Paper CC59

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 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 your programs can automatically update your processing log. If you have ever wondered who ran a program that overwrote your data, SAS has the answer! And If you don't want to be tracing back through a year's worth of code to produce a codebook for your client at the end of a contract, SAS has the answer!

INTRODUCTION

Who cares about metadata? Any SAS programmer should care! Knowing and being able to track your data is vital. By using SAS metadata in conjunction with careful 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!

DOCUMENT, DOCUMENT, DOCUMENT!

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

```
Title1 "SYSFUNC(GETOPTION(SYSIN)) run &SYSDATE - &SYSTIME - by &&SYSUSERID";
or
Footnote1 "SYSFUNC(GETOPTION(SYSIN)) run &SYSDATE - &SYSTIME - by &&SYSUSERID";
```

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.

```
*****;
** save a compiled header macro;
*****;
OPTIONS MSTORED SASMSTORE=MYSTORE;
LIBNAME MYSTORE "C:\Sample\COMPILED_MACROS";

%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;
```

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

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

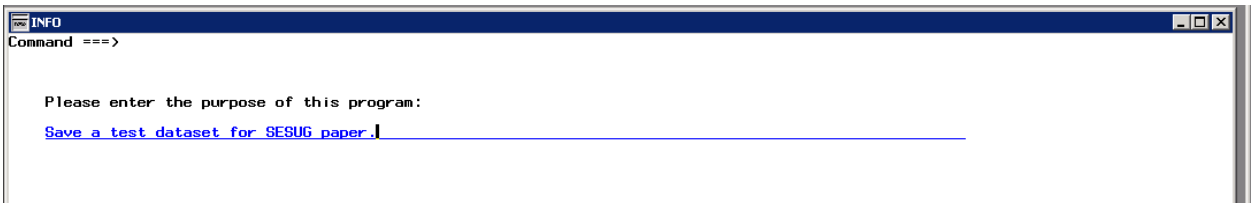
Now your logs will contain metadata in an easily located header:

```
0          ************************************************
0          ** Project: Sample Project
0          ** Program:
0          ** Run by:  GlassR
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 and called with %include to automatically update your log. It can of course be customized to save the information that would be useful to you, and can include empty fields which you can edit manually.

```
*** program name: Autodoc.sas:
libname doc 'C:\Users\GlassR\Desktop\samples';

* prompts the user for purpose and reason each time program is run in batch mode *;

* %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;

%let lengths = program_name $ 100   run_date $ 9      run_time $ 8
               run_by $ 32          purpose $ 100     sas_version $10
               system $20           input_files $500  output_files $500;

* collect run information for this execution of the program;
data userline(drop= word_n path_program);
      length &lengths.;

      run_by="&sysuserid";
      run_date="&sysdate";
      run_time="&systime";
      purpose="&purpose";
      sas_version="&sysver";
      system="&sysscpl";
      input_files=' ';
      output_files=' ';

      path_program = "%sysfunc(getoption(sysin))";
      word_n = countc(path_program,"\.");
      program_name = scan(path_program,word_n,"\." );
run;

* assuming you edited the spreadsheet, and want to retain what you put in there;
* read current doc excel file into sas;
proc import dbms=excel out = testdocA
    datafile = "C:\Users\GlassR\Desktop\samples\testdoc.xlsx";
run;

data testdoc;
      length &lengths.;
      set testdocA;
run;

* append the new information *;
proc append data=userline base=testdoc force;
run;

* re-save the updated testdoc SAS file;
data doc.testdoc;
      set testdoc;
run;

