

## SAS macro %COUNT

Jim Groeneveld, OCS Biometric Support, the Netherlands.

### ABSTRACT

To count the occurrence of specific distinct values of a number of specified variables within records (cases) the statistical software package SPSS® uses the command COUNT. Such user friendly functionality always has been lacking in SAS®. A similar result can only be obtained by complex coding in the data step. In this paper a macro is presented that converts the SPSS COUNT syntax into SAS code. It counts the occurrence of specific values (value lists) in specified variables (variable lists) within records. Results (per record) are stored in user specified variables. The macro %\_Count\_ supports the full syntax and identical functionality of the SPSS command. It even supports syntax and functionality beyond the SPSS COUNT command. Its main features are: constant character values with any content, missing value support with the \_SYSMIS\_ keyword, \_TO\_-convention for implied variable lists, \_THRU\_-convention for value ranges, including infinite \_LO\_ and \_HI\_ keywords, lexicographic comparison, variable specification instead of values, specification of array elements and SAS name literals.

### INTRODUCTION

At least since the seventies the statistical software package SPSS (originally Statistical Package for the Social Sciences) contains a command COUNT that counts the occurrence of a list of specified values in a list of specified variables within records. Its syntax is in its **simple** form:

```
COUNT Countvar = var_a var_b var_c var_d [...] (value_1, value_2, [...]).
```

and in its more **complex, general** form:

```
COUNT Cntvar1 = var_a (value_1, [...]) var_b (value_2 THRU value_5, [...]) [...] /
      Cntvar2 = var_c var_d TO var_f (value_6 value_7, [...]) [...] / [...].
```

in which [...] denotes similar continuations. The THRU keyword indicates value ranges. It is a powerful command. SPSS missing values, which may be any values defined as such, may also be counted, both individually and together. If counting all missing values at once the value(s) may be specified as MISSING and SYSMIS with SPSS.

SAS has always been lacking similar functionality. To obtain the same results in SAS several lines of code have to be written using many IF statements and the like. As an example we could have:

```
Countvar = (Var_a = Value_1) + (Var_a = Value_2) + (Var_b = Value_3) + [...];      OR:
Countvar = (Var_a IN (' Value_1 ', ' Value_2 ')) + (Var_b IN (' Value_3 ')) + [...];
```

It is clear that even such a solution will result in a lot of code in case the number of variables and values is large. The IN operator can be used, with exclusive values and returns a true/false (1/0) result that can be added.

### %COUNT

#### 1. REQUIREMENTS

In order to simplify the coding to obtain counts of values of variables within records a prototype macro %Count has been developed. The requirements were to minimize the number of variables and values to be programmed and to allow for both numeric and character variables and values. The macro initially has been developed to support functionality analogous to the **simple** form of the SPSS equivalent. The (minimum) information to be passed to the macro is:

- the name of the result variable to obtain the count (e.g. Countvar);
- the list of variable names to count values from;
- the list of values to count.

#### 2. SYNTAX

Based on the previous requirements the following syntax to call the macro was chosen:

```
%Count (CountVar=Countvar, VarList=Var_a Var_b [...], ValList=Value_1 Value_2 [...]);
```

The macro supports both numeric and character variables, but only one type at a time. The user has to take care to specify only one type of (existing) variables and values per macro call. The variable Countvar may or may not exist, while the other variables should exist; otherwise SAS will generate its own error reports in the log. Based on this macro call the %MACRO statement for the macro %COUNT is:

```
%MACRO Count (CountVar=, VarList=, ValList=);
```

The complete macro code can be found below.

## PhUSE 2008

### 3. MISSING VALUES

As already explained the COUNT command in SPSS can also count missing values, both user defined (MISSING) and the system missing value (SYSMIS). It may be desirable to count only one (or some) of the user specified missing values (or ranges). In that case SPSS allows to specify the actual value, whether specified missing or not, e.g. 99 or 'unknown'.

