Building Systems of Macros
The Print System and The Macro Reporting System (MRS)

John Iwaniszek and George DeMuth – Stat-Tech Services, LLC. Chapel Hill, North Carolina

Abstract

The SAS/Macro facility can be used to create complex open-ended reporting systems. Global macro variables can be used to create SAS PROC style interfaces by passing key summary information within a system of macros. The Print System and the Macro Reporting System (MRS) provide two examples of systems, each designed with a programming interface that enables users to assemble statistical summary tables from their primitive elements. In this case, the primitives are the macros that constitute the horizontal and vertical dimensions of the table (independent and dependent variables) and the types of summaries that populate the cells (univariate statistics, frequencies, test statistics). Different macros serve to initialize the table, specify the rows, and execute the table.

Introduction

This paper describes a method of constructing complex applications in Base SAS. Two systems that the authors have written for their own use and for general distribution will serve as examples.

The user interface in each application is designed as an analog to the program's final product. Each macro represents a component of the table or listing produced by the application. In the Print System, the majority of the macros captures output parameters and post them to global macro variables. In MRS, global summary parameters, such as treatment group, are stored in global macro variables for identification in individual summary macros. The summary macros produce output in the form of summary data sets that conforms to a standard format. A final macro assembles the analysis data sets and produces a final table.

Programs make much use of both macro variable arrays and subscripting of macro variables. These systems are easy to use, but require some familiarity with higher level programming concepts and the SAS macro language.

Statistical Reporting in Clinical and Social Research

The authors work in the field of clinical research involving drug and device development and have experience in other areas such as social sciences and academics. The common element in this experience is the need to produce concise reports that include statistical tables, listings, and individual subject (i.e. patient, survey respondent, employee, or other unit of analysis) profiles.

The authors have developed various programming tools in Base SAS that use relatively simple aspects of the SAS Macro Facility. These tools are complex systems of macros that minimize programming time and produce custom and standard tables and listings.

The Applications

The following discussion will use examples from two applications, the Macro Reporting System (MRS) and the Print System. These applications are used to generate statistical summaries and display results as printed text. The applications are collections of macros that initialize the system and collect parameter settings to control features of the finished product. The overall system has three classes of macros: an initialization macro, several different utility macros, and a driver macro.

The Print System

The print system is a set of macros that can be used to produce ASCII listings from a SAS data set. The system was created to generate output features not generally available in a single SAS procedure. For example, the macros can create fully-hooded (box) style output, continuation messages, header and footer lines, handle pagination of BY columns, and produce mixed-style footnotes. Further, the Print macros handle column widths with less user input than PROC REPORT.

The Print System is comprised of a set of macros that initialize the system, identify which SAS data set to print, what variables to include in the listing, the formatting options for the variables, and any spanning headers. All the macros except for the last one in the system collect parameter settings and store them in global macros variables and global macro variable arrays.

The %PRINT macro at the end is the driver macro and employs the settings to produce the finished listing. The generic structure for a print system program is described below.

```
%initbox( data=, outstyle=,  …Initializes system
with user defined and default settings ) ;

%column( var=, form=, label=, width=,  j=
…Identifies the variables to be displayed and
sets their properties) ;

%span( text= …Spanning header to apply to
following columns);

    %column( var=, …);

    %span( text= …. NOTE:  Spanning headers
may be nested and contain split characters.);
            %column( var=, …);
            %column( var=, …);
    %spend;

%spend; (terminates the most recent spanning
header. This macro has no parameters.)

%print;  (Collates the information entered through
the above macro calls and produces listing as
specified.)
```

As described above, the PRINT macro uses the information captured in the previous macro calls to generate a DATA _NULL_ program to produce the final product. For further operational details of the Print System.

The Macro Reporting System

The MRS is a system for capturing and printing results from multiple SAS procedures or custom user summaries. The MRS was created to address the issue of generating common descriptive summaries that constitute the majority of statistical reports in the clinical trials setting.

