PharmaSUG 2014 - Paper AD09

Macro Programming via Parameter Look-Up Tables

Joseph Hinson, Accenture Life Sciences, Berwyn, PA, USA

ABSTRACT

SAS® Macros provide a unique means of creating one-size-fits-all programs and many large institutions rely on secure non-editable macros for generating standardized clinical reports. In such situations, programmers are shielded from the standard macros and are instead simply made to manipulate an external source of parameters in order to produce customized reports. This approach can further be extended by organizing report macros into domain-neutral classes based only on the structural layout of the reports, and another set based on analysis types, such that a call to a parameter look-up table would provide the appropriate combination of structural and analysis class macros pertinent to a particular report. With such an approach, just a dozen macro classes can rely on a large set of parameters in a look-up table to generate hundreds of different customized report across domains and therapeutic areas. The present paper aims to offer the concepts behind such an approach with the hope that application developers and programmers can gain from them innovative ideas for efficient macro programming for the generation of clinical reports.

INTRODUCTION

Often in very large organizations, SAS® programmers are made to rely on standard macros, especially for reports, in order to attain consistency and production speed. In such situations, the standard macros, to some extent, rely on parameters from spreadsheets maintained by users. In spite of such systems, organizations still find themselves dealing with several hundreds of standard macros.

This idea of maintaining a core set of standard, non-editable macros, and having the parameters supplied via an external document accessible to programmers can further be extended by changing the entire programming paradigm: considering reports as possessing dual entities, *layout* and *analysis*. Many report tables and listings share common structural features regardless of the domain or statistical analysis that was performed. Thus by making parameters indicate which combination of layout and analysis is needed, only a handful of macros would be required to produce hundreds of reports. A parameter look-up table can therefore be the workhorse in disguise, providing analysis and layout data to global macros that transform them into report-specific programs. This is the main concept behind this paper.

The paper is organized into three sections:

1. The use of a Parameter Library to configure global macros
2. The classification of reports according to structural layouts and analysis types
3. The coordination of the entire process with the %Pilot macro

I. THE PARAMETER LIBRARY

Macros typically depend on parameters for functioning. When just a handful of parameters are required, it is quite convenient to simply list parameters as arguments in the macro definition statement. Others choose to create macro variables with the %LET command at the top of the macro body. However, there are often situations where macro parameters can run into hundreds. Parameters that define a custom report could include titles, row-labels, column headers, footnotes, as well as data sources, treatments, visit names, etc. For such situations, a parameter look-up table, called in this paper “Parameter Library”, could be very convenient. This ordinarily would be an Excel spreadsheet that acts as a look-up table and is accessible to all programmers. The rows in the spreadsheet are organized into blocks of parameter data, with each block having its own ID (“BlockID”). Each block is assigned to a particular report output and would contain all the information that would configure the standard macros for that unique output.

A section of the Parameter Library (colored green in Table 1 below) is a reserved block with global parameters and locked from editing, except by the macro standards team. This is the block that would hold the standard macro names as well as various global data that are applicable to any report.

