

PROC DOC III: Self-generating Codebooks Using SAS®

Louise Hadden, Abt Associates Inc.

ABSTRACT

This paper will demonstrate how to use good documentation practices and SAS® to easily produce attractive, camera-ready data codebooks (and accompanying materials such as label statements, format assignment statements, etc.) Four primary steps in the codebook production process will be explored: use of SAS metadata to produce a master documentation spreadsheet for a file; review and modification of the master documentation spreadsheet; import and manipulation of the metadata in the master documentation spreadsheet to self-generate code to be included to generate a codebook; and use of the documentation metadata to self-generate other helpful code such as label statements. Full code for the example shown (using the SASHELP.HEART data base) will be provided upon request.

INTRODUCTION

The most onerous task any SAS programming professional faces is to accurately document files and processes. The truth is that there are no easy answers to the documentation quandary. It takes hard, painstaking work! By setting careful standards at the outset of a programming task, documenting your processes, labelling your data files and variables, providing value labels (formats) for your variables when appropriate, and using the many tools the SAS® system provides to assist in the documentation process, producing codebooks can be a piece of cake.

ON THE OTHER SIDE OF THE MOUNTAIN: CODEBOOK GENERATION

You've done a lot of hard work documenting every aspect of your programming project, and now it is time to reap your rewards. There are a number of ways that you can present information from PROC CONTENTS and PROC DATASETS covered in many other papers, including some of my own. We are going to focus on the use of an intermediate spreadsheet to drive creation of a robust codebook with self-generating code.

STEP 1

It is important to review and evaluate the metadata associated with the data set to be documented. Data sets should be labeled accurately. Variables should be labelled accurately. If variables have informats or formats, that information should be available and accurate. There should be a program available to create a permanent format library with a two level catalog name, if applicable – and those formats should be accurate. For our example, we create an age category variable that we wish to format, and write a program to generate a format in a permanent, two-level format catalog.

Code snippet from 1gen_formats_MWSUG_2017_RF07.sas:

```
TITLE1 "MWSUG 2017 PAPER RF07";
FOOTNOTE1 "%SYSFUNC(GETOPTION(SYSIN)) - &SYSDATE - &SYSTIME - run by
&SYSUSERID in &SYSPROCESSMODE";
RUN;

LIBNAME dd '.';
LIBNAME library '.';
FILENAME odsout '.';
RUN;

PROC FORMAT LIBRARY=LIBRARY.HEART;
VALUE startage 25 - 34='25 to 34 years'
              35 - 44='35 to 44 years'
```

```

                45 - 54='45 to 55 years'
                55 - 64='55 to 64 years';
VALUE agefmt 1='25 to 34 years'
            2='35 to 44 years'
            3='45 to 54 years'
            4='55 to 64 years';

RUN;

```

STEP 2

In the example shown below, a Microsoft Excel® spreadsheet with selected variables from PROC CONTENTS output is generated using PROC EXPORT in program 2gen_metadata_MWSUG_2017_RF07.sas. I am using a modified copy of SASHELP.HEART as the sample data set for several reasons, one of which is that not all variables are labelled, requiring some changes. Another reason is that this data set is available to all SAS users.

Code snippet from gen_metadata_MWSUG_2017_RF07.sas:

```

DATA dd.heart (LABEL="Copy of SASHELP.HEART for MWSUG 2017 PAPER RF07-
created by %SYSFUNC(GETOPTION(SYSIN))
- &SYSDATE - &SYSTIME - run by &SYSUSERID in &SYSPROCESSMODE");
LENGTH dslabel $ 200 source $ 32;
SET sashelp.heart;

/* put in some missing labels */

dslabel="Copy of SASHELP.HEART for MWSUG 2017 PAPER RF07- created by
%SYSFUNC(GETOPTION(SYSIN))
- &SYSDATE - &SYSTIME - RUN by &SYSUSERID in &SYSPROCESSMODE";

source="&dsname";

IF 25 LE ageatstart LE 34 THEN age=1;
IF 35 LE ageatstart LE 44 THEN age=2;
IF 45 LE ageatstart LE 54 THEN age=3;
IF 55 LE ageatstart LE 64 THEN age=4;

IF ageatstart ge 85 THEN age=7;
FORMAT age agefmt.;

LABEL cholesterol='Cholesterol level'
      diastolic='Diastolic blood pressure'
      height='Height'
      sex='Gender'
      smoking='Cigarettes per day'
      status='Wanted, dead or alive'
      systolic='Systolic blood pressure'
      weight='Weight'
      source='Data set name'
      dslabel='Data set information'
      age='Age at Start Category'
      ;

RUN;

```

```

. . . PROC EXPORT DATA = dd.heart_cb DBMS = excel
OUTFILE = ".\heart_db.xlsx" REPLACE;
RUN;

```

Of course, you want to review the results of your spreadsheet creation in Excel and maybe modify a label or format assignment. Note that I have created a variable indicating a specialized variable type (VARTYPE), as I want to treat formatted variables differently from unformatted variables.

