PharmaSUG 2019 - Paper BP-255

End of Computing Chores with Automation: SAS® Techniques That Generate SAS® Codes

Yun (Julie) Zhuo, PRA Health Sciences

ABSTRACT

SAS programmers often have to confront computing chores that require repetitive typing. Manual coding is time consuming and inefficient. It also almost inevitably introduces human errors which compromise quality. While SAS has no shortage of automation techniques, it does not always occur to SAS programmers to write SAS codes that generate SAS codes. In this paper, we focus on three code-generating techniques. Each technique will be demonstrated using a simple practical example of automating variable label creation. Code samples will be provided. This paper also provides programming tips, explores other applications, and compares the three techniques using the simplicity, flexibility, and efficiency metrics.

INTRODUCTION

In this paper, we are going to introduce, demonstrate, and compare the following three techniques with code-generating capabilities:

- Create and resolve a macro variable
- Call Execute routine
- Put statements that write codes to an associated file

We will demonstrate that the code generating techniques will improve efficiency and accomplish repetitive tasks that may not be easily accomplished otherwise.

THE EXAMPLE

This paper uses a simple practical example of creating SAS variable labels to demonstrate the automation techniques. Creating analysis datasets with descriptive variable labels is a daily routine for statistical programmers. Traditionally, programmers type up variable labels using label statements, as shown in Display 1. The ability to automatically label variables instead of manually typing up many lines of codes in a label statement will be a great time saver.

```
create table Analysis.&pgrm(label='Subject Level Dataset') as
 select STUDYID label='Study Identifier', USUBJID label='Unique Subject Identifier',
        SUBJID label='Subject Identifier for the Study', SITEID label='Study Site Identifier',
        AGE label='Age', AGEU label='Age Units', SEX label='Sex',
        RACE label='Race', RACEN label='Race (N)',
        ETHNIC label='Ethnicity', ETHNICN label='ETHNICN (N)', COUNTRY label='Country',
        HEIGHT label='Height (cm)', WEIGHT label='Weight (kg)', BMI label='Body Mass Index (kg/m2)',
        SYSBP label='Systolic Blood Pressure (mmHg)', DIABP label='Diastolic Blood Pressure (mmHg)',
        HR label='Heart Rate (beats/min)', TEMP label='Temperature (celsius)',
        SMOKING label='smoking History', NYHACLAS label='NYHA Classification',
        NUMCAT3 label='CIRS-G Categories at Level 3 Severity', NUMCAT4 label='CIRS-G Categories at Level 4 Severity',
        TOCAT label='Number of CIRS-G Categories Endorsed',
        NHLHIST label='NHL Histology', ENTRCRT label='Entry Criterion',
        BULKYDIS label='Presence of Bulky Disease', PREPI3K label='Previous Treatment with PI3K Inhibitors',
        ECOG label='ECOG Performance Status', HIST label='Histology',
        STAGE label='Stage at Study Entry', DIAGDTC label='Date of Initial Diagnosis',
        DIAGRAND label='Time since Initial Diagnosis (m)',
        FSTPDTC label='Date of First Progression', MRCTPDTC label='Date of Most Recent Progression',
        TfirstProg label='Time since First Progression (Month)', TrecProg label='Time since Most Recent Progression (Month)',
        LTRRCPM label='Time from Lst Trt to Most Recent Prg (m)',
        FASFL label='Full Analysis Set Population Flag', SAFFL label='Safety Population Flag',
```