BlockID	ParameterName	ParameterValue	CallingMacro	Creator	LastModified
G0000	AE	Counts	%Analyzer	(Read Only)	
G0000	VS	DescriptStats	%Analyzer	(Read Only)	
G0000	LB	DescriptStats	%Analyzer	(Read Only)	
G0000	EG	DescriptStats	%Analyzer	(Read Only)	
G0000	CM	Counts	%Analyzer	(Read Only)	
G0000	DS	Counts	%Analyzer	(Read Only)	
G0000	PageSize	80	%ReportGenerator	(Read Only)	
G0000	Title9	CLINICAL RESEARCH COMPANY	%ReportGenerator	(Read Only)	
G0000	Footnote9	Confidential	%ReportGenerator	(Read Only)	
T3007	DatasetName	ADVSX	%DataImporter	John Doe	12/30/2013
T3007	Title1	A Multi Center Double-Blind Randomized Cross Over Trial	%ReportGenerator	John Doe	12/30/2013
T3007	Title2	Amyotropic Lateral Sclerosis	%ReportGenerator	John Doe	12/30/2013
T3007	Title3	Efficacy	%ReportGenerator	John Doe	12/30/2013
T3007	ReportType	Summary	%Analyzer	John Doe	12/30/2013
T3007	Domain	VS	%Analyzer	John Doe	12/30/2013
T3007	ProgrammerName	John Doe	%ReportGenerator	John Doe	12/30/2013
T3007	Footnote1	N=total number of treated subjects	%ReportGenerator	John Doe	12/30/2013
T3007	Footnote2	Run on SAS 9.2	%ReportGenerator	John Doe	12/30/2013
T3007	ByVariables	usubjid, treatment, visit	%Subsetter	John Doe	12/30/2013
T3007	ColumnHeaders	TRT-A, TRT-B, TRT-C, TOTAL	%ReportGenerator	John Doe	12/30/2013
T3007	WhereClause	where (TRTEMFL="Y")	%Subsetter	John Doe	12/30/2013
T3007	Protocol	CRC123	%CoverPage	John Doe	12/30/2013
L5001	DatasetName	ADSL	%DataImporter	Jane Doe	1/1/2014
L5001	Title1	A Multi Center Double-Blind Randomized Cross Over Trial	%ReportGenerator	Jane Doe	1/1/2014
L5001	Title2	Amyotropic Lateral Sclerosis	%ReportGenerator	Jane Doe	1/1/2014
L5001	ProgrammerName	Jane Doe	%ReportGenerator	Jane Doe	1/1/2014
L5001	ReportType	Listing	%Analyzer	Jane Doe	1/1/2014
L5001	Domain	DM	%Analyzer	Jane Doe	1/1/2014
L5001	Title3	Demographics Listing	%ReportGenerator	Jane Doe	1/1/2014
L5001	Footnote1	N=total number of treated subjects	%ReportGenerator	Jane Doe	1/1/2014
L5001	ByVariables	usubjid, treatment, visit	%Subsetter	Jane Doe	1/1/2014
L5001	WhereClause	where (TRTEMFL="Y")	%Subsetter	Jane Doe	1/1/2014
L5001	Protocol	CRC445	%CoverPage	Jane Doe	1/1/2014

Table 1. The Parameter Library as a look-up table

Each row of the library would have a “ParameterName” and a “ParameterValue” column. The parameter value is the data retrieved by a macro (through a helper macro called “%GetParamValue”) from the library, using the BlockID and the parameter name.

INFORMATION RETRIEVAL FROM THE PARAMETER LIBRARY

Hash objects are great as look-up tables and are used for retrieving specific parameters from the Parameter Library. First, the entire Parameter Library Excel sheet (Table 1) is read into SAS®, becoming the dataset “paramlibrary” , which is then loaded into a hash object. The hash method object.find() then presents BlockID and ParameterName as keys, which then triggers the ParameterValue associated with the ParameterName to be returned. The entire code is wrapped in the helper macro %GetParamValue, as shown below:

```
%macro GetParamValue(blockid, pname);
data  null ;
       length blockid $11 parametername $27 parametervalue $58;
       if(1=2)then set paramlibrary;
       call missing(of _all_);
       declare hash pl(dataset:"paramlibrary");
       pl.defineKey("BlockID","ParameterName");
       pl.defineData("ParameterValue");
       pl.defineDone();

       rcc=pl.check(key:&blockid., key:&pname.);
       if rcc eq 0 then do;
       rcf=pl.find(key:&blockid.,key:&pname.);put ParameterValue=;
       end;
run;
%mend GetParamValue;
```

CALLING THE MACRO TO OBTAIN PARAMETER VALUES:

```
%GetParamValue("&blockid.","Title1");
%GetParamValue("&blockid.","DatasetName");
%GetParamValue("&blockid.","ByVariables");
%GetParamValue("&blockid.","WhereClause");
%GetParamValue("&blockid.","ReportType");
%GetParamValue("&blockid.","Domain");
%GetParamValue("&blockid.","Protocol");
%GetParamValue("&blockid.","Footnote1");
%GetParamValue("&blockid.","Footnote2");
%GetParamValue("&blockid.","ProgrammerName");
```