Figure 1: Screenshot of Microsoft Excel® worksheet created by program 2gen_metadata_MWSUG_2017_RF07.sas

	A	B	C	D	E	F	G	H	I	J	K
1	varnum	vartype	name	label	format	length	npos	type	source	dsinfo	
2	1	2	dslabel	Data set information		200	88	2	HEART	Copy of SASHELP	
3	2	2	source	Data set name		32	288	2	HEART	Copy of SASHELP	
4	3	2	Status	Wanted, dead or alive		5	320	2	HEART	Copy of SASHELP	
5	4	2	DeathCau	Cause of Death		26	325	2	HEART	Copy of SASHELP	
6	5	3	AgeCHDDi	Age CHD Diagnosed		8	0	1	HEART	Copy of SASHELP	
7	6	2	Sex	Gender		6	351	2	HEART	Copy of SASHELP	
8	7	3	AgeAtStar	Age at Start		8	8	1	HEART	Copy of SASHELP	
9	8	3	Height	Height		8	16	1	HEART	Copy of SASHELP	
10	9	3	Weight	Weight		8	24	1	HEART	Copy of SASHELP	
11	10	3	Diastolic	Diastolic blood pressure		8	32	1	HEART	Copy of SASHELP	
12	11	3	Systolic	Systolic blood pressure		8	40	1	HEART	Copy of SASHELP	
13	12	3	MRW	Metropolitan Relative Weight		8	48	1	HEART	Copy of SASHELP	
14	13	3	Smoking	Cigarettes per day		8	56	1	HEART	Copy of SASHELP	
15	14	3	AgeAtDea	Age at Death		8	64	1	HEART	Copy of SASHELP	
16	15	3	Cholester	Cholesterol level		8	72	1	HEART	Copy of SASHELP	
17	16	2	Chol_Stat	Cholesterol Status		10	357	2	HEART	Copy of SASHELP	
18	17	2	BP_Status	Blood Pressure Status		7	367	2	HEART	Copy of SASHELP	
19	18	2	Weight_St	Weight Status		11	374	2	HEART	Copy of SASHELP	
20	19	2	Smoking_	Smoking Status		17	385	2	HEART	Copy of SASHELP	
21	20	1	age	Age at Start Category	AGEFMT	8	80	1	HEART	Copy of SASHELP	
22											
23											
24											
25											

You can then reimport the modified spreadsheet for use in the next step to: (a) write code to be included to generate a codebook with output varying by variable type; (b) write code to generate a label statement; and (c) write code to generate a format assignment statement, among other normally onerous tasks.

STEP 3

The codebook generation program, 3gen_codebook_MWSUG_2017_RF07.sas, starts with reimporting the edited version of the metadata spreadsheet, shown above. A number of macros are then

constructed: to report on “header information” (i.e. variable name, label, etc.), missing values, and then details on non-missing values, differential by variable type (character, continuous, categorical). Additionally, the program accesses the metadata and outputs text files with macro calls to the macros created above conditional upon the variable type in the metadata and reporting macros, that are then reused in the program as include files.

Code snippet from 3gen_codebook_MWSUG_2017_RF07.sas:

```
DATA _null_;
  FILE out1 LRECL=80 PAD;
  LENGTH include_string $ 80;
  SET dd.heart_cb (KEEP=varnum name vartype);

  include_string=CATS('%header(',name,",",",varnum,");");
  PUT include_string;
RUN;

. . .

DATA _null_;
  FILE out4 LRECL=80 PAD;
  LENGTH include_string $ 80;
  SET dd.heart_cb (KEEP=varnum name vartype);

  IF vartype=1 THEN include_string=CATS('%printtable(',varnum,");");
  IF vartype=2 THEN include_string=CATS('%printtablec(',varnum,");");
  IF vartype=3 THEN include_string=CATS('%printblurb(',varnum,");");

  PUT include_string;
RUN;
```

Macros are written to report on each variable, creating an RTF codebook. These printing macros are utilized in the %include files written by the program inside a TAGSETS.RTF sandwich.