In SAS numeric missing values work considerably different from SPSS. SAS only has system missing values, 28 numerical ones and one character one (a space), of which the numeric single period (.) is best known. The numeric missing values in ascending order are: . \_ . .a to .z (dot underscore, dot, dot A to dot Z), the last one being the largest. So selecting all numeric missing values is best done by an IF statement like:

```
IF (Variable LE .z) THEN .....;           OR:           IF (MISSING(Variable)) THEN .....;
```

The last form also supports the character missing value (space) to be checked if the variable is of character type.

The macro %Count may count single numeric missing values by specifying them as values in the macro call. If one wants to count all missing values the keyword \_SYSMIS\_ may be specified which causes to count all numeric values LE .z and character values consisting of a space.

### 4. MACRO CODE

The code of the initial macro %Count basically consists of:

```
%* Macro Count counts value lists for numeric and character dataset variables ;
%* (C) Jim Groeneveld, 1 December 2005, version 0.1;

%MACRO Count (CountVar=, VarList=, ValList=);
  %LOCAL I Var J Val;
  &CountVar = 0;
  %LET I = 1;
  %LET Var = %SCAN ( &VarList, &I, %STR( ) );
  %DO %WHILE (&Var NE);
    %LET J = 1;
    %LET Val = %SCAN ( &ValList, &J, %STR( ) );
    %DO %WHILE (&Val NE);
      %IF (%UPCASE(&Val) EQ _SYSMIS_) %THEN
        %DO;
          IF (MISSING(&Var)) THEN &CountVar = &CountVar + 1;
        * (all 28 SAS numerical missing values or character space are checked here);
        %END;
      %ELSE
        %DO;
          IF (&Var EQ &Val) THEN &CountVar = &CountVar + 1;
        * (no sum statement to avoid forced implicit RETAIN from this macro);
        %END;
      %LET J = %EVAL (&J + 1);
      %LET Val = %SCAN ( &ValList, &J, %STR( ) );
    %END;
    %LET I = %EVAL (&I + 1);
    %LET Var = %SCAN ( &VarList, &I, %STR( ) );
  %END;
%MEND Count;
```

### 5. EXPLANATION

In its simplest form the name of a (new) counting variable is passed to the macro along with a list of existing variable names and a list of values to check for their occurrence with each variable of the variable list within the same record. The variable list is scanned for single variable names, which are in turn compared to each of the scanned values from the value list. All matches are counted and the sum of the matches is stored in the counting variable. Multiple specification of variables results in multiple counts.

### %\_COUNT\_

The more recent, full code of the macro %Count is available from the author's website (see References below). It is yet subject to changes, possible bug fixes and additions. In order to avoid interference with a possible future internal SAS macro function %Count the macro discussed here has been renamed to **%\_Count\_**. This macro, version 0.9, has evolved to support the full syntax and the same functionality of SPSS with even a few extensions. Its main code is presented below (without an extensive, informative header). The basics to count values are the same as above, but many more additional features are supported. It is not the intention to explain the full macro code here, but some programming strategies are discussed with specific features. Versions of the macro on the web may contain much more explanatory comments and error checking code than the code presented below. The current features of the macro are summarised below the macro code. Note that the code also contains two auxiliary macros (serving as subroutines for more or less general purposes): **%DisClose** and **%\_ArrayNr**. Having them in the same source code file causes them to be compiled along when the main macro is called, which in its turn calls the auxiliary macros.