Display 1. Create Variable Label with Label Statements

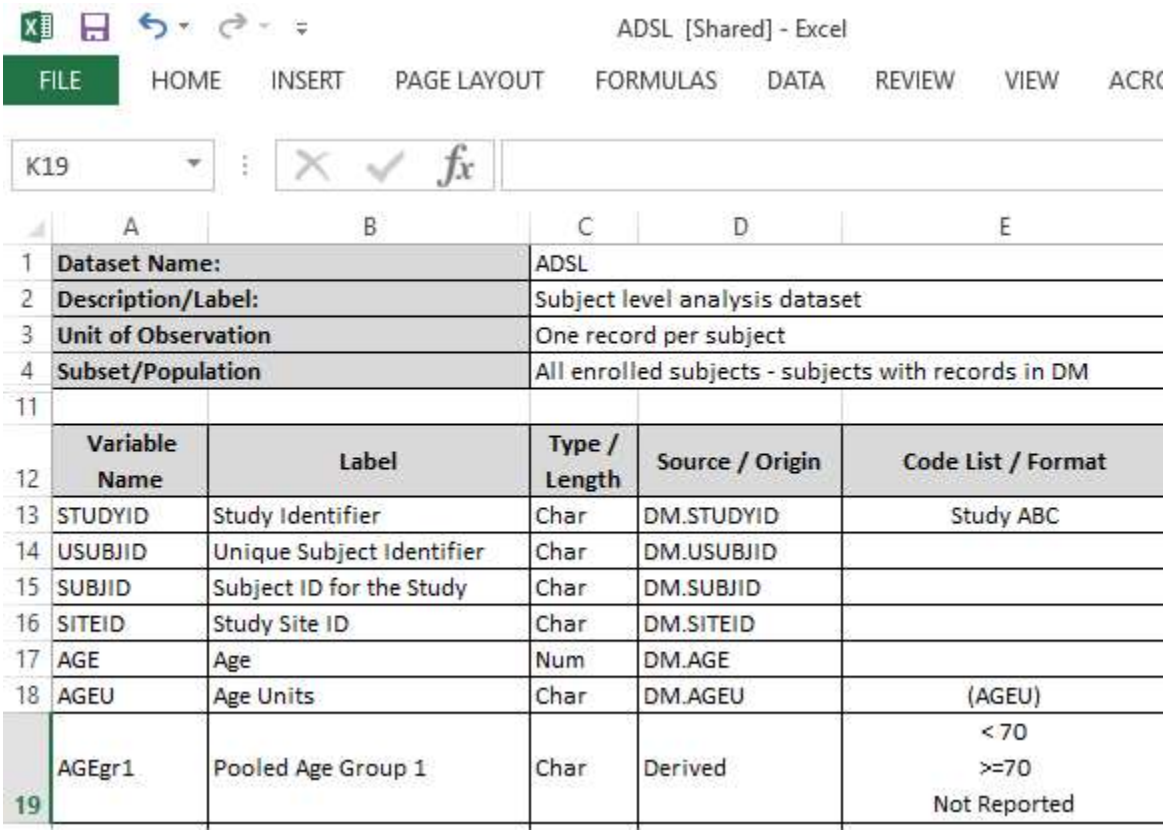
In our example, we will use a specification spreadsheet and an intermediate data set with missing labels as our input sources:

- A specification EXCEL® spreadsheet. The data dictionary stores specifications such as data set variable name, variable label, variable type, variable origin, and variable derivation logic. A partial snapshot of a specification EXCEL file is illustrated in Display 2.

  We use the IMPORT procedure to read the specification spreadsheet into SAS. Note that we only need the first two columns starting at row number 12. We name the imported data set 'META'. Sample code is below.

```
PROC IMPORT OUT= WORK.META
            DATAFILE= "<file directory and name>"
            DBMS=EXCEL REPLACE;
     RANGE="<sheet name>.$A12:B200";
     GETNAMES=YES;
     MIXED=NO;
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;
```

- An interim SAS data set with all the variables created but all the labels still missing. For demonstration purpose, we name the data set as interim_data.sas7bdat.



	A	B	C	D	E
1	**Dataset Name:**		ADSL		
2	**Description/Label:**		Subject level analysis dataset		
3	**Unit of Observation**		One record per subject		
4	**Subset/Population**		All enrolled subjects - subjects with records in DM		
11					
12	**Variable Name**	**Label**	**Type / Length**	**Source / Origin**	**Code List / Format**
13	STUDYID	Study Identifier	Char	DM.STUDYID	Study ABC
14	USUBJID	Unique Subject Identifier	Char	DM.USUBJID	
15	SUBJID	Subject ID for the Study	Char	DM.SUBJID	
16	SITEID	Study Site ID	Char	DM.SITEID	
17	AGE	Age	Num	DM.AGE	
18	AGEU	Age Units	Char	DM.AGEU	(AGEU)
19	AGEgr1	Pooled Age Group 1	Char	Derived	< 70 >=70 Not Reported

Display 2. Specification Data in Excel spreadsheet format.

