ABSTRACT
SAS programmers often need to perform computing chores that require a lot of repetitive typing. Examples include labeling variables for a large data set or transposing a large number of variables. This paper introduces and compares two methods to automate repetitive tasks. The reliability, efficiency, as well as potential pitfalls of the two methods will be demonstrated with two examples. The audience should be comfortable with base SAS and the basic SAS macro language.

INTRODUCTION
Although typing is an inseparable part of programming, repetitive typing does not have to be. Manual typing is not only tedious and time consuming, it also introduces human errors that always compromise quality. Although SAS has an abundance of automation techniques, it does not always occur to SAS programmers that we can write SAS codes to generate SAS codes. SAS code generators include the following:

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

In this paper, we are going to introduce, demonstrate, and compare the first two techniques with two examples. In the first example, we are going to automate the creation of data set labels through a driver data set, which is a specification excel spreadsheet. In the second example, we are going to use the same techniques to transpose variables from long format to wide format. In this scenario, we do not have a readily available driver data set, but we are able to create one with some manipulation in the input data set.

EXAMPLE 1
SAS programmers create a large amount of data sets. Adding descriptive variable labels to SAS data sets is necessary for them to be useful downstream. Typing up variable labels using label statements is a tedious and daunting task. The traditional manual approach is demonstrated in Display 1 in page 2.

In this section, we are going to demonstrate the creation of variable labels using SAS code generators. The input sources are the following:

- 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 in page 2.
- 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.

THE MACRO VARIABLE TECHNIQUE
With the macro variable technique, we follow the steps in Figure 1 to accomplish the task.

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Label</th>
<th>Type / Length</th>
<th>Source / Origin</th>
<th>Code List / Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDYID</td>
<td>Study Identifier</td>
<td>Char</td>
<td>DM.STUDYID</td>
<td>Study ABC</td>
</tr>
<tr>
<td>USUBJID</td>
<td>Unique Subject Identifier</td>
<td>Char</td>
<td>DM.USUBJID</td>
<td></td>
</tr>
<tr>
<td>SUBJID</td>
<td>Subject ID for the Study</td>
<td>Char</td>
<td>DM.SUBJID</td>
<td></td>
</tr>
<tr>
<td>SITEID</td>
<td>Study Site ID</td>
<td>Char</td>
<td>DM.SITEID</td>
<td></td>
</tr>
<tr>
<td>AGE</td>
<td>Age</td>
<td>Num</td>
<td>DM.AGE</td>
<td></td>
</tr>
<tr>
<td>AGEU</td>
<td>Age Units</td>
<td>Char</td>
<td>DM.AGEU</td>
<td></td>
</tr>
<tr>
<td>AGEGr1</td>
<td>Pooled Age Group 1</td>
<td>Char</td>
<td>Derived</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt; 70</td>
<td>&gt; 70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not Reported</td>
<td></td>
</tr>
</tbody>
</table>

Display 2. Specification Data in Excel spreadsheet format.
To start off, 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’.

```sas
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;
```

Then we create a global macro variable named ‘dataset_label_list’ using the SQL procedure. In this step, SAS interacts with the META data set to retrieve variable names and labels and then store them in the ‘dataset_label_list’ macro variable.

Finally 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 process is demonstrated in Display 3 below. The first block in the display contains SAS codes that create and resolve the macro variable. The second block contains the generated codes in the log. The last block is the output from a CONTENT procedure showing variables and the newly created labels.

Display 3. Create Variable Labels with the Macro Variable Technique.
THE CALL EXECUTE ROUTINE

The CALL EXECUTE routine builds SAS codes dynamically as data step iterates, making it an effective SAS code generator. In this section, we are going to demonstrate the code-generating capacity of the SAS routine.

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.

Display 4 illustrates the process. 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 second block shows the CALL EXECUTE generated codes in the log. The last block is the PROC CONTENT output showing variable names and the newly created labels.

Display 4. Create Variable Labels with the CALL EXECUTE Routine.
EXAMPLE 2

In the previous example, a specification spreadsheet serves as a convenient driver data set that feeds into the macro variable or the Call Execute routine. In cases where a convenient driver data set is not available, it is possible to generate a driver in order to work with the code generating techniques. In this section, we are going to demonstrate such an example that transposes multiple variables through code generators.

In this example, our input source is a SAS data set in the vertical format, as shown below in Display 5. For illustration purpose, we only display three hypothetical subjects (SUBJID 001, 002, and 003) with three variables related to their adverse events (AEs) in a clinical trial. The goal is to transpose all the AE variables for the subjects to facilitate the creation of a listing output which is in the horizontal format.