LOG SHOWING IMPORTATION OF THE PARAMETER LIBRARY EXCEL SHEET INTO SAS®:

```
193  proc import file="C:\Documents and Settings\jhinson\Desktop\ParameterLibrary.xls"
193! out=paramlibrary dbms=xls replace;
194  run;
NOTE: The import data set has 41 observations and 6 variables.
NOTE: WORK.PARAMLIBRARY data set was successfully created.
```

RETRIEVED PARAMETER VALUES (IN BLUE) DISPLAYED IN THE LOG:

```
308  %GetParamValue("&blockid.","Title1");
parametervalue=A Multi Center Double-Blind Randomized Cross Over Trial

309  %GetParamValue("&blockid.","DatasetName");
parametervalue=ADVS

310  %GetParamValue("&blockid.","ByVariables");
parametervalue=usubjid, treatment, visit

311  %GetParamValue("&blockid.","WhereClause");
parametervalue=where (TRTEMFL="Y")

312  %GetParamValue("&blockid.","ReportType");
parametervalue=Summary

313  %GetParamValue("&blockid.","Domain");
parametervalue=VS

314  %GetParamValue("&blockid.","Protocol");
parametervalue=CRC123

315  %GetParamValue("&blockid.","Footnote1");
parametervalue=N=total number of treated subjects

316  %GetParamValue("&blockid.","ProgrammerName");
parametervalue=John Doe
```

II. CLASSIFICATION OF REPORTS ACCORDING TO LAYOUTS AND ANALYSES

Even though large organizations streamline their coding needs by reliance on standard macros, often the number of such macros easily runs into several hundred. This is mainly because reports are seen as unique entities, often with the user only supplying titles and footnotes. A novel concept being promoted in this paper is that reports have dual characteristics: analysis type AND layout type.
Let's consider the three tables below:

(1) VS:

	Trt-A	Trt-B
RESP.RATE		
N	xxx	xxx
Mean	xx	xx
SD	x.x	x.x
Median	xx	xx
Min	xx	xx
Max	xx	xx

(2) AE:

	Trt-C	Trt-D
CARDIAC	xx	xx
Palpitations	x	x
Tachycardia	x	x
CHF	x	x
Arrhythmia	x	x
Angina	x	x
Atrial Fib.	x	x

(3) LB:

	Grade 0	Grade 1	Grade 2
HEMATOLOGY			
Hemoglobin	xx	xx	xx
WBC	xx	xx	xx
RBC	xx	xx	xx
Hematocrit	xx	xx	xx
Neutrophil	xx	xx	xx
Platelet	xxx	xxx	xxx

One can see that they all have a similar pattern of row labels: “One-Parent Several-Children Rows” or “pcRows”. So programmatically they all belong to the same layout class, regardless of the number and type of columns. By classifying reports according to structural layouts, only a few layout macros would need to be created.

Analysis types:

Most clinical reports involve just a handful of analysis types, mostly: descriptive-statistics, inferential-statistics, categorical-frequency, shift-tables, survival-analysis, etc. The Parameter Library can therefore make available a few analysis macros classes. So for a specific report, a block would contain the proper combination of the analysis macro class and the layout macro class particular for that report as illustrated by the two tables below for Adverse Events and Vital Signs. From the Parameter Library the two tables retrieve the same class of layout macro: “%pcRows”, but different classes of analysis macros: “%Counts” for Adverse Events and “%DescStats” for Vital Signs. The actual row labels and column headers would be data-driven and not dependent on the macros. However, a programmer can choose to include those in the Parameter Library as well, retrievable by a helper macro.