Code snippet from 3gen_codebook_MWSUG_2017_RF07.sas:

```
%MACRO printblurb(order);

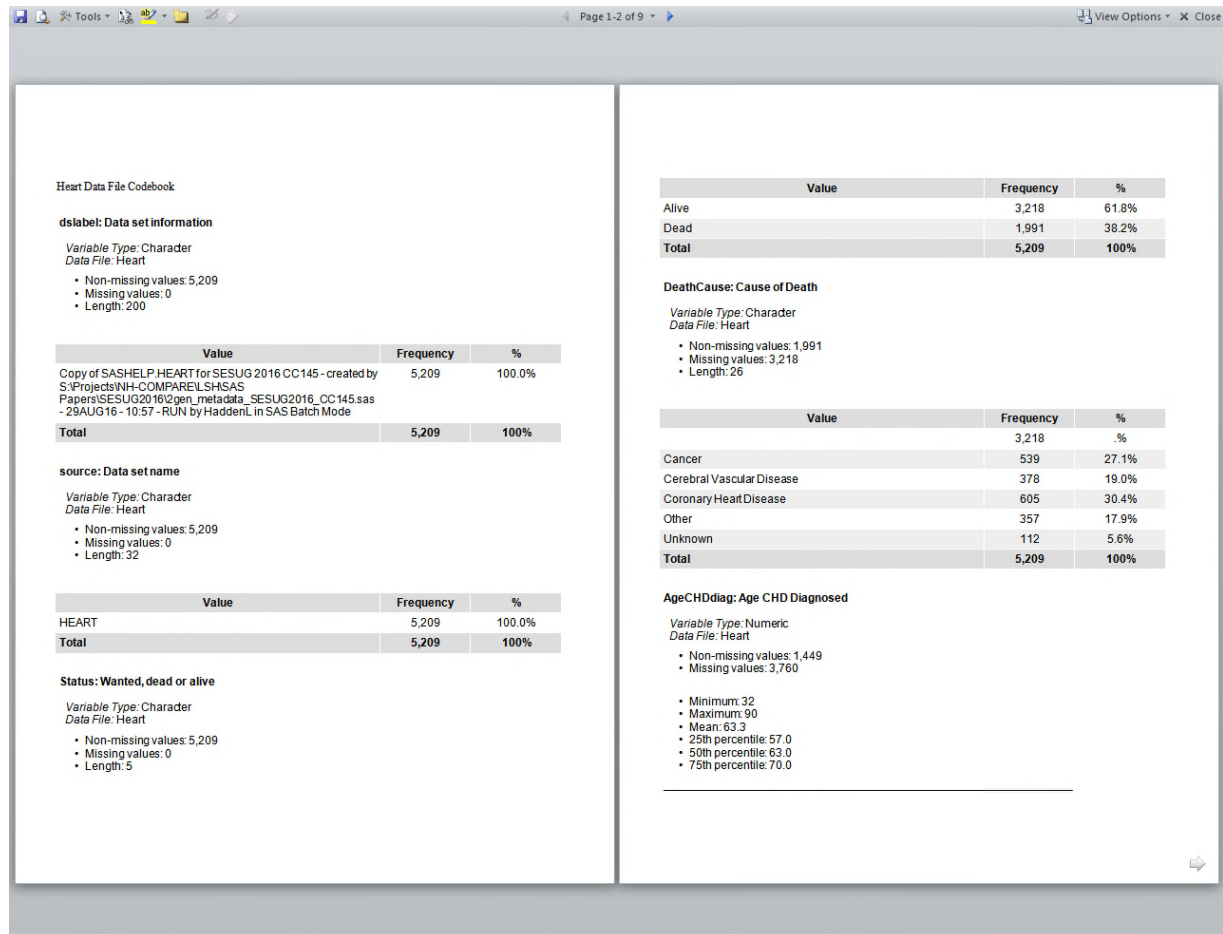
ODS TAGSETS.RTF STYLE=styles.noborder;
ODS STARTPAGE=no;

PROC REPORT NOWD DATA=print&order
  STYLE(report)=[cellpadding=3pt vjust=b]
  STYLE(header)=[just=center font_face=Helvetica font_weight=bold
                 font_size=10pt]
  STYLE(lines)=[just=left font_face=Helvetica] ;
  COLUMNS blurb ;
  DEFINE blurb / style(COLUMN)={just=1 font_face=Helvetica
                               font_size=10pt cellwidth=988 }
                style(HEADER)={just=1 font_face=Helvetica
                               font_size=10pt };
RUN;

ODS STARTPAGE=no;

%MEND;
```

