An efficient method to create a large and comprehensive codebook

Wen Song, ICF International, Calverton, MD
Kamya Khanna, ICF International, Calverton, MD
Baibai Chen, ICF International, Calverton, MD

ABSTRACT

At the conclusion of many data management projects, providing a client with an informative codebook is a common task for data managers or SAS programmers. When faced with a particularly large dataset containing thousands of variables or when faced with clients who demand many details about their variables, even the most experienced programmers might find this to be a daunting task. This paper provides a method of using SAS to create a customized codebook that includes the names, labels, types, values, formatted values, frequencies and percentages for every variable in a dataset.

INTRODUCTION

Codebooks are informative data summary reports which provide non-SAS user clients a comprehensive view of their data, and they often play a significant role in a project. When you have several projects in hand, and each of them requires you to create a brand new codebook all within a short amount of time it is easy to feel overwhelmed. Upon closer look at these different codebooks, we might find that although the output format for the codebooks may vary from simple Rich text file to complicated HTML output, most codebooks consist of very similar data information: labels, types, values, frequencies, and basic statistical information (Figure 1 demonstrates a simple codebook).

These consistencies allow developers to write a modifiable code that is independent of such knowledge as what data sets to use or what variables exist in the data sets prior to execution. Thanks to the capability of MACRO Language within SAS® system and the employment of PROC FORMAT, we can create a code that may be applied to any datasets with minimum revision, and produce codebooks in a matter of minutes. In this paper, we will outline a few simple steps – and a couple of handy tricks – to do just that.

TRICK 1: CHOOSE THE BEST METHOD TO CREATE MACRO VARIABLES

After classifying your variables into three or four general categories (e.g. numeric variables with specified formats, continuous or discrete numeric variables which are requested to display statistical information, and character variables which only need simple frequencies and percentages for missing and non-missing values), it becomes clear...
An efficient method to create a large and comprehensive codebook

that a certain set of operations (PROC FREQ or PROC MEANS) must be performed on a series of many variables. We need to use MACROs to do the job.

**PROC SQL? PROBABLY NOT!**

Most programmers have a preferred way to create MACRO variables and I am no different. So when I started to write my code, the first thing occurred to me was the PROC SQL function. By using PROC SQL, I placed all variables name into the MACRO variable &names1, and created &nvar to detect how many variables are in the data set. The MACRO variable &name is created by the MACRO function %SCAN, which is a means to extract every variable name out from &names1. A MACRO do loop then executes the PROC FREQ procedure for every variable which belongs to the first category and inserts variable name to the output data set that is generated.

```sql
proc sql noprint;
select name into:names1 separated by '#' from a1;
select count(name) into :nvar from a1;
quit;
%MACRO freq;
    %do i=1 %to &nvar;
        %let name=%SCAN(&names1,&i,'#');
        proc freq data=mmpst;
        tables &name / missing out=&name;
        run;
    %end;
%mend;
%freq
```

```
Output 1. Partial Output from the MACRO code above

<table>
<thead>
<tr>
<th></th>
<th>COUNT</th>
<th>PERCENT</th>
<th>var</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15</td>
<td>29.4118</td>
<td>ENS16v</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>11.7647</td>
<td>ENS16v</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
<td>56.8627</td>
<td>ENS16v</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.9608</td>
<td>ENS16v</td>
</tr>
</tbody>
</table>
```

The program succeeded in performing this first step. The values, frequencies and variable names are in the output as requested, but I soon found that there are certain limitations in this code. When I wanted to follow the same strategy to create another MACRO variable called label to store variable label information, I got an unwelcome warning: “The quoted string currently being processed has become more than 262 characters long. You might have unbalanced quotation marks”. This is not even to mention the lengthy error message I got when I tried to apply %SCAN function to this MACRO variable! I knew I could get the label information by merging the PROC FREQ output with PROC CONTENTS output, but I was unwilling to try this relatively clumsy method since my goal is to create a clear and concise MACRO code.

**CALL SYMPUT DATA DEFINED MACRO VARIABLES? A SENSIBLE CHOICE!**

After several experiments, I discovered that a series of operations or procedures can be easily applied to several different data sets by using &var&i operations. The advantages of using CALL SYMPUT DATA defined MACRO variables include not only assigning a sequential number to each variable or each data set, but also providing a convenient way to store the compound character values. In the following program, the order of observation _N_ is used to assign a sequential number to each variable name, label and format. The total number of observations is also retained in &nvar. Note that the resolution to &format&i is not same as the format value for variable &varname&i, it is actually a compound value, i.e. a concatenation of format value and the character “.”. A MACRO do loop executes PROC FREQ for every variable that belongs to the first category, then inserts variable name, variable label and formatted value to the output data set which is generated by PROC FREQ, and finally formats the count and percentage values.
An efficient method to create a large and comprehensive codebook