# PhUSE 2008

## 1. MACRO CODE

```

%* ~~~~~;
%* Macro _Count_ counts occurring value lists for dataset variables ;
%* _____;

%* (C) Jim Groeneveld, 17 August 2008, version 0.9 for SAS vs. 9+;

%MACRO _Count_ (SPSScode); %* just one (main) positional argument;
  %LOCAL H SubCount Counter SpecList VarList Vallist I Var J Val ThruVal HighVal
    ToVar StopVar Array;

  %* remove enclosing parens;
  %LET SPSScode = %Disclose (Enclosed=&SPSScode, MaxDepth=9);
  %IF (%QSYSFUNC(SCANQ(&SPSScode,-1,.)) EQ) %THEN %LET SPSScode =
    %BQUOTE( %SUBSTR(&SPSScode, 1, %LENGTH(&SPSScode)-1) );

  %LET H = 1;
  %LET SubCount = %QSYSFUNC ( SCANQ ( &SPSScode, &H, / ) );
  %DO %WHILE (%BQUOTE(&SubCount) NE);
%* Extract counter var name (and remove its leading and trailing blanks);
  %LET Counter = %QSYSFUNC ( SCANQ ( &SubCount, 1, = ) );
  &Counter = 0; * if not implicitly (re)set by sum statement;
/* %LET SpecList = %QSUBSTR(&SubCount,%EVAL(%INDEX(&SubCount,=)+1)); */
  %LET SpecList = %QSYSFUNC ( SCANQ ( &SubCount, 2, = ) ); %* no more = !;
%* No more = characters expected outside constant string values, if more: error;
  %IF (%QSYSFUNC ( SCANQ ( &SubCount, 3, = ) ) NE ) %THEN %PUT Error: SPSScode;
%* Extract elements and remove leading and trailing blanks in them;
  %LET VarList = %QSYSFUNC ( SCANQ ( &SpecList, 1, %STR(%) ) );
  %LET Vallist = %QSYSFUNC ( SCANQ ( &SpecList, 2, %STR(%) ) );
  %IF (&Vallist EQ) %THEN %DO;
    %PUT Empty value list, VarList=&VarList;
  %END;
  %ELSE %DO;
    %LET I = 1;
    %LET Var = %QSYSFUNC ( SCANQ ( &VarList, &I, %STR(, ) ) );
    %DO %WHILE (%BQUOTE(&Var) NE);

%* Support for _TO_ convention using arrays (with incrementally numbered names);
    %LET ToVar = %QUPCASE(%QSYSFUNC(SCANQ(&VarList, &I+1, %STR(, ) )));
    %LET StopVar = %QSYSFUNC(SCANQ(&VarList, &I+2, %STR(, ) ));
    %IF ((%BQUOTE(&ToVar) EQ _TO_ OR %BQUOTE(&ToVar) EQ %STR(--))
      AND %BQUOTE(&StopVar) NE /* _TO_ convention, var list */
      AND %SYSFUNC(SCANQ(&Var, 2, {[%STR(%)]) EQ /* Var no array element */
      AND %SYSFUNC(SCANQ(&StopVar, 2, {[%STR(%)]) EQ /* StopVar neither */
      ) %THEN %DO;
      %LET Array = %_ArrayNr; %* use of auxiliary macro _ArrayNr;
      DROP _0; %* drop temporary and macro defined index variable;
%* unique incremental array name, no valid order check of variables;
      %UNQUOTE ( ARRAY _0&Array &Var -- &StopVar; ) %* create unique array;
%* variable list itself and its amount is unknown to macro;
      %UNQUOTE ( DO _0_ = 1 TO DIM(_0&Array); ) %* start of data step loop;
      %LET Var = _0&Array[_0_]; %* define array elements, redefine Var;
      %LET I = %EVAL ( &I + 2 ) ; %* finally skip over _TO_ and StopVar;
    %END;
    %ELSE %DO; %* single variable;
      DO; %* nothing special, just to start a DO-END block with one iteration;
    %END;

    %LET J = 1;
    %LET Val = %QSYSFUNC ( SCANQ ( &Vallist, &J , %STR(, ) ) );
    %DO %WHILE (%BQUOTE(&Val) NE);
      %LET ThruVal = %QUPCASE(%QSYSFUNC(SCANQ(&Vallist, &J+1, %STR(, ) )));
      %LET HighVal = %QSYSFUNC(SCANQ(&Vallist, &J+2, %STR(, ) ));
      %IF (%BQUOTE(&ThruVal) NE _THRU_) %THEN %DO;
        %IF (%QUPCASE(&Val) EQ _SYSMIS_) %THEN %DO;
          IF (MISSING(&Var)) THEN &Counter + 1;
        %END;
      %END;
    %ELSE %DO;
% (all 28 SAS numerical missing values or character space are checked here);
    %END;
    %ELSE %DO;

