Automatic Documentation: Using SAS® to Create the Data Dictionary, Data Structure, and Process Flow Tables

Arthur L. Carpenter

California Occidental Consultants

KEY WORDS

CONTENTS, DATA DICTIONARY, DOCUMENTATION, DATA STRUCTURE, PROCESS FLOW

ABSTRACT

The documentation of SAS programs and data sets is critical, yet it is always the last thing that is done (when it is done at all). Documentation that does exist is rarely updated and often not accurate. Usually this is because the process of producing and maintaining the documentation is too labor intensive and too time-consuming. If we expect our programmers to produce and maintain the documentation for the programs they write, the documentation process needs to be as automatic and painless as possible.

Welldocumented programs, and especially systems of programs, should include a Data Dictionary (definition and attributes of each SAS variable and an exhaustive list of the data sets in which it occurs), a Data Structure Table showing the variables contained in each SAS data set and a Process Flow Description Table.

This paper discusses the types of information expected in well-structured programs and their associated documentation. Two SAS programs are presented that can be used to automatically generate the three tables described above. The generation of these tables need no longer be a burden: SAS can do the hard work for us.

INTRODUCTION

The process of developing and maintaining documentation for programs can be tedious, time-consuming, and labor intensive. Very often the documentation is developed long after the programs have been written, and although we as programmers have good intentions, the quality of the documentation may leave something to be desired. Later, when the programs are modified, the documentation rarely is kept up to date.

A set of tools is needed so that the documentation process can be made easier and more accurate. These tools are not provided directly as part of the SAS System. However, SAS does provide some aides and can be programmed to create these tables. Once automated, the documentation process becomes substantially easier. The programmer can generate a new data dictionary and associated tables easily each time the programs are changed.

SAS maintains internal documentation on each data set. This information is stored on the descriptor record and can be displayed using PROC CONTENTS or the VAR window. This information is usually displayed in the OUTPUT window or printed. However, most SAS programmers are unaware that PROC CONTENTS will also create a data set describing the attributes of each data set in the system. When cross-referenced across data sets, this information can be used to generate the Data Dictionary and the Data Structure Tables.

This process is illustrated for this paper by documenting a SAS program (AIRQUAL3.SAS) which was written to summarize temperature and humidity data collected within California over the last two decades. This program is one of several forming a system of programs to monitor and analyze various aspects of California air quality information.

PROGRAMMING REQUIREMENTS

In order to take full advantage of the various aspects of internal documentation available within SAS, the SAS programmer must conform to several program standards. Once these standards are in place, it is possible to write documentation programs using SAS to create the documentation.

The following standards have been adopted by California Occidental Consultants and are utilized by the programs that follow. Standards can vary and the programs that perform the automatic documentation can be easily adapted.

Data step requirements

- Each step will start with at least one comment containing that step's Process Step Number.

- A KEEP (not a DROP) statement will be used whenever a data set is created.
- Each step should end with a RUN; (or QUIT;).

Data set requirements

- Each data set name will be unique within a system of programs.
- Each data set will have a data set label specified.
- Data set labels will contain the Process Step Number.

Variable requirements

- Each variable name will be unique within a use.
- Each variable will have a label.

Data Step Requirements

The data step requirements provide more for a uniform style than as an aide to the programs discussed here. The exception is the use of the Process Step Number.

In structured methodologies, programming steps are identified by a series of levels of increasing complexity. Level 0 is usually a broad brush overview of the program system. Level 1 is usually a sequence of steps; for most SAS jobs, level 2 identifies each individual job step (PROC step or DATA step). Starting each step with at least one comment containing that step's Process Step Number(s) allows both the programmer and the Autodoc program to identify steps.

A KEEP (not a DROP) statement is used whenever a data set is created because the KEEP statement tells the programmer what variables are in the data set. The DROP only tells us what is not in the data set.

Ending each step with a RUN; (or QUIT;) helps the programmer identify breaks within the program. Usually these are redundant, however, the presentation of the comments in the log will then be associated with the correct step.

Data Set and Variable Requirements

The data set and variable requirements are more important to the AUTODOC programs, because the program takes advantage of the uniqueness of data set and variable names when establishing relationships.

Within SAS, each data set and variable name can be used over and over again, however if a name is unique within a system of programs the programmer is less likely to become confused. The creation of the data structure table (discussed below in more detail) is also simplified. The AUTODOC program creates a cross reference among data sets and their variables and the process becomes unwieldy if names repeat within different contexts. Throughout a system of programs, variable and data set names should have one and only one definition.

