

Developing Data-Driven SAS® Programs Using Proc Contents

Robert W. Graebner, Quintiles, Inc., Kansas City, MO

ABSTRACT

It is often desirable to write SAS programs that adapt to different data set structures without being modified. Such programs are referred to as data-driven programs because they assess the structure of the data set they are working with and automatically adapt to that structure. In SAS, the macro language can be used in conjunction with PROC CONTENTS to produce such programs. In this paper examples are provided to illustrate how this technique can be used to reduce programming and maintenance effort in a variety of situations. This paper is intended for experienced SAS programmers who have a basic understanding of the SAS macro language.

INTRODUCTION

The SAS macro language provides powerful capabilities for writing flexible programs that can behave differently depending on the parameters passed to them. A common use of macros is to reduce repetition in programs. An example from the pharmaceutical industry is the need to produce summary listing for each subject with the subject ID included in the report title. To accomplish this, you could write PROC REPORT code and make a copy for each subject then change the ID in each title statement. A more efficient way would be to put the PROC REPORT code in a macro, pass subject ID as a parameter and then use macro variable substitution to place the ID in the title. This method is simple when there are only a few differences between each report, but what do you do when you need reports for many different data sets with different structures? PROC CONTENTS provides a simple solution with its capability of storing data set structure information in a data set. This information can then be stored in macro variables and used to build SAS programming statements tailored to the data set you are working with. For example, in PROC REPORT, the variable names could be used to construct the COLUMN statements. In addition to making your program more generic, you also eliminate many errors. Because the variable names are obtained from PROC CONTENTS, you are assured that all variables will be included and that they will all be spelled correctly. Information on type, length, label and format can be used in a similar fashion to produce a DEFINE statement for each variable. This method can be used to generate SAS programming statements for any SAS procedure that utilizes data set structure information.

There are two basic ways in which this process can be used in your programs. The first is to use macro variable substitution in the source code run by your current session. This has the advantage that your program can be generic and self contained. The second method is to use a DATA _NULL_ step with a series of PUT statements that

use macro variable substitution to create a text file containing SAS source code statements. An advantage of this method is that you can modify the program before you run it. This is helpful when you are not able to handle all coding needs in your macro. It also allows you to give the source code to clients without giving away your macro technology.

METHOD

PROC CONTENTS has several features that make it useful in developing data-driven applications. It can determine the structure of any data set in a SAS library by using the DATA= LIBNAME.MEMBER option or it can process all data sets in a library at once by specifying DATA= LIBNAME._ALL_. By using the OUT= LIBNAME.MEMBER and NOPRINT options you can send the output to a SAS data set and suppress printed output. The resulting data set contains a series of variables describing the data set structures with an observation for each variable in each data set. The most useful variables are listed below.

PROC CONTENTS Variable	Description
LIBNAME	SAS Library Name
MEMNAME	SAS Library Member Name
NAME	Variable Name
TYPE	Variable Type (1= numeric, 2= character)
LENGTH	Variable length
LABEL	Variable Label
FORMAT	Variable Format
FORMATL	Format Length
FORMATD	Format Decimals
INFORMAT	Variable Informat
INFORMATL	Informat Length
IINFORMATD	Informat Decimals

Creating a structure data set is very simple, an example is given below.

```
proc contents
    data= &saslib..&ds
    out=struct
    position
    noprint;
run;
```

The macro variables that indicate the SAS library and data set name to be used are passed as parameters to the macro that contains the call to PROC CONTENTS. The PROC CONTENTS output is stored in a temporary data set called struct. The position option specifies that the observations will be ordered by the location of variables

in the data set rather than alphabetically by variable name. The NOPRINT option suppresses printed output of the PROC CONTENTS results.

The next step is to place the desired information into SAS macro variables. Because these variables are often used in iterative processes, it is desirable to have them in an array. While the SAS macro language does not support arrays, you can simulate arrays (sometimes called pseudo arrays) by using multi-pass macro variable resolution. The following source code creates a pseudo array containing the variable names and types from the data set struct. The SYMPUT function is used to store the data set variables NAME and TYPE into macro variables that have the observation number added to the end of variable name (e.g. var1, var2, etc.) to facilitate referencing them in a %DO loop. The last observation number is stored as well to serve as the upper limit for the %DO loop.

```
data _null_;
  set struct end=last;
  call symput('var' || left(_N_), name);
  call symput('type' || left(_N_), type);
  if last then call symput('numrec', _N_);
run;
```

As mentioned earlier, one use of this information is to use macro variable substitution to form the necessary SAS source code when the macro is resolved. The following example loops through all variables in the data set and calls PROC FREQ for each one.

```
%do i = 1 %to &numrec;
  proc freq data= &saslib..&ds;
    tables &&var&i;
  run;
%end;
```

The macro variable reference &&var&I will be resolved in two passes. When I = 1, the first pass will resolve to &var1 and the second pass will resolve to the string stored in &var1 which will be the name of the first variable in the data set.