THE TECHNIQUES

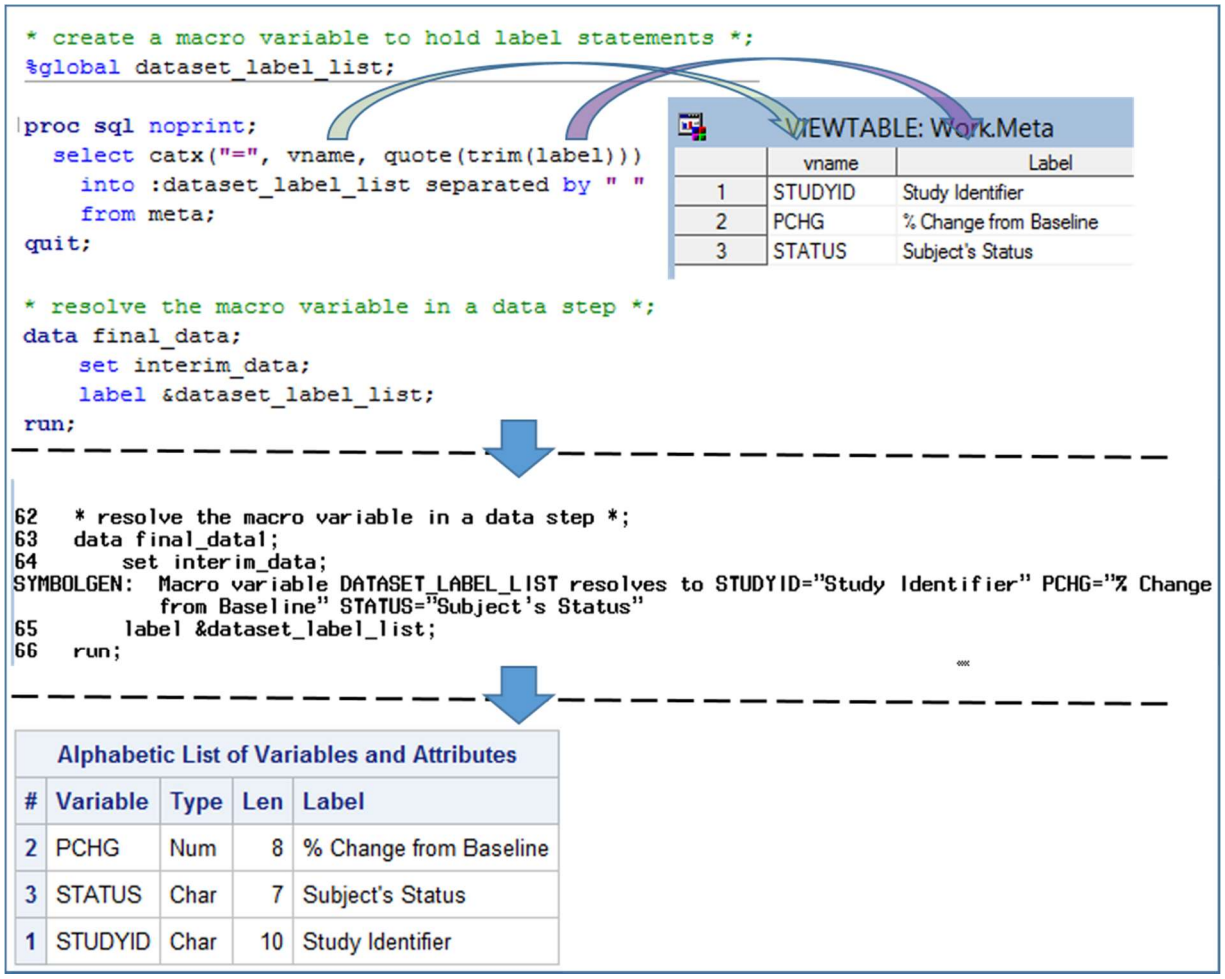
Three code-generating techniques are available to automate the process of label creation. We will demonstrate that they all produce the same results while each technique has its own strength and weakness.

