A Data-Driven Macro Automating the Data Presentation Process by Generating Tailored, Customizable SAS Code

Relax, let %TABGEN do your work!

John Morrill, Quintiles, Inc., Kansas City, MO.
Kristi Wiser, Quintiles, Inc., Kansas City, MO.
Jay Zhou, Quintiles, Inc., Kansas City, MO.

ABSTRACT

In many clinical trials it is useful or even required, perhaps for quality control purposes, to generate a selected data printout of desired variables in any/all datasets for a given directory. This paper presents a simple-to-use yet feature-laden macro designed to lighten this time-consuming task. The macro, %TABGEN, generates for each dataset an independent SAS program containing a tailored, customizable PROC REPORT. %TABGEN gleans dataset attributes using PROC SQL and SCL functions to generate a detailed DEFINE statement with appropriate width, format, and label options. %TABGEN allows sub-setting, the global dropping or keeping of variables, and the specification of ID and order status as well as line and page break variables. Several header, report type, and directory specification options are also available, and empty datasets are processed to show dataset structure. %TABGEN produces auxiliary files containing set-up information and code to run all tabulations together. %TABGEN is robust across multiple platforms. This utility is a valuable tool for both beginner and expert, and the material is presented with this range in mind.

INTRODUCTION

In clinical trial studies, data collected by the case report forms (CRF) are double-entered into database management systems. To ensure that the database structures have been set up correctly and all CRF data have been keyed into data modules appropriately, a common practice is to manually check data for randomly selected subjects against their CRFs. This process is generally done with PROC PRINT, and all CRF data have been double-entered into database management systems. In clinical trial studies, data collected by the case report forms (CRF) are double-entered into database management systems. This task is generally done with PROC PRINT, and all CRF data have been double-entered into database management systems. This process is often done with PROC PRINT, and all CRF data have been double-entered into database management systems.

For this reason, it is desirable to use SAS to programmatically generate source code that creates a tabulation for each dataset. There are several SUG papers discussing how to automate this task by using SAS to generate SAS code. Yao (2000) discussed several techniques using PROC SQL to create macro variables for patient numbers and dataset names and to generate a dataset containing dataset names and variables for sorting and formatting purposes. These techniques could be employed in automating the process of audit listing production. Johnson (2001) discussed some techniques how to generate SAS code for data and procedure steps by utilizing data from the view tables in the SASHELP library. Reading (2001) described a method with which ASCII files containing SAS code for the variable names from a spreadsheet are called with `%include statements in a data step for a data transfer task. Graebner (1998) demonstrated how to obtain dataset structure information from PROC CONTENTS to automate data tabulation production for all datasets of a given data library. Two macros are used for this. One macro generates basic PROC SQL or PROC REPORT code which can be used to create a simple data listing with all variables of a dataset. The other macro creates the macro variable names used in the `do loop for the macro calls to the code generator. The macros reduce programming time of PROC SQL or PROC REPORT significantly. However, the generated code needs to be altered if changing the variable display order or dropping variables.

This paper presents a simple-to-use yet feature-laden SAS macro, %TABGEN, designed to lighten the time-consuming task of creating data tabulations. The macro generates for each dataset an independent SAS program containing a tailored, customizable PROC REPORT and uses it to generate nicely-formatted output for the purpose of a data audit. Before discussing the automation process and technical details, the macro parameter specifications of %TABGEN will be explained. Example calls to the macro will also be given to demonstrate the utility of the macro. The macro, along with all SAS code presented in the paper, was developed and tested with SAS V6.12 and V8.2 under the Windows/NT and UNIX platforms.

MACRO PARAMETER SPECIFICATIONS

%TABGEN has 13 keyword parameters that provide optimal processing flexibility to control file location and selection operations, titling features, population and variable exclusion options, and enhanced PROC REPORT options. The full macro specification is as follows (defaults shown):

```
%tabgen(datalib=, data=, prglib=, outlib=, rtype=tabulation, subset=, drop=, split=*, idvar=, corder=, skipvar=, pagevar=, header=both)
```

The first four parameters are clerical parameters used to specify input and output directories and designate the individual datasets to process. The `datalib` parameter is an input parameter specifying the directory containing the raw data. `prglib` and `outlib` are output parameters designating the target locations of the programs and tabulations, respectively. These three parameters may be either SAS librefs or physical path names. The `data` parameter specifies one or more raw dataset(s) to process, or allows specification of datasets to be excluded. By default, all datasets in the input directory are processed.