```

## PhUSE 2008

```

        %UNQUOTE ( IF (&Var EQ &Val) THEN DO;
            &Counter + 1; )
        END;
    %* can be simplified to: &Counter + (&Var EQ &Val);
        %END;
    %END;
    %* _THRU_ support;
        %ELSE %DO; %* ThruVal = _THRU_, no _SYSMIS_ value expected here!;
    %* _LO_, _HI_ support;
        %IF (%BQUOTE(&HighVal) NE) %THEN %DO;
            IF ( %IF (%QUPCASE(&Val) EQ _LO_) %THEN %DO;
                NOT MISSING(&Var) AND %END;
                %ELSE %DO; %UNQUOTE(&Var GE &Val AND) %END;
                %IF (%QUPCASE(&HighVal) EQ _HI_) %THEN %DO; 1 %END;
                %ELSE %DO; %UNQUOTE(&Var LE &HighVal) %END;
            ) THEN &Counter + 1;
    %* can be simplified to: &Counter + ((interpreted) condition);
        %END;
        %ELSE %PUT _COUNT_ NOTE: Value lacking after keyword _THRU_, no count!;
        %LET J = %EVAL (&J + 2); %* Proceed to after range spec;
        %END;
        %LET J = %EVAL (&J + 1);
        %LET Val = %QSYSFUNC ( SCANQ ( &Vallist, &J, %STR(, ) ) );
    %END;
        END; %* of loop through var list from _TO_ convention or normal;

        %LET I = %EVAL (&I + 1);
        %LET Var = %QSYSFUNC ( SCANQ ( &VarList, &I, %STR(, ) ) );
    %END;
    %END;
    %LET H = %EVAL (&H + 1);
    %LET SubCount = %QSYSFUNC ( SCANQ ( &SPSScode, &H, / ) );
    %END;
%MEND _Count_;
%*~~~~~;
%* Auxiliary Macro Disclose removes enclosing parentheses ;
%*~~~~~;
%MACRO Disclose (Enclosed=, MaxDepth=1);
%* Remove leading and trailing spaces beforehand;
    %LET Enclosed = %BQUOTE(%UNQUOTE(&Enclosed));
    %IF (&Enclosed EQ %STR(%)) %THEN; %* if ( ) then return nothing or msg;
        %* to prevent warning message;
    %ELSE %IF (%QSUBSTR(&Enclosed,1,1) EQ %STR(%( )
        AND %QSUBSTR(&Enclosed,%LENGTH(&Enclosed),1) EQ %STR(%)) %THEN %DO;
        %IF (&MaxDepth LE 1) %THEN
/* truncate surrounding parentheses and remove leading and trailing spaces */
        %BQUOTE( %SUBSTR(&Enclosed, 2, %LENGTH(&Enclosed)-2) );
    %ELSE /* recursively */
        %Disclose (Enclosed=%QSUBSTR(&Enclosed, 2, %LENGTH(&Enclosed)-2),
            MaxDepth=%EVAL(&MaxDepth-1));
    %END;
    %ELSE &Enclosed; %* return disclosed or original value;
%MEND Disclose;
%*~~~~~;
%* Auxiliary macro _ArrayNr creates or increments global macro variable _ArrayNr ;
%*~~~~~;
%MACRO _ArrayNr (_Counter= _ArrayNr);
    %IF (%NRBQUOTE(&&& _Counter) EQ %NRSTR(&)_Counter) %THEN %DO; %* Not existing;
        %* [ %SUPERQ(&_Counter) then is empty, but that is not enough ] ;
        %* WARNING: Apparent symbolic reference &_Counter not resolved. ;
        %PUT NOTE: Ignore WARNING about unresolved symbolic reference &_Counter above;
        %* Create GLOBAL macro variable &_Counter and initialize to 1 ;
        %GLOBAL &_Counter ; %LET &_Counter = 1;
    %END;
    %ELSE %DO; %* Existing;
        %* Increment &_Counter;
        %LET &_Counter = %EVAL ( &&&_Counter + 1 ) ;
    %END;
    &&&_Counter /* return value */
%MEND _ArrayNr;

```

## 2. FEATURES

The features of version 0.9 are:

### 1. Full support of the SPSS syntax

The complete SPSS syntax and functionality of the COUNT command is supported. The command consists of (multiple) variable and value lists per count (SPSS's case) (and possibly multiple counts per record). Value lists are enclosed in parentheses, preceded by variable lists, and multiple counts are separated by a slash (/). The final period (.), ending the SPSS command may, for the sake of syntax compatibility, be included or omitted; it will be ignored (*filtered out*). See the examples in the next paragraph below. The syntax of the macro %\_Count\_ in general is:

```
%_COUNT_ (( /* note: two consecutive left parentheses / start of SPSS-like syntax */
    Counter1 = variable_list_1 (value_list_1) variable_list_2 (value_list_2) [...] /
    Counter2 = variable_list_3 (value_list_3) variable_list_4 (value_list_4) [...] /
    [...]).
/* end of SPSS-like syntax / note: two consecutive right parentheses: */ );
```

### 2. Support for constant character values

Constant character values containing any characters between single or double quotes may be specified, including commas, spaces and those used as delimiters (parentheses and slash). This also applies to numeric values defined by a quoted value, directly followed by a quoted string identifier, for example a date value in the form of '31dec1948'd. (*Initially this feature seemed difficult to implement. The macro %SCAN and %QSCAN functions were firstly used to separate parts of the specified code text. These functions do not discriminate between separators wherever in the text, whether within normally quoted values (specified in the code text, surrounded by single or double quotes) or not. A scan function was needed to separate elements using the delimiters outside the quoted values only. Initially it was intended to write a suitable macro (function) %SCANQ for that purpose. However, since SAS 9 there is the SAS function SCANQ doing exactly that in SAS code. It has been applied together with the %QSYSFUNC macro function*);

### 3. Missing value support with the \_SYSMIS\_ keyword

Missing values are supported by specifying them either individually or as a group of 28 numerical missing values or space as character missing value using the \_SYSMIS\_ keyword (*generating: IF (MISSING(Var)) THEN ...*);

### 4. Support for SPSS's TO convention for implied, consecutive, same type variable lists.

Variable ranges specified with TO in SPSS and with -- and - in SAS differ rather a lot regarding their interpretation between SPSS and SAS. In SPSS you can't choose: if the variables exist, TO denotes an existing variable range (from the PDF, like SAS's --), if they (both) don't exist, TO denotes a range indicated by names with sequence numbers (like SAS's -). With the SPSS COUNT command all variables, to count values from, however, should exist, so in this case the interpretation is straightforward: a successive variable range from the PDF. The TO convention in variable lists is supported by the macro using either the keyword \_TO\_ (after SPSS) or -- (double dash, after SAS). The user has to make sure that he knows which ordered variable list will be generated. It is not allowed to specify array elements as the first and last entries of an implied list. (*It works like generating an ordered list of variables as they occur in the PDV, though the algorithm actually works somewhat different from that. Due to the way the feature has been programmed (SAS ARRAY and DO loop) it is not possible and not necessary for the macro to know the actual variable list and its size. Therefore checks on correct variable specifications can not be performed and the user has to rely on SAS's own error reporting, e.g. wrong order of starting and ending list variables. More details about the programming strategy of this feature is described below in a separate paragraph*);

### 5. Support of the SPSS's THRU convention for values

The equivalent macro's \_THRU\_ convention in value lists generates value ranges (GE and LE). If applied to character variables the range to compare will be interpreted as a lexicographical one by SAS. If the second value (or specified variable's value) is lower than the first one the value range will be empty and the resulting increment will be 0; no error report will be generated. (*If the subsequent value of any value consists of the keyword \_THRU\_ the value thereafter and the current one will be used in a range comparison*);

### 6. Support for the SPSS keywords LOWEST or LO and HIGHEST or HI

Lowest and Highest values only apply to ranges and thus should occur together with the keyword \_THRU\_ to indicate (one and two sided) infinite ranges. They are callable as \_LO\_ and \_HI\_ keywords. The \_LO\_ keyword causes the exclusion of all 28 numerical and single character SAS missing values; only valid values are included. (*If the range limits consist of the keywords \_LO\_ or \_HI\_ then the range comparison is adapted accordingly*);

