

Building Sequential Programs for a Routine Task with Five SAS® Techniques

Gongmei Yu and Paul LaBrec, 3M Health Information Systems.

ABSTRACT

When a task needs to be implemented on a regular basis and has a tight timeline, the SAS® programs should be designed with minimum required updates to the code and maximum automation in each processing step.

This paper intends to illustrate the process of developing such programs with five SAS techniques: (1) Macro variable; (2) Macro program; (3) Conditional and iterative statement; (4) %Include statement; (5) DDE (Dynamic Data Exchange) outputting.

This paper assumes basic knowledge of SAS procedures, macro language and the use of program logic. The emphases of the paper is to illustrate how the five techniques can be applied to the specific task of calculating healthcare payment weights. The code may include other techniques not addressed in this paper.

INTRODUCTION

In one multiple year contract, we are required to implement a task twice a year, with the most recently available data. The main goal of this task is to recalibrate long-term care diagnosis-related groups (LTC-DRG) relative weights which will be applied for LTCH prospective payment calculation in next Federal fiscal year. Specifically, valid claims from LTC population are first classified into distinct diagnostic groups (MS-DRGs--Medicare Severity-Diagnosis Related Group), then relative weights are estimated with cost related information for each DRG. A few adjustment factors are also calculated based on simulations of payments under six scenarios.

The whole calculation process is complex and the task also has a very tight timeline. Within one week or less from the time we receive data, we are required to produce the final weights, output the final results, and include some intermediate results into over 20 TABs in a pre-formatted Excel file. Also, in the middle of the process, we need to send intermediate results to the client. The client uses the interim data to make judgments on final steps and provides us new input information for next processing steps.

High accuracy and quick turnaround are critical for this task. To achieve these two goals, we designed our SAS code in a way that requires minimum code updates and achieves maximum automation in the implementation process.

This task was designed to be implemented with 6 sequential steps. A core SAS program is developed for each step. Figure 1 displays the structure of each sequential program and its function. A core SAS program could be one program or comprised of multiple sub-programs. Each sub-program has its objective and serves as a block in the core program. All the core programs are relatively independent but required to execute in the sequential order. P0 and P1 are designed for updating data. P2 to P4 implement three major sub-tasks which involve extensive data manipulation, iterative calculation, and simulations. P5 outputs results into a predefined Excel template.

For each update, a new set of folders that house the SAS programs, input data, and output will be constructed, parallel with folders constructed for previous updates. Figure 2 below illustrates the folder structure for this project (LTC). The folders at the left (highlighted as dark blue) served as the base folder and they are shared by every execution of the project. On the right side, (highlighted as light blue) are folders for this current update. The one highlighted as green represents folders for past updates and the one highlighted as orange represents folders for next year's update.

With this folder structure, we keep all the input/output SAS data file names the same since they are saved in different folder for each update. For each update, only programs P0 and P1 need to be modified to

update information associated to the current update. No code change are needed for the rest of the programs unless there is a change in the methodology or output requirement.