The `rtype` parameter triggers the type of display of titles and footnotes. The parameter can be defined with one of three values to specify the type of output to be generated: tabulation, listing, or table. Sponsor, project, protocol, filename, date, page numbering, and special publishing characters are all candidates for inclusion in titles.
Both subset and drop parameters act as filters to screen the data to be reported. The subset parameter allows the user to generate a report for a portion of the observations. For instance, only the data for Visit 1 are reported with if subset = %str(visit=1). The drop parameter is used to drop variables in the output. Both are quite useful in avoiding unneeded output. For example, these are useful if all output for a single subject or a 10% quality control review is requested. These may also be valuable if a data validation request comes with the preference to ignore certain system administrative variables common to many datasets. To avoid warnings and errors, %TABGEN ascertains for each dataset whether the subset and drop requests make sense before applying them.

The remaining six parameters enhance the PROC REPORT output. The split parameter specifies the split character used for the SPLIT option in PROC REPORT. The idvar parameter specifies the variables used in the DEFINE statement with the ID and ORDER options. The values of idvar variables are ordered and not repeated from one row to the next until the value changes. They ensure that each row of the report can be identified when a single page cannot contain all columns. The corder parameter specifies the order of variables in the COLUMN statement, subject to the variables defined by idvar taking precedence in case of overlap. The skipvar and pagevar parameters specify arguments and options, respectively, for the BREAK statement. They control line and page breaks each time the value of the variables specified changes. In all of these parameters, adjustment is made for variables that do not exist in a given dataset. The header parameter simply specifies whether the variable name, the label associated with that name, or both are to be used as the DEFINE statement’s column-header.

THE AUTOMATION PROCESS

Figure 1 illustrates the use of %TABGEN. In its present use, a single copy of %TABGEN is maintained in a central location so that updates to it are immediately available to users. Users access %TABGEN through REPORT.SAS which is simply a macro call.

Once the call is made, %TABGEN performs various administrative tasks such as assigning libraries, determining the operating system, and handling case sensitivity issues. %TABGEN then creates SETTINGS.SAS as a repository for libraries and other set-up information; it is used by the programs which produce the PROC REPORT output. %TABGEN then accesses the target directory, pulls off dataset names, and produces RUNALL.SAS, which is simply a “batch” file with a %include for every dataset in the target directory. The main part of %TABGEN looks only at the attributes of the data, not the data itself.

%TABGEN is then ready to produce the individual programs. The name, type, length, format, and label of each variable in each dataset requested is obtained via SCL commands from the target directory data dictionary. With this information, %TABGEN produces for each dataset a stand-alone, ready-to-run PROC REPORT tailored to the specifications of the user. Each PROC REPORT is in a file bearing the same name as the dataset and contains detailed COLUMN, DEFINE, and BREAK statements populated with information from the %TABGEN macro call and the information gleaned from the data dictionary. Finally, these programs are included by %TABGEN to generate the PROC REPORT outputs and are directed to the specified directory.

TECHNICAL IMPLEMENTATION

We now explain the technical implementation with the basic code used in five areas of %TABGEN.

1. Creating the set-up program.

This section shows the code used to create the SETTINGS.SAS program. Note that &prgpath&opstr resolves to the destination location of the settings.sas program and is sensitive to the operating system previously detected. Also note that the system macro variable &sysncp is used to determine the operating system and the location of ancillary files. Other macro variables are parameters specified in the %TABGEN call. SETTINGS.SAS initializes pathnames and other particulars common to all programs. The final individual programs are designed to run independently so look to SETTINGS.SAS to provide this standardization.

```sas
/** Generate the settings.sas program ***/;
filename ssasref1 "&prgpath&opstr.settings.sas";
  data _null_;  
  file ssasref1 notitle;  
  put @1 '/' @3 69'**';  
  put @1 '/' @95 'Program: ' @120 "settings.sas" @60 "&sysver" ;  
  put @1 '/' @95 'Author: ' @120 '%TABGEN macro';  
  put @1 '/' @95 'Created: ' @120 "&sysdate9" ;  
  put @1 '/' @95 'Revision: ' ;  
  put @1 69'**' @70 '*/';  
  put;  
  put @1 'libname datalib 'datalpath'';  
  if &sysncp eq EG WIN then do;  
  put @1 '** Include title and SAS to Word macros **';  
  put @1 "%inc 'c:\development\maclib\tf.sas''";  
  put @1 "%inc 'c:\development\maclib\tf.sas''";  
end;  
put;  
put @1 '** Define global macro variables used in the individual programs **';  
put @1 ' %let outpath= "soutpath'';  
put @1 ' %let prgpath= "&prgpath'';  
put @1 ' %let subset= "&subset'';  
run;
```

Figure 1. Overall process of %TABGEN
2. Creating the batch program to run the final individual programs.