The system utilizes global macro variables to pass parameters about the overall appearance of the table: summary groups or sub-groups, column widths, table format (boxed versus line), and other parameters. Individual summary macros generate successive rows for the final table.

The actual output is assembled at the end of the program and depending upon the assembly macro used it can be in ASCII, RTF, or HTML format. Hence, the MRS follows a more complex strategy than the Print System because it includes summary macros that do a data processing job as well as simply collecting and storing parameters.

Each of the summary macros performs a categorical or numerical summary and places the results in an intermediate data set built according to a standard structure.

An MRS program has the following generic structure.

```
%itab( data= , grp=, …Initializes the table with
user defined and default settings);

    %rowcount( var=, …one of several MRS
macros that produce summary rows in the table);

    %summary macro
    %summary macro

    (Other summary macros include categorical,
univariate, multi-response, and event
summaries.)

%ASSEMBLE( This macro collates output from
the summary macros into a table.  Output can be
ASCII, RTF or HTML depending on how this
macro is implemented)
```

The User Interface – Analog to Output

The user input via the macro parameters is the primary means to get information into the systems described above. Other means include hardcoding global macro variables, creating special intermediate data sets, and creating user defined formats. The guiding principle in developing the systems was to break the components of the desired output into related elements. The resulting systems present close analogies to the expected output and the programming standards SAS users expect when using Base SAS.

Figure 1 below was generated with MRS and will serve as an example for the following discussion. Most displays of data listings and statistical tables can be described as two-dimensional objects containing row and column elements.

In a basic table the input data set, the treatment group variable (if any), and column labels are examples of non-repeating entities. However the row elements are repeating entities in that there may be several different types of rows included all with different entries into the same classes of parameters. Repeating elements include summary type, position in the table, and summary label.

In Figure 1 below, the rows include two categorical summaries and two univariate summaries. The structure of the program that produced this table is schematically similar to those described in the section above. The non-repeating elements (including the column dimension) are initialized in %ITAB while the repeating elements are initialized in the summary macros requesting and generating the row summaries.

The following program produced the table in Figure 1

```
%itab(all,grp=trt,grplbl=Treatment
Group,grpfmt=trt.,vwidth=30) ;

    %* Standard summaries for all
patients ;

%unisum(var=age,label=%bquote(Age
(yrs))) ;

%catsum(var=gender,fmt=$6.,label=Gen
der) ;

%unisum(var=wtkg,label=%bquote(Weigh
t (kg))) ;

%catsum(var=race,fmt=$16.,label=Race
) ;


%titlgen(progname=DEMOGO1,ref=LO1,pr
intto=ON) ;   %* Get title and print
;

%assemble( method=LINE) ;

%pageit(pp=ON) ;
```

Note that %TITLGEN and %PAGEIT are special utility macros that access a database of project titles and paginate the final table, respectively.

All parameters are passed between macros via individual global macro variables and arrays of global macro variables. The following section describes the simple techniques used to accomplish this.

Summary of Demographic Characteristics

Characteristic	Treatment Group: Placebo	Active	Total
Age (yrs)			
N	100	86	186
Mean	45.3	44.1	44.7
Std. Dev.	7.30	8.11	7.68
Median	45.0	44.0	45.0
Gender			
Female	47 (45%)	39 (41%)	86 (43%)
Male	58 (55%)	56 (59%)	114 (57%)
Total	105	95	200
Weight (kg)			
N	105	95	200
Mean	79.3	82.1	80.6
Std. Dev.	14.95	15.19	15.09
Median	80.5	80.7	80.6
Race			
African American	16 (15%)	17 (18%)	33 (17%)
Asian	13 (12%)	15 (16%)	28 (14%)
Caucasian	23 (22%)	18 (19%)	41 (21%)
Hispanic	23 (22%)	19 (20%)	42 (22%)
Other	28 (27%)	23 (24%)	51 (26%)
Total	103	92	195

Figure 1: Example Table Generated by MRS.