* re-save the excel file;
proc export data= doc.testdoc
      outfile= "C:\Users\GlassR\Desktop\samples\testdoc.xlsx"
      DBMS=EXCEL label replace;
      sheet="program log";
run;
```

Save an empty Microsoft Excel® file or generate a SAS file and export to an Excel file called testdoc.xlsx with the following columns: program_name, run_date, run_time, run_by, purpose, sas_version, system, input_files and output_files before using autodoc.sas for the first time. The autodoc program prompts the user to enter the purpose of the program, reads the existing Excel® log file into SAS, appends the information for this run of the program, and saves the updated excel log. A SAS data set version of the log is saved as a backup should the excel file not save correctly.



In our example we do not automatically populate the input and output data fields. The information is added manually. As you can see from the results of running three programs, the manually entered information is retained when the excel file is updated by autodoc.

	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	GlassR	Save a macro that creates headers for programs.	9.4	X64_ES08R2	none	macro %hdr	
3	docpaper_2	30JUL15	09:12	GlassR	Add a format for treatment variable to format library.	9.4	X64_ES08R2	none	format %trtmt	
4	docpaper_3	30JUL15	09:14	GlassR	Save a test dataset for SESUG paper.	9.4	X64_ES08R2			
5										

SAS DATASETS

When you receive a SAS file from outside your company you can add a label using PROC DATASETS without having to save a new copy.

```
PROC DATASETS LIBRARY=DD;
    MODIFY Enrollment_Intake
    (LABEL="Random Assignment of Study Subject - Received from client 01JUL2015");
    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. The Contents and Dataset procedures 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 on &sysdate - &systime - by &sysuserid -
by program %sysfunc(getoption(sysin))");

      input @1 ID $2.
      @3 Treatment $1.
      @4 Baseline_Cost 5.2
      @9 Followup_Cost 5.2
      ;

      label ID                = 'Intake: Identification Number'
            Treatment         = 'Intake: Randomly assigned treatment'
            Baseline_Cost     = 'Claims: Baseline Cost'
            Followup_Cost     = 'Claims: Follow-up Cost'
                  ;

      Cards;
      01T15530
      02C16725
      03T18986
      04T15342
      05T16612
      06C21587
      07C25309
      08T20382
      09C19943
      10C19617
      ;
run;

proc sort data=dd.test;
      by ID;
run;

proc contents data=dd.test;
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" :