Figure 2: Screenshot of two pages from RTF codebook created by program gen_codebook_MWSUG_2017_RF07.sas



The codebook construction can take some time. Arrange to send yourself a text message with the condition code of your job when it finishes, and get a cup of coffee.

Code snippet from 3gen_codebook_MWSUG_2017_RF07.sas:

```
FILENAME msg EMAIL TO="0000000000@txt.att.net"
FROM = "Big Nerd <louise_hadden@abtassoc.com>"
SUBJECT="All Systems Go (or not)?" ;

DATA _null_;
FILE msg;
PUT "Program Path and Name: %SYSFUNC(GETOPTION(SYSIN))" ;
PUT "RUN &SYSDATE - &SYSTIME - by &SYSUSERID in &SYSPROCESSMODE" ;
PUT "Condition Code is &SYSCC." ;
RUN;
```

STEP 4

Similarly, metadata can be accessed to create label, format, and length, etc. statements.

Code snippet from 4gen_label_fmt_stmt_MWSUG_2017_RF07.sas:

```
DATA temp1;
  LENGTH include_string $ 180;
  SET dd.heart_cb;

  label=COMPRESS(label, '');

  qlabel=CATS('', label, '');
  include_string=CATX(' ', name, '=', qlabel);
RUN;

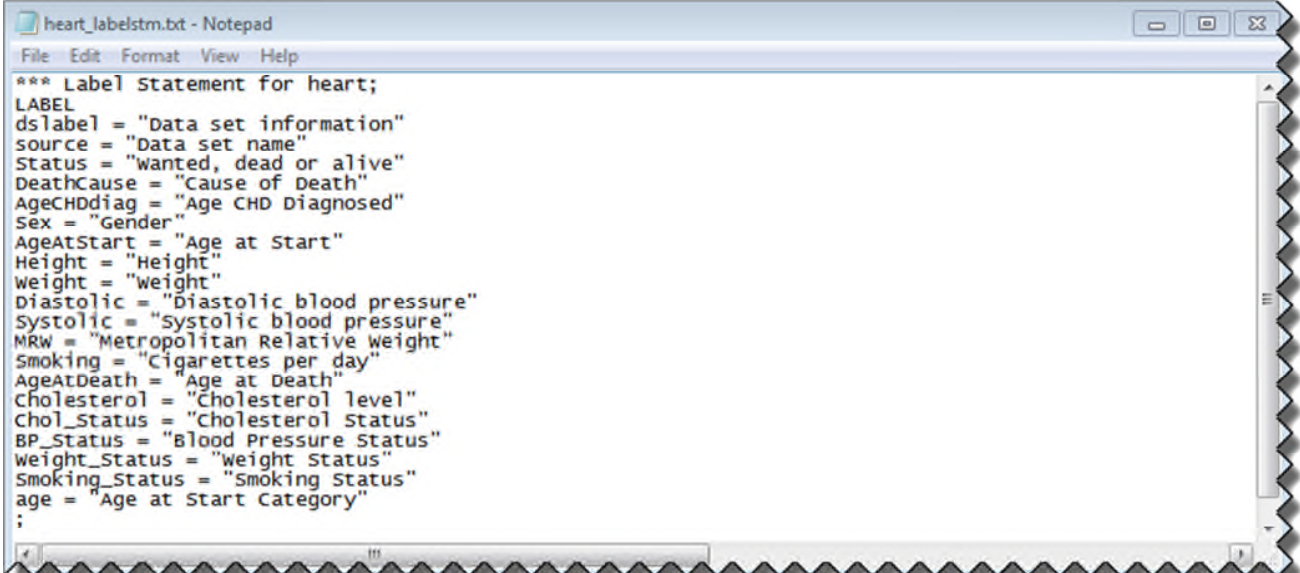
DATA templabel (KEEP=include_string);
  FILE out1 LRECL=180 PAD;
  LENGTH include_string $ 180;
  SET runlabel temp1 runrun;
  PUT include_string;
RUN;

DATA temp2;
  LENGTH include_string $ 180;
  SET dd.heart_cb (WHERE=(format NE ''));
  qformat=CATS(format, '.');
  include_string=CATX(' ', name, qformat);
RUN;

DATA tempfmt (KEEP=include_string);
  FILE out2 LRECL=180 PAD;
  LENGTH include_string $ 180;
  SET runformat temp2 runrun;
  PUT include_string;
RUN;
```

The resulting statement, example shown below, can be included in other programs seamlessly.