Capturing User Input - %Global Macro Variables and Repeated Structures

The hallmark of this and other computer applications is that it captures data from its environment and uses them to guide its intended process. SAS macro applications can receive input through a variety of means including macro parameters, global macro variables, local macro variables, data sets in SAS or other formats, and operating system settings.

The chief means by which users communicate with the macros described above is via the macro parameters. The macros also register their position and other attributes in macro variable arrays that are initialized in the opening macro, updated by the various utility macros, and used by the system driver to produce the f

final product. The following describes the various means by which information is accumulated and travels between the macros in these systems.

Using Macros to Gather Input

Input to the macro systems (other than the analysis data sets) is primarily via macro parameters. These macro parameters are transmitted to other parts of the system (most prominently to the driver macro) by global macro variables and arrays of global macro variables.

The unifying theme is that data are transmitted via globalized macro variables which are declared and initialized in the initialization macro. The global macro variables are populated from user input to the initialization and utility macro parameters.

Sufficient documentation on global macro variables is present in the SAS language manual (see key word %GLOBAL). In short, the contents of a global macro variable are available to all parts of a SAS program following the section of code that declares and populates it. There is no native macro variable array in the SAS macro language so macro variable arrays are constructs that are built via programming to be described below. Arrays are iterative structures and require two components; elements containing the data and indexes to keep track of array elements' relative position to one another.

In the Print system described above, global macro variable arrays are constructed by first initializing a counter. The counter (&_pcoln in this example) for the total number of columns is a global macro variable that is set to zero in the initialization macro (%INITBOX).

```
%global
      _pcoln
      ;

%let _pcoln =0;
```

The variable &_pcoln is used to keep track of the number and order of calls to the %COLUMN macro. Each additional call to %COLUMN increments &_pcoln by one, using the following assignment statement:

```
%* Update count of total columns in the table ;
%let _pcoln = %eval(&_pcoln + 1) ;
```

The %COLUMN macro also contains assignment statements that add new elements to various macro variable arrays that govern appearance and content.

In the following example, the global macro variables are populated by input from the %COLUMN macro's parameters. Note that appending the current value of &_pcoln forms the index of each global macro variable element in an array of global macro variables. This results in the creation of a new global macro variable called _var2 for the print variable identified on the second %COLUMN macro call in a table program.

```
%macro column(
      var=,       /* variable name REQUIRED */
            form=,          /* format */
            label=,         /* label */
      j=C,          /* Justification values C L or R */
      type=DISPLAY, /* type of variable DISPLAY
or BY are only valid types */
      width=,       /* Optional width  */
      wrapind=0,    /* Indent for wrapped rows
(after 1st row) */
      wrap=N        /* request for overline when
OUTSTYLE=LINE */
      ) ;

%* Update count of total columns in the table ;
%let _pcoln = %eval(&_pcoln + 1) ;

%* create global variables ;
%global _var&_pcoln _frm&_pcoln _lab&_pcoln
_typ&_pcoln
            _wid&_pcoln _wrp&_pcoln _jst&_pcoln
_wrpi&_pcoln ;

%* set values ;
%let _var&_pcoln = &var ;
%let _frm&_pcoln = &form ;
%let _lab&_pcoln = &label ;
%let _jst&_pcoln = %upcase(%substr(&j,1,1)) ;
%let _typ&_pcoln = %upcase(&type) ;
%let _wid&_pcoln = &width ;
%let _wrp&_pcoln = &wrap ;
%let _wrpi&_pcoln = &wrapind ;
```

Collating Parameters and Building the Final Table from its Components

A driver macro collates the parameters stored in the various macro variables and macro variable array and generates the final product.

The following example shows how a series of SAS data sets can be concatenated using a macro %do loop and a subscript (&_i) built into the data set name. The global macro variable &_varcnt contains the number of summary rows requested in an MRS table program. In the MRS, each summary macro creates a data set called _iset#. Here, # in an index value corresponding to the order the summary macro was called. E.g. the i^{th} summary macro creates a data set with name _isetI. The special data sets conform to an application standard that is named according to a preset convention. Temporary files _iset1 to _iset&_varcnt are concatenated in the loop

```
data _null_ ;
  set %do _i = 1 %to &_varcnt ;
        _iset&_i (keep=_col)
        %end ;
```

The following shows another example of a macro variable array. The MRS allows up to 5 additional user created columns to be added to a standard table via the parameter ADDSET1= to ADDSET5=.

