrows, maxCol=&nVars)

/* BUT LEFT JUSTIFY FIRST NAMES */
data _null_;
  file ddeCmds;
  put %unquote(%bquote('[select("r4c1:r&nrows.c1")]'));
  put '[alignment(2)]';
run;

/* FORMAT HEIGHT */
%formatNumber(fmt=%str(#0.0), mincol=c4, maxCol=c4)

/* FORMAT WEIGHT */
%formatNumber(fmt=%str(#000.0),mincol=c5, maxCol=c5)

/* INSERT THIN GRAY LINE UNDER THE HEADERS. */
%insertLineDivider(row=4, maxCol=&nvars)

/* HIDE ROW #2 -- VARIABLE NAMES */
data _null_;
  file ddeCmds;
  put '[row.height(0,"R2",FALSE,1)]';
run;

%saveAndClose(yVsN=&doIt);
```

	A	B	C	D	E
1	SASHelp.Class Formatted in EXCEL with SAS Macros				
3	*First Name*	*Sex*	*Age*	*Height (inches)*	*Weight (pounds)*
5	Alfred	M	14	69.0	112.5
6	Alice	F	13	56.5	084.0
7	Barbara	F	13	65.3	098.0
8	Carol	F	14	62.8	102.5
9	Henry	M	14	63.5	102.5
10	James	M	12	57.3	083.0
11	Jane	F	12	59.8	084.5

Summary and Conclusions

Single-purpose macros for formatting EXCEL spreadsheets have been described in this paper. Their description was preceded by discussions about the DDE *triplet* and *doublet* and with a more in-depth look at X4ML commands that are used for formatting EXCEL spreadsheets from inside SAS.

The macros exemplify modularity which reduces complexity by hiding internal X4ML code from the applications programmer. Because of their modularity, the macros can be easily modified to meet specific business requirements. Their relatively narrow focus also promotes re-use and enables adherence to the principle of LILO, discussed at length in the paper.

Copyright Statement

Your Turn: Working with associated files on the NESUG 2004 CD (also available by request):

What's There

1)The calling program: **N04CallingPgm.sas**

2)Contents of SASAUTO subdirectory:
  The 11 single-purpose macros
  HTML files that make up the Macro Descriptor:
    ClickOn.HTML
    Contents.HTML
    SASauto.HTML

3)Contents of the XLS subdirectory:

ExcelAutoInX4ML.XLS
  18 EXCEL tables containing row and column totals are displayed remotely on separate spreadsheets by executing X4ML auto-format commands. Then they are duplicated elsewhere on the worksheet with a reapplication of the corresponding auto-format in EXCEL 2002. Spreadsheet tabs are labeled with the auto-format's NAME.

XLSAutoClass.XLS:
  Separate spreadsheets are used to show SASHELP.CLASS data set in each of the 18 different X4ML auto-formats.

EXCELColorMap.XLS:
  The 56 X4ML color numbers are mapped to their actual colors. The color list is compared to the current set of colors available in EXCEL 2002.

Instructions for Set-up

1)Place MacroFun.Hlp on your desktop. Downloading and installation instructions are provided in the paper.

2)Store the HTML and the 11 single-purpose macros together on the same user-defined SASAUTOs subdirectory. Press **ClickOn.HTML** and minimize the file. Compile (run) the 11 macro programs.

3)Go into **N04CallingPgm.sas** and change the path to the user-defined SASAUTOs subdirectory. Also adjust the full name of the output XLS file to reflect your directory structure.

Running N04CallingPgm.sas

1)Run the program incrementally module by module, checking how the output EXCEL spreadsheet changes as you go. The two screen snapshots in Figure 1 were captured from such an incremental run. If you follow this procedure, you will see how X4ML works, and if there is trouble, you will be more able to pinpoint it if you go through the program, step-by-step.

2)Re-route the output from a Data _NULL_ step by switching from the DDE fileref to PRINT. This way you will observe the commands that are actually passed to EXCEL. This is helpful when your commands contain SAS-macro variables that have to be resolved.

3)Observe the LOG after each incremental run. Because the doublet is interpreted, you will get a line count for the number of commands that successfully made it through to EXCEL. If that count *m* is less than the total number of lines in the data _NULL_ step *n*, then an X4ML error will be found in line *m+1*.