### 7. Extraneous functionality beyond the SPSS functionality

- Lexicographic comparison** of character values with the \_THRU\_ keyword (character set, platform dependent);
- Extended support for **variable specifications** instead of values in the **value (range) list**. The values may thus consist of both constant numeric or character values and variable names. The \_THRU\_ convention in value lists also applies with variables and defines variable value ranges that may differ between records. So while the \_THRU\_ convention applies to a value list, even with variable names, the \_TO\_ convention will not apply there;
- The reverse is also supported: **constant values** (without \_TO\_) instead of variable names are allowed in the **variable list**;
- Instead of variables **array elements may be specified** too as long as their subscripts are not contained within parentheses, but within **braces or brackets** instead. Array elements may not be used as starting and ending variables of a TO-list; yet there will be no error report if this rule is violated but \_TO\_ will then be regarded as a variable by SAS (-- in such a case will lead to a SAS error);
- SAS name literals**, variable names in the form 'any characters'n, are supported too. These are allowed (in SAS vs. 9.1 at least) with OPTIONS VALIDVARNAME=ANY. (*This feature was added along with the support for constant character values, using the %QSYSFUNC(SCANQ(... construct))*).

### 8. Extensively documented macro header

The macro header, in the newer versions, specifies the arguments, their descriptions, their syntax, the development history and other information, including a reference to this article.

## PhUSE 2008

The supported macro call to count specific occurring values is for example (original SPSS syntax code in *italics*):

```
%_COUNT_ (( /* note: two consecutive left parentheses / start of SPSS-like syntax */
  Cntvar1 = var_a (value_1, [...]) var_b (value_2 THRU value_5, [...]) [...] /
  Cntvar2 = var_c var_d TO var_f (value_6 value_7, [...]) [...] / [...].
/* end of SPSS-like syntax / note: two consecutive right parentheses: */ ));
```

Note the double parentheses at the start and at the end of the macro call. These are mandatory because the inner parentheses and all code between them define and distinguish the **first (positional) macro argument (SPSScode)** containing the whole SPSS-like command, which is **parsed**. Any additional arguments may be specified just before the final right parenthesis (in between the last two parentheses). Additional arguments are not discussed here, though they will be implemented as part of the newer production version(s) of the macro.

### 3. EXAMPLES

The macro %\_COUNT\_ is called from within a SAS data step involving the variables from the dataset being read. Below some examples with their explanation are presented; *the SPSS-equivalent code is printed in italics*. The example starts with some example data. Note that variable names and values within lists are separated by commas and/or spaces.

```
OPTIONS VALIDVARNAME=ANY;
```

```
DATA _NULL_;
  INPUT a b c d $ e $ f $;
  ARRAY abc a b c;
  One = 1; Two = 2;
  ParenL='('; ParenR=')'; BracketL='['; BraceR=']";
  'copy of: E as "SAS name literal" 'n = E;
  IF (E EQ '.') THEN E = ' '; * make period (.) into missing space;
* PUT (_ALL_)(=);
CARDS;
1 2 3 ( [ {
4 5 6 ) ] }
7 8 9 D E F
. .d . _ _ . _ /* last three not at all missing character values */
;
RUN;
```