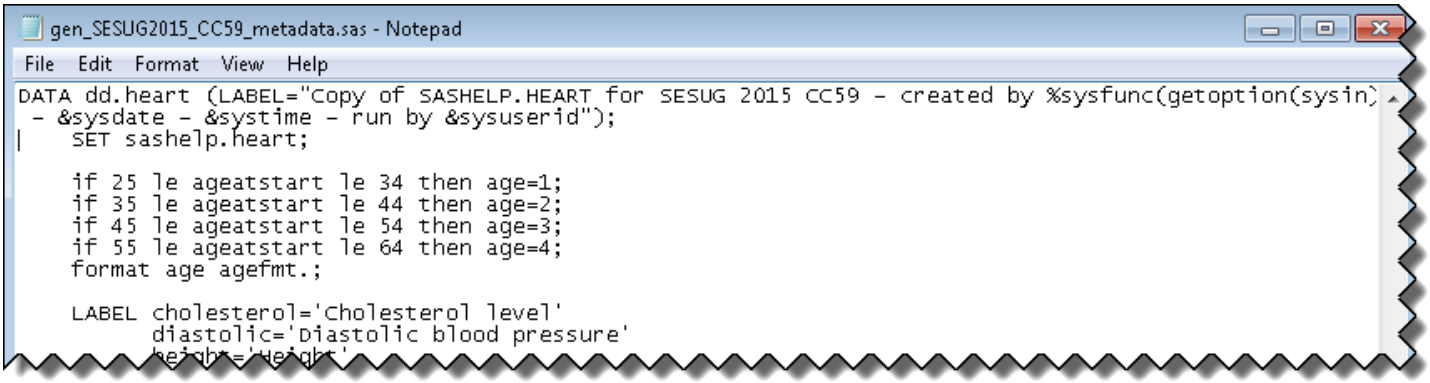
```
LIBNAME Fmtlib   "C:\Users\GlassR\Desktop\samples\Fmtlib";

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.

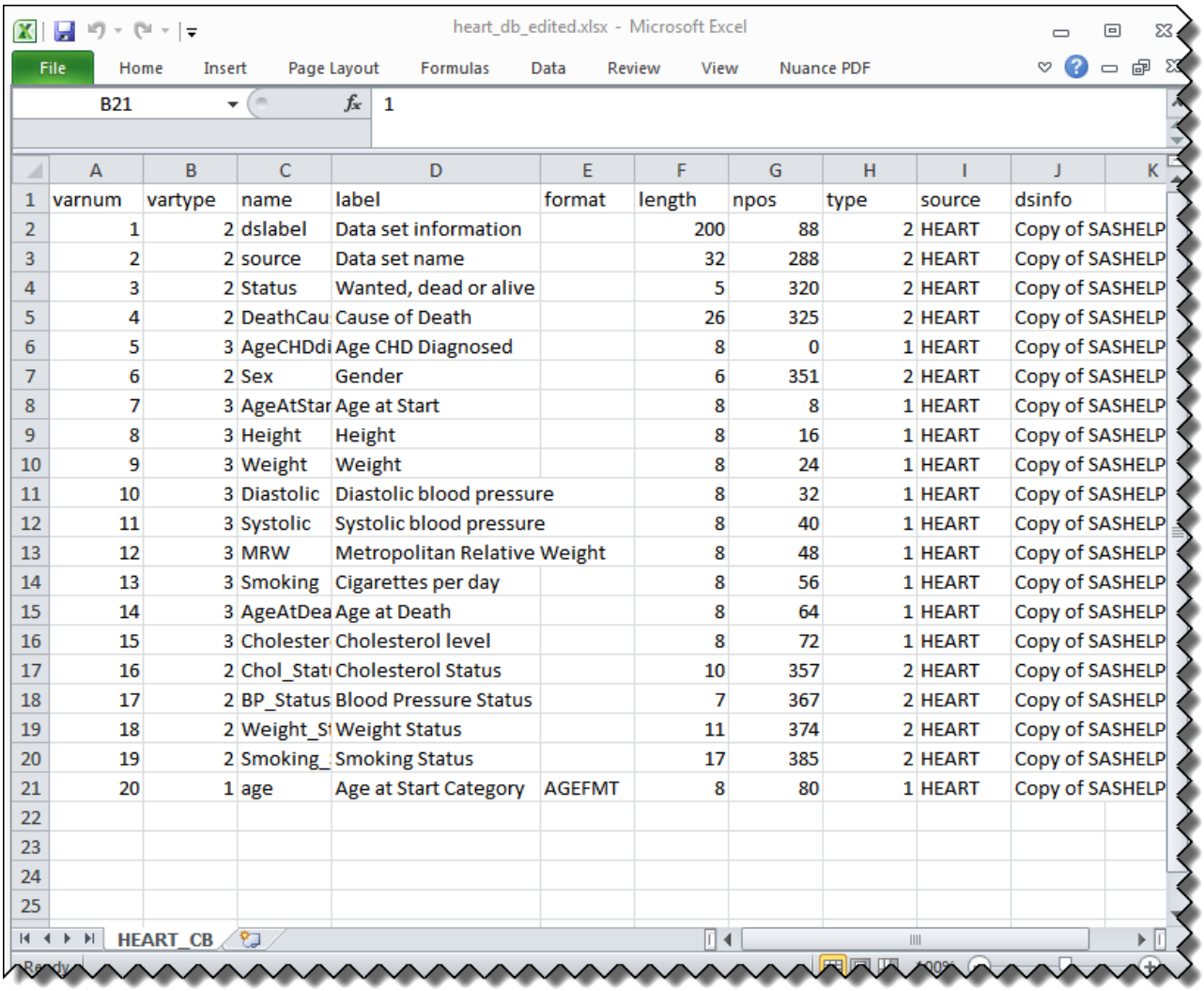
```
OPTIONS MSTORED FMTSEARCH=(Fmtlib.test);

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, an Excel spreadsheet with selected variables from PROC CONTENTS output is generated using PROC EXPORT. 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, and another is that this data set is available to all users.

