PhUSE 2010

Paper AD09

Phase-III Macro System

Wolf-Dieter Batz, Phenix-MTK, Kelkheim, Germany

ABSTRACT

The Phase-III Macro System is a flexible, data independent and parameter controlled set of SAS macros.

Module size is kept small (three screen pages at maximum) for maintainability and avoids hard-coded references to any application related information like data types, labels and formats. Coding style makes broad use of automatic documentation and generation of meta data and lookup tables at runtime.

INTRODUCTION

The Phase-III Macro System is aimed at serving as a base for an extendable system that provides mechanisms for shaping input datasets, processing calculations and generating SAS datasets with ready made text content. The following requirements are met:

- Produce a wide variety of output with a minimum set of modules.
- Minimize maintenance efforts through self-documenting and limited program code.
- Be prepared to add new output structures without substantial delay.

The Phase-III Macro System is a highly interactive collection of macro modules providing transformation methods for study emergent datasets making use of all the information available in the description part of the dataset processed.

It provides subroutines that care for data types, formats, labels, headers, missing values, loops and more. Runtime generated information used to control processing is kept in standardized data structures using macro variable lists ("mlists"), SAS formats and datasets.

The user is provided with (an) output dataset(s) containing character columns with standard names and externally controlled attributes. Finally the Phase-III Macro System provides pre- and post processing functionality such as condense, struct and missline.

SCOPE

The Requirements, Ideas, Architecture and Solution described here have been taken from the statistical programming part of a clinical study for which I was contracted a few years ago. As part of the excellent and precisely defined processes the study was based on, a booklet, the so-called table shell, served as unique reference for the total of all tabulation to be performed and later addressed to the health authorities for approval.

Total subjects treated	2821 (100)	2823 (100)
Start of study medication within 24 hrs after randomization		
Yes	2751 (97.5)	2775 (98.3)
No	66 (2.3)	47 (1.7)
Surgery delayed	59 (2.1)	41 (1.5)
Other	13 (0.5)	13 (0.5)
Missing	- (-)	- (-)
Missing	4 (0.1)	1 (0.0)
Study medication only from kit allocated		
Yes	2821 (100)	2823 (100)
Missing	- (-)	- (-)
Number of subjects who did not receive full loading dose	26 (0.9)	12 (0.4)
Reason:		
Adverse event	3 (0.1)	3 (0.1)
Technical failure / dosing error	17 (0.6)	8 (0.3)
Missing	6 (0.2)	1 (0.0)
At least 75% of loading dose administered		
Yes	2808 (99.5)	2819 (99.9)
No	8 (0.3)	4 (0.1)
Missing	5 (0.2)	- (-)
Number of subjects with maintenance dose interruptions longer than 1 hour	86 (3.0)	101 (3.6)
Number of maintenance dose interruptions		
1	79 (2.8)	98 (3.5)
2	4 (0.1)	4 (0.1)

SUSTAINABLE APPROACH

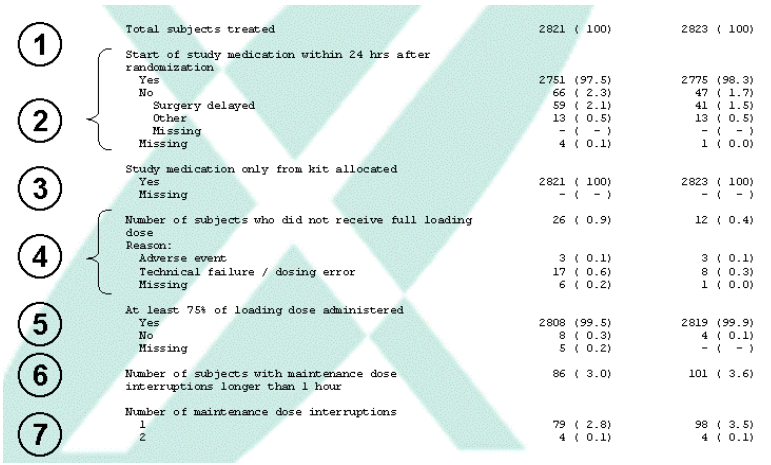
The ordinary way to go would have been to immediately start programming in order to generate the first table according to the definition from the table shell and then continue one-by-one. But for some reason we did not do that. Maybe because the number of tables was too large to accomplish programming both, in-time and in high quality, or maybe because we were expecting a number of minor last-minute-changes to eat up more time than was available after database closure or, worse, after un-blinding.