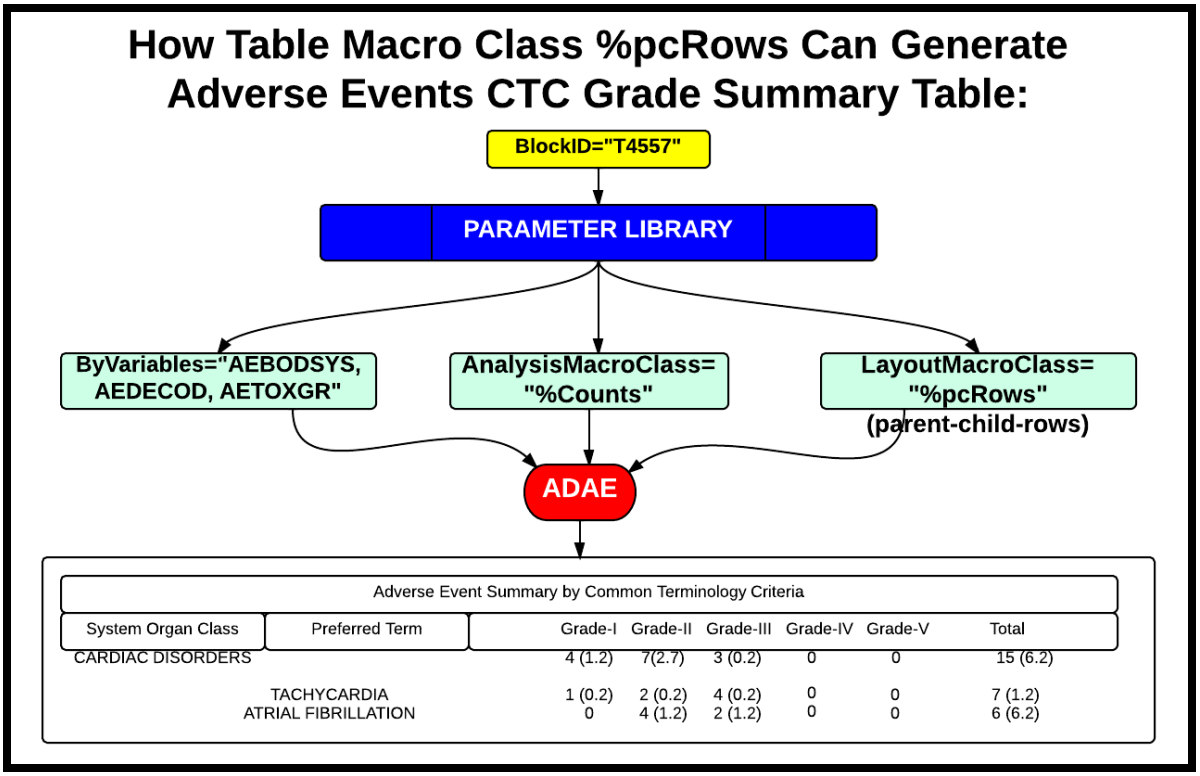

Figure 1. AE Summary Table with Layout Macro Class “pcRows” and Analysis Macro Class “%Counts”.

