Paper 1314-2017

Document and Enhance Your SAS® Code, Data Sets, and Catalogs with SAS Functions, Macros, and SAS Metadata

Roberta Glass, Abt Associates Inc., Cambridge, MA

Louise S. Hadden, Abt Associates Inc., Cambridge, MA

ABSTRACT

Discover how to document your SAS® programs, data sets, and catalogs with a few lines of code that include SAS functions, macro code, and SAS metadata. Do you start every project with the best of intentions to document all of your work, and then fall short of that aspiration when deadlines loom? Learn how SAS system macro variables can provide valuable information embedded in your programs, logs, lists, catalogs, data sets and ODS output; how your programs can automatically update a processing log; and how to generate two different types of codebooks.

INTRODUCTION

Who cares about metadata? Any SAS programmer should care! Knowing and being able to track the development of your data files is vital. From answering client questions about the definition of a single variable to defending your company from a lawsuit, careful detailed documentation will save time and money in the long run.

By using SAS metadata in conjunction with automated documentation, you can find out when a program was last run, who ran it, what variables were created, whether the data set is sorted or indexed, and more. You can use your metadata to write portions of your programs, and to generate codebooks. We will give you a whirlwind tour of tools, tips and techniques to enhance your SAS programming toolkit!

Every time a SAS session is initiated, a wealth of metadata becomes available to users, and this metadata can be used to help document your processes and output. We will discuss methods of documenting in these three areas:

- Programs, logs, and Output,
- SAS datasets, and
- SAS Catalogs.

PROGRAMS, LOGS & OUTPUT

It is good practice to place the name of the program in your program, log, and output; but how many of us are guilty of re-using code and forgetting to change the program name in the documentation section? Have you ever been presented with a table you created two years ago and asked how a statistic was computed? Using system functions and macro variables can save you time and insure that every program you run has a .log, .lst, and table that contain the correct program name, date and time it was run, and the user ID of the person who ran it.

- SYSFUNC(GETOPTION (SYSIN)) returns the path and name of the program
- &SASDATE returns the date the program began
- &SYSTIME returns the time the program began and
- &SYSUSERID returns the user ID of the programmer who submitted the job.

To make sure that your output is easily linked to the program which created it, this information can be placed in either a title or footnote:

```
TITLE1 “SYSFUNC(GETOPTION(SYSIN)) run &SYSDATE - &SYSTIME – by &SYSUSERID”;
FOOTNOTE1 “SYSFUNC(GETOPTION(SYSIN)) run &SYSDATE - &SYSTIME – by
    &SYSUSERID”;
```

To link data files with the program that creates it, this information can also be placed in a variable or variables, or as part of a data set label.

A simple macro can insure that every program has a header section which contains this information. First, save a compiled version of the macro to a macro catalog. Note that you can also store a description of your macro using the DES= option:

```
%INCLUDE "C:\Users\GlassR\Desktop\samples\autodoc.sas";

LIBNAME Macrolib "C:\Users\GlassR\Desktop\samples\Macrolib";
LIBNAME Fmtlib "C:\Users\GlassR\Desktop\samples\Fmtlib";
libname dd "C:\Users\GlassR\Desktop\samples";

OPTIONS MSTORED SASMSTORE=Macrolib ;

*****;
** save a compiled header macro;
*****;

%MACRO hdr/ STORE SOURCE DES="Program Header";
%PUT 0              ************************************************;
%PUT 0              ** Project: Sample Project;
%PUT 0              ** Program: %SYSFUNC(GETOPTION(SYSIN));
%PUT 0              ** Run by: &SYSUSERID;
%PUT 0              ** Run Date/time: &SYSDATE - &SYSTIME;
%PUT 0              ************************************************;
%MEND;
RUN;
```

Then include a call to the stored header macro in all of your programs:

```
OPTIONS MSTORED SASMSTORE=MYSTORE;
LIBNAME Macrolib "C:\Sample\compiled_macros";
%hdr;
```

Now your logs will contain metadata in an easily located header:

```
0       ************************************************
0       ** Project: Sample Project
0       ** Program: SAMPLE1.SAS
0       ** Run by: RGlass
0       ** Run Date/time: 27JUL15 – 15:09
0       ************************************************
```