REQUIREMENTS ANALYSIS

Instead, we started by investing some time to have a closer look at the table definitions. Not at all surprising, we identified a limited number of similarities and dissimilarities that resulted in two lists: The first one listed table structures found and the second one listed parameters. After that every single entry from the table-shell could be expressed as variation and combination from on one or more structures.

GRANULARITY

As a result from structural analysis, it was regarded most useful to have table definitions formulated as grid of super-cells. These are cells that contain one or more values. Super-cells aggregate horizontally to form super-rows. Super-rows aggregate vertically to form an output table.



CELL TYPES

Super-cell contents are managed on a character string level. Nevertheless, three categories are defined to reflect data types of original data reported. These are continuous variables ("cont"), categorial variables ("catv") and boolean variables ("bool").

IMPLEMENTATION

For each of these cell types SAS macros are coded to produce super-rows.

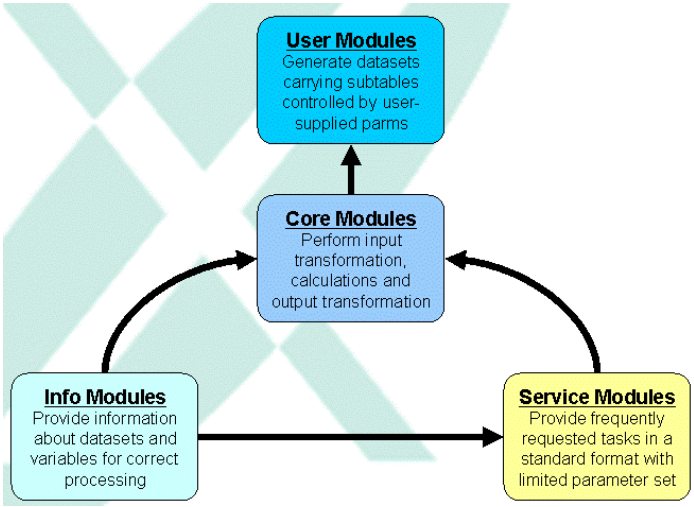
As can be seen from this example, a table is most likely to be comprised from several super-rows of different type and depth.

This term "depth" probably needs to be explained in more detail since it is crucial for the understanding of the entire logic used in the Phase-III Macro System.

DEPTH

Let's accept for a moment the idea that complex structures may be generated by repeating simple structures plus adding other simple structures, that is, logical multiplication and addition. By iterating this operation we can deliberately increase complexity without any need for new functions aside multiplication and addition.

The architecture of the Phase-III Macro System is based on this concept and hence, is capable to produce a theoretically unlimited set of tables with theoretically unlimited complexity.



ARCHITECTURE

To provide tabulation for clinical studies the architecture described here was found to be totally sufficient:

1. All tabulation is performed by User Modules. 2. These make use of functions provided by the Core Modules. Core Modules generate data tables that could be output but would not look well formatted. 3. Service Modules do not produce such output data but carry functions that are needed by Core Modules to work properly. 4. Finally, Info Modules provide information that they obtain from the data dictionary and other repositories like dataset headers etc.