Then the `_Count_` macro can be called from inside a data step as follows:

```
%_COUNT_ (( /* note: two consecutive left parentheses / start of SPSS-like syntax */
  Counter1 = a, b c (3, 5 7, _sysmis_) /* _SYSMIS_ here means all 28 missing values */
  b, a b (1, 9) /* multiple variable specification counts multiple */
  d (ParenL '_' ) /* (temporary) var name instead of literal value */
  e f (BracketL BraceR _sysmis_ 'E') /* _SYSMIS_ here means empty, space */
/ Counter2 = a c (1 THRU 8) /* THRU value range, 1 up to 8 including the limits */
  b (7 THRU 4) /* legal, but never incrementing the counter */
  d (0 thru ) /* lacking second value: no counting but NOTE: */
/ Counter3 = a c (THRU 8) /* there is no variable THRU, but value 8 may exist */
  b (7 THRU thru ) /* no var THRU: SAS reports uninitialized variable */
/ Counter4 = d e ('D' THRU 'E') /* lexicographic comparison */
  f ('F' THRU 'E') /* legal, but not incrementing the counter */
  a b (One thru Two) /* Value list (vars in record), not variable list */
/ Counter5 = "D" "E" "F" (D E F) /* reverse, values with variables, same result */
/ Counter6 = 1 (0) /* constant values, not matching here */
/ Counter7 = 1 2 (One thru Two, 0) /* reverse value list, not variable list */
/ Counter8 = d, f ('=' '*' ) /* some allowed special characters */
/ Counter9 = abc[2] abc{3} (2 3 abc[1]) /* array elements */
/ Counter10 = a b (lo thru hi ) /* keywords LO and HI (with num & char type) */
/ Counter11 = a b c (2 thru hi ) /* all values 2 and higher */
/ Counter12 = a b c (lo thru 2) /* all values 2 and lower, excluding SAS missings */
/ Counter13 = d TO f (LO THRU 'E') /* all lexicographically LE "E", no missing */
/ Counter14 = a -- c (4 thru 6) /* consecutive variables in TO-list of same type */
/ Counter15 = a to c d -- f (_sysmis_) /* first list numeric, second one character */
/ Counter16 = d TO f ('/' '(' " ." "" ) /* all kinds of characters incl. delimiters */
/ Counter17 = ') ' (d) /* constant value in variable list, reverse specification */
/ Counter18 = 'copy of: E as "SAS name literal" 'n E ('[' ']' '.' ) /* SAS name literal */
/* end of SPSS-compatible syntax / note: two consecutive right parentheses: */ ));
```

## PhUSE 2008

### 4. PROGRAMMING STRATEGY OF IMPLIED, CONSECUTIVE, SAME TYPE VARIABLE LISTS (SPSS TO-CONVENTION)

At first sight implementing a TO-convention for variables, generating an explicit variable list to use, seemed virtually impossible. While PROC CONTENTS, with its output written to a dataset, can easily yield the desired information, this is not possible from within a SAS data step. Other possible solutions, applying the functions VARNUM and VARNAME, need to know the current dataset name, which isn't known to the macro. And even if known it would be necessary to constantly open, read and close the dataset from the macro code for every (implied, consecutive, same type) variable list, using the %SYSFUNC(VARNUM(..)) and %SYSFUNC(VARNAME(..)) constructs. This is not very efficient.

Hence, quite another solution was devised and developed. This solution started from the point of view that it isn't really necessary for the macro to know all the intermediate (and same type) variable names between the first and last variables of a TO-list specification. It was sufficient to develop code with which SAS (not the macro) can find the variables involved. Such code needs an ARRAY specification with only the (known) first and last variables of the list. Subsequently SAS could be driven by the macro to loop through that list of array elements, which would be involved in the comparisons and counts. So this solution needs to create a (uniquely named) SAS array for every implied TO-list.

The next problem that emerged was that it isn't possible to use the same array over and over again for all TO-lists as once an array has been created in a data step it can't be removed and recreated (with a different specification) anymore. This required defining a new and still uniquely named ARRAY for every TO-list. For that purpose a prefix for an array name was chosen that quite likely is very unique (\_0), to which a sequence number (generated by the auxiliary macro \_ArrayNr) is appended. As any subsequent call of the macro should be prohibited from using the same sequence numbers again it is necessary to store the sequence number in a global macro variable with a unique name (\_ArrayNr). The first time the value of that macro has to be checked, while it doesn't exist yet, yields a WARNING on a not resolved symbolic reference. In that case the macro generates a NOTE on ignoring the warning. Another solution here to even avoid the warning would be to apply the macro function %SYMEXIST (and possibly %SYMGLOBL), both in SAS version 9.x, but the warning and the note in the log aren't very disturbing. Initially, it was the intention to have the macro compatible with at least SAS version 8.x, but the use of the 9.x SAS function SCANQ (see above) changed this.