When there is a time crunch, many of us find ourselves with a backlog of programs to add to our process log. But what if each program could automatically add itself to the log? The following program autodoc.sas can be saved as a compiled macro and can be called at the end of every program to automatically update a Microsoft Excel® documentation log. The program assumes that a documentation log has already been saved as a spreadsheet. It can of course be customized to save the information that would be useful to you, and can include empty fields to be edited manually. The autodoc.sas program is shown below:

```
%MACRO autodoc(docfile)/ STORE SOURCE DES="Creates program log";

LIBNAME doc 'S:\Projects\Sample';
```

```
t       LENGTHS = program_name $ 100 run_date $ 9 run_time $ 8 run_by $ 32 purpose
            $ 500 input_files $500 output_files $500;
        * collect run information for this execution of the program;

        DATA userline;
        LENGTH &lengths.;
        run_by="&sysuserid";
        run_date="&sysdate";
        run_time="&systime";
        purpose="&purpose";
        input_files=" ";
        output_files=" ";
        path_program = "%sysfunc(getoption(sysin))";
        word_n = countc(path_program,"\.");
        program_name = scan(path_program,word_n,"\." );
        DROP word_n path_program;
        RUN;

        PROC PRINT data=userline noobs;
        TITLE1 “Documentation line saved for this program”;
        RUN;

        * assuming manual edits have been made to the spreadsheet;
        * read current doc excel file into SAS;

        PROC IMPORT dbms=excel out = backupdocA datafile = "&docfile.";
        RUN;

        DATA backupdoc;
        LENGTH &lengths.;
            SET backupdocA;
        RUN;

        * append the new information *;

        PROC APPEND data=userline base=backupdoc force;
        RUN;

        * re-save the updated backupdoc SAS file;

         DATA doc.backupdoc;
             SET backupdoc;
         RUN;

        * re-save the excel file;

        PROC EXPORT data= doc.backupdoc outfile= "&docfile." DBMS=EXCEL label
        replace;
            sheet="program log";
        RUN;

       %MEND;
```

When the autodoc macro is called , the user specifies the path and file name of the documentation log:

```
%autodoc(path/filename.xlsx);
```

The macro variable purpose must also be defined. This can be accomplished in two ways.

A %LET statement can be used to explain the purpose of the program:

```
%LET purpose = Test example for SAS GLOBAL paper.;
```

Alternatively, a windows prompt may be displayed prompting the user to enter the value of the purpose macro variable. The following code may be inserted at the beginning of a program, before the %autodoc macro call:

```
%global purpose docfile;
* prompts the user for purpose each time program is RUN *;
* %WINDOW defines the prompt *;

%WINDOW info
   #4 @5 'Please enter the purpose of this program:'
   #6 @5 purpose 100 attr=underline display=yes auto=no color = blue;

* %DISPLAY invokes the prompt
*;

%DISPLAY info;

%PUT &purpose.;
```

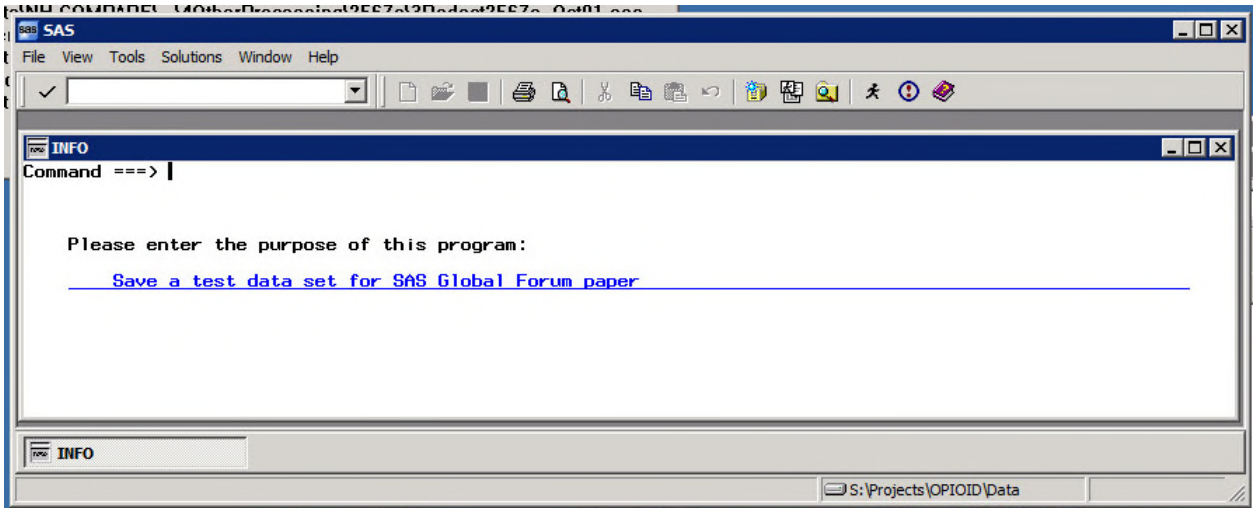
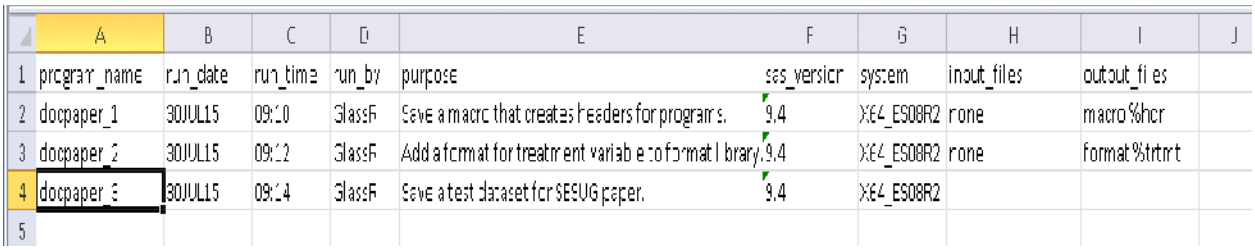