The labels of both variables and data sets are accessed and noted by the AUTODOC program. If the programmer fails to include this information, the documentation becomes much less valuable. Since data sets will always be first created within a specific step, including that Process Step Number within the data set label adds an additional cross reference.

PROCESS FLOW DESCRIPTION TABLE

The Process Flow Description Table provides the programmer with an overview of the flow of the program. Ideally this table will accompany a Process Flow Diagram (flow chart). The Process Flow Description Table usually takes the form of an outline with a description of each program step.

TABLE 1 is a portion of a Process Flow Description Table. The Level 0 description for the weather station data is P3. The Level 1 process, (P3.1) in this example, processes ASCII data and the second Level 1 process (P3.2 - not shown) estimates missing temperature data. Within the Level 1 processes are Level 2 processes. Shown here is P3.1.1 which reads specific ASCII temperature data. In most SAS programs Level 2 processes are individual SAS steps.

The SAS program (MAKEPROF.SAS) was used to generate the above table reads SAS code as data and interprets comments as the input for the table. A portion of the program (AIRQUAL3.SAS) from which the outline was generated is shown in TABLE 2.

Comments beginning with a double asterisk (**) are also included in the Process Flow Description Table. The MAKEPROF.SAS program automatically indents the process levels.

```
P3 Process the weather station data
   P3.1 Read the ASCII file and create a SAS dataset of temp
       and hum.
        P3.1.1 Read the ASCII temperature data
               stations include OAK, SJC, SFO, BUR, LAX, LGB,
              or SAN
        P3.1.2 Read the ASCII humidity data
        P3.1.3 Eliminate overlapping observations for each
            data type
```

TABLE 1 Portion of a Process Flow Description Table.

```
*P3 Process the weather station data;
*P3.1 Read the ASCII file and create a SAS dataset of temp and
hum.;
*P3.1.1 Read the ASCII temperature data;
data teml (keep=stn date maxtem mintem meantem yy mm
           label='P3.1.1 Monthly temperature data');
infile rawtdata missover;
input @1  stn $3.
      @5 yy 2. mm 3.
      @10 maxtem 5.1 mintem 5.1 meantem 5.1
      ;
** stations include OAK, SJC, SFO, BUR, LAX, LGB, or SAN;
if stn='OAK' or stn='SJC' or stn='SFO' or stn='BUR' or
stn='LAX' or stn='LGB' or stn='SAN';
```

TABLE 2 A portion of the program (AIRQUAL3.SAS) from which TABLE 1 was generated.

```
WORK     TEM1      P3.1.1 Monthly temperature data

      DATE       sample date
      MAXTEM     maximum monthly temperature
      MEANTEM    mean monthly temperature
      MINTEM     minimum monthly temperature
      MM         month of sample
      STN        three digit weather station code
      YY         year of sample


WORK     TEM2      P3.1.3 Eliminate overlapping values

      DATE       sample date
      MAXTEM     maximum monthly temperature
      MEANTEM    mean monthly temperature
      MINTEM     minimum monthly temperature
      MM         month of sample
      STN        three digit weather station code
      YY         year of sample
```

TABLE 3 Data Structure Table

```
DATE      sample date

     Attributes
       TYPE   LENGTH        INFORMAT      FORMAT    JUST.
     ______________________________________________________
        N        8                        DATE7.      R

     Database Cross Reference
     SASDATA.WEATHER        WORK.HUM1           WORK.HUM2
     WORK.HUM3              WORK.HUM4           WORK.TEM1
     WORK.TEM2              WORK.TEM3           WORK.TEM4


MEANTEM  mean monthly temperature

     Attributes
       TYPE   LENGTH        INFORMAT      FORMAT    JUST.
     ______________________________________________________
        N        8                                    R

     Database Cross Reference
     SASDATA.WEATHER        WORK.TEM1           WORK.TEM2
     WORK.TEM3              WORK.TEM3MEAN       WORK.TEM4
```

TABLE 4 Portion of the Data Dictionary

DATA STRUCTURE TABLE

The Data Structure Table provides the user with information concerning the structure of individual data sets. This information is very similar to that which is generated using PROC CONTENTS, and in fact this table is often created by using the listing generated with a PROC CONTENTS. The program used to generate the Data Structure Table (TABLE 3) is called MAKEDICT.SAS, and it saves the information from the PROC CONTENTS in a SAS data set and is therefore able to print the table in a user defined format.

The table is ordered by LIBREF and data set name and includes the data set label (when provided) and a list of all the variables within that data set. In this example the data set label also contains the Level 2 process number which provides a cross reference to the Process Flow Description Table.