Another option is to use a DATA _NULL_ step and PUT statements to generate SAS source code. The example below uses this method to generate PROC REPORT code. The source code is put in the text file referenced in the FILE statements. The MOD option is used so that each successive DATA step will append to the file rather than overwrite it.

```
data _null_;
  set struct end=last;
  file sascode mod;
  if _N_ = 1 then do;
    put / "proc report data=&saslib..&ds
      missing nowindows headline headskip
      split='\ ' column ' @;
  end;
  linelen + length(name);
  if linelen >= 70 then do;
    put / +9 @;
    linelen = 10;
  end;
  put name @;
  if last then put ' ';
run;

data _null_;
  set struct end=last;
  file sascode mod;
  length clabel $ 120 coltype $ 7;
  vnwidth = length( trim(name));
  if name in('PATNO','VISIT') then
    coltype = 'order';
  else coltype = 'display';
  select;
    when(length <= 4)
      clabel = "define " || name || " / " ||
      coltype || " width=" ||
      put( max(vnwidth, 4), 2.)
      || "left;";
    when(4 < length <= 20)
      clabel = "define " || name || " / "
      || coltype || " width=" || put(
      max(vnwidth, length), 2.) || "
      left;";
    when(length > 20)
      clabel = "define " || name || " / "
      || coltype || " width=20" || " left
      flow;";
  end;
  put @3 clabel;
  if last then do;
    put " title1 'QC Listing Report for
      &ds';" /
    'run;';
  end;
run;
```

To start the program generation, a PROC REPORT statement and the associated options are written. Because this is only needs to be done once, before the variable-specific statements are written, an IF statement is used so that this line is only written when `_N_` equals one. The macro variables `&SASLIB` and `&DS` contain the SAS library name and the data set name. Because these macro variables need to be resolved as the source code is generated, double quotes are used to surround the string that contains them. If you need to include a macro variable reference in the source code you generate, use single quotes to enclose the string. The remaining statements in this DATA step are used to create a COLUMN statement that contains the names of all the variables in the data set to be reported from.

The second DATA step is used to create DEFINE statements for each column. This section illustrates how conditional processing can be used to generate source code that is dependent on each variable's attributes. When the length of a variable is less than or equal to four, the column width is set to the maximum of the length of the variable name and four. This guarantees that you will not have any columns narrower than four spaces. When the length is greater than four, but less than or equal to 20, the column width is equal to the maximum of the length of the name and the length of the variable. When the length is greater than 20, the column width is set to 20 and the FLOW option is used for column wrapping. After the last variable is reached, a TITLE statement is generated.

This source code is part of a macro that receives the data set name in the parameter DS. If you need to perform generate source code for multiple data sets, you can write another macro that creates a pseudo array containing the names of the required data sets and then calls the source code generating macro for each data set. An example of such a macro is given below.

```
%macro qcrptgen (saslib, codefile);
  options nolabel nofmterr;
  filename sascode "&codefile";
  /**** CREATE PROGRAM HEADER ****/
  data _null_;
    file sascode mod;
    gendate = put( today(), date9.);
    put "/*****";
    put "  QC Listing for: &saslib";
    put "  Program name   :: &codefile";
    put "  Authors name   :: ";
    put "  Date started   :: " gendate /;
    put "  Source code generated by the";
    put "  QCList Macro.";
    put "*****"/;
run;
```

```
/**** CREATE A DATASET CONTAINING THE NAMES
OF ALL DATASETS IN THE LIBRARY SASLIB ****/
```

```
proc contents
  data=&saslib._all_
  out = libmem
  (keep= memname varnum)
  position
  noprint;
run;
proc sort data=libmem nodupkey;
  by memname;
run;
```

```
/**** CREATE A PSUEDO-ARRAY (D1..Dn) OF
MACRO VARIABLES CONTAINING EACH DATASET
NAME ****/
```

```
data _null_;
  set libmem end=last;
  call symput('d'||left(_N_), memname);
  if last then call symput('numrec',
    _N_);
run;
```

```
/**** CALL THE SOURCE CODE GENERATOR FOR
EACH DATASET ****/
```

```
%do i = 1 %to &numrec;
  %prgen(&d&i);
%end;
%mend qcrptgen;
```

This macro has two parameters; SASLIB, which contains the name of the SAS library to use, and CODEFILE, which contains the full path and filename of the source code file you want to create. The first DATA step generates a program header. The next step uses PROC CONTENTS to create a data set that contains the data structure information for all data sets in the specified SAS library. The purpose of this data set is to provide a list of data sets to generate PROC REPORT source code for. The structure data set created by PROC CONTENTS will have one observation for each variable in each data set. PROC SORT with the NODUPKEY option, using MEMNAME (data set name) as the by variable, is used to create a data set with one observation per data set in the specified library.

The next DATA step uses the SYMPUT function to store the data set names in a pseudo array of macro variables. This allows a %DO loop to be used to call the source code generating macro for each of the data sets.

CONCLUSION

The methods presented in this paper illustrate how PROC CONTENTS and the SAS macro language can be used to increase programming efficiency by creating data-driven programs or by generating SAS source code.

ACKNOWLEDGMENTS

SAS is a registered trademark or trademark of SAS Institute, Inc. in the USA and other countries. ® Indicates USA registration.

CONTACTING THE AUTHOR

Robert Graebner
Quintiles, Inc.
P.O. Box 9708
Kansas City, MO 64134-0708

Email: bob.graebner@quintiles.com
graetech@grapevine.net

Web Site: Quintiles.com
www.grapevine.net/~graetech

M W S U G

Data Management