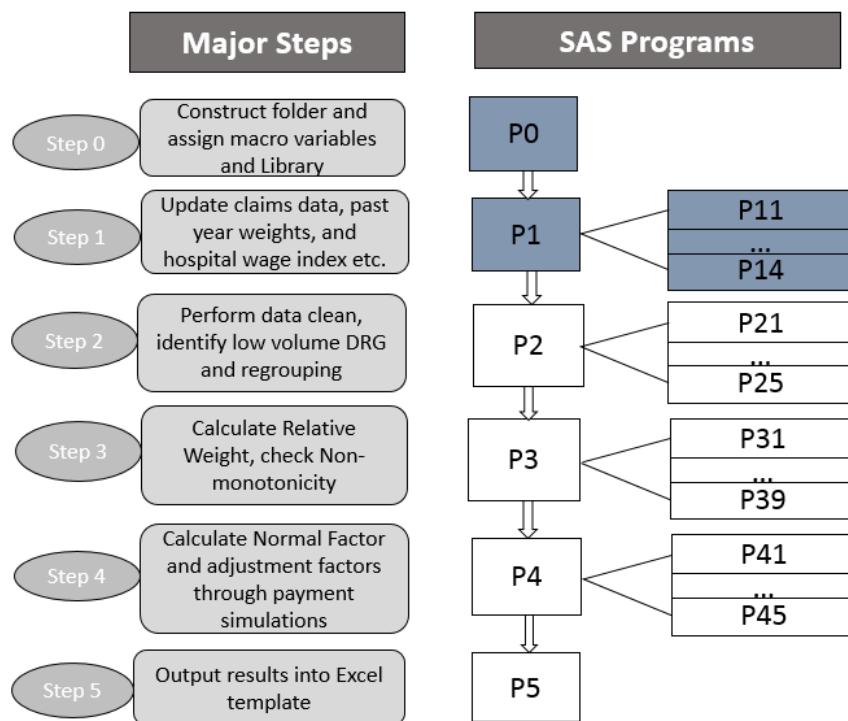


Figure 1 Structure of Sequential Programs and Their Function

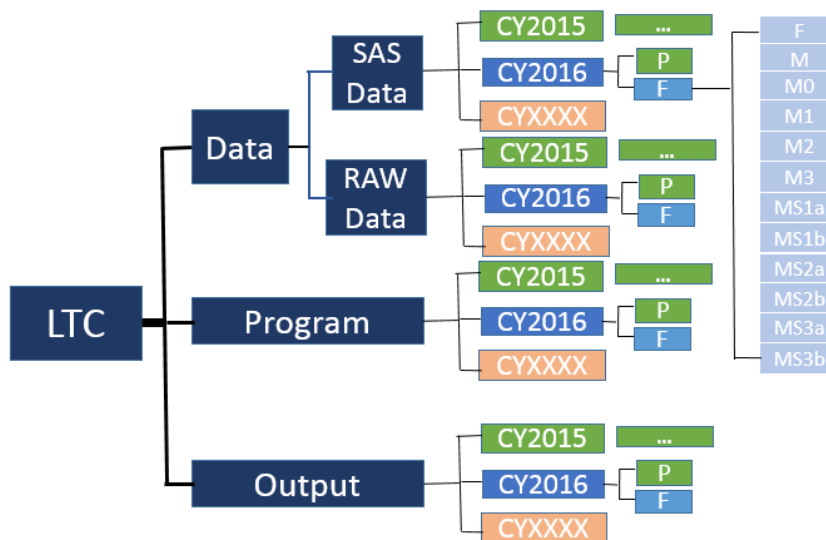


Figure 2 Folder Structure

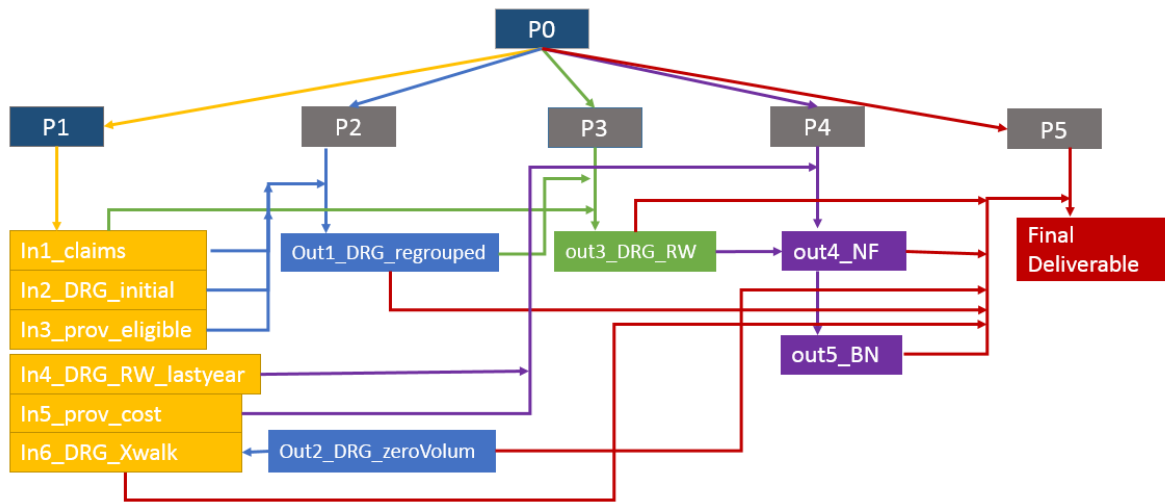


Figure 3 Relative Weight Calculation Program Flow Chart

Figure 3 displays the interaction between the programs, input data and output results. P0 is on top of all other programs since it assigns global macro variables and library names that will be applied in all other programs. P1 to P5 need to be executed in the sequential order since the output needs to be used as input data for the next step. However, they are also relatively independent. For example, when out5_BN need to be updated due to the update of In5_prov_cost, we only need to execute part of the program in P1 to update in5 and then execute P4. P2 and P3 do not need to be re-run since they are unaffected by the change of in5.

To implement the analysis design, five SAS techniques were extensively applied in the SAS coding. The following sections will illustrate how these five techniques were utilized to achieve minimum updating and maximum automation.

TECHNIQUE 1: MACRO VARIABLE

A macro variable is a short hand way of referring a string. In the program P0_XXXX below, a few macro variables are defined to update year, version, and parameters. These macro variables could be referenced later in the definition of other macro variables, libraries, and in the rest programs. Each time a macro variable is referenced, the macro variable is substituted with its value.

The program P0_XXXX serves as the must run program in the sequential programs. It assigns all the global macro variables and library used in P1-P5.

This coding design achieved minimum updating. Each time, once the value of macro variables in part ① are updated (highlight in blue), the value of other macro variables in ② and library names in ③ and all other references in sequential programs will be updated automatically without changing the code.