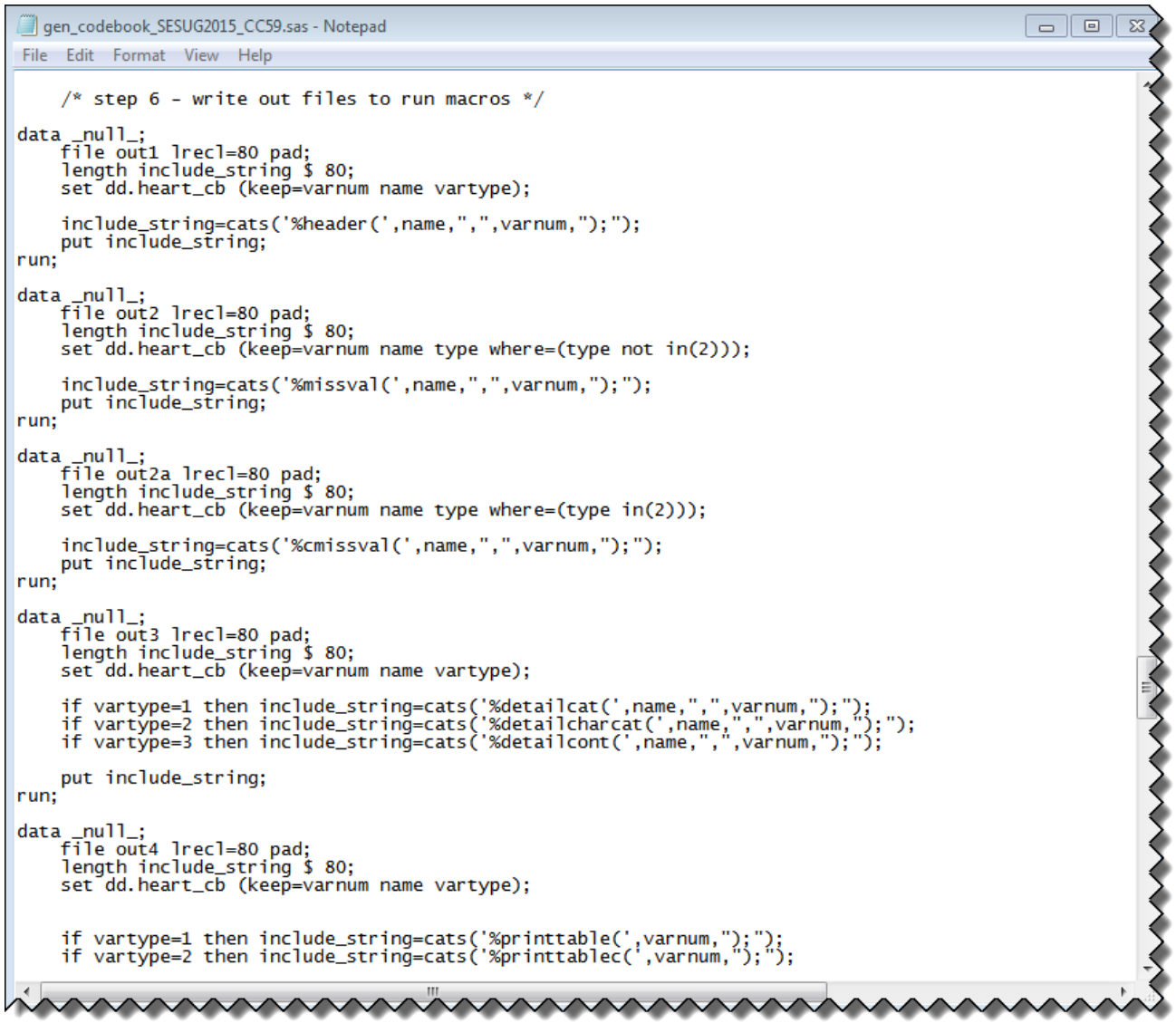


Of course, you want to review the results and maybe modify a label or format assignment. You can then reimport the modified spreadsheet , and use the information to: (a) write code to be included to generate a codebook with output varying by variable type; (b) write code to generate a label statement; and (c) write code to generate a format assignment statement, among other normally onerous tasks.