The code that generates RUNALL.SAS is shown below. The SQL code creates macro variables of dataset names from the dictionary table. This SQL code is performing a similar function that others have used PROC CONTENTS to perform (Graebner, 1998). The data null creates a header then one %include for every dataset requested. The purpose of RUNALL.SAS is to enable the user to execute the individual programs all at one time. As code for the individual programs may have been manually customized, running %TABGEN again would wipe out these changes. For this reason, the intent is for %TABGEN to be executed only once.

```sas
filename sasref2 "&prgpath&opstr.runall.sas";
proc sql noprint;
create table _dataset as
select memname
from dictionary.tables
where libname eq "&datalib";
quit;
data _null_; set_dataset end=_eof;
file sasref2 notitle;
if _n_ eq 1 then do;
put '@1 /*include "&prgpath'&opstr=settings.sas"';
end;
file sasref3 col=col notitles;
if _n_ eq 1 then do;
put '@1 /*include "&prgpath'&opstr.settings.sas"';
end;
memname=lowcase(memname);
put '@1 /*include "&prgpath'&opstr=settings.sas"';
run;
```

3. Obtaining data attributes.

The code below generates the dataset used to populate the PROC REPORT. With &i as a looping variable and &sasin=%scan(&files,&j,%str( )), the following portion of a DATA STEP illustrates the use of SCL functions to obtain dataset attributes. This code produces a file containing an observation for every variable in every dataset requested. Several statements have been omitted but the variables that are created in this DATA STEP for later use are name, type, length, format, header, byvar, pagevar, skipvar, varpos, and width. The _null_ that creates the PROC REPORTs uses nothing more than this _content dataset and the %TABGEN parameter specifications. The explanation of the SCL functions is outside the scope of this paper; see SAS Component Language Specifications. The explanation of the SCL functions is outside the scope of this paper; see SAS Component Language Specifications.

```sas
executed only once.
changes. For this reason, the intent is for %TABGEN to be
individual programs may have been manually
customized, running %TABGEN again would wipe out these
changes. For this reason, the intent is for %TABGEN to be
executed only once.

/* obtaining data content with SCL functions */
data _content;
length name $8 type $1 format $12 header $51;
if label ne '' then
  header=trim(name)||'|'||trim(label)||'';
else header=name;
end;
dsid=close(dsid);
r
```

4. Generating PROC REPORT code.

With the dataset containing all the relevant information in hand, the fourth area is code used to actually generate the separate files containing a PROC REPORT for each requested dataset. The first data _null_ creates the header information and the PROC REPORT up to and including a column statement populated with the requested variables. The second data _null_ picks up with the DEFINE statements, thoroughly populating each line with information such as format, justification, label, etc. It then places the break lines and closes the program. For details on PROC REPORT, see Smith (2000).

```sas
/* code to generate the PROC REPORT code */
filename sasref3 "&prgpath&opstr&filename..sas";
data _null_; set _content end=_eof;
file sasref3 col=col notitles;
if _n_ eq 1 then do;
  put '@1 /*include "&prgpath'&opstr.settings.sas"';
end;
memname=lowcase(memname);
put '@1 /*include "&prgpath'&opstr=settings.sas"';
run;
```
5. Processing the entire directory.

Surrounding part four above, the code that generates the PROC REPORT code, is code which loops through each requested dataset in the specified directory:

```
proc sql NOPRINT;
   select trim(libname)||'.'||memname into :files separated by ' ' from dictionary.tables
   where libname EQ "sasprg" %if %length(&data) GT 0 %then
      %str(and memname IN ("&data"));
   quit;
   %let i=%eval(&i+1);
   %do %while (%length(&sasin) GT 0)
      %let filename=%lowcase(%scan(&sasin,2));
      %let i=%eval(&i+1);
      %end;
%include sasreport furnishings=%files;
%let sasinp=%scan(&files,&i,%str( ));
%do %while (%length(&sasin) GT 0)
   %let filename=%lowcase(%scan(&sasin,2));
%end;
%include sasreport furnishings=%files;
%let i=%eval(&i+1);
%let sasinp=%scan(&files,&i,%str( ));
%end;
```