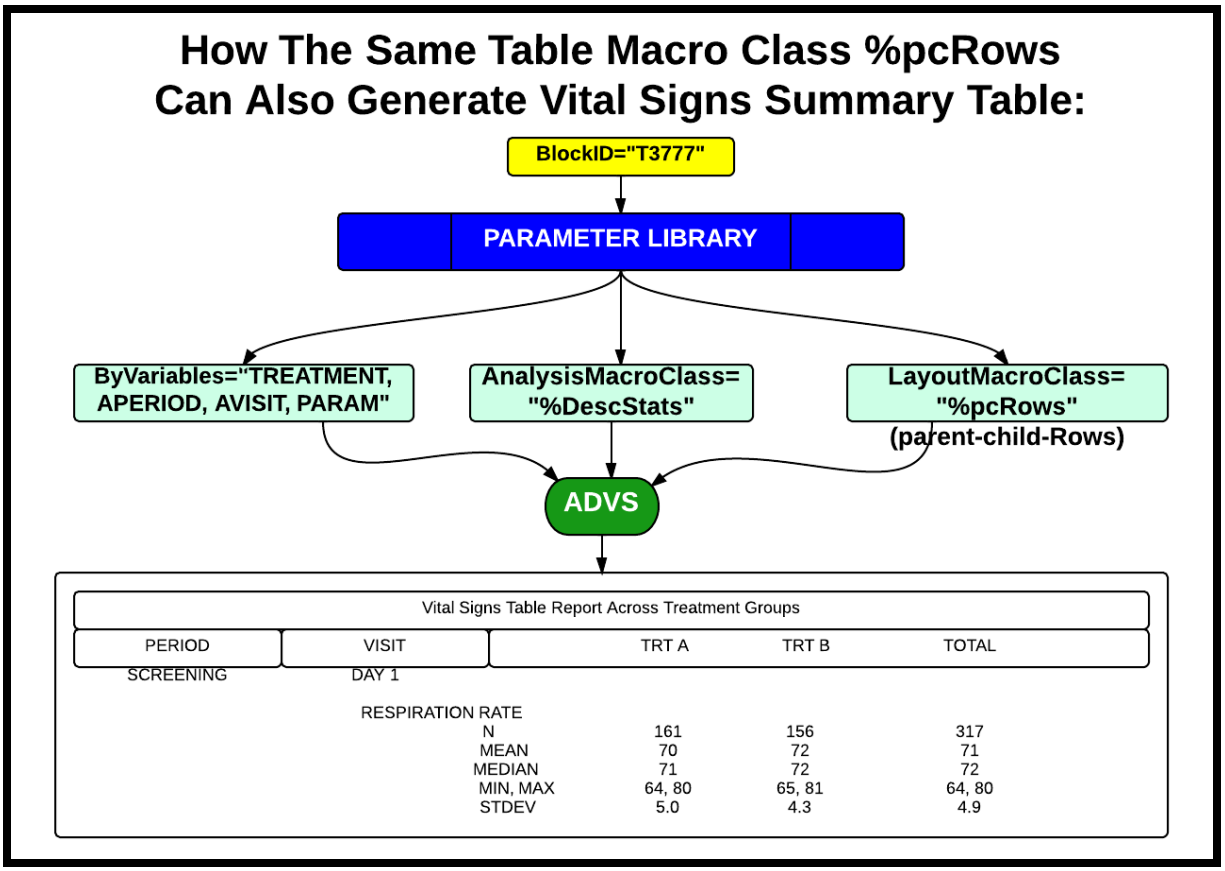

Figure 2. VS Summary Table with Layout Macro Class "pcRows" and Analysis Macro Class "%DescStats".

The problem of varying number of columns:

A robust report program should be able to accommodate a varying number of columns to be useful, and yet output procedures like Proc Report, by the nature of their syntax, require a fixed number of items in the COLUMN statement and a corresponding fixed number of DEFINE statements. The macro facility allows this problem to be easily solved as shown below:

First, items for the column statement are retrieved from the Parameter Library:

```
%GetParamValue("&blockid.","ColumnHeaders");
```

which would yield, for BlockID T3007, &columnheaders as: TRT-A, TRT-B, TRT-C, TOTAL.

Using the *countw* function, the number of items can be computed, enabling a macro do loop and the %scan function to iterate to supply the items for the column statement, as well as the items for the define statement, as shown below:

```
%macro ReportGenerator();
%GetParamValue("&blockid.","ColumnHeaders");
%let itemcount=%sysfunc(countw("&columnheaders", ","));
%do n=1 %to &itemcount;
%let item&n=%scan(%bquote(&columnheaders.),&n.,",");
%end;

proc report data=RRD nowd;
column  %do n=1 %to &itemcount;&&item&n %end;;
%do n=1 %to &itemcount;define &&item&n / width=10;%end;
run;
%mend ReportGenerator;

%ReportGenerator;
```

III. COORDINATING THE ENTIRE PROCESS

Even with the aid of an external look-up table like the Parameter Library, the process of generating clinical reports can be quite intricate. Several helper macros might need to be involved to manage not only access to various sources of input data and to perform "housekeeping" functions, but also to synchronize the various macro calls. A coordinator macro, called "%Pilot" is proposed in this paper for such roles.