```

/*****
/* program name: P0_assign_MacroVariable.sas */
/* Purpose: assign macro variables, library name */
*****/

%let CY=CY2016;
%let FY=FY2018;
%let version=F; /* P for Prelim and F for Final */
%let DRGv1=v33;
%let DRGv2=v34;
%let DRGv3=v35;
  
```

①

```

%let ssmin=8 /* Minimum LOS value Inclusion */;
%let LTCH_Finalrate_full=42901.17;
%let LTCH_Finalrate_reduced=42051.65;
%let Rootpath=%str(E:\Projects\LTC);

/* Other Macro variable definition */
%let rawfilepath=%str(&Rootpath.\Data\Rawdata\&CY.\&version);
%let outfilepath=%str(&Rootpath.\Output\&CY.\&version);
%let SASdatapath=%str(&Rootpath.\Data\SASdata\&CY.\&version);
%let programpath=%str(&Rootpath.\Program\&CY.\&version);
%let Exelname=%str(&FY._&version..xlsx);

/* Assign library name */
Libname LTC_M "&SASdatapath.\M"; /* for intermediate results */
Libname LTC_F "&SASdatapath.\F"; /* for final results */

Libname LTC_M0 "&SASdatapath.\M0"; /* for initial run */
Libname LTC_M1 "&SASdatapath.\M1"; /* for first run if necessary */
Libname LTC_M2 "&SASdatapath.\M2"; /* for second run if necessary */

libname LTC_MS1a "&SASdatapath.\MS1a"; /* for payment simulation step1a */
libname LTC_MS1b "&SASdatapath.\MS1b"; /* for payment simulation step1b */
libname LTC_MS2a "&SASdatapath.\MS2a"; /* for payment simulation step2a */
libname LTC_MS2b "&SASdatapath.\MS2b"; /* for payment simulation step2b */
libname LTC_MS3a "&SASdatapath.\MS3a"; /* for payment simulation step3a */
libname LTC_MS3b "&SASdatapath.\MS3b"; /* for payment simulation step3b */

```

Macro variables also help accomplishing automation and make your program data driven. The value of a macro variable could be extracted from an intermediate data set generated in the process and used to determine the path of execution. This application will be illustrated later in the discussion of TECHNIQUE 3 (P37_XXXX and P3_XXXX).