Figure 1: Screenshot of SAS window created by autodoc_SGF2017_Session1314.sas code

The SAS driven prompt offers you the opportunity to describe the purpose of the program. The text that is entered is stored as a macro variable and can be used in subsequent code.



	A	B	C	D	E	F	G	H	I	J
1	program_name	run_date	run_time	run_by	purpose	sas_version	system	input_files	output_files	
2	docpaper_1	30JUL15	09:10	GlassF	Save a macro that creates headers for programs.	9.4	X64_ES08R2	none	macro %hdr	
3	docpaper_2	30JUL15	09:12	GlassF	Add a format for treatment variable to format library.	9.4	X64_ES08R2	none	format $trtfmt	
4	docpaper_3	30JUL15	09:14	GlassF	Save a test dataset for SESUG paper.	9.4	X64_ES08R2			
5										

Figure 2: Screenshot of Microsoft Excel® worksheet modified by autodoc_SGF2017_Session1314.sas macro

In the example above the input and output data set fields are not automatically updated to illustrate that information may be added manually. As you can see from the results of running three programs, the manually entered information is retained when the file is updated by the autodoc macro.

If you wish to automate the data set names as well, you can customize the program by defining the variables input_files and output_files using macro variables, and then defining the macro variables each time you execute the program. For example change these two lines in the macro code as follows:

```
input_files="&input_files";
output_files="&output_files";
```

Then define these macro variables in the body of each program:

```
%global ouput_files input_files;
%LET in1 = dd.sample1; %LET in2 = dd.sample2;
%LET out1 = dd.sample2;
%LET input_files = &in1. &in2.;
%LET output_files = &out1.;
```

SAS DATASETS

When you receive a SAS file from another SAS user inside or outside of your company you can add an informative label using PROC DATASETS without having to save a new copy:

```
PROC DATASETS LIBRARY=DD;
   MODIFY Enrollment)Intake
  (LABEL=”Random Assignment of Study Subjects – Received from client”);
   CONTENTS DATA=Enrollment_Intake;
RUN;
```

For files that you create, the DATA step label option can be used to store metadata that documents the creation of a data set. PROC CONTENTS and PROC DATASETS will then be able to provide you with information on how and when the dataset was created as well as how it is sorted or indexed:

```
DATA dd.test (label=”Test data set created &SYSDATE - &SYSTIME – by
                    &SYSUSERID – by %SYSFUNC(GETOPTION(&SYSIN))”);
    INPUT @1 ID $2.
          @3 treatment $1.
          @4 baseline_cost 5.2;

    LABEL id              = 'Intake: Identification Number'
          Treatment       = 'Intake: Randomly assigned treatment'
          Baseline_Cost   = 'Claims: Baseline Cost';
    CARDS;
```

```
    A1Y11111
    A2N22222
    A3N33333
    A4Y44444
    A5Y55555;
RUN;

PROC SORT DATA=dd.test;
    BY id;
RUN;
```

If you are meticulous about labeling every variable and maintain a catalog of variable formats, you will have the material needed to create a codebook at the end of the file building process. In the example above the variable labels contain the source of the information ('Intake' for data produced by the random assignment enrollment procedure and 'Claims' for data derived from insurance claims. Other examples of useful information to include in a variable label are the question/item number of variables coming from a survey or administrative form and whether the variable has been recoded.