The central concept presented in this paper is that, macros involved in the clinical report generation are global in scope, but become study-specific as they receive parameters from the Parameter Library. The process, coordinated by %Pilot, is illustrated in the schematic (Figure 3) shown below:

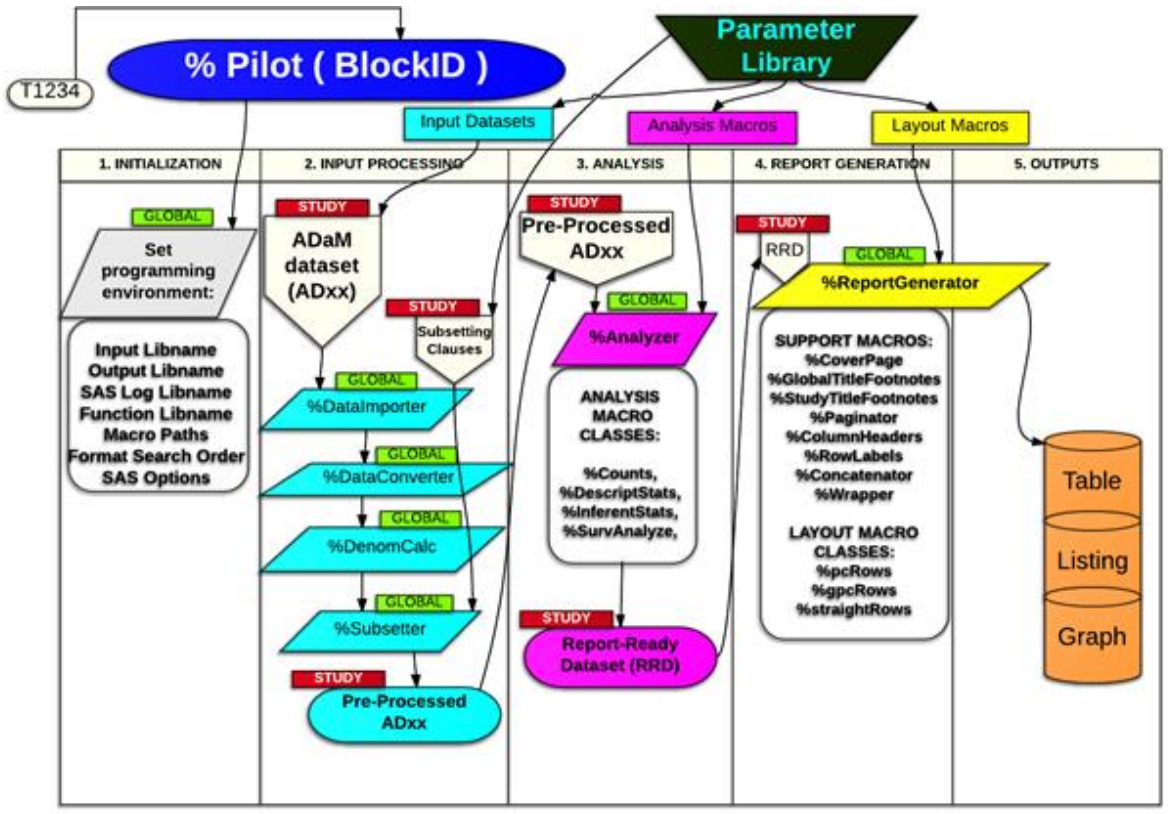

Figure 3. A high-level schematic of the concept of driving four stages of report creation with a parameter look-up table, coordinated by %Pilot

1. %Pilot accepts a BlockID (eg. "T1234") as parameter which then becomes available to the auxiliary global macros.

2. The auxiliary macros use the BlockIDs to retrieve study-specific parameters from the Parameter Library.

3. %Pilot also triggers "housekeeping" functions such as initialization, which configures the programming environment, establishing libnames, macro paths, SAS® options, SDTM and ADaM data paths, format catalogs, etc.

4. Input processing auxiliary macros bring in ADaM dataset files (ADxx) converting them to work datasets, and selecting appropriate records by subsetting according to study and analysis-specific parameters supplied by the Parameter Library via the BlockID.