DATA DICTIONARY

The Data Structure Table provides information concerning individual data sets, however it does not give the user/programmer any information on the individual variables. This is done in the Data Dictionary which provides not only additional PROC CONTENTS type information, but also a cross reference list of all the data sets in which that variable appears.

The segment of the Data Dictionary shown in TABLE 4 highlights two of the variables shown in the above data sets.

In addition to noting each LIBREF.data-set which contains the variable, the variables individual attributes have been printed. These include type (Character or Numeric), length, informat, format, and justification.

USING THE PROGRAMS

Two short programs were written to create these three tables. MAKEPROF.SAS creates the Process Flow Description Table by reading the SAS code of the program to be documented. The Data Dictionary and the Data Structure Table are created by a second program (MAKEDICT.SAS) that uses PROC CONTENTS to obtain the necessary information.

The MAKEPROF.SAS program searches a SAS job for comments beginning with '*P' or '**' and reprints the comment as a character string. The '*P' comments are indented for the various levels of the process. Level 0 process statements will take the form of '*Px' where x is the level 0 process number. Level 1 statements are expected to be in the form of '*Px.y' where y is the level 1 process number. And similarly for level 2 which takes the form of '*Px.y.z'. Indentations are handled automatically when the statements are printed and the input program steps are assumed to be in order.

The data set information used by MAKEDICT.SAS to create the Data Dictionary and the Data Structure Table is obtained through PROC CONTENTS with a DATA=_ALL_ option.

Prior to running MAKEDICT.SAS, the programs to be documented must be executed. This establishes all the work files at one time so that variable relationships may be determined. The program used in this paper checks two libraries, one of which is the WORK library. Additional libraries could easily be specified and OPTIONS OBS=0 can be used to create the data sets for this step.

ABOUT THE AUTHOR

Arthur L. Carpenter has over fifteen years of experience as a statistician and data analyst and has served as a senior consultant with California Occidental Consultants, CALOXY, since 1983. His publications list includes a number of papers and posters presented at SUGI and he has developed and presented several courses and seminars on statistics and SAS programming.

CALOXY offers SAS contract programming and in-house SAS training nationwide.

AUTHOR

Arthur L. Carpenter
California Occidental Consultants
4239 Serena Avenue
Oceanside, CA 92056-5018