SAS CATALOGS

Storing all of your variable formats in a catalog doesn't just save the time of finding and copying code, it also serves as valuable documentation. You can save formats in one format library, or save separate format libraries specific to a particular dataset or phase of your project. For instance, to save a format for the Treatment variable in a catalog specific to dataset test, we specify "LIBRARY=fmtlib.test" instead of just "LIBRARY=fmtlib":

```
LIBNAME Fmtlib   "C:\Users\GlassR\Desktop\samples\Fmtlib";

PROC FORMAT LIBRARY = Fmtlib.test;
    VALUE $trtmt
       'C' = "Control"
       'T' = "Treatment"
       other = "Error";
RUN;
```

To use the format, specify the format library in the option statement. If you create your format library before creating the data file, you can assign the formats to variables with a format statement , thus storing the association permanently. Since in this example the dataset already exists, the format statement is used in the PROC FREQ procedure:

```
OPTIONS MSTORED FMTSEARCH=(Fmtlib.test);
PROC FREQ data-dd.test;
     TABLES treatment;
FORMAT treatment $trtmt.;
TITLE2 "Treatment Status";
RUN;
```

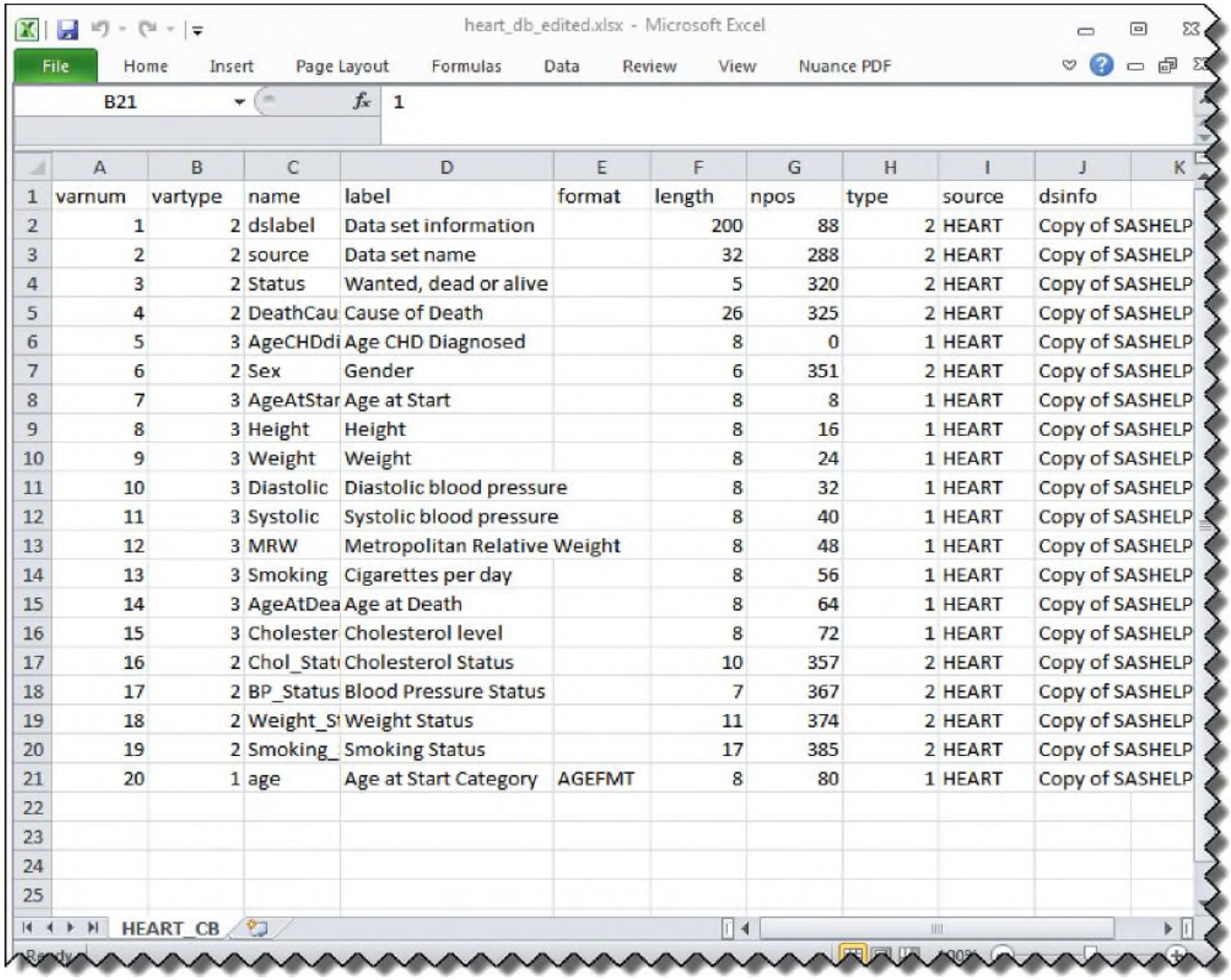
CODEBOOK GENERATION

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. In the example we show here, a Microsoft Excel spreadsheet with selected variables from PROC CONTENTS output is generated using PROC EXPORT in program gen_metadata_SGF2017_Session1314.sas. We are using a modified copy of SASHELP.HEART as our 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 users.

