

Paper 164-30

An Easy-to-Use SAS® Table Formatting Macro: Stand-Alone, Flexible, and Quick

Zbigniew Kadziola, Clinical Outcomes and Research Institute (CORI) Eli Lilly Australia P/L,
Vienna, Austria

ABSTRACT

A reporting macro was created with the following requirements: functionality similar to PROC TABULATE, much better than PROC TABULATE for controlling output layout, ease of incorporating non-PROC TABULATE statistics, and overall ease of use. The table definition mimics the final output and includes text, macro variables, macro calls, and special characters (cell and row separators, decimal tab, line feed, repeated column characters). The sub-macros %c(*cname*) and %e define a categorical variable *cname* and its scope within the table definition. To define a continuous variable, a pair of sub-macros, %v(*vname*) and %e, are used. The variables can be nested in a similar way to PROC TABULATE. If within the table definition there are macro variables that have names equal to the PROC TABULATE statistical keywords, the macro will call PROC TABULATE to resolve them. To build the call, the macro uses the information about the location of the particular statistical keyword within the table definition in relation to the scope of the *cname/vname* variables. After resolution of the table specific macro variables, the table definition is executed by the macro processor; that is, the macro variables are substituted, and the user macros called. During the execution the characters between the %c(*cname*) and %e will be processed for each category of *cname*, and for each iteration the category name will be available as &*cname*. The resulting string of characters is divided into rows and cells and constitutes the data set that reflects the initial table definition. That dataset is the input for the required final ODS processing: RTF, PDF, HTML, etc. Utilizing this macro to its full potential enables the user to produce diverse professional looking reports with varied and complex statistics with relative ease.

INTRODUCTION

Formatting tables in SAS for clinical trial data presents a challenge especially when maximum flexibility of combining summary statistics, confidence intervals, and p-values is desired. PROC TABULATE provides a broad range of summary statistics but is not that flexible with regard to table layout or incorporating other externally generated information. Other procedures require very detailed data manipulation before the desired table can be produced. One solution to this problem is to construct a table-submitter macro where the macro call actually mimics the table layout desired.

TEST DATA

The following data set will be used to create examples:

```
data test;
  retain seed 117;
  do i=1 to 1000;
    if ranuni(seed)<.5 then group='Group A'; else group='Group B';
    if ranuni(seed)<.5 then gender='M'; else gender='F';
    if ranuni(seed)<.5 then origin='Caucasian'; else origin='Hispanic ';
    if ranuni(seed)>.005 then age=60+5*normal(seed); else age=.;
    output;
  end;
run;
```

INTRODUCTORY EXAMPLE

The paragraph below shows a call of the main macro: %table and the RTF output produced by the call:

```
%table(data=test, rtf=Y, title="Example",
tabdef=%nrstr(
  " |                | Total \ (N=&N) | "
  " |                |                | "
  " | Gender, N(% ) |                | "
%c(gender) " | &gender      | `&N` (&Pctn)  | "%e
  " |                |                | "
  " | Age, years     |                | "
%v(age)    " | N              | `&N`          | "
  " | Mean (±SD)    | `&Mean` (±&Std) | "%e
));
```

Note that parameter *tabdef* actually mimics the layout of the final output.

Example	
	Total (N=1000)
Gender, N(%)	
F	492 (49.20)
M	508 (50.80)
Age, years	
N	993
Mean (±SD)	59.94 (±5.026)

OTHER ELEMENTS OF TABLE DEFINITION

REPEATED COLUMN CHARACTERS

Most tables of clinical trial data include comparisons between groups (e.g. treatments). With the `%table` macro, each group category column is repeated using the same format/layout and the category specific information.

The `%table` macro provides a parameter `coldef` which allows easy definition of such columns:

```
%table(data=test, rtf=Y, coldef=%nrstr(%c(group, char1=[, char2=])),
tabdef=%nrstr(
  " | &group\ (N=&N) ] | "
  " | Gender, N(% ) ] | "
  %c(gender) " | &c [ | `&N` (&ColPctn) ] | "%e));
```

Each occurrence of `[` and `]` within the `&tabdef` will be replaced by `%c(group)` and `%e` calls, respectively.