5. Specific classes of analysis macros are also invoked from the Parameter Library by macros based on the BlockID.

6. The report-generating macro also retrieves report-specific parameters from the Parameter Library to create a specific table, listing, or figure. Figure is an example of a display or screen capture.

The schematic below (Figure 4) illustrates the process of information retrieval with the blockID.

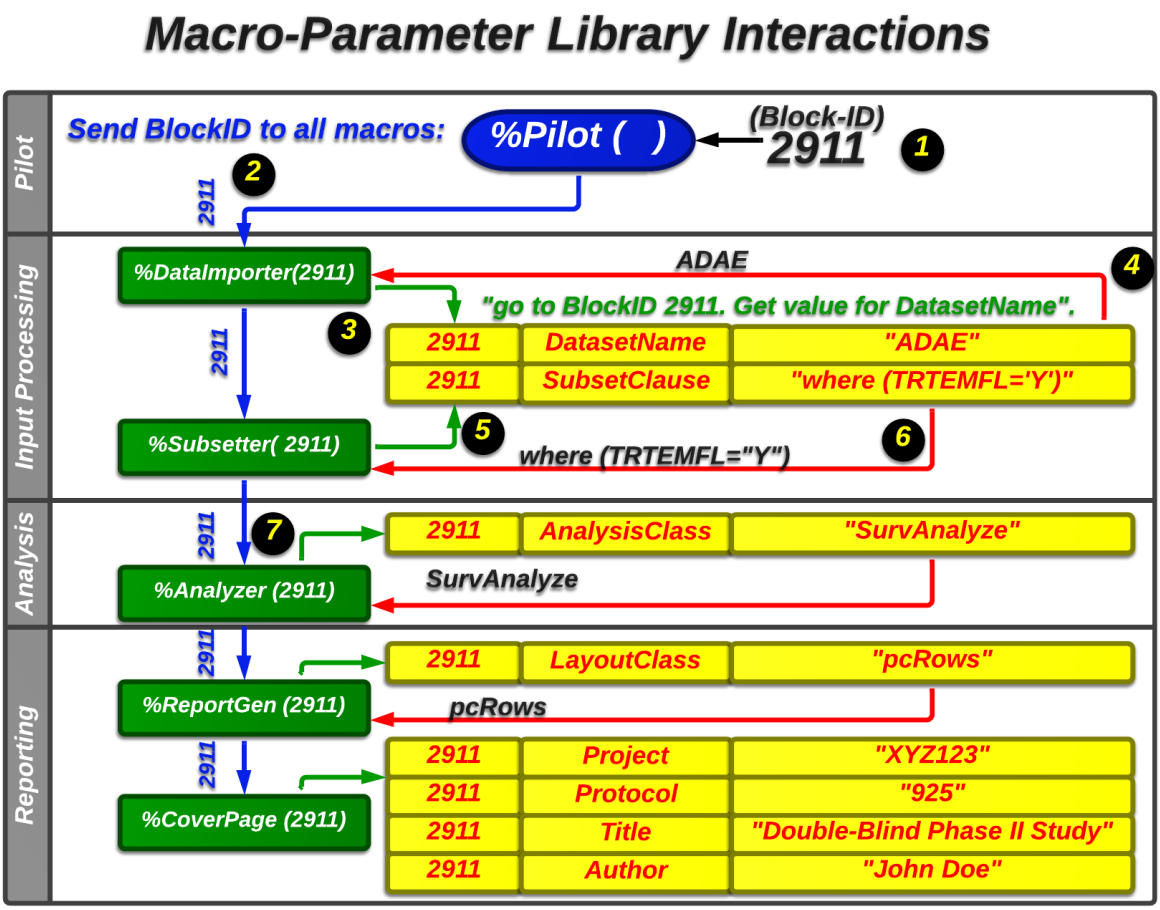

Figure 4. BlockID-based Retrieval of Information by Macros

All the auxiliary macros of %Pilot use %GetParamValues to obtain their specific parameter values the BlockID (“bid”) and the ParamerName (“pname”) are passed to a hash table as earlier described. %GetParamValues creates global macro variables using Call Symputx:

```
%macro GetParamValue(bid, pname);
          data _null_;
          length blockid $11 parametername $27 parametervalue $58;
          if(1=2)then set paramlibrary;
          call missing(of _all_);
          declare hash pl(dataset:"paramlibrary");
          pl.defineKey("BlockID","ParameterName");
          pl.defineData("BlockID", "ParameterName", "ParameterValue");
          pl.defineDone();
          rcc=pl.check(key:&bid., key:&pname.);
          if rcc eq 0 then do;
          rcf=pl.find(key:&bid.,key:&pname.);put ParameterName= ParameterValue=;
          end;
          call symputx(ParameterName,ParameterValue,'g');
          run;
    %mend GetParamValue;
```

The calling auxiliary macros then directly use the parameter macro variables for processing. For example,

(a) the %Analyzer macro would request values for dataset name, domain, and report type as follows:

```
%macro analyzer();
          %GetParamValue("&blockid.","DatasetName");
          %GetParamValue("&blockid.","ReportType");
          %GetParamValue("&blockid.","Domain");
```

(b) then the macro variables are used within the macro to select the appropriate analysis procedure

```
%if ((&domain. in DM VS LB EG CM PE PM) AND (&reporttype. in Summary)) %then
          %do;
          proc means data=&datasetname.;
          class param;
          var basex;