TECHNIQUE 2: MACRO PROGRAM

Macro programs offer more programming 'flexibility' than macro variables. You can pass information to parts of the macro program via macro parameters. The M1_Macros_XXXX below illustrates a simple example of macro program that read data from excel file into SAS. The excel file name, TAB name, and SAS data set name are passed through macro parameters while the folder path of the excel file is passed through a global macro variable (defined in P0_XXXX).

```

/*****
/* Macro program name: M1_Macros_readexcel.sas */
/* Purpose: read data from Excel file into SAS */
*****/

%macro Readexcel(infilename,Tabname,outfile);
  proc import out=&outfile
    datafile="&rawfilepath.\&infilename"
    dbms=excel replace;
    sheet="&Tabname";
    getnames=yes;
  run;
%mend;

```

```

/*****
/* program name: P13_update_Hosp.sas      */
/* Purpose: update HOSP file */
*****/

%include "&programpath.\P0_assign_MacroVariable.sas"; ❶
%include "&Rootpath.\Program\Macros\M1_Macros_readexcel.sas";

/* step 1: read raw file into SAS */
%readexcel(Provlist_3M_18FR_0705.xlsx,data,LTC_F.HOSP_raw); ❷

/* step 2: Rename variables with standard name and create new variables */
Proc contents data= LTC_F.HOSP_raw;
  Run;

data prov (keep=provider quality prov_tot_rcc  Prov_final_br);
set LTC_F.HOSP_raw;
rename  prov=provider totccr=prov_tot_rcc; ❸
quality=quality_18fr;
  if quality=1 then Prov_final_br=&LTCH_Finalrate_full; ❹
  else quality=0 then Prov_final_br=&LTCH_Finalrate_reduced;
run;

/* step 3:  update final hosp file */
data LTC_F.HOSP_final;
set prov;
run;

```

The program p13_XXXX is one of the programs in step 1 (Figure 1) that updates hospital information received from the client. Highlighted contents are the updates for the current run. They are either input file name or variable names in the input file received from the client.

- ❶: Include the program that define all macro variables and macro program.
- ❷: Execute macro program with specified file and TAB name to read data in excel file into SAS.
- ❸: As needed, rename variables as standard variable name used in sequential programs.
- ❹: Update variable value by referencing macro variable defined in P0_XXXX.

The Macro program M1_Macros_XXXX was extensively used in other sub programs in step 1 to update other information. All the sub-programs in step 1 share the same structure as P13_XXXX. Ultimately, all input data file names and variable names in the data files are updated with standard file name and variable names. These standard file names and variable names will be used programs in the remaining steps.

TECHNIQUE 3: CONDITIAL AND ITERATIVE STATEMENT

SAS macros provide us with the flexibility to use a piece of code multiple times by simply calling the macro. This flexibility can be exploited to reach the next level of sophistication with use of conditional statements and Do loops. Very often, we need to implement the same calculation multiple times with slight changes in the input data or parameters. Sometimes, the time of the iteration is unknown. The iteration stop when a specific condition meets. For example, in step 3 in Figure 1, after initial calculation of RW (relative weight) for each DRG, the monotonicity of DRG RW within the base DRG (higher lever category of DRG should have higher weights) needs to be checked. When Non-Monotonicity DRG is found, the DRG groups need to be modified based on the predefined logic. Then with the new defined DRGs, the whole calculation of relative weight will be repeated. The process of both monotonicity check and weight calculation may need to be conducted a few times until no DRG violates monotonicity within

its base DRG.