THE MACRO VARIABLE METHOD

The Macro Variable method creates a macro variable to store the codes for the label statement. Resolving the macro variable after a label statement in a data step will generate the SAS codes we need to create labels for all the variables from the META dataset. With the SYMBOLGEN option, we are able to see the generated codes in the log. The diagram below in Figure 1 demonstrates the three-step process of the technique.

Figure 1. Steps to Create Labels with the Macro Variable Technique.

In Display 3. Create Variable Labels with the Macro Variable Technique.below, we show the SAS program, the SAS log, and the SAS output involved in the process.



Display 3. Create Variable Labels with the Macro Variable Technique.

The first block in Display 3 contains SAS codes that create and resolve the macro variable. As shown in the display, we start off creating a global macro variable named 'dataset_label_list' using the SQL procedure. Then, SAS interacts with the META data set to retrieve variable names and labels, and store them in the 'dataset_label_list' macro variable.

The second block contains the generated codes in the log. In a data step, we resolve the 'dataset_label_list' macro variable after a label statement. With the SYMBOLGEN option turned on, we can view the generated codes in the log.

The last block is the output from a CONTENT procedure showing variables and the newly created labels.

THE CALL EXECUTE METHOD

The CALL EXECUTE routine builds SAS codes dynamically as data step iterates, making it an effective SAS code generator. Figure 2 illustrates the work process of the technique. It is of note that the Call Execute routine has the capacity of combining the following two steps in one data step: it builds SAS codes while iterating through the input dataset, then it executes the codes automatically.

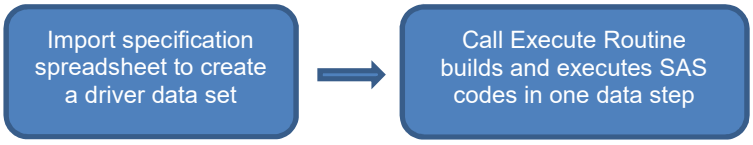
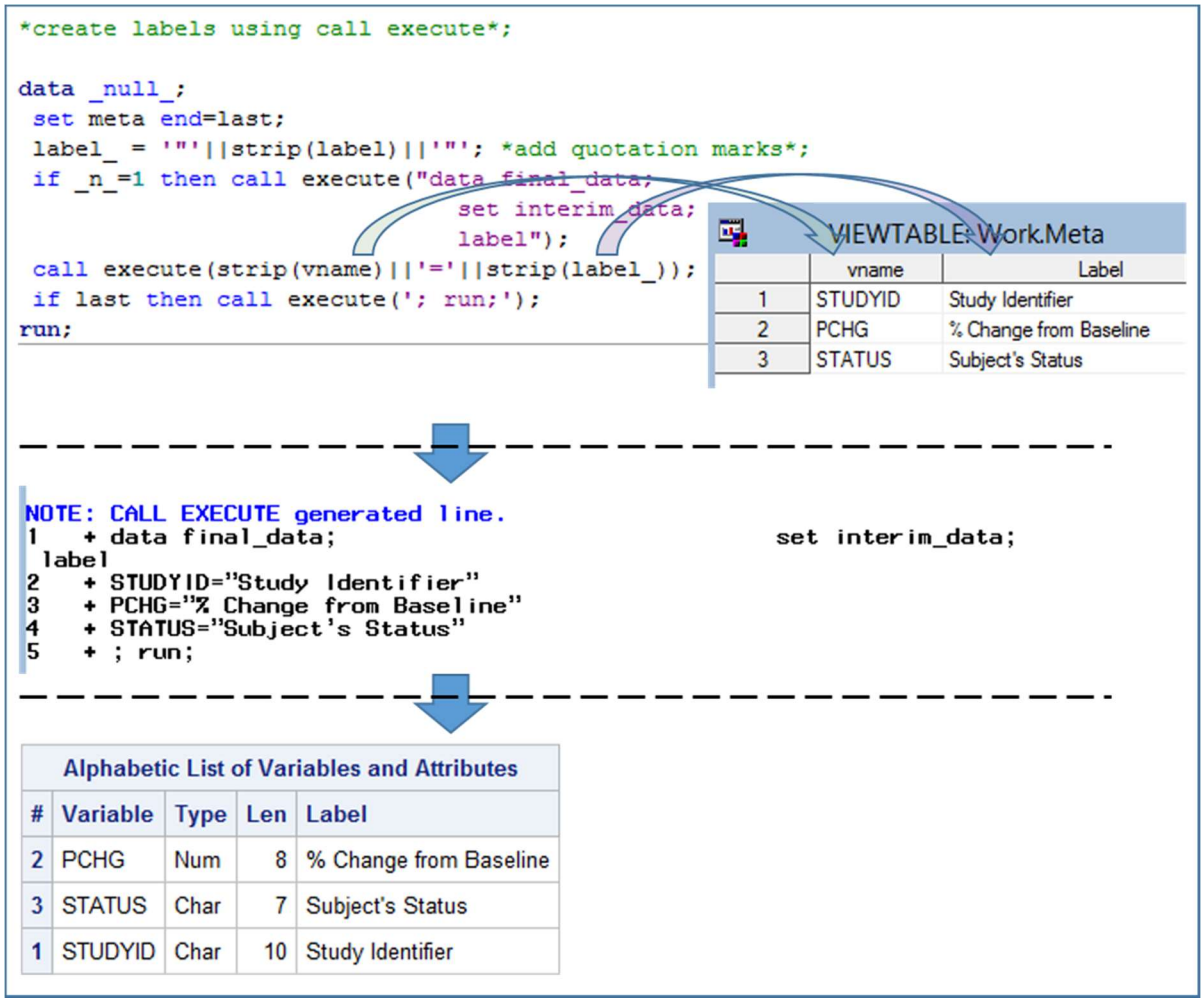