So the above table definition is equivalent to:

```
tabdef=%nrstr(
  " | %c(group) | &group\ (N=&N) %e | "
  " | Gender, N(% ) %c(group) | %e | "
  %c(gender) " | &c %c(group) | `&N` (&ColPctn) %e | "%e)
```

And the result is as follows:

	Group A (N=494)		Group B (N=506)	
Gender, N(%)				
F	257	(52.02)	235	(46.44)
M	237	(47.98)	271	(53.56)

The `[` and `]` are the defaults for `%table`.

SUBSET

It often happens that we want to show only selected categories in a table. The second positional parameter of the `%c` submacro will be treated by the macro processor as a logical condition. So the actual `%c` syntax is `%c(cname, lcondition)`. The `lcondition` will be evaluated for each category of variable `cname`, and the text between `%c` and `%e` brackets will be processed only when the condition is true.

So the following code:

```
%table(data=test, rtf=Y, coldef=%nrstr(%c(group)), tabdef=%nrstr(
  " | [ | &c\ (N=&N) ] | "
  " | [ | ] | "
  %c(gender, &c=F) " | Female Gender, N(% ) [ | `&N` (&ColPctn) ] | "%e));
```

Will produce the following output:

	Group A (N=494)		Group B (N=506)	
Female Gender, N(%)	257	(52.02)	235	(46.44)

The `%c` which defines the repeated columns can use the selection condition, too. I.e. the

```
... coldef=%nrstr(%c(group, &c=Group A))...
```

will not produce the *Group B* column.

DENOMINATOR

The denominators for calculating percentages can be defined in the same way as for PROC TABULATE:

```
%table(data=test, rtf=Y, coldef=%nrstr(%c(group)), tabdef=%nrstr(
  " | [ | &c\ (N=&N) ] | "
  %c(origin) " | [ | ] | "
  " | &c Origin [ | N=`&N` ] | "
  " | Gender, N(% ) [ | ] | "
  %c(gender) " | &c [ | `&N` (&Pctn<gender>) ] | "%e%e));
```

As the result each *origin* is treated as 100% for *gender* percentages:

	Group A (N=494)	Group B (N=506)
Caucasian Origin	N= 261	N= 237
Gender, N(%)		
F	133 (50.96)	98 (41.35)
M	128 (49.04)	139 (58.65)
Hispanic Origin	N= 233	N= 269
Gender, N(%)		
F	124 (53.22)	137 (50.93)
M	109 (46.78)	132 (49.07)

F SUB-MACRO

The %f submacro provides a way of including external (i.e. not from PROC TABULATE) information in the table. The syntax is %f(dname,vname,vformat). When called during processing of the table definition, %f opens the dname data set and returns the text which is the result of reading vname variable using the given format vformat.

When %f call occurs within a scope of %c(cname1), %c(cname2),... calls, then it opens dname with a where=(cname1=&cname1 and cname2=&cname2 ...) condition. When >1 record is available then the reads are separated by Line Feeds.

The following PROC FREQ call will produce a p-value for each *origin* category:

```
proc freq data=test noprint;
  table origin*gender*group/fisher;
  output out=xf fisher;
run;
```

The %f is nested within the %c(origin) and its %e brackets, so %f will be called twice: once for *Caucasian*, once for *Hispanic* origin; the first call will open xf(where=(origin="Caucasian")), the second: xf(where=(origin="Hispanic")):

```
%table(data=test, rtf=Y, coldef=%nrstr(%c(group)), tabdef=%nrstr(
  %c(origin) " | [ &c\ (N=&N) ] | p-Value* | "
  " | [ | ] | "
  " | &c Origin [ N= `&N ] | "
  " | Gender, N(% ) [ ] | `%f(xf, xp2_fish, pvalue6.) | "
  %c(gender) " | &c [ `&N` (&Pctn<gender>) ] | "%e%e | "
  " | [ ] | "
), footnote="* - Fisher's exact test");
```

And the result:

	Group A (N=494)	Group B (N=506)	p-Value*
Caucasian Origin	N= 261	N= 237	
Gender, N(%)			0.0384
F	133 (50.96)	98 (41.35)	
M	128 (49.04)	139 (58.65)	
Hispanic Origin	N= 233	N= 269	
Gender, N(%)			0.6544
F	124 (53.22)	137 (50.93)	
M	109 (46.78)	132 (49.07)	

* - Fisher's exact test

U SUB-MACRO

The `%table` macro makes use of the `%unquote` twice when the table definition `tabdef` is processed. This fact makes it more complicated to place within the table definition a 'proper' call to a user-written macro that refers to the table specific macro variables (like `&cname`, `&N`, etc.). One way is to use `%nrstr` calls, which will delay the execution of the user macro. A second way is to use the `%u` macro, since the `%table` will synchronize the execution of calls to `%u` with the moment when all the table specific macro variables are resolved.