Code snippet from gen_metadata_SGF2017_Session1314.sas – supplied in proceedings:

```
DATA dd.heart (LABEL="Copy of SASHELP.HEART for SAS GLOBAL FORUM 2017
Session 1314 - created by %SYSFUNC(GETOPTION(SYSIN)) - &SYSDATE –
&SYSTIME - run by &SYSUSERID");
LENGTH dslabel $ 200 source $ 32;
    SET sashelp.heart;

    dslabel="Copy of SASHELP.HEART for SAS GLOBAL FORUM 2017 Session
    1314 – created by %SYSFUNC(GETOPTION(SYSIN)) - &SYSDATE - &SYSTIME
    - RUN by &SYSUSERID";
      source="&dsname";

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



	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											

Figure 3: Screenshot of Microsoft Excel® worksheet created by program gen_metadata_SGF2017_Session1314.sas

Of course, you want to review the results of your spreadsheet creation and maybe modify a label or format assignment. You can then reimport the modified spreadsheet, and use the information 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.

Our codebook generation program, gen_codebook_SGF2017_Session1314.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 gen_codebook_SGF2017_Session1314.sas – supplied in proceedings:

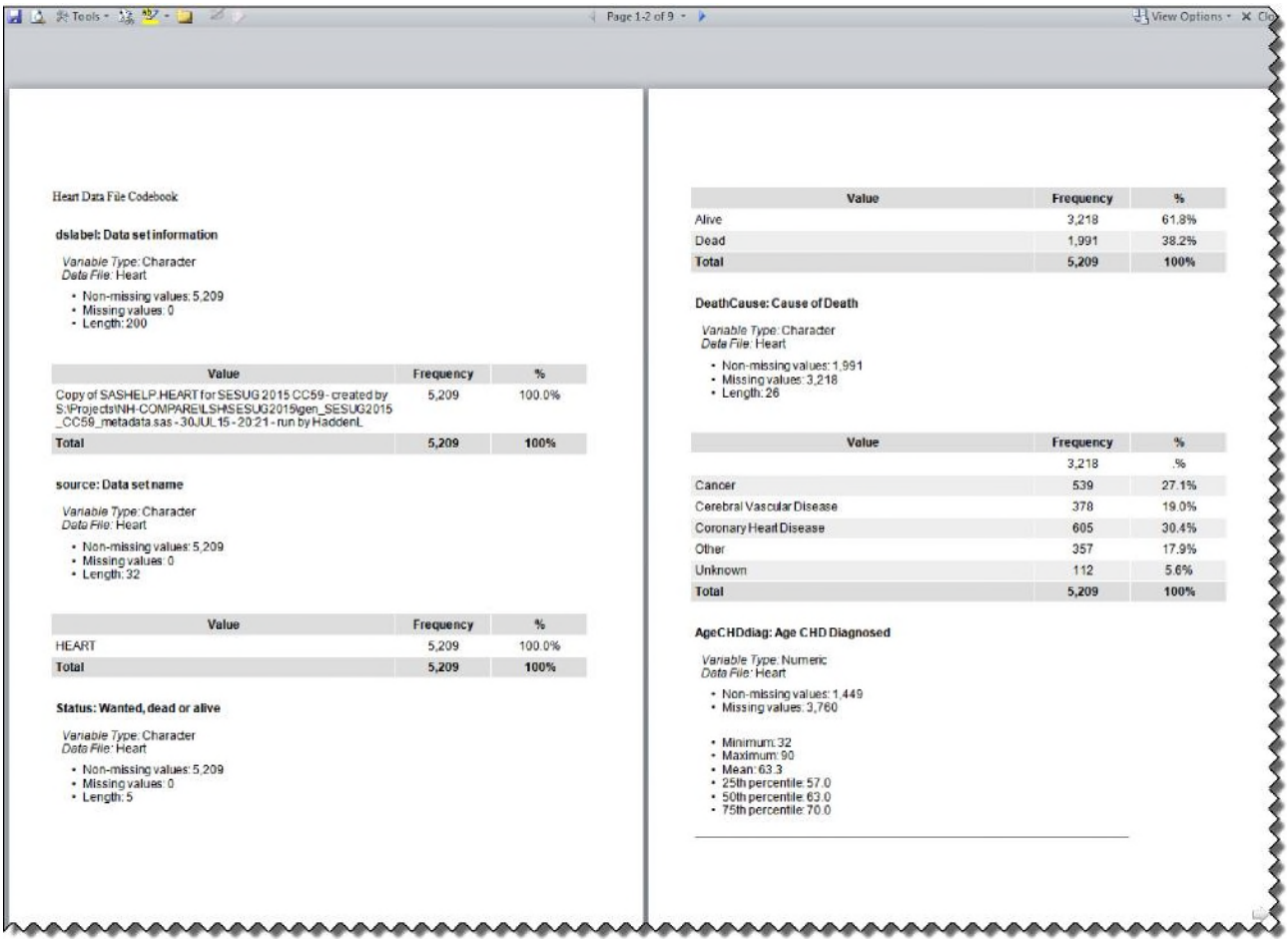
```
/* step 6 - write out files to run macros */
DATA _null_;
FILE out1 LRECL=80 PAD;
LENGTH include_string $ 80;
    SET dd.heart_cb (KEEP=varnum name vartype);
    include_string=CATS('%header(',name,",",varnum,");");
    PUT include_string;
RUN;