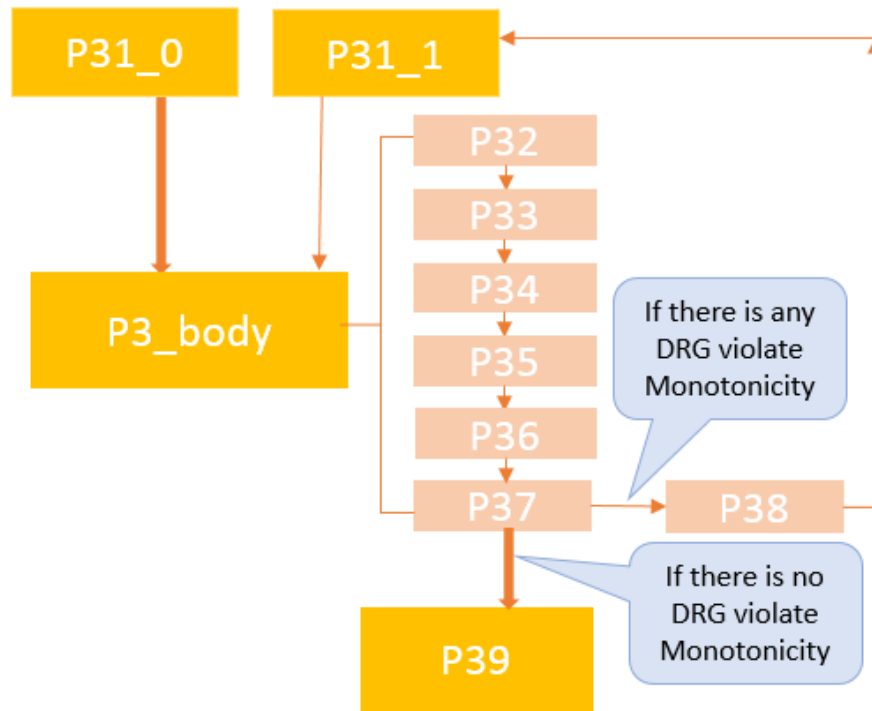


Figure 4 RW Calculation Program Flow Chart

Fig 4 displays the program flow chart in step 3 in Figure 1. P31_0 prepare data for initial run, and P31_1 update data for a new run within the iteration. P32-P35 conduct the weight calculation. P36 implements Monotonicity check. The condition check is nested in P37. If the condition is met, P38 will be executed, followed by P31_1 and a new round of calculations. Otherwise, the iteration stops and flow moves to P39. P39 arranges results from a few results files for outputting.

The program P37_XXXX below illustrates how a condition check is implemented with %IF, %Then statement.

```

/*****
/* program name: P37_DRG_RV_Monoton_findfix */
/* Purpose: Check and identify Non- Monotonicity DRG */
*****/

/*****/
Other program codes
/*****/

Data DRG_needfix;
Set DRG_RW;
If fix>0;
Run;

%let total_fix =0;
proc sql;
select sum(fix) into: total_fix ①
from DRG_needfix;

```

```
quit;

%macro auto_process();
%if &total_fix>0 %then %do; ❷
%include "&programpath.\P38_DRG_RV_mono_newgrouping.sas";
%end;
%mend;

%auto_process();
```

The data set DRG_needfix contains non-Monotonicity DRGs (fix=1) that need to be modified. In ❶, a macro variable total_fix is defined to extract the total number of non-Monotonicity DRGs. The condition check is implemented with a %IF-%then within a macro program Auto_process In ❷, If there is any Non-Monotonicity DRG (total_fix>0), program P38 will be executed to modify the grouping for these DRGs. Then another round of iterations will be started from executing P31_1. Otherwise, nothing will be done when executing macro auto_process.

The iteration is implemented with %DO-%WHILE statement. It could also implemented with %DO-%TO or %DO-%UNTIL. The program p3_XXXX below is the core program that implements the whole process of weight calculation in step 3. It connects all programs displayed in Figure 3. P3_body simply bundled P32 to P37.

- ❶ defined macro variable libname as "LTC_M0" which allow all the output generated in the initial RW calculation saved in this folder. This macro variable libname will be updated within the Do loop in Macro program simu_RW for other iterations so all outputs are saved in corresponding folder.
- ❷ Macro program simu_RW is developed to implement the iterations with %DO %WHILE statement. Under each iteration, a macro variable oldlibname is defined refer folder that hold the output in previous iteration. The macro variable libname is updated thus all outputs are saved in the corresponding folder.
- ❸ the conditional check is conducted at the beginning of the DO LOOP. The same macro variable, total_fix (obtained in P37_XXXX) is used to determine whether to proceed. If total_fix>0, another round of calculation will be implemented with updated information. Otherwise, the DO LOOP is terminated.

```
/* **** */
/* program name: P3_CalculateRV */
/* Purpose: Calculate RV */
/* **** */

/* Step 1: Initial calculation */
%let libname=LTC_M0; ❶
%include "&programpath.\P31_0_preparedata.sas";
%include "&programpath.\P3_body.sas";

/* Step 2: one or more iteration of RV calculation if needed */
%let i=0;

%Macro simu_RW(); ❷

%do %while (&total_fix >0); ❸

%let ii=%eval(&i+1);
%let oldlibname=%str(LTC_M&i);
%let libname=%str(LTC_M&ii);

%include "&programpath.\P31_1_preparedata.sas";
%include "&programpath.\P3_body.sas";
```

```

        %let i=%eval(&i+1);
    %end;
%mend;

%simu_RW();

/* Step 3: Create final RV */
%include "&programpath.\P39_1_DRG_RV_beforeXwalk.sas";
%include "&programpath.\P39_2_DRG_RV_Xwalk.sas";
%include "&programpath.\P39_3_DRG_RV_Final.sas";