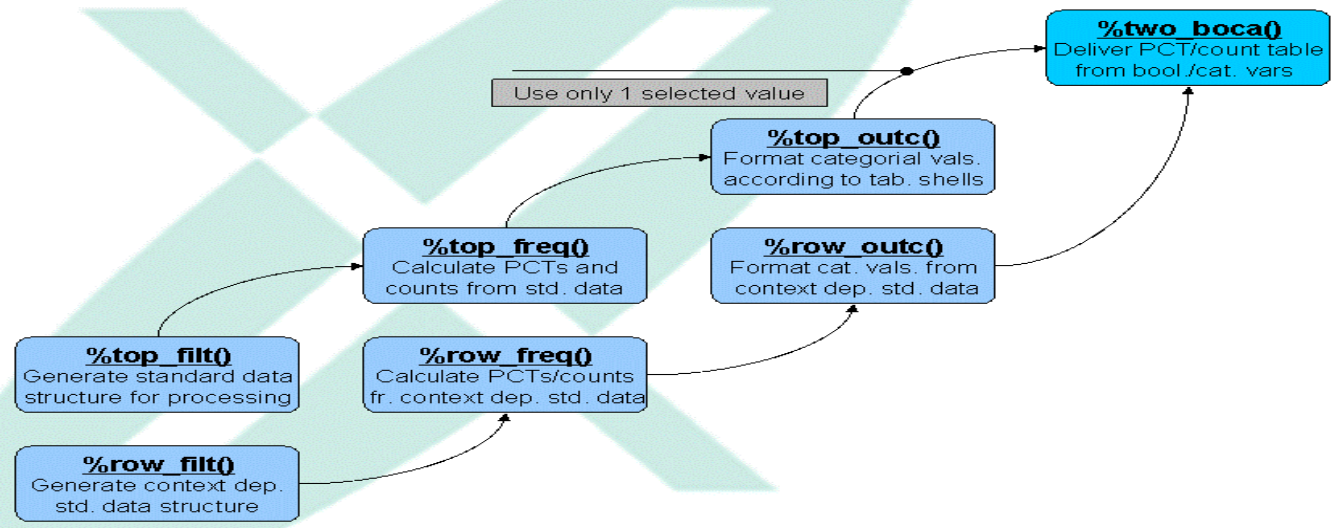
Quite obviously depth in software or system architecture supports the concepts of module reusability and system maintainability. Moreover they provide a means to limit module source code which in turn facilitates validation, error prevention and code maintenance.

EXAMPLE: TABLE COMPOSITION FROM USER MODULES

```
1. %BLK_CATV(dsn=_in_put_, row=sdw24h, rev=n, col=trnoat, space=2, total=o, head=n, indent=0,
   num=1, stat=n)
2. %TWO_BOBO(dsn=_in_put_, use=, use2=N, row=sdw24h, head=n, row2=sdsudl#Y other#1 miss_5#1,
   space=3, col=trnoat, indent=2, indinc=2, num=2, total=n, stat=n, weight=y, rev=y)
3. %BLK_CATV(dsn=_in_put_, row=sdon_der, rev=y, col=trnoat, space=2, total=n, head=y,
   indent=0, num=3, stat=n)
4. %TWO_BOCA(dsn=_in_put_, use=0, row=loadfull, head=n, row2=nofusdrs, space=3, col=trnoat,
   indent=0, indinc=0, num=4, total=n, stat=n, weight=y)
5. %BLK_CATV(dsn=_in_put_, row=vol75, rev=y, col=trnoat, space=2, total=n, head=y, indent=0,
   num=5, stat=n)
6. %ROW_BOOL(dsn=_in_put_, row=infint1h, col=trnoat, space=0, total=n, indent=0, num=6,
   use=1, stat=n)
7. %BLK_CATV(dsn=_in_put_, row=infintno, rev=n, col=trnoat, space=, total=n, head=y,
   indent=0, num=7, stat=n)
```

Numbers 1 to 7 refer to the numbers in the table shown above. The naming convention used supports recognition of the output structure generated:
“ROW” indicates that only one output line is produced.
“BLK” indicates that several output lines are produced with block style appearance.
“TWO” indicates that several output lines are produced with appropriate indentation from nested categories.