The macro variable &files is created to contain the datasets to be processed. This contains both the pathname and the dataset name. The delimiter is specified as a space so the &files macro variable is of the form &files=lib.a lib.b lib.c lib.d where lib is a libref and a through d are dataset names. If the &data parameter is specified in the %TABGEN call, then the %if %length(&data) GT 0 condition is satisfied and the additional memname IN ("&data") is added to the where clause, limiting what ends up in the &files macro variable. Otherwise, all datasets in the directory are listed in &files. To loop through each of the datasets listed in &files, the iteration variable is initialized to 1 which references the first dataset. Since the delimiter to the %scan function is specified to a blank, the macro variable &sasin is initialized by scanning &files for the characters preceding the first blank. This results in &sasin being set to the first libref.filename combination. A while loop is initialized to process libref.filename combinations until &sasin is empty. The filename assignment pulls off the dataset name since the period serves as a default delimiter of the %scan function. This allows iteration through each dataset, creating the PROC REPORT code for each one. The %include statement runs each individual PROC REPORT program in turn, something that could have been accomplished equivalently by running the RUNALL.SAS program once at the end. After generating the output for the first dataset, the iteration variable iterates to 2. The &sasin variable takes the second dataset name and control is returned to the while statement. This loop continues in this manner until &sasin has incremented through all dataset names.

**EXAMPLE %TABGEN CALL**

Assuming the %TABGEN macro is defined in the autocall facility, the code below represents the entire contents of REPORT.SAS to illustrate a simple %TABGEN call. The PROC REPORT output is found in the Appendix.

```
options ps=57 ls=157;
%let dossier=n;
%let client=ABC;
%tabgen(datalib=e:\ab\project\study\data, rtype=tabulation, subset=%str(patno<='0002004'), idvar=subid vsnorep crfrpgno pgnorep repno, drop=day vsnorep, header=both)
```

The options and macro variable definitions shown prior to the %TABGEN call illustrate tasks which can be accomplished in conjunction with %TABGEN. In this case, the page size, line size, and client name is globally set for all outputs. We also select an ancillary file parameter which customizes titles and footnotes. This ancillary file is beyond the scope of this discussion but may be the topic of a future paper.

The required datalib parameter gives the location of the source data. Since neither prglib nor outlib are specified, the programs and outputs are all directed to same directory that the REPORT.SAS program containing this macro call resides. Since data is not specified, all datasets in the source directory are processed. The subset parameter reduces the number of observations processed to those matching the criteria shown. The idvar parameter specifies the ID and ORDER options in the PROC REPORT DEFINE statement of the given variables. Note that in the example output in the appendix only subid is found in this dataset and the specification of other variables in the macro call does not present a problem. As specified in the drop parameter, variables day and vsnorep will be omitted from datasets containing them. Finally, with header specified as both, both the variable names and labels will be placed in the header of each variable.

**DISCUSSION**

In its present form, %TABGEN automatically detects and adjusts for the Windows and UNIX platforms and could easily be modified to cover others. The raw datasets could be pre-processed, merging in treatment or a population variable which could then be used as a subset variable. The %TABGEN parameter corder specifies the order of the variables with the variable number as the default; an alternative choice of alphabetic order could be added. Finally, %TABGEN could be modified to sort and display an order variable's values by the formatted value rather than only the unformatted value itself.
CONCLUSION

We have shown a method of generating data-driven tabulations that is both simple and full of features. With just a few macro specifications, this method produces PROC REPORT code which can later be modified separately and provided to clients or approving authorities.

REFERENCES


ACKNOWLEDGMENTS

The authors would like to thank Pete Anderson for significant early development work on this macro.

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

CONTACT INFORMATION

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

John Morrill
Quintiles, Inc.
P.O. Box 9708
Kansas City MO 64134-0708
Phone: (816) 767-6000
E-mail: John.Morrill@Quintiles.com

Kristi Wiser
Quintiles, Inc.
P.O. Box 9708
Kansas City MO 64134-0708
Phone: (816) 767-6000
E-mail: Kristi.Wiser@Quintiles.com

Jay Zhou
Quintiles, Inc.
P.O. Box 9708
Kansas City MO 64134-0708
Phone: (816) 767-6000
E-mail: Jay.Zhou@Quintiles.com

Appendix. Output from the example %TABGEN call