Figure 2. Steps to Create Labels with the CALL EXECUTE Routine.

In Display 4 below, we show the SAS program, the SAS log, and the SAS output involved in the process.



```
*create labels using call execute*;

data _null_;
 set meta end=last;
 label_ = '"'||strip(label)||'"'; *add quotation marks*;
 if _n_=1 then call execute("data final_data;
                             set interim_data;
                             label");
 call execute(strip(vname)||'='||strip(label_));
 if last then call execute('; run;');
run;
```

VIEWTABLE: Work.Meta

	vname	Label
1	STUDYID	Study Identifier
2	PCHG	% Change from Baseline
3	STATUS	Subject's Status

```
NOTE: CALL EXECUTE generated line.
1    + data final_data;                                  set interim_data;
 label
2    + STUDYID="Study Identifier"
3    + PCHG="% Change from Baseline"
4    + STATUS="Subject's Status"
5    + ; run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
2	PCHG	Num	8	% Change from Baseline
3	STATUS	Char	7	Subject's Status
1	STUDYID	Char	10	Study Identifier

Display 4. Create Variable Labels with the CALL EXECUTE Routine.

In the first block of Display 4, in a data step that reads the META data set, we use the Call Execute routine to dynamically build code using the variable names and labels in the META data set. The Call Execute routine interacts with the META data set to retrieve information.

The second block shows the CALL EXECUTE generated codes in the log. By default, the Call Execute routine will automatically print generated codes to the log.

The last block is the PROC CONTENT output showing variable names and the newly created labels.

THE PUT STATEMENT METHOD

The PUT statement has the capacity of writing text, making it possible to generate SAS codes. Figure 3 below demonstrates the work process. This method involves three steps. Firstly we associate a file name to a SAS program. Then we build SAS codes into the associated file using the PUT statement. Finally, we use %INCLUDE to execute the generated codes.

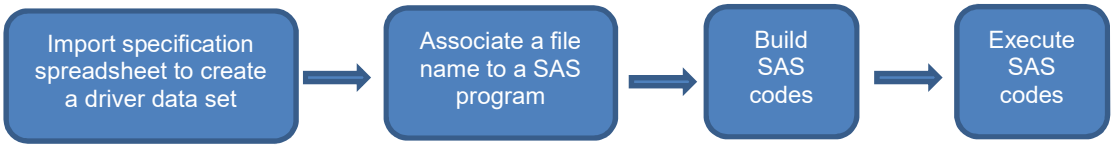
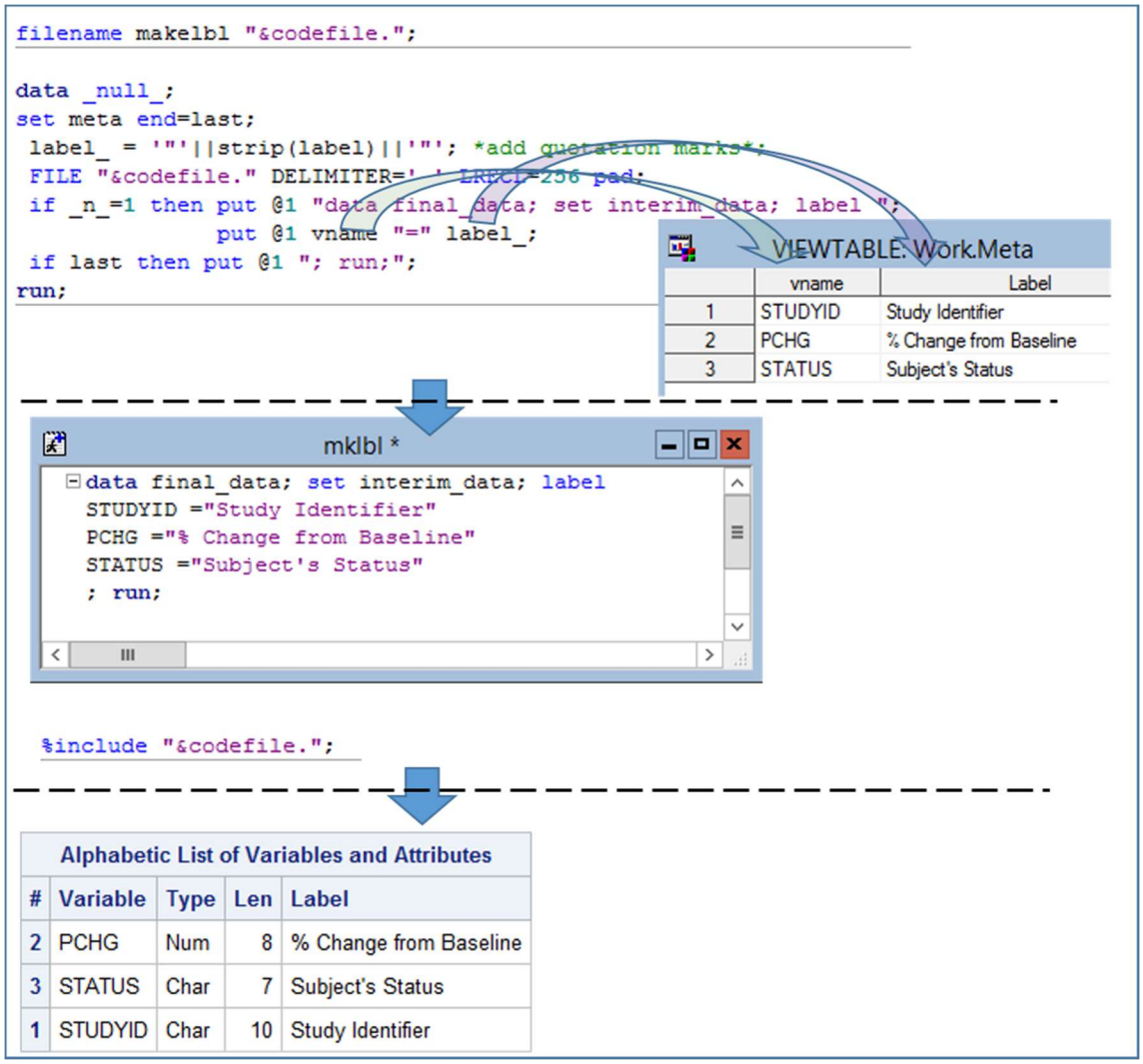


Figure 3. Steps to Create Labels with the Put Statement Method.



Display 5. Create Variable Labels with the Put Statement Method.

In Display 5 above, we show the code-writing SAS program, the SAS-generated SAS program, and the SAS output involved in the process.

The first line of the SAS code uses the FILENAME statement. The purpose of the statement is to associate a SAS name (in this example we name it 'makelbl') with an external file. The macro variable &codefile in our example will resolve to a valid file specification referencing the SAS program that we are going to generate.

Following the FILENAME statement, we use a data step to interact with the input META data set. We then use a series of PUT statements to write SAS codes, which will be written to the output file specified in a FILE statement.