EXAMPLE: MACRO %TWO_BOCA()
To allow a deeper understanding of how the architecture relies on nesting (depth structuring), one of the User Modules from the example above is shown.



1. declares and upper level processing

```
%MACRO TWO_BOCA(dsn=, use=, row=, row2=, col=, indent=0, num=, total=T, stat=Y, weight=Y,
space=3
, condense=, struct=, struct2=, head=Y, head2=Y, misslin2=, indinc=2) / store des="";
%LOCAL n_grp v_grp n name;
%LET name=TWO_BOCA;
%IF &STRUCT  eq %THEN %LET struct =&DSN;
%IF &STRUCT2 eq %THEN %LET struct2=&DSN;
%TOP_FILT(dsn=&DSN, grp=&ROW, by=&COL, grplvl=&NUM,var=, condense=&CONDENSE);
%TOP_FREQ(dsn=top_filt, struct=&STRUCT, grp=&ROW, by=&COL);
%TOP_OUTC(dsn=top_freq, head=&HEAD, total=&TOTAL, stat=&STAT, indent=&INDENT, grp=&ROW,
rev=n, use=&USE, by=&COL);
```

2. loop for lower level processing

```
%GRP_DESC(dsn=&DSN, grp=&ROW, miss=n);
%DO n=1 %TO &N_GRP;
  %IF %SCAN(&V_GRP,&N) eq &USE %THEN %DO;
    %ROW_FILT(dsn=&DSN, context=&ROW, subgrp=&N, grp=&ROW2, by=&COL, var=, miss=n);
    %ROW_FREQ(dsn=row_filt, sum=top_freq, struct=&STRUCT2, context=&ROW, grp=&ROW2,
by=&COL, weight=&WEIGHT);
    %ROW_OUTC(dsn=row_freq, sum=, head=&HEAD2, stat=&STAT, indent=%EVAL(&INDENT+&INDINC),
context=&ROW, grp=&ROW2, by=&COL, missline=&MISSLIN2);
  %END;
%END;
```

3. care for naming and send completion mail

```
%IF &TAB_NAME ne %THEN %DO;
  data &TAB_DEF&NUM%SUBSTR(&TAB_NAME,5,4);
   set
    %IF &SPACE eq 1                  %THEN dummy ;
     row&NUM._0
    %IF &SPACE eq 2                  %THEN dummy ;
     row&NUM._&CURSUB
    %IF &SPACE eq 2                  %THEN dummy ;
    %IF &SPACE eq 3                  %THEN dummy ;
   ;
  run;
%END;
%GEN_MAIL(name=&NAME);
%MEND TWO_BOCA;
```

PARAMETERS

COMMON

DSN: Name of input dataset or view. This may be any valid SAS dataset name (one-level or two-level) not accompanied by dataset options or other SAS syntax components.

COL: Name of variable used to construct columns. The variable is checked for number of levels and an appropriate number of columns are generated.

ROW: Name of variable to construct rows, superrows, and subtables. Modules capable of processing more than one variable accept a list here.

HEAD: Optionally specify “N” to suppress output of the header line for the row variable generated from their label. In categorial processing the header is an additional 1st line whereas in continuous processing the header text is written left-hand to the 1st stats line output. Default is “Y”.

STAT: Optionally select “Y” to generate an output column which contains the names of statistics generated. Default is “N”.

INDENT: Optionally select a positive integer to indent the rows generated as one block. Default is “0”.

SPACE: Optionally select spacing mode for one-level subtables: 0=no blank lines; 1=1st output line is blank; 2=last output line is blank. Default is “2”. For two-level subtables: 2=insert additional blank line between upper and lower level output; 3=only last output line is a blank line. Default is “3”.

NUM: Assigns a unique number to the output generated. Only one digit is allowed here.