%MACRO count1;
   data _null_; 
   set index1; 
   call symput(compress("varname"||_N_),trim(name)); 
   call symput(compress("nvar"),compress(_N_)); 
   call symput(compress("vlabel"||_N_),trim(label)); 
   call symput(compress("fformat"||_N_),compress(format)||'.'); 
   run;
   %do i=1 %to &nvar;
      proc freq data=noformat noprint;
      tables &varname&i/missing out=&varname&i;
      run;
      data &varname&i(drop=count percent &varname&i);
      set &varname&i;
      length label$100 varname no per $20;
      fvalue=&&varname&i;
      format fvalue &&fformat&i;
      varname="&&varname&i";
      var=&&varname&i;
      label="&&vlabel&i";
      no=put(count,8.);
      per=compress(put(percent,5.1))||"%";
      run;
   %end;
%mend;
%count1

Output 2. Partial Output from the MACRO code above

The output is almost perfect! Not only do we have nice labels in the data set, but we also have the formatted values of the data. This code is good enough to use for creating a codebook with separated tables for all variables (Output 2 is a good example of this type codebook). However if your goal is to create an integrated codebook such as the one in Figure 1, the program above still has to incorporate one small modification.

TRICK 2: GENERATING NEW VARIABLES WITH PROC FORMAT

When you look at the Output 2 above, it is hard to figure out what there is left to adjust in the code. But if you try to set all of the output data sets together, the problems become clear - the formatted values look very strange. The following output shows the result of setting two variables together. Since those two variables have different formats, the formatted values depicted here are inappropriate.
An efficient method to create a large and comprehensive codebook

Output 3. Output from setting Ens16v and Ens17 together. Note that the values for fvalue are inappropriate.

At first I was stumped, trying to figure out how to resolve this problem. I asked myself, "should I use an IF/THEN/ELSE code to assign the appropriate values to the variables?" After some research and discussion with my colleagues, I quickly rejected this idea since that is quite a tedious process. Instead I found out that there is in fact a very simple way to generate new variables using PROC FORMAT - a method which seems too simple to be true. The PUT function calls the Format &format%i to generate a new variable called fvalue, like so:

\texttt{fvalue=put(&&varname%i,&&fformat%i);} 

<table>
<thead>
<tr>
<th>fvalue</th>
<th>label</th>
<th>varname</th>
<th>no</th>
<th>per</th>
<th>var</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKIPPED</td>
<td>ENS16v - Groups address - Local wellness pol</td>
<td>ENS16v</td>
<td>15</td>
<td>29.4%</td>
<td>A</td>
</tr>
<tr>
<td>NOT ANSWERED</td>
<td>ENS16v - Groups address - Local wellness pol</td>
<td>ENS16v</td>
<td>6</td>
<td>11.8%</td>
<td>C</td>
</tr>
<tr>
<td>NO</td>
<td>ENS16v - Groups address - Local wellness pol</td>
<td>ENS16v</td>
<td>29</td>
<td>56.9%</td>
<td>1</td>
</tr>
<tr>
<td>SKIPPED</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>1</td>
<td>2.0%</td>
<td>2</td>
</tr>
<tr>
<td>NOT ANSWERED</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>5</td>
<td>9.8%</td>
<td>C</td>
</tr>
<tr>
<td>1 OR 2 TIMES</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>6</td>
<td>11.8%</td>
<td>2</td>
</tr>
<tr>
<td>3 OR 4 TIMES</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>17</td>
<td>27.5%</td>
<td>3</td>
</tr>
<tr>
<td>5 OR 6 TIMES</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>7</td>
<td>13.7%</td>
<td>4</td>
</tr>
<tr>
<td>MORE THAN 6 TIMES</td>
<td>ENS17 - Meetings during past 12 months</td>
<td>ENS17</td>
<td>4</td>
<td>7.8%</td>
<td>5</td>
</tr>
</tbody>
</table>

Output 4. Output from setting Ens16v and Ens17 together. Note that fvalue now has appropriate values.

The complete code is now shown below with the %IF/%THEN %DO loop added to the program to SET all of the output data sets together.

```plaintext
%MACRO count1;
data _null_; 
set index1; 
call symput(compress("varname"||_N_),trim(name));
call symput(compress("nvar"),compress(_N_));
call symput(compress("vlabel"||_N_),trim(label));
call symput(compress("fformat"||_N_),compress(format)||'.');
run;

%do i=1 %to &nvar;
proc freq data=noformat noprint; 
tables &&varname&i/missing out=&&varname&i; 
run;
data &&varname&i(drop=count percent &&varname&i); 
set &&varname&i; 
length fvalue $50 label$100 varname no per $20 ; 
fvalue=put(&&varname&i,&&fformat&i); 
varname="&&varname&i"; 
var=&&varname&i; 
label="&&vlabel&i"; 
no=put(count,8.); 
per=compress(put(percent,5.1))||"%"; 
run;
%if i=1 %then %do;
data totall; 
set &varname1; 
run; 
%end; 
%else %do; 
data totall; 
set totall &&&varname&i; 
run; 
%end; 
%end;
```
FORMAT THE OUTPUT

The final step to create a codebook is to apply the appropriate OUTPUT FORMATs to the data set that we have already generated. PDF, EXCEL or RTF are all good choices for a printable copy of a codebook, as well as for wider electronic distribution. There are many different ways of enhancing the appearance and readability of codebooks. One way is to create a web output that provides readers with additional links to access more detailed information about the variables in the codebook. As Figure 2 shows below, the variable names are used as the links to the variable information. When you click the variable name you want to explore, you are directed to the appropriate destination where a table containing even more detailed variable information is located. To further enhance the HTML output, you can use various style elements available in SAS. Alternatively, you can add a compute block along with a style statement. If you are very comfortable with HTML, you even can use Hypertext Markup Language directly in SAS. The Appendix shows a complete code to get a codebook that resembles the style depicted in Figure 2 by embedding Hypertext Markup Language in SAS program.