	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											

Our codebook generation program 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.


gen_codebook_SESUG2015_CC59.sas - Notepad

File Edit Format View Help

```
    /* 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 out2 lrecl=80 pad;
    length include_string $ 80;
    set dd.heart_cb (keep=varnum name type where=(type not in(2)));

    include_string=cats('%missval(',name,",",varnum,");");
    put include_string;
run;

data _null_;
    file out2a lrecl=80 pad;
    length include_string $ 80;
    set dd.heart_cb (keep=varnum name type where=(type in(2)));

    include_string=cats('%cmissval(',name,",",varnum,");");
    put include_string;
run;

data _null_;
    file out3 lrecl=80 pad;
    length include_string $ 80;
    set dd.heart_cb (keep=varnum name vartype);

    if vartype=1 then include_string=cats('%detailcat(',name,",",varnum,");");
    if vartype=2 then include_string=cats('%detailcharcat(',name,",",varnum,");");
    if vartype=3 then include_string=cats('%detailcont(',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,");");
```

Macros are written to report on each variable, creating an RTF codebook.

Blah

gen_codebook_SESUG2015_CC59.sas - Notepad

File Edit Format View Help

```
%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;
```

Two pages from the codebook are shown below.

Tools Page 1-2 of 9 View Options

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%**

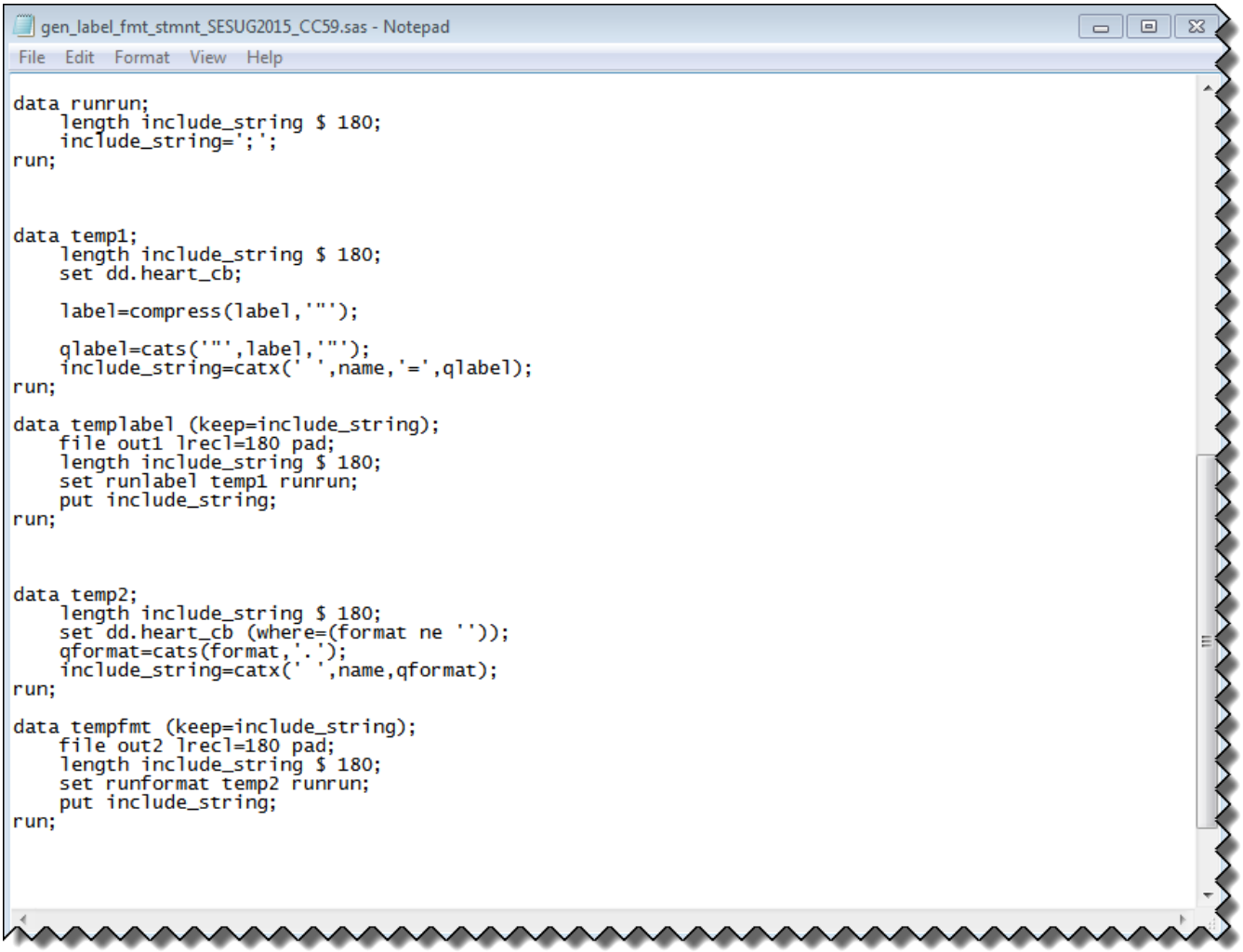
AgeCHDdiag: Age CHD Diagnosed

Variable Type: Numeric
Data File: Heart

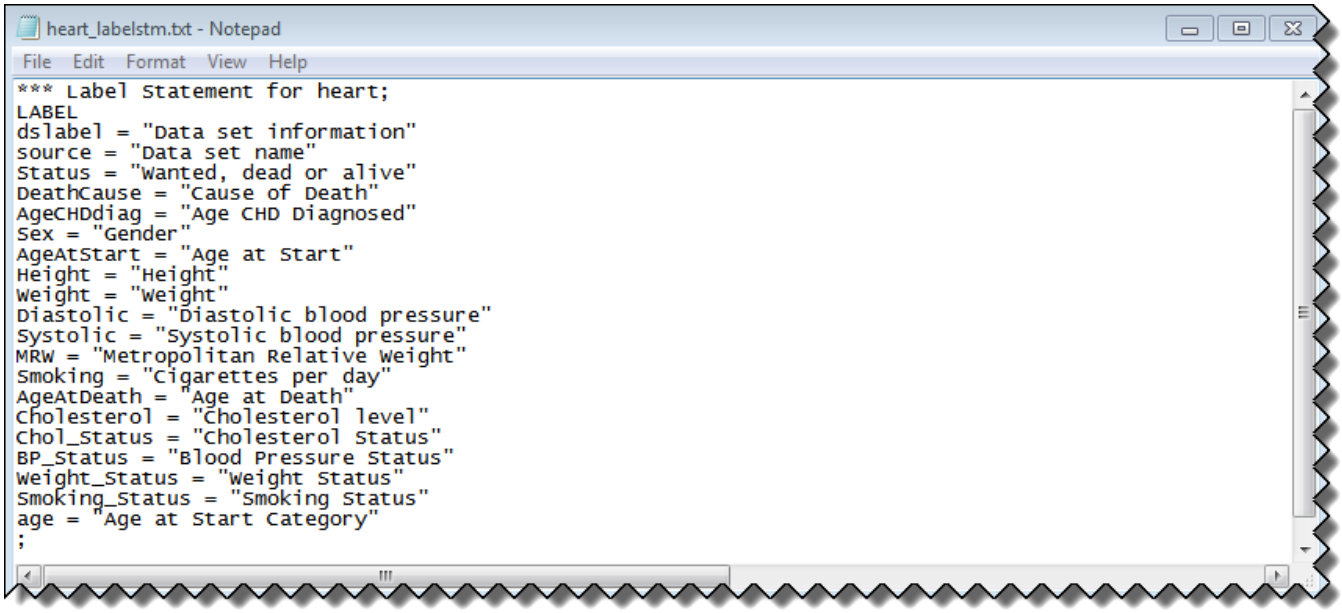
- Non-missing values: 1,449
- Missing values: 3,760

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

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



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



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 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.

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 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.

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 acknowledges the helpful work of Kathy Fraeman, Michael Raithel, Patrick Thornton 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