. . .

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 gen_codebook_SGF2017_Session1314.sas – supplied in proceedings:

```
/* step 9 - create printing macros */


%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=l font_face=Helvetica
```

```
          font_size=10pt cellwidth=988 }
    style(HEADER)={just=l font_face=Helvetica font_size=10pt};
RUN;

ODS STARTPAGE=no;

%MEND;
```



Heart Data File Codebook

dslabel: Data set information

Variable Type: Character
Data File: Heart

- Non-missing values: 5,209
- Missing values: 0
- Length: 200

Value	Frequency	%
Copy of SASHELP.HEART for SESUG 2015 CC59 - created by S:\Projects\NH-COMPARE\LSH\SESUG2015\gen_SESUG2015_CC59_metadata.sas - 30JUL15 - 20:21 - run by HaddenL	5,209	100.0%
Total	**5,209**	**100%**

source: Data set name

Variable Type: Character
Data File: Heart

- Non-missing values: 5,209
- Missing values: 0
- Length: 32

Value	Frequency	%
HEART	5,209	100.0%
Total	**5,209**	**100%**

Status: Wanted, dead or alive

Variable Type: Character
Data File: Heart

- Non-missing values: 5,209
- Missing values: 0
- Length: 5

Value	Frequency	%
Alive	3,218	61.8%
Dead	1,991	38.2%
Total	**5,209**	**100%**

DeathCause: Cause of Death

Variable Type: Character
Data File: Heart

- Non-missing values: 1,991
- Missing values: 3,218
- Length: 26

Value	Frequency	%
	3,218	.%
Cancer	539	27.1%
Cerebral Vascular Disease	378	19.0%
Coronary Heart Disease	605	30.4%
Other	357	17.9%
Unknown	112	5.6%
Total	**5,209**	**100%**

AgeCHDdiag: Age CHD Diagnosed

Variable Type: Numeric
Data File: Heart

- Non-missing values: 1,449
- Missing values: 3,760

- Minimum: 32
- Maximum: 90
- Mean: 63.3
- 25th percentile: 57.0
- 50th percentile: 63.0
- 75th percentile: 70.0