```

TECHNIQUE 4: %INCLUDE STATEMENT

In the example programs discussed in TECHNIQUE 3, another macro statement, %include, is extensively used. The %include statement is equivalent as copying and inserting all the code in the file into current place. However, it makes the main program short and easy to read.

%include statement could be used to access lengthy macro programs that stored in a separate file. This application was illustrated in ❶ in program p13_XXXX (TECHNIQUE 2).

With %include statement, you can also bundle a series of programs into one when all these programs need to be run together multiple times. Program P3_body below use %include statements to bundle 6 sequential programs. The program P3_body appeared twice in core program P3_CalculateRV.

```

/*****
/* program name: P3_body.sas */
/* Purpose: link files */
*****/

%include "&programpath.\P32_DRG_trimthreshold.sas";
%include "&programpath.\P33_DRG_RV_Initial.sas";
%include "&programpath.\P34_DRG_RV_hospitaladjust.sas";
%include "&programpath.\P35_DRG_RV_std.sas";
%include "&programpath.\P36_DRG_RV_Monoton_prepare.sas";
%include "&programpath.\P37_DRG_RV_Monoton_findfix.sas";

```

My favorite part of using %include statement is to connect sub-programs into a core program for a major step. Compare with developing a lengthy program with hundreds lines, I prefer to develop short programs for each sub step and then connect them into one core program. When there is need for changing methodology, you only need to modify the relevant program without touching other sub programs.

The core program style can also serves as documentation, make the program easy to read. The RW calculation process illustrated in Figure 3 could be easily understood from program P3_CalculateRV.

TECHNIQUE 5: DDE (DYNAMIC DATA EXCHANGE)

While one could use Proc EXPORT to output SAS data into one TAB in an excel file, outputting results with DDE allow automatically outputting multiple data sets into one TAB with predefined template. Specifically, with DDE, the SAS user can interact with Excel to write data directly into specific Excel worksheet cells. With this technique, Tugluke Abdurazak developed a macro program EXCELOUT (Appendix 1) to accomplish the outputting task.

When a task requires extensive output, especially with specific format, outputting with DDE will become an efficient tool. Once all the results is produced and organized with required format, Program P5_outputing below automatically produces up to 20 TABs with the macro EXCELOUT within one minute.

```

/*****
/* program name: P5_outputing */
/* Purpose: output SAS outputs into Excel template for final report */
*****/

X "COPY &Rootpath.\Output\&CY.\Final_Template.xlsx
&outfilepath.\&Exelname"; ❶

%include "&Rootpath.\Program\Macros\macro_EXCELOUT.sas";

%let tabname=%str(Budget Neut Factor); ❷
%EXCELOUT(LTC_MS1a.output_Brate,&tabname,&outfilepath\& Exelname,1,3); ❸
%EXCELOUT(LTC_MS1a.output_parameter,&tabname,&outfilepath\&Exelname,4,2);
%EXCELOUT(LTC_MS1b.output_parameter,&tabname,&outfilepath\&Exelname,10,2);
%EXCELOUT(LTC_MS2a.output_parameter,&tabname,&outfilepath\&Exelname,17,2);
%EXCELOUT(LTC_MS2b.output_parameter,&tabname,&outfilepath\&Exelname,25,2);

/* output for other TABs */

```

- ❶ Make a copy of the excel template for outputting
- ❷ Specify TAB name
- ❸ Call macro EXCELOUT with SAS data name, TAB name, excel file name/path (specified with global macro variables in P0), and the cell location.