(619) 724-8579

TRADEMARK INFORMATION

SAS is a registered trademark of the SAS Institute, Inc., Cary, NC, USA.

```
* makeprof.sas
*
*************************************************************************
* Purpose of the program
* Read SAS code as data and keep the Process descriptions.  Write out
* the descriptions to an ascii file.
*
* Written by:  Art Carpenter
*              CALOXY
*              (619) 724-8579
* Written on:  20 June 1991
*
* Modified on: 11sep91
* notes:       Prepared for sugi.
*
*
*************************************************************************;
filename outtxt '\sugi17\autodoc\procflow.txt';
*************************************************************************;

proc datasets library=work;
delete all;
run;

%macro readcode(job);
*
* read sas code with a fn of &job and save the process statements.
*;
filename job "\sugi17\autodoc\&job..sas";

**************  statements are of the form*********;
*P1 Process gaseous air quality data;
*P1.1 Read the ASCII file and create a SAS dataset containing CO and O3;
*P1.1.1 Read the ASCII gas data;
** this type of comment, with a double (**), will be included also;
*****************************************************;

* read a sas job, save the process statements;
data a1 (keep=indent str);
infile job missover;
length str $80;
input @1 process $9. @;
if process =:  '*P' then input @2 str $char.;
else if process =: '** ' then input @4 str $char.;
else delete;
if substr(process,4,1)=' ' then indent=1;
else if substr(process,6,1)=' ' then indent=4;
else if substr(process,8,1)=' ' then indent=9;
else if substr(process,3,1)=' ' then indent=16;
* strip off the trailing semi-colon;
str=substr(str,1,length(str)-1);
run;

proc append base=all data=a1;
%mend readcode;
*************;
 %readcode(aq300)      *read the sas job as data;
*--------------------------------------------------------------------*;

* print the process statements;
```

```
data _null_;
set all;
file outtxt ps=55 ls=78 noprint;
if indent=1 then put;
put @indent str;
run;
```

```
* makedict.sas
*
*************************************************************************
* Purpose of the program
*
* create a data dictionary from a series of work & perm. files;
*
*
* Written by:  Art Carpenter
*              CALOXY
*              (619) 724-8579
* Written on:  19Jun91
*
* Modified on: 11sep91
* notes:  changed the appearance of the output page.
*
*
* Prior to execution the user should execute all aspects of the program
* to be documented, so that all work files and permanent files will be
* fully defined to this documentation program.;
************************************************************************;
libname sasdata '\sugi17\autodoc';
filename datastr '\sugi17\autodoc\datastr.txt';
filename datadic '\sugi17\autodoc\datadic.txt';
************************************************************************;
options obs=max;

* use contents to id all variables in the requested datasets;
proc contents data=work._all_ memtype=data noprint out=conwork;
proc contents data=sasdata._all_ memtype=data noprint out=conperm;

* combine and select appropriate information ;
data con1;
set conwork conperm;
run;
*----------------------------------------------------------------------*;

* create the data structure;
proc sort data=con1 out=str1;
by libname memname name;

* make a text file containing the data structure;
data _null_;
set str1;
by libname memname name;
file datastr print notitle ps=52 ls=72 header=hdr linesleft=lleft;
if first.memname then do;
   if lleft<4 then put _page_;
   put @;
   i+1;
   if i>1 then do;
      put // ;
      put @1 libname @10 memname @20 memlabel/;
   end;
end;
put @7 name @17 label;
return;
hdr:
   put @1 libname @10 memname @20 memlabel/;
   if first.memname then i=0;
return;
run;
*----------------------------------------------------------------------*;

* create the data dictionary;
proc sort data=con1 out=dic1;
by name libname memname;

* determine the members that this variable is in;
* assumes that a variable name has a unique set of characteristics
*    e.g. the same name should not be used in different datasets with
*         different meanings (labels etc.);
data dic2; set dic1;
by name libname memname;
array mem $17 mem1-mem69;
retain mem1 mem2 mem3
       mem4 mem5 mem6
       mem7 mem8 mem9
       mem10 mem11 mem12
       mem13 mem14 mem15
       mem16 mem17 mem18
       mem19 mem20 mem21
       mem22 mem23 mem24
       mem25 mem26 mem27
       mem28 mem29 mem30
       mem31 mem32 mem33
       mem34 mem35 mem36
       mem37 mem38 mem39
       mem40 mem41 mem42
       mem43 mem44 mem45
       mem46 mem47 mem48
       mem49 mem50 mem51
       mem52 mem53 mem54
       mem55 mem56 mem57
       mem58 mem59 mem60
       mem61 mem62 mem63
       mem64 mem65 mem66
       mem67 mem68 mem69;
drop i;
if first.name then cnt=0;
cnt+1;
if cnt le 68 then mem{cnt} = left(compress(libname||'.'||memname));
else if cnt=69 & not last.name then mem69='etc.';
else if cnt=69 & last.name then
     mem{cnt} = left(compress(libname||'.'||memname));

if last.name then do;
   output;
   do i = 1 to 69;
      mem{i}=' ';
   end;
end;

* make a text file containing the data dictionary;
data _null_;
set dic2;
by name;
length infmt fmt $15;
file datadic ps=55 ls=72 notitle print header=hdr linesleft=lleft;
array mem $17 mem1-mem69;

* setup for writing;
if type=1 then vtype='N'; else vtype='$';

infmt= informat;
if informl>0 then infmt=left(compress(infmt||put(informl,2.)||'.'));
if informd>0 then infmt=left(compress(infmt||put(informd,2.)));

fmt= format;
if formatl>0 then fmt=left(compress(fmt||put(formatl,2.)||'.'));
if formatd>0 then fmt=left(compress(fmt||put(formatd,2.)));

if just=1 then vjust='R';
else if just=0 then vjust='L';
else vjust=' ';

* create one section for each variable;
if lleft le 5 then do;
   put _page_;
   i=1;
end;
* count number of variables so far on this page;
put @;
i+1;
if i>1 then do;
   put // ;
   put @1  name @10 label /;
end;
put   @5 'Attributes';
put   @5 '  TYPE   LENGTH      INFORMAT     FORMAT  JUST. '
    / @5 '  ____________________________________________ ';
put   @9 vtype @16 length 2. @26 infmt @39 fmt @50 vjust;
put / @5 'Databese Cross Reference';
put @5 mem{ 1} @25 mem{ 2} @45 mem{ 3};
done=0;
do j=4 to 67 by 3 until(done);
   if mem{ j} = ' ' then done=1;
   if not done then put @5 mem{ j} @25 mem{ j+1} @45 mem{ j+2};
end;
return;
hdr:
   put @1  name @10 label /;
   i=0;
return;
run;
```