We can open the associated file to view the generated SAS codes, which is demonstrated in the middle block of the display. In our example, the name of the SAS program is mklbl.sas. The middle block of the display also contains the %INCLUDE which we use to invoke and execute the SAS program.

Lastly the third block of the display contains the PROC CONTENT output which is the same as the outputs we generated using the previous two methods.

PROGRAMMING TIPS

A common feature of the three techniques is the creation, concatenation, and manipulation of text strings which make up the SAS codes. In this section, we will share a few programming tips to avoid some potential pitfalls commonly encountered during text manipulation.

1. Quotation marks:

   Quotation marks are required for both Macro Variable method and Put Statements method to enclose the text strings in the labels. Without quotation marks, the generated codes will hang while execution. Although quotation mark is optional for the CALL EXECUTE routine, it is recommended in order not to lose apostrophe marks in the labels. Here are two examples of adding quotations shown in previous displays:

   select catx("=", vname, quote(trim(label))) into :dataset_label_list

   label_ = ""||strip(label)||"";

2. %NRSTR quoting function:

   The CALL EXECUTE routine has the capacity of resolving macro references. As a result, If the generated codes contain non-macro language constructs for assigning macro variables during run time (e.g., call symput or proc sql into:), we recommend using the %NRSTR quoting function to avoid errors on unresolved macro variable references.

3. Text concatenation functions:

   A good understanding of the concatenation functions is necessary in order to write SAS from SAS. Please note the following default behaviors of the four concatenation functions.

   - CATS: removes both leading and trailing blanks from each string before concatenation.
   - CATX: removes both leading and trailing blanks from each string before concatenation; in addition, a delimiter to be added between the strings is specified in the first function parameter.
   - CATT: removes only trailing blanks from strings to be joined
   - CAT or the traditional double bars ||: does not remove any blank, leading or trailing

   Although the ability to automatically remove blanks come in handy in many situation, for our purpose of writing SAS codes, leading and/or trailing blanks are often necessary. We need to be aware of situations where we should use CAT or the traditional double bars in order to keep, instead of remove, the blanks. For example, the following code-generating codes cannot use CATS or CATX because we need the leading blanks (in yellow highlight).