Figure 4: Screenshot of two pages from RTF codebook created by program gen_codebook_SGF2017_Session1314.sas

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

Code snippet from gen_label_fmt_stmnt_SGF2017_Session1314.sas – supplied in proceedings

```
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.

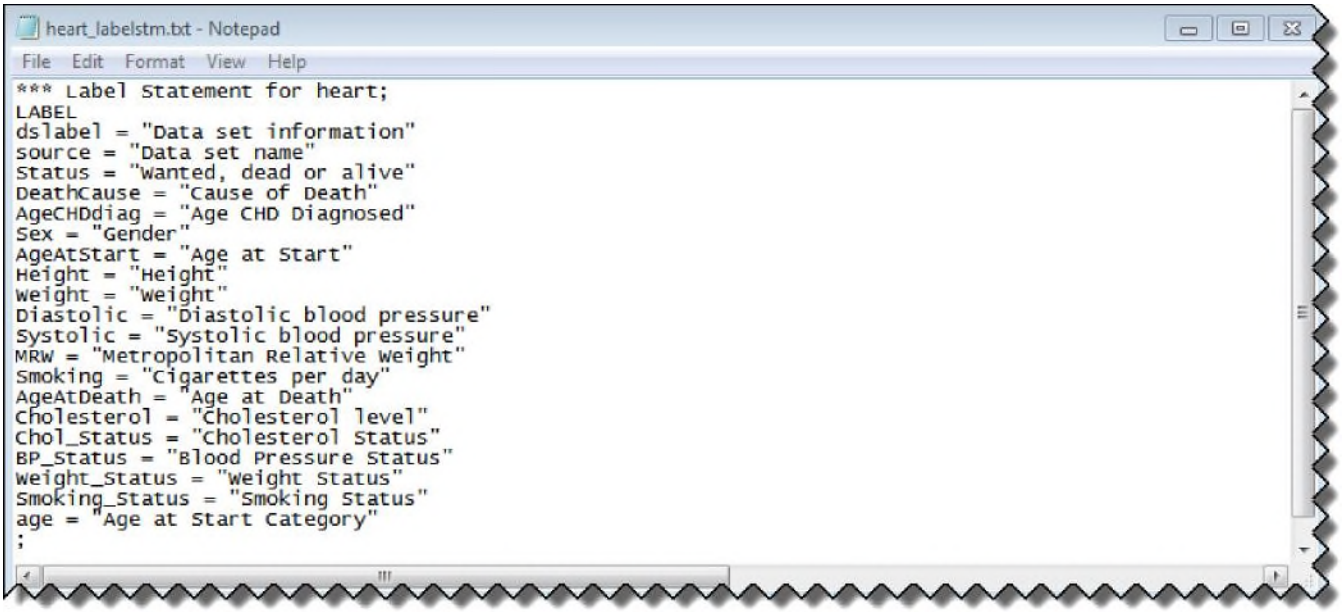


Figure 5: Screenshot of label statement created by program gen_label_fmt_stmnt_SGF2017_Session1314.sas

For those of us who deliver data to internal and external clients, careful documentation results in easy transfers with the help of SAS metadata.

Only code snippets are shown here: full code is available in the proceedings and from the authors upon request.

CONCLUSION

With attention to documentation from the start of a project, you can automatically keep a processing log updated, label your data sets and variables, and identify the code that created datasets, .logs, .lsts, and tables. This will allow you to take advantage of the PROC

DATASETS (as well as PROC CONTENTS), SAS Dictionary Tables and SASHELP.VIEWS 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.

Fecht, Marje, 2016. “Be More Productive! Tips and Tricks to Improve your SAS® Programming Environment.” Proceedings of SAS Global Forum 2016.

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.

Hughes, Troy Martin. September 2017. *SAS® Data Analytic Development: Dimensions of Software Quality.* Wiley and SAS Business Series, First Edition.

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 V!” 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.

Schacherer, Chris, 2013. “SAS® Data Management Techniques: Cleaning and transforming data for delivery of analytic datasets.” Proceedings of SAS Global Forum 2013.

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 authors gratefully acknowledge the helpful work of Kathy Fraeman, Michael Raithel, Patrick Thornton, Troy Martin Hughes and Kirk Paul Lafler, among others.

CONTACT INFORMATION

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

Roberta Glass: Roberta_Glass@abtassoc.com

Louise Hadden: Louise Hadden@abtassoc.com