PROCESSING SPECIFIC

STATS (continuous): Specify a list of statistics to calculate. Currently this may be a list of statistics output by Proc Univariate; output formats are preset as follows: N, NMISS, MIN and MAX: 6.; MEAN and MEDIAN: 8.1; STD: 9.2. No default list is assigned. Preset formats may be overwritten by attaching a positive integer to the statistic requested using a “#” sign. Hence MIN#3 outputs minimum values formatted with three (3) decimal positions.

REV (categorial): Optionally reverse order of decodes in output subtable. This does not affect results for missing values which always are output as bottom line. Default is “N”.

USE (boolean): Select value to be used as “true value” or the “logical 1”. Since a value of “YES” may be coded as “1”, “Y” or something else this is a necessary parameter for all modules performing boolean processing. Hence no default value is assigned.

USE (%two_bobo()): Optionally select a “true value” from the upper hierarchy level. When no selection is made all decodes from the upper level variable are calculated and output as in categorial processing. No default is set.

TOTAL (categorial & boolean): Optionally select totals mode: O=generate totals line only; T=place totals line on top of subtable generated with one blank line intermitted; I=output totals line as first line of subtable generated and equally indented; B=like “I” but make totals the last line; N=no totals line shall be output. Calculation of percentages may be suppressed by optionally specifying “C” as 2nd digit of totals mode as in “OC”, “TC”, “IC” and “BC”. Default is “T”.

STRUCT (categorial): Specify the name of a dataset that contains all categorial levels of the &ROW variable at all levels of the &COL variable to appear in the output. Default is set to &DSN.

MISSLINE (categorial): Suppress the generation of dummy missing lines normally intended to make tables appear complete by specifying “N”. Default is “Y”.

CONDENSE (categorial with no primary key): Define the “true” value for the &ROW variable according to variable#value when variable is not unique. No default is set.

NULLTXT (continuous): Optionally specify “Y” to write a value of &NUM._N_ in the first column of an output from continuous processing beginning at the second row. These values will be hidden by a format $nulltxt which maps them to blank but show up visible by assigning another format, usually $longtxt. Default is “N”.

PVALUE (continuous): Optionally specify whether non-parametric testing for &VAR is performed on &COL. The resulting p-value is output in the rightmost column of an additional line and formatted appropriately. Currently only “Wilcoxon” is available. No default is set.

EXCLUDE ((%two_catv()): Optionally specify value from the upper hierarchy level for which no output shall be generated. No default is set.

LEVEL SPECIFIC

HEAD2 (two-level categorial): Optionally specify an “N” to suppress output of a header line from lower level processing. Module specific defaults are set for %two_catv(), %two_boca() and %two_bobo().

INDINC (two-level categorial & boolean): Indent increment for lower level subtable. A value of “0” will produce alligned upper and lower level results. Default is “2”.

MISSLIN2 (two-level categorial): Suppress the generation of dummy missing lines in lower level processing by specifying “N”. Default is “Y”.

ROW2 (two-level categorial): Specify a composite list of lower level variables and their true values (var1#Y var2#1 var3#N). Each list element triggers generation of one line for output. No default is set, neither for variables nor for true values.

STRUCT2 (two-level categorial): Specify name of structure dataset to be used for lower level procesing. Default is set to &DSN.

USE2 (two-level categorial): Select upper level value to be used as lower level context and weight. No default is set.

WEIGHT (two-level categorial & boolean): Optionally use upper level results (percentages) as weights for lower level results. Default is “Y”.

MODULES

INFO MODULES

Get_Attr: Reads dataset header and returns attributes as undeclared macro variables using the requested attributes names. Information becomes available when the particular variable is declared in the calling environment using a %global or %local statement.

Grp_Desc: Investigates given categorial variable and provides results using undeclared macro variables: &n_grp - number of distinct values; &v_grp – structured list of distinct unformatted values; &l_grp – structured list of distinct formatted values.

