Generating SAS® Source Code with SAS® Macros
Robert W. Graebner, Pharmaceutical Research Associates, Inc., Lenexa, KS

ABSTRACT

The SAS macro language is a useful tool for developing utility applications that can be reused on different projects to perform common tasks. This paper illustrates how the SAS macro language can be used in combination with PROC CONTENTS to create a simple SAS source code generator. The program presented in this paper can significantly reduce the development time needed for common processing and reporting tasks that involve all variables in all data sets in a SAS library. This paper is intended for experienced SAS programmers who have a basic understanding of the SAS macro language.

INTRODUCTION

Often it is necessary to generate SAS programs that perform some process for every data set in a SAS library. Examples of this include post processing data sets to match client specifications and generating data set listing reports. Through the use of the SAS macro language and PROC CONTENTS, it is possible to generate a text file that contains SAS source code for procedures such as PROC SQL and PROC REPORT. Such a file can automatically contain statements calling the required procedure for each data set in the specified library. Each of these procedure calls can then specify each variable in a given data set in the necessary statements, such as a SELECT statement in PROC SQL or the COLUMN and DEFINE statements in PROC REPORT. The text file created serves as a starting point for program development and can eliminate a large amount of tedious typing. Once the source code file has been created, it can be opened with the SAS program editor and each section can be modified as necessary.

METHOD

The application presented in this paper consists of two SAS macros, CODEGEN and PRGBUILD. When the PRGBUILD macro is run, it first determines which SAS data sets are contained in the specified library. It then calls the CODEGEN macro for each data set to generate the specified type of SAS source code. PRGBUILD is called with three parameters as follows:

```
%prgbuild(saslib, codefile, codetype)
```

The first parameter is the name of an existing SAS library that contains the data sets to be processed by the program being generated. The second parameter is the name of the source code file to be generated, including the full path. The third parameter is specified as either SQL or RPT, depending on whether you want to generate PROC SQL or PROC REPORT source code.

PRGBUILD uses PROC CONTENTS to create a data set which contains the names of all SAS data sets in the specified library. By specifying DATA = &saslib.._ALL_, the data set created by PROC CONTENTS will contain an observation for each variable in each of the data sets in the saslib library. The output data set is then sorted by memname using PROC SORT with the NODUPKEY option so that there will be only one observation per data set. A WHERE clause allows you to exclude the specified data sets, which in this case are system files from a data management application. The SYMPUT function is used to put all of the data set names into a pseudo array of macro variables. The macro variable d1 contains the first data set name, d2

contains the second and so on for all data sets in the library. The total number of data sets to be used is stored in the macro variable numrec. When the CODEGEN macro is called, the name of the data set to be used is specified with the macro variable reference &&d&i, which is resolved in two passes. When i equals 1, the first pass will resolve to &d1. On the second pass, &d1 will be replaced with the data set name contained in the macro variable d1. A DO loop is used to call the CODEGEN macro for each data set, passing the name of the data set to be used for each call.

```
/*************** PRGBUILD MACRO **************/

%macro prgbuild (saslib, codefile, codetype);
  proc contents data=&saslib.._all_ out = libmem
  (keep= memname varnum) position noprint;
  run;
  proc sort data=libmem nodupkey;
    by memname;
    where memname not in('CORRECT','DIRLIST',
                         'ENROLL','INVEST',
                         'JOURNAL','PAGES',
                         'RSLVLOG','STATLOG',
                         'TRACK','TRANJRN1',
                         'TRANJRN2','_TRACK_');
  run;
  data _null_;
    set libmem end=last;
    call symput('d'||left(_N_), memname);
    if last then call symput('numrec', _N_);
  run;
  %do i = 1 %to &numrec;
    %codegen(&&d&i);
  %end;
%mend prgbuild;
```

CODEGEN generates a text file that includes either PROC SQL or PROC REPORT statements which specify all variables in the data set whose name was passed. PROC CONTENTS is used to create a data set that contains the names, lengths and labels of all the variables in the current data set. A DATA NULL step is then used to process each observation, using PUT statements to add the necessary SAS statements to the output text file. It is important to use the correct type of quotation marks in the PUT statements, double quotes when the included macro variable references are to be resolved during the execution of CODEGEN and single quotes when they are to be copied verbatim to the source code text file.