EXAMPLE CODEBOOK

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDATE</td>
<td>Interview date</td>
</tr>
<tr>
<td>START</td>
<td>Start Time</td>
</tr>
<tr>
<td>RACE</td>
<td>Racial group</td>
</tr>
<tr>
<td>ATLCTY</td>
<td>Eligibility, Atlanta county of residence</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>Country of birth</td>
</tr>
<tr>
<td>SPECBORN</td>
<td>Specify country of birth</td>
</tr>
<tr>
<td>YEARUSA</td>
<td>Year first came to US</td>
</tr>
<tr>
<td>MSTAT</td>
<td>Marital status</td>
</tr>
<tr>
<td>MSTAT_M</td>
<td>Marital status clarify</td>
</tr>
<tr>
<td>SCHOOL</td>
<td>Education</td>
</tr>
</tbody>
</table>

IDATE

Label: Interview date

Universe: Apply to all

Note: Will use it to recalculate Age at the closeout

<table>
<thead>
<tr>
<th>Length</th>
<th>Label</th>
<th>Value</th>
<th>Formatted Value</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Interview date</td>
<td>MMDDYY</td>
<td>DATE9</td>
<td>12535</td>
<td>99.94</td>
</tr>
<tr>
<td>8</td>
<td>Interview date</td>
<td>.</td>
<td>Missing</td>
<td>8</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Figure 2. Codebook in HTML format

CONCLUSION

This paper serves as an introduction to generating clean and comprehensive codebooks that contain information for a large number of variables. The programs presented in this paper can be easily modified to fit the needs of different users. I hope to have provided sufficient explanation for even a beginner level SAS programmer to learn how to handle creating such an extensive report. Although my methods are basic, they are extremely useful even in an advanced user’s day-to-day programming.
APPENDIX – SAMPLE CODES TO GENERATE HYPERLINK EMBEDDED HTML OUTPUT

%MACRO output;
data _null_
set AF;
call symput(compress("varname"||_N_),trim(name));
call symput(compress("nvar"),compress(_N_));
call symput(compress("vlabel"||_N_),trim(label));
varlabellen=length(compress(label));
call symput(compress("varlabellen"||_N_),trim(varlabellen));
run;
/* HTML - Write header */
%let tab='09'x;
data _NULL_
file "C:\Documents and Settings\codebook.html";
put "<html>";
put "<head>";
put "<h1>HET2 CODEBOOK</h1>";
put "</head>";
put "<body>";
put "<p>";
put '<table border cellspacing="0" cellpadding="2">';
%do i=1 %to &nvar ;
put &tab. '<tr>

put &tab. &tab. '<td align="left"><a href="#' "&&&varname&i." ">' &&&&varname&i.
'</a></td>

%if &&&varlabellen&i.>0 %then %do;
put &tab. &tab. &tab. '<td>&&&vlabel&i.</td>
%end;
%else %do;
put &tab. &tab. &tab. '<td><i>(This variable has no label.)</i></td>
%end;
put &tab. '</tr>

%end;
put '</table>

run;
/* HTML - Write each variable */
%do i=1 %to &nvar 

data _NULL_
file "C:\Documents and Settings\wsong\Desktop\New Codebook\codebook.html" mod;
put "<p>";
put '<big><b><a name="' "&&&varname&i." ">' &&&&varname&i.
'</a></big></b></big>
<br><br>
%if &&&varlabellen&i.>0 %then %do;
put "Label: &&&vlabel&i";
%end;
%else %do;
put "<i>(This variable has no label.)</i>
<br><br>
%end;

run;
DATA T (drop=varnum name);
SET &&varname&i;
RUN;
proc sql noprint;
select count(flength) into: lastobs from T;
quit;
data _NULL_
SET T;
file "C:\Documents and Settings\wsong\Desktop\New Codebook\codebook.html" mod;
if _N_=1 THEN DO;
put '<table border cellspacing="0" cellpadding="2">';
put &tab. '<tr>';
ACKNOWLEDGMENTS

I would like to thank my boss Tonja Kyle and our client (CDC/OID/NCHHSTP) for providing me the challenges over the years. I also want to thank my colleagues Kamya Khanna and Baibai Chen, who are the co-authors for this paper, for their helpful discussions.

CONTACT INFORMATION

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

Name: Wen Song  
Enterprise: ICF International  
Address: 11785 Beltsville Dr, suite 300  
City, State ZIP: Beltsville, MD, 20705  
Work Phone: 301-572-0963  
E-mail: wen.song@icfi.com

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.