Chk_List: Reads supplied list of tokens and returns undeclared macro variables: &n_lst - number of list elements; &v_lst – structured list of supplied elements. Input list elements may be separated by blank and comma only.

SERVICE MODULES

Gen_Intv: Reads list of ranges and generates two formats; **&fmt_name.g** for correct sorting; **&fmt_name.f** for using the supplied ranges as a group label during tables generation.

Gen_Type: Generates list of _type_ values from supplied number of variables. Provided _type_ values are calculated with respect to generation of hierarchical weights.

Gen_Wgts: Generate matrix of weights to provide hierarchical percentages in table generation.

Gen_Reco: Read sequence of blank separated assingments to be used in Proc Format. A character format named according to the supplied &fmt_name is produced.

Exp_Info: Generates format from variable and primary key from dataset. Format generation is type sensitive.

Imp_Info: Generates new variable by mapping a primary key with a format and makes it available using an sql view.

Addgroup: Transports information from dataset_A to dataset_B by combining the services from %exp_info and %imp_info.

Gen_Mail: For batch execution immediately report which module executes in what report environment. Specific return codes supplied send appropriate messages.

CORE MODULES

Top_Filt: Read from source dataset or view and perform filter and select operations

Row_Filt: Read from source dataset or view and perform filter and select operations with respect to a given context variable‘s value.

Top_Freq: Read output from *_filt module and perform calculations of frequencies and counts.

Row_Freq: Read output from *_filt module and perform calculations of frequencies and counts.

Row_Univ: Read output from *_filt module and perform calculations of univariate statistics according to given list with respect to their sequence.

Top_Outc: Read output from *_freq module and generate an output file with character formatted columns, totals, header and indentation.

Row_Outc: Read output from *_freq module and generate an output file with character formatted columns, indentation and weighted percentages. Add counts and percentages from context variable as headline.

Row_Outv: Read output from *_univ module and generate an output file with character formatted columns, stat labels and header.

USER MODULES

Row_Bool: Select single (‚true‘) value for categorial processing from *_freq and *_catv modules.

Blk_Bool: Loop with %row_bool over array of categorial values using same name prefix and ‚true‘ value and output results as one block.

Tab_Bool: Loop with %blk_bool over groups of variables using diferent name prefixes.

Blk_Catv: Use categorial processing from *_freq and *_catv modules with restricted paramater set.

Tab_Catv: Loop with %blk_catv over array of names using same processing parameters.

Blk_Conv: Use continuous variable processing from *_univ and *_conv modules with restricted paramater set.

Tab_Conv: Loop with %blk_conv over array of names using same set of processing parameters.

Two_Catv: Perform nested processing of two categorial variables looping the context variable from the row_* modules over the categories of the ‚outer‘ categories.

Two_Boca: Perform nested processing with only one (‚true‘) value select from the outer category.

Two_Bobo: Perform nested processing with boolean (‚true value‘) selection from the outer category and an array of boolean selections inside like in %blk_bool. True values may be chosen for each inside variable separately.

CONCLUSION

The system presented has been implemented using the SAS Systems Macro Facility.

This functionality has long time been overseen in favor of Screen Control Language by SAS Institute and the programmer community as well. The Macro Facility supports very well an object oriented system design, although not all OO-Characteristics are present and supported due to the age of the platform.

For us it is breathtaking efficient what has been accomplished in the extremely short period of eight months, once the preparation in terms of modularization functions and granularity of output had been done:

Less than 30 Modules, none of which extended three screen pages, are sufficient to meet the entire tabulation requirements for a large Phase III clinical study.

CONTACT INFORMATION

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

Author Name	Wolf Dieter Batz
Company	Phenix-MTK GmbH
Address	Wiesengrund 8
City / Postcode	D-69234 Dielheim
Work Phone:	+491772163609
Fax:	+496222770095
Email:	batz@phenix-mtk.com
Web:	www.phenix-mtk.com