If PROC SQL code is generated, a SELECT statement is created with two lines for each variable in the data set. The first line specifies the variable name and the second line specifies the label associated with that variable. This facilitates easy modification for applying formats to variables with the INPUT function, creating derived variables, transforming variables or deleting unwanted variables. Note that both of the lines for a given variable end with a comma and therefore the program generated will not run without modification. This was done because most of the variables are simply copied over to the output data set without being transformed. For those variables, the entire line with the label statement is deleted. For variables that require the label specification, the comma at the end of the first line must be deleted.

If PROC REPORT code is generated, a COLUMN statement listing all variables in the data set is created and then DEFINE

statements are created for each variable as well. The formatting of the report columns depends on the length of the variable. If the length is 5 or less, the column width is set to 5 and the column is centered. If the length is from 6 to 20, then the column is left justified. If the length is greater than 20, the column width is set to 20 and text flow is used to wrap text within the column.

```
/*************** CODEGEN MACRO ***************/

%macro codegen(ds);
  filename sascode "&codefile";
  proc contents data= &saslib..&ds out=struct
    position noprint;
  run;
  proc sort data=struct;
    by varnum;
  run;
  %if &codetype = SQL %then %do;
    data _null_;
      set struct end=last;
      file sascode mod;
        if _N_ = 1 then
          put /"proc sql;"/
              "   create table dest.&ds as"/
              "      select";
          clabel ="label='"||trim(label)||"',";
          cname = trim(name) || ",";
          put @10 cname /  @10 clabel;
          if last then
            put /"       from &saslib..&ds" /
                "       order by barcode;";
    run;
  %end;
  %else %if &codetype = RPT %then %do;
    data _null_;
      set struct end=last;
      file sascode mod;
      retain linelen;
      if _N_ = 1 then do;
        put /"proc report   data=&saslib..&ds"||
            "missing nowindows headline "||
            "headskip split='\';"/
            '   where patid = &pid;'/
            '   column ' @;
        linelen = 0;
      end;
      if name in('PROTO') then delete;
      linelen = linelen + length(name);
      if linelen >= 70 then do;
        put / +10 @;
        linelen = 10;
      end;
      put name @;
      if last then put ';';
    run;
    data _null_;
      set struct end=last;
      file sascode mod;
      if name in('PROTO') then delete;
      select;
        when(length <= 5)
          clabel = "define " || name ||
                   "/group width= 5" ||
                   " center    '" ||
                   trim(compress(label,"'")) ||
                   "';";

        when(5 < length <= 20)
          clabel = "define " || name ||
                   " /group width=" ||
                   put(length, 2.) ||
                   " left     '" ||
                   trim(compress(label,"'")) ||
                   "';";
        when(length > 20)
          clabel = "define " || name ||
                   " /group width=20" ||
                   " left flow '" ||
```

```
                   trim(compress(label,"'")) ||
                   "';";
      end;
      put @4 clabel;
      if last then do;
        put "    title1 'Listing for &ds';" /
            'run;';
      end;
    run;
  %end;
%mend codegen;
```

CONCLUSION

The programming techniques presented in this paper can be used to create utilities that generate source code for a wide variety of applications in SAS. While the source code generated generally requires modification to produce the exact results required, significant savings in development time can be realized due to the elimination of tedious, redundant typing. The next step planned in the development of these macros is to enhance CODEGEN with an algorithm that processes the label in the PROC REPORT section, inserting split characters based on the width of the columns.

ACKNOWLEDGMENTS

CONTACTING THE AUTHOR

Robert Graebner
PRA, Inc.
16400 College Blvd.
Lenexa, KS 66219

graebnerbob@pra-ww.com
rwgraebner@msn.com