Figure 3: Screenshot of label statement created by program 4gen_label_fmt_stmnt_MWSUG_2017_RF07.sas



```
heart_labelstm.txt - Notepad
File Edit Format View Help
*** Label Statement for heart;
LABEL
dslabel = "Data set information"
source = "Data set name"
Status = "wanted, dead or alive"
DeathCause = "cause of death"
AgeCHDdiag = "Age CHD Diagnosed"
Sex = "Gender"
AgeAtStart = "Age at Start"
Height = "Height"
weight = "weight"
Diastolic = "Diastolic blood pressure"
Systolic = "systolic blood pressure"
MRW = "Metropolitan Relative Weight"
Smoking = "Cigarettes per day"
AgeAtDeath = "Age at Death"
Cholesterol = "Cholesterol level"
Chol_Status = "Cholesterol Status"
BP_Status = "Blood Pressure Status"
weight_Status = "weight Status"
Smoking_Status = "Smoking Status"
age = "Age at Start Category"
;
```

CONCLUSION

Only code snippets are shown here: full code is available from the author upon request or by visiting the paper's page on sascommunity.org, http://www.sascommunity.org/wiki/PROC_DOC_III:_Self-generating_Codebooks_Using_SAS%C2%AE.

The SAS system provides numerous opportunities for creating self-documenting data sets. With care at the onset of a project, SAS programmers can utilize the power of the SAS system to ensure quality data and accurate documentation. Simple documentation is not enough to ensure quality data and analyses. SAS can show us the way to create user-friendly documentation, and generate components of your SAS programs without typing a word.

REFERENCES

- Carey, Helen and Carey, Ginger, 2011. "Tips and Techniques for the SAS Programmer!" Proceedings of SAS Global Forum 2011.
- Crawford, Peter, 2013. "A Day in the Life of Data – Part 3." Proceedings of SAS Global Forum 2013.
- Fraeman, Kathy Hardis, 2008. "Get into the Groove with %SYSFUNC: Generalizing SAS® Macros with Conditionally Executed Code." Proceedings of NESUG 2008.
- Hadden, Louise, 2014. "Build your Metadata with PROC CONTENTS and ODS OUTPUT", Proceedings of SAS Global Forum 2014.
- Huang, Chao, 2014. "Top 10 SQL Tricks in SAS®." Proceedings of SAS Global Forum 2014.
- Karafa, Matthew T., 2012. "Macro Coding Tips and Tricks to Avoid "PEBCAK" Errors." Proceedings of SAS Global Forum 2012.

Kuligowski, Andrew T. and Shankar, Charu, 2013. "Know Thy Data: Techniques for Data Exploration." Proceedings of SAS Global Forum 2013.

Lafler, Kirk Paul, 2014. "Powerful and Hard-to-find PROC SQL Features." Proceedings of SAS Global Forum 2014.

Murphy, William C., 2013. "What's in a SAS® Variable? Get Answers with a VI!" Proceedings of SAS Global Forum 2013.

Raithel, Michael A., 2011. "PROC DATASETS: the Swiss Army Knife of SAS® Procedures." Proceedings of SAS Global Forum 2011.

Thornton, Patrick, 2011. "SAS® DICTIONARY: Step by Step." Proceedings of SAS Global Forum 2011.

Zhang, Jingxian, 2012. "Techniques for Generating Dynamic Code from SAS® Dictionary Tables." Proceedings of SAS Global Forum 2012.

ACKNOWLEDGMENTS

The author gratefully acknowledges the helpful work of Kathy Fraeman, Michael Raithel, Patrick Thornton, Troy Martin Hughes, Richann Watson, Roberta Glass and Kirk Paul Lafler, and my colleague Tom McCall, among others.

CONTACT INFORMATION

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

Louise Hadden: Louise.Hadden@abtassoc.com



Scan me for sample code and additional papers!

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.