```
proc sql noprint;
 select distinct catt('driver (where=(n=', N,
                  ') rename=(AEDECOD=AEDECOD', N,
                  ' AESER=AESER', N,
                  ' AETOXGRN=AETOXGRN', N,
                  '))')
      into :newcode separated by ' '
      from driver;
quit;
```

COMPARISON

Selection of an automation technique depends on a number of factors including your comfort level with each technique. This paper will discuss the simplicity, flexibility, and system efficiency of each technique to aid in your decision.

The Macro Variable method and the Call Execute method both require similar amount of coding. By contrast, the Put Statement method requires a larger amount of coding. For example, in our example of creating variable labels, the macro variable technique takes 26 words and 201 characters to do the job, the Call Execute technique takes 25 words and 192 characters, while the Put Statement technique takes 41 words and 248 characters.

The Call Execute technique has the advantage of combining SAS code generation and execution in one data step, making a more simplified process. By contrast, the Put Statements has a more complex process as it involves three distinct steps including file association, code generation, and SAS execution.

The Macro Variable technique is subject to the limitation that the length of macro variables cannot exceed 65,534 characters. By contrast, the other two techniques do not have such limitations.

The Put Statement method is the only method that actually generates a physical SAS program which we can open, review, and even modify. As a result, the technique gains a significant amount of flexibility and clarity by sacrificing simplicity.

To evaluate and compare system efficiency of the three techniques, we run each technique 20 times, 10 times in interactive mode and 10 times in batch mode, on an interim SAS data set with 121 variables and 40,000 records during various times when host system resources are unlikely to be in use. We use the following codes as stopwatch placed in the beginning and the end of each SAS run.