As an example let's assume that we want to calculate the difference in age between group A and B:

```
proc ttest data=test;
  class group;
  var age;
  ods output statistics=s(where=(class='Diff (1-2)'));
run;
data _null_;
  set s;
  call symput('diff', '' || put(mean,best5.) || ' '(±' || put(stderr,best5.) || ')');
run;
```

... and to show it in column *Group A*:

```
%macro u(group);
%if %unquote(&group)=Group A %then &diff;
%mend u;
%table(data=test, rtf=Y, coldef=%nrstr(%c(group)), tabdef=%nrstr(
  " |&group\ (N=&N) ] | "
  " |Age, years [ | "
%v(age) " | N [ | `&N ] | "
  " | Mean (±SE) [ | `&Mean` (±&StdErr) ] | "%e
  " | Mean Diff. (±SE) [ | %u(&group) ] | "));
```

And the result will be:

	Group A (N=494)	Group B (N=506)
Age, years		
N	490	503
Mean (±SE)	59.71 (±0.227)	60.16 (±0.224)
Mean Diff. (±SE)	-0.45 (±0.319)	

NUMBER OF DECIMAL PLACES

The number of decimal places shown in a table for a PROC TABULATE specific statistic is controlled as follows:

- 2 decimal places for percentages
- for a statistic related to a continuous variable
 - o `best5.` format if the variable is not formatted
 - o number of decimal places of the formatted variable increased by `&dstat`, where `stat` is the name of the PROC TABULATE statistical keyword. E.g. For `mean`, the default `&dmean` is 1, for `stderr` the `&dstderr` is 2. The defaults can be changed within the `%table` call: e.g. `%table(...,dmean=2,...)`.

In the below example the format of the variable `age` is `5.`, i.e. with 0 decimal places:

```
%table(data=test, rtf=Y, coldef=%nrstr(%c(group)), format=age 5.,
  tabdef=%nrstr(
  " |&group\ (N=&N) ] | "
  " |Age, years [ | "
%v(age) " | N [ | `&N ] | "
  " | Mean (±SE) [ | `&Mean` (±&StdErr) ] | "%e));
```

So the `&Mean` will be displayed with `0+1=1`, and the `&StdErr` with `0+2=2` decimals:

	Group A (N=494)	Group B (N=506)
Age, years		
N	490	503
Mean (±SE)	59.7 (±0.23)	60.2 (±0.22)

HOW DOES THE %TABLE WORK ?

Let's start again with the known example, and let's trace how the *Pctn* macro variable gets its values.

PROC TABULATE CALL

For the below code:

```
%table(data=test, rtf=Y, title="Example",
tabdef=%nrstr(
    " |                | Total\ (N=&N) | "
    " |                |                | "
    " | Gender, N(% ) |                | "
%c(gender) " | &gender | `&N` (&Pctn) | "%e
    " |                |                | "
    " | Age, years |                | "
%v(age) " | N | `&N | "
    " | Mean (±SD) | `&Mean` (±&Std) | "%e));
```

the *%table* will create and execute the following PROC TABULATE call:

```
proc tabulate data=test missing out=__tab;
class gender;
var age;
table all*(N)/printmiss;
table gender*(N PCTN)/printmiss;
table age*(N MEAN STD)/printmiss;
run;
```

Note that regardless the number of variables and regardless the type of variables (continuous/categorical) there will be only ONE call to PROC TABULATE (instead of, for example, calling PROC FREQ/UNIVARIATE for each row of the table).