Thus every TO-list creates its own SAS array, incrementally numbered, even across different data steps in a SAS program (open SAS session). Next SAS needs an index variable to serve as the subscript for array elements. This index variable may be the same one every time, but should be unique to the remaining variables used in the data step (\_0\_). With it SAS is able to refer to the variables in the implied, consecutive, same type variable list (TO-list) by their respective array element specifications. The size of the array is not known to the macro itself but it is known to the SAS code as DIM(array). This way SAS is able to perform the comparisons and counts for each element of the array(s) perfectly well. The arrays disappear automatically at the end of the data step(s), but the index variable would remain if not DROPPed explicitly by the code generated by the macro (DROP \_0\_ ;). The global macro variable disappears automatically at the end of the program (or SAS session). If the TO-convention is not used these variables and arrays are not generated at all.

The drawback of this solution is that it is never known for sure whether the generated macro and SAS variable and array names are really unique and don't interfere with the other user specified data step code and existing variables in datasets used. The names have been chosen as rather uncommon, but in the later versions (1.0 and up) these names (and prefix) will be made user definable (with defaults). In the macro code presented here these features have not been incorporated as they don't contribute to the understanding and operation of the macro and its strategy. There will be other user definable arguments to the macro as well, all of which will be keyword arguments, specified after the first and most important positional argument SPSScode (containing the SPSS COUNT compatible code).

### CONVERSION DEMONSTRATION

To see the generated, converted SAS code of a call to the macro %\_Count\_ (of any macro unconditionally generating SAS code actually), instead of running it (as part of a data step), adapt and run the following code:

```
%PUT {Start of generated code} %QUOTE(  
/* the complete %_Count_ call here */  
) {End of generated code};
```

All generated SAS code will appear in the log, not neatly formatted, but logically working. The generated code may be run directly and the resulting log may be studied for debugging purposes.

### CONCLUSION

Imitating the SPSS COUNT command in a SAS macro, both its syntax and its functionality, is very well possible. The syntax ultimately can be the same, a SAS macro call is different from an SPSS command, but the SPSS-like command can be used embedded in the SAS macro call (its first and positional argument to be exact). The functionality can also be regarded as identical; there are only differences in the way SPSS and SAS deal with missing values. These differences in the concept and the handling of missing values are no obstacle at all to imitate the SPSS syntax and functionality as much as possible in SAS. After all, it is not the intention to create an exact copy of SPSS's COUNT command in SAS, but merely to create a simple and efficient way to count occurring values in SAS. If using dedicated SAS code it may need quite some amount of code and programming effort to reach the desired result, while a macro reduces the amount of programming work considerably. Any syntax could have been chosen while calling a macro developed to that extent, but as the SPSS equivalent COUNT was already known to be very easy and simple that syntax has been selected as the syntax of choice. There is no need, wish or claim to be fully compliant with SPSS regarding missing values. The macro %\_Count\_ can also be viewed as a syntax converter from simple (SPSS-like) counting code to complex (SAS) code.

### REFERENCES

- 1) SAS Institute Inc. 2004. Base SAS® 9.1.3 Procedures Guide. Cary, NC: SAS Institute Inc.
- 2) SPSS for Windows, Rel. 11.0.1. 2001. Chicago: SPSS Inc.
- 3) <http://listserv.cc.uga.edu/cgi-bin/wa?A2=ind0609&L=spssx-l&D=0&F=P&P=43849&F=> (question on SPSSX-L)
- 4) <http://tinyurl.com/parmbuff-at-comp-soft-sys-sas> or <http://tinyurl.com/parmbuff-at-SAS-L> (discussion on SAS-L)
- 5) [http://home.hccnet.nl/jim.groeneveld/software/SASmacro/\\_Count\\_.zip](http://home.hccnet.nl/jim.groeneveld/software/SASmacro/_Count_.zip) (the newest version of the macro)

### CONTACT INFORMATION

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

Author Name : Y. (Jim) Groeneveld  
Company : OCS Biometric Support  
Address : Schipholweg 78  
2316 XD Leiden  
The Netherlands  
Work Phone : +31 (0)71 572 1828  
Fax : +31 (0)71 576 5040  
Email : [info@ocs-biometricsupport.com](mailto:info@ocs-biometricsupport.com)  
Web : [www.ocs-biometricsupport.com](http://www.ocs-biometricsupport.com)

Brand and product names are trademarks of their respective companies.