```
%PUT Start: %SYSFUNC(datetime(),datetime23.3);
%let Start = %SYSFUNC(time());
%PUT End: %SYSFUNC(datetime(),datetime23.3);
%let End = %SYSFUNC(time());
```

Then we use the following formula to calculate real run time of each SAS run.

```
Runtime = %SYSFUNC(round(%SYSEVALF(&end-&start),.001))
```

The results are illustrated in Table 1 below. Within the interactive mode, the Macro Variable method runs more efficiently compared to the other methods. Within the batch mode, the Call Execute method is the most efficient.

The Call Execute method runs faster in the batch mode, but slower in the interactive mode, which is most likely due to the fact that the Call Execute routine slows down when writing generated codes to the log window in the interactive mode.

Mode	Method	Mean	SD	Minimum	Maximum
Interactive	Macro Variable	0.092	0.004	0.083	0.098
	Call Execute	0.120	0.004	0.111	0.125
	Put Statement	0.107	0.007	0.101	0.124
Batch	Macro Variable	0.082	0.008	0.077	0.104
	Call Execute	0.079	0.002	0.077	0.084
	Put Statement	0.089	0.002	0.087	0.094

Table 1. Comparison of Real Run Time in Seconds

It is of note that each SAS port yields different performance statistics based on the host operating system and the network resources. Therefore, the measurements above only serve the purpose of comparing efficiency for the techniques in discussion.

DISCUSSION

In this paper, we chose a simple example of creating variable labels for ease of demonstration. The code generating techniques have a wide array of applications on both simple and complex programming tasks. Here are a few examples where the techniques have proven to improve efficiency:

1. Transpose a large number of variables automatically
2. Import a variety of raw data files into SAS with one code-generating SAS program
3. Automatic adverse event toxicity edit check in clinical programming
4. Specification driven lab CTC grading in clinical programming
5. Meta data driven mapping for SDTM and ADaM data sets

Therefore, when encountered with repetitive tasks or tedious typing, programmers are encouraged to consider the code generating approach as an automation option.

CONCLUSION

In this paper, we use a simple example of creating variable labels to demonstrate that automatic SAS code generators are reliable and efficient methods for repetitive programming tasks. They will significantly cut down on programming time. Higher efficiency will also lead to better quality. All three techniques introduced by this paper will generate the same results, with each technique having its own advantages and disadvantages.

The code-generating approach has served us well. It can be applied in a wide range of programming tasks, with the potential of eliminating repetitive programming chores for SAS programmers.

REFERENCES

Batkhan, L. 2017. “SAS Blogs: CALL EXECUTE Made Easy for Data-Driven Programming.” Accessed July 26, 2018. https://blogs.sas.com/content/sgf/2017/08/02/call-execute-for-sas-data-driven-programming/

Dai, Y and Li, Y 2018. “SAS® automation techniques – specification driven programming for Lab CTC grade derivation and more” Proceedings of the PharmaSUG Conference 2018. Available at https://www.pharmasug.org/proceedings/2018/BB/PharmaSUG-2018-BB12.pdf

Gau, L. 2004. “Write SAS® Code to Generate Another SAS® Program A Dynamic Way to Get Your Data into SAS®” Proceedings of the SAS User Group International 2004, Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi29/175-29.pdf

Shan, X. 2015. “Transpose Dataset by MERGE.” Proceedings of the SAS Global Forum 2015, Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings15/

Zhuo, Y. 2018. “Automate Repetitive Programming Tasks: Effective SAS® Code Generators.” Proceedings of the Western Users of SAS Software 2018, Cary, NC: SAS Institute Inc. Available at https://www.lexjansen.com/wuss/2018/73_Final_Paper_PDF.pdf

ACKNOWLEDGMENTS

I appreciate my former colleagues at Axio Research for their inputs.

CONTACT INFORMATION

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

Yun (Julie) Zhuo
PRA Health Sciences
ZhuoYun@prahs.com
Kite Pharma
jzhuo@kitepharma.com

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