Figure 5 below displays the template and contents in TAB “Budget Neut Factor”.

	A	B	C	D	E
1			Base Rate Full Quality=1	\$	
2			Base Rate Full Quality=0	\$	
3		Simulation A			
4		RW Simulation Name	FY18P34X		
5		Payment Simulation Name	PaySim1		
6		Relative Weight Simulation	FY2017 Final Rule CMS		
7		Baserate		\$	
8					
9		Simulation AA			
10		RW Simulation Name	FY18P34X		
11		Payment Simulation Name	PaySim2		
12		Relative Weight Simulation	FY2017 Final Rule CMS		
13		Target Budget Payment		\$	
14		HCO Threshold		\$	
15					
16		Simulation B			
17		RW Simulation Name	FY18P35C		
18		Payment Simulation Name	PaySim1		
19		Relative Weight Simulation	FY18P35C		
20		RW Normalization Factor			
21		Payment Simulation Target	FY18P34X-PaySim1		
22		Budget Neutrality Factor			
23					
24		Simulation BB			
25		RW Simulation Name	FY18P35C		
26		Payment Simulation Name	PaySim2		
27		Relative Weight Simulation	FY18P35C		
28		RW Normalization Factor			
29		Payment Simulation Target	FY18P34X-PaySim2		
30		HCO Threshold			
31		Budget Neutrality Factor			
32					

Figure 5 Output of TAB “Budget Neut Factor”

CONCLUSION

Utilization of Macro variables, Macro programs, and Macro statements can achieve a lot of goals. This paper illustrates how these techniques were applied to a task repeated yearly to achieve high automation, high accuracy, and quick turnaround. Although the program presented is designed for a specific scenario, the techniques presented can be applied to other similar repeated tasks.

REFERENCES

Carpenter, Art, 2004. *Carpenter's Complete Guide to the SAS® Macro Language 2nd Edition*, Cary, NC: SAS Institute Inc.,2004.

Abdurazak, T. 2002. "Using SAS Macros to Create Automated Excel Reports Containing Tables, Charts and Graphs". *Proceedings of the twenty-seventh Annual SAS Users Group International Conference, paper 126, 2002.*

Appendix I

```
/* **** */
/* program name: macro_EXCELOUT.sas      */
/* Purpose: output SAS data into Excel template */
/* **** */

%MACRO EXCELOUT(SDS,XLSSHT,XLSF, ROW,COL ) ;
PROC CONTENTS DATA=&SDS NOPRINT OUT=CNT;
PROC SORT DATA=CNT ;
BY VARNUM ;
PROC SQL NOPRINT;
SELECT NAME
INTO: VARS SEPARATED BY ' '
FROM CNT ;
SELECT COUNT(DISTINCT NAME)
INTO: COLS SEPARATED BY ' '
FROM CNT ;
SELECT NOBS
INTO: ROWS
FROM CNT
WHERE VARNUM = 1;
QUIT;

OPTIONS NOXWAIT NOXSYNC ;
X "&XLSF" ;
DATA _NULL_ ;
X=SLEEP(2);
RUN ;

FILENAME TEMP DDE "EXCEL|&XLSSHT.!R&ROW.C&COL.:R%TRIM(%EVAL(&ROWS+&ROW-1))C%TRIM(%EVAL(&COLS+&COL))" notab ;

DATA _NULL_ ;
SET &SDS ;
FILE TEMP dlm='09'x;
PUT &VARS ;
RUN ;

/* DDE option is used with FILENAME statement to read data from SAS
and write to Excel ; */
```

```
FILENAME CMDS DDE 'EXCEL|SYSTEM' ;  
DATA _NULL_ ;  
FILE CMDS ;  
PUT '[SAVE()]' ;  
PUT '[QUIT()]' ;  
RUN ;  
%MEND EXCELOUT ;
```

CONTACT INFORMATION

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

Gongmei Yu
3M Health Information Systems
Gyu5@mmm.com