          output out=dstax;
          run;
          %end;
%mend analyzer;
```

Thus, data from the Parameter Library are able to cause the selection of a particular analysis type by an analysis macro.

THE HIERARCHY OF MACRO CALLS

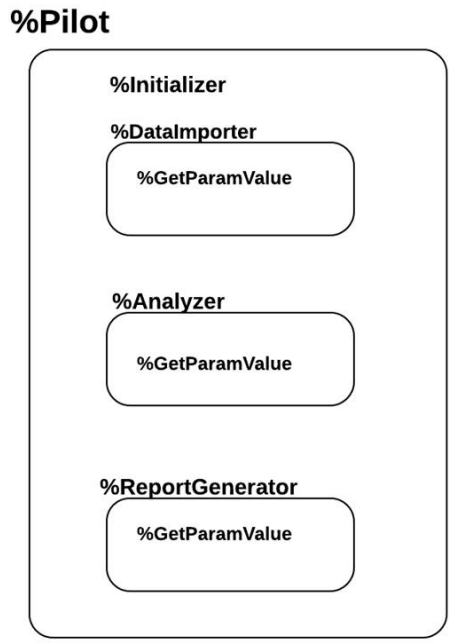

Figure 5. Key Macro Calls Involved in Parameter Library-mediated Report Processing

CONCLUSION

The present paper has proposed a novel approach for handling macro-based clinical report creation. The approach involves the reliance on an external look-up table containing a variety of macro parameters, which are retrieved by a few global macros. The proposed approach has also considered the classification of clinical reports based on structural layouts as well as analysis types. The author believes by having available to macros an external table with a large body of parameters, just a few global macros are requires to generate a large collection of diverse report types. Maintenance, updates, and troubleshooting also become more manageable since parameters are separated from the body of the macro program.

A few of the concepts discussed are already in use in some large organizations. However the concepts are by no means restricted to such organizations, where they deal with just one “client”, which is themselves. Contract Research Organizations (CROs), which tend to deal with a variety of clients, each with its own unique requirements for clinical reports, could also implement the ideas presented in this paper. It is quite conceivable that a CRO could create a separate Parameter Library for each client, and if the global macros are well designed for flexibility, could even utilize the same set of global macros for all clients.

Future considerations for enhancement of the approaches presented include using machine learning and other artificial intelligence techniques for analyzing and classifying report layouts, for the sake of robust global macro design.

CONTACT INFORMATION

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

Joseph W. Hinson, PhD
Accenture Life Sciences
1160 W. Swedesford Rd
Berwyn, PA 19312
1-610-407-7580
joehinson@outlook.com