MACRO VARIABLES RESOLUTION

The PROC TABULATE output data set

	gender	_TYPE_	_PAGE_	_TABLE_	N	PctN_0	age_N	age_Mean	age_Std
1		0	1	1	1000
2	F	1	1	2	492	49.2	.	.	.
3	M	1	1	2	508	50.8	.	.	.
4		0	1	3	.	.	993	59.936	5.025766

will be used to resolve the table specific macro variables:

```
...
TABLE C1L1_PCTN_0 49.20
TABLE C1L2_PCTN_0 50.80
...
```

TABLE DEFINITION EXECUTION

The table definition (i.e. the *&tabdef*) is executed twice using *%unquote*. After the first execution it looks like:

```
" || Total\ (N=&&_c.&_v._N) | "
" || | "
" | Gender, N(% ) | | "
%c(gender,1,%nrstr(" | &gender | `&&_c.&_v._N` (&&_c.&_v._PCTN_0) | "))
" || | "
" | Age, years | | "
%v(age,1,%nrstr(" | N | `&&_c.&_v._N | "
" | Mean (±SD) | `&&_c.&_v._MEAN` (±&&_c.&_v._STD) | "))
```

After the second:

```
" || Total\ (N=1000) | "
" || | "
" | Gender, N(% ) | | "
" | F | `492` (49.20) | " | M | `508` (50.80) | "
" || | "
" | Age, years | | "
" | N | `993 | "
" | Mean (±SD) | `59.94` (±5.026) | "
```

DATA SET WITH TABLE CONTENT

The resulting text is divided into rows/cells and a data set is created:

	_col1	Total\N=1000)
1		
2	Gender, N(%)	
3	F	492 (49.20)
4	M	508 (50.80)
5		
6	Age, years	
7	N	993
8	Mean (\pm SD)	59.94 (\pm 5.026)

DATA SET WITH FORMATTED TABLE CONTENT

Up to this point the processing is independent from the destination format.

The `%table rtf=Y` parameter will force RTF formatting, which will create another data set:

	_col1	Total\N=1000)
1	^R"\keepn"	
2	Gender, N(%)^R"\keepn"	
3	F^R"\li200 " ^R"\keepn"	^R"\tab " ^R""492^R"\tab " ^R""(49.20)^R"\tqdec\tx650 " ^R"\tqdec\tx1300 "
4	M^R"\li200 "	^R"\tab " ^R""508^R"\tab " ^R""(50.80)^R"\tqdec\tx650 " ^R"\tqdec\tx1300 "
5	^R"\keepn"	
6	Age, years^R"\keepn"	
7	N^R"\li200 " ^R"\keepn"	^R"\tab " ^R""993^R"\tqdec\tx650 " ^R"\tqdec\tx1300 "
8	Mean (\pm SD)^R"\li200 " ^R"\keepn"	^S={cellwidth=3.3177083333cm} ^R"\tab " ^R""59.94^R"\tab " ^R""(\pm 5.026)^R"\tqdec\tx650 " ^R"\tqdec\tx1300 "

TEMPLATES AND FILE PRINT ODS=

The formatted data set will be printed using FILE ODS=(TEMPLATE=...) statement:

Example	
	Total (N=1000)
Gender, N(%)	
F	492 (49.20)
M	508 (50.80)
Age, years	
N	993
Mean (\pm SD)	59.94 (\pm 5.026)

DISCUSSION

The `%table` macro has been used extensively to produce thousands of pages of output summarizing clinical trials. It has proved its usefulness, however some weak points have become apparent. One of them is because the table definition is stored in a macro variable. That means that at each stage of processing the table – which ‘lives’ as a text string – cannot be longer than 32K characters in SAS 8 (64K in SAS 9). This problem has been partially addressed by delaying the resolution of a macro variable if the resolved text is longer than the macro variable name. A second weakness is that the internal names of the table specific macro variables can easily reach the 32 characters limit, for example if there is high degree of nesting (approximately ≥ 4 levels). But generally the `%table` macro works fast and efficiently, and substantially reduces the amount of text produced in the log file.

CONCLUSION

1. The `%table` macro builds similar tables to PROC TABULATE but has much greater flexibility regarding the layout
2. By using macro variables and macro calls within the table definition, the user can incorporate any kind of statistics into the final table
3. Output can be easily redirected to the different destinations (rtf, pdf, html,...)
4. The macro builds the table using only one call of PROC TABULATE (instead of, for example, calling PROC FREQ/UNIVARIATE for each row of the table).

ACKNOWLEDGMENTS

Many thanks to Bruce Basson for his help and support.

CONTACT INFORMATION

Contact the author at:

Zbigniew Kadziola
Eli Lilly Regional Operations Ges.m.b.H.
Kölblgasse 8-10
A-1030 Wien, Austria
Work Phone: +43 1 71178 496
Fax: +43 1 71178 220
Email: kadziolaz@lilly.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.