4)If SAS completely freezes up, go into EXCEL. The problem probably can be resolved over there.

5)All files are offered on an 'as-is' basis.

Questions to Consider

1)How would you revise the macro **FormatTitle** so that it could print titles in any color? Could you move the WORKBOOK.ACTIVATE and SELECT commands? Could the two ALIGNMENT commands be collapsed into one without changing the output? Must FORMAT.FONT precede ALIGNMENT?

2)In **FormatBody**, will the AUTO-FIT formats work if the macro is applied before the data are transferred from SAS to EXCEL?

3)What happens if you exercise **InsertLineDivider** early in a calling program?

4)How would you write a single-purpose macro that would format a table with row and column totals to resemble the *Classic 3* Auto-Format in EXCEL 2002?

5)What would you do to improve the file save routines in **N04CallingPgm.sas**?

References

[1]Bodt, Mark. *Talking to PC Applications Using Dynamic Data Exchange.* Observations™, Volume 5, No. 3, 1996, pp. 18-27.

[2]Carpenter, Arthur. *Building and Using Macro Libraries*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #17.

[3]Carpenter, Art. *Carpenter's Complete Guide to the SAS® Macro Language, Second Edition.* Cary, NC: SAS Institute Inc., 2004.

[4]Carpenter, Arthur. *Macro Functions: How to Make Them - How to Use Them*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #100.

[5]McConnell, Steve. *Code Complete.* Microsoft Press: Redmond, WA, 1993.

[6]Microsoft Corporation. *Function Reference Microsoft EXCEL Spreadsheet with Business Graphics and Database: Version 4.0 for Apple® Macintosh® Series or Windows™ Series.* Document AB26298-0592. Printed in the United States of America, 1992.

[7]Microsoft Corporation. *Visual Basic User's Guide: Automating, Customizing, and Programming in Microsoft Excel with the Microsoft Visual Basic Programming System, Applications Edition*. Document XL57927-0694. Printed in the United States of America, 1993.

[8]Murphy, William C. Give *Your Clients What they Want: Fill Report Tables with the SAS® System and DDE.* Proceedings of the 15th Annual Northeast SAS Users Group Conference, Buffalo, NY, 2002, paper # PS025.

[9]Roper, Christopher A. *Intelligently Launching Microsoft Excel from SAS, using SCL functions ported to Base SAS*. Proceedings of the Twenty-Fifth SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2000, paper #97.

[10]SAS Institute Inc., *SAS Companion for the Microsoft Windows Environment, Version 8.* [Chapter 11: "Using Dynamic Data Exchange (DDE), pp. 217-226]. Cary, NC: SAS Institute Inc., 1999.

[11]Schreier, Howard. *Getting Started with Dynamic Data Exchange*. Proceedings of the Sixth Annual Southeastern SAS Users Group Conference, pp. 207-215, 1998.

[12]Vyverman, Koen. *Creating Custom Excel Workbooks from Base SAS® with Dynamic Data Exchange: A Complete Walkthrough*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #190.

[13]Vyverman, Koen. *Using Dynamic Data Exchange to Export Your SAS® Data to MS Excel - Against All ODS, Part 1 -*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002, paper #5.

[14]Watts, Perry. *Highlighting Inconsistent Record Entries in EXCEL: Possible with SAS® ODS, Optimized in Microsoft® DDE*. Proceedings of the 17th Annual Northeast SAS Users Group Conference. Baltimore, MD, 2004, paper #io01.

[15]Wexler, Steve and Julianne Sharer. *Microsoft® Excel Macros Version 4 for Windows™ and the Macintosh®.* Microsoft Press: Redmond, WA, 1992.

[16]Whitlock, Ian. *A Serious Look at Macro Quoting*. Proceedings of the 16th Annual Northeast SAS Users Group Conference. Washington,DC, 2003, paper #at012.

Trademark Citation

Contact Information

The author welcomes feedback via email at perryWatts@comcast.net. Please contact her to obtain a copy of the associated files from the NESUG 2004 CD.