To start off, we need to create a driver data set. In the driver data set (Display 6), we added a counting variable named ‘N’.

```
proc sort data=AE;
  by subjid aedecod;
run;

data driver;
  set AE;
  by subjid;
  N+1;
  if first.subjid then N=1;
run;
```


THE MACRO VARIABLE TECHNIQUE

In order to use the macro variable technique, we create a macro variable named ‘newcode’ to store the SAS codes. Note that with a ‘select distinct’, we will be able to generate codes for only distinct numbers in the variable N.
```sql
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;
```

We would like to take a look at the new macro variable with the %PUT statement.

```text
%put &newcode.;
```

Finally we resolve the macro variable 'newcode' in a MERGE statement.

```text
data ae_transposed(drop=n) ;
merge &newcode.;
by subjid;
run;
```

The desired output in the horizontal format is demonstrated below in Display 7.

<table>
<thead>
<tr>
<th>AEDECOD1</th>
<th>AESER1</th>
<th>AETOXGRN1</th>
<th>AEDECOD2</th>
<th>AESER2</th>
<th>AETOXGRN2</th>
<th>AEDECOD3</th>
<th>AESER3</th>
<th>AETOXGRN3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1 Fatigue</td>
<td>Y</td>
<td>Insomnia</td>
<td>N</td>
<td>Rash</td>
<td>2</td>
<td>N</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2 Fatigue</td>
<td>N</td>
<td>1 Renha</td>
<td>Y</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Insomnia</td>
<td>Y</td>
<td>2 Renha</td>
<td>N</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Display 7. Transposed AE Data Set in the Horizontal Format.

THE CALL EXECUTE ROUTINE

Firstly, we create a data set with distinct number of variable N.

```text
proc sort data=driver(keep=n) out=n nodupkey;
  by n;
run;
```

Then we use the CALL EXECUTE routine three times in a data step.

```text
data _null_; 
  set n end=last;
  if _n_=1 then call execute('data ae_transposed(drop=N); merge ');
call execute(catt('driver(where=(n=', N,
    ') rename=(AEDECOD=AEDECOD', N,
    ' AESER=AESER', N,
    ' AETOXGRN=AETOXGRN', N,
    '))'));
  if last then call execute( ';' by subjid; run;');
run;
```

After execution, the codes will generate the same desired output as shown above in Display 7.
COMPARISON

SAS code generators have proven to be effective techniques to generate quality desired results with a significantly less amount of manual coding. Both techniques demonstrated in this paper require similar amount of coding. For example, in the case of creating variable labels, the macro variable technique takes 26 words and 201 characters to do the job, while the Call Execute technique takes 25 words and 192 characters.

The macro variable technique is subject to the limitation that the length of macro variables cannot exceed 65,534 characters. By contrast, the Call Execute technique does not have such limitations. In addition, the Call Execute technique generates SAS codes and executes them with one data step, making the process a bit simpler.

To evaluate and compare system efficiency of the two 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 100 variables and 271 records during various times when host system resources are not 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.

\[
\text{Runtime} = \%SYSFUNC(\text{round}(\%SYSEVALF(\&end-\&start),.001))
\]

The results are illustrated in Table 1 below. In general, batch model runs more efficiently. The Call Execute method runs faster in batch mode, but slower in interactive mode.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Method</th>
<th>Mean</th>
<th>SD</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive</td>
<td>Macro Variable</td>
<td>0.087</td>
<td>0.017</td>
<td>0.062</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>Call Execute</td>
<td>0.197</td>
<td>0.031</td>
<td>0.144</td>
<td>0.235</td>
</tr>
<tr>
<td>Batch</td>
<td>Macro Variable</td>
<td>0.025</td>
<td>0.008</td>
<td>0.015</td>
<td>0.032</td>
</tr>
<tr>
<td></td>
<td>Call Execute</td>
<td>0.013</td>
<td>0.007</td>
<td>0.000</td>
<td>0.016</td>
</tr>
</tbody>
</table>

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.

CONCLUSION

In this paper, we 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. The code-generating approach has served us well. It has the potential of eliminating repetitive programming chores for SAS programmers.

REFERENCES


ACKNOWLEDGMENTS

I appreciate insights from Paul Stutzman, Manager of Programming Infrastructure at Axio Research. I would like to thank the management at Axio Research for their support in writing and presenting this paper.

CONTACT INFORMATION

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

Yun (Julie) Zhuo
Axio Research, LLC
yunz@axioresearch.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.