The code below is used to sort the additional data sets. The array consists of elements named &addset1 to &&addset&_addc where &_addc is the number of elements in the macro variable array addset. The individual elements of the array are resolved in two passes so that the first element (&&addset&_addc) is resolved as &addset1on the first pass, and then resolved to whatever its contents (as text) are on the second pass.

```
%do _i = 1 %to &_addc ;
      %if &&addset&_i ne %then %do ;
         proc sort data=_add&_i ;
            by &_pg &_by _morder _varcnt _rorder
;
            run ;
   %end;
```

The following shows a more complex example where the elements of macro variable arrays are macro variables themselves. This code is used in the MRS to print a message the user if formats are being taken from an input data set. The macro variables _pg, _by, _grp, and _sgrp are global variables that correspond to user input for summary modifiers of a page-by variable, a by-variable, a group variable, and a sub-group variable. For each of the variables, the user has the option of specifying a format or letting the macros obtain the formats from the input data set.

```
   %let ivars=     _pg    _by    _grp   _sgrp ;
   %let iforms=    _pgfmt _byfmt _grpf  _sgrpf;

   %do i=1 %to 4;

         %let ivar=%scan( &ivars, &i);
         %let iform=%scan( &iforms, &i);

         %if &&&ivar> and &&&iform= %then %do;
            %put NOTE: Format is not specified for
variable &&&ivar;
                  %put NOTE: Format
will be taken from format on input
data set;

               %end;
     %end;
```

The lists assigned to &ivars and &iforms are macro variables containing the names of variables and their associated formats. The list is parsed from left to right and the macro variables are resolved in two passes. In this process &&&ivar becomes &_pg in the first pass and in the second resolves to its ultimate value, which in this application is a SAS variable name.

Programming and debugging programs that use these systems

Programming with the systems presented above is straightforward if the programmer internalizes the heuristic that the table program is analogous to the final table or listing. Knowledge and experience using the SAS macro language is necessary to take advantage of the advanced features of these systems. Thorough understanding of SAS programming in general and data step programming in particular greatly enhance the utility of these systems.

Debugging any macro system is can be tricky and is made more difficult by the cryptic error messages and lack of line numbers in the SAS log. Attention to a few details can improve the probability of successful program execution.

The program schematics above provide an illustration of the basic structure of programs using the systems. The interface is made up of classes of macros that are analogous to the components of the finished product. The initialization macros govern basic table formatting, input data sets, by variables, treatment groupings, and other parameters that are non-repeating entities. The utility macros set up repeating entities like summary rows (in the case of MRS) and variables for listing columns (in the case of The Print System). The driver macros collate all the information into a single package and generate the final display.

The user should have a good understanding of basic SAS macro programming and some idea of how SAS macros are processed. Correct syntax is very important because of the cryptic nature of the error messages and the lack of line numbering in the portion of the SAS program log that contains the macro code execution. Simply dropping a parenthesis in a utility macro call can result in laborious debugging. Obviously, careful attention to details such as parentheses, semicolons and other standard syntax is very important. Familiarity with macro quoting is also

very important. Any character in a label, for example, that could be construed as an operator (*, and, or, etc.) or reserved word might result in a difficult to detect syntax error and should be quoted with whatever quoting function that is appropriate.

Conclusion

The paper demonstrates how using global macros and simple indexing can be used to create extended systems using only SAS/Base. This approach allowed the authors to build useful systems of macros capable to addressing a large percent of production programming work. These systems have reduced programming time, despite the additional run time the programs require to execute when compared to native SAS procedures. Finally, the systems have improved standardization of the programming process, appearance of output, and quality of output. By implementing these strategies, the programmer can obtain additional power by achieving more open-ended specification of parameters and more expansive systems.