Paper p74-26

Taking Control and Keeping It: Creating and Using Conditionally Executable SAS® Code

Justina M. Flavin, Pfizer Global Research & Development, La Jolla Laboratories, San Diego, CA
Arthur L. Carpenter, California Occidental Consultants, Oceanside, CA

ABSTRACT

Have you ever wanted to selectively execute chunks of program code? Do you want to leave debugging statements in your program for future debugging sessions while preventing their execution without commenting them manually? Would you like your program to automatically start at a specified time or perhaps delay the start for some period of time? Do you want to have error conditions in one step determine which, if any, of the remaining parts of the program are to be executed? These concepts and others will be discussed in the context of controlling how and when all or a portion of a program runs using conditionally executable SAS code.

KEYWORDS

conditional execution, conditional processing, cancel, abort, endsas, errorabend

INTRODUCTION

Usually the topic of conditional execution is the discussion of IF-THEN-ELSE processing within the DATA step. Since these statements are limited to the DATA step, their use is limited when one attempts to control the overall flow of the program. Fortunately there are a number of other types of statements, options, functions, and programming alternatives that are well suited to controlling how and when all or a portion of a program executes.

The various tools available fall into three broad areas. These include the various kinds of actual and effective comments, the use of dates and times in the decision process, and the termination of all or part of the job based on some criteria.

As you learn to use the techniques discussed in the following paper, you will find that they expand very readily to many other types of programming situations. They will allow you to take charge by creating flexible code that is conditionally executable.

I. COMMENTS

In addition to the two standard types of comments that are part of the Base system and the one type of comment in the Macro Language, there are other ways to use the concept of comments to control what code is executed. From a practical standpoint, a comment masks the code so that it will not be compiled and executed. It is also possible to effectively comment code without using any one of these styles of comments.

(i) * comment; STYLE COMMENTS

Comment lines are useful for suspending execution of particular lines of code without deleting the code. To suppress execution of a single line of code, the statement is preceded with an asterisk.

```
data vitals;
  set db.vitals;
  * if patid=1 and visit=1 then wgt=162;
  * if patid=9 and visit=3 then wgt=150;
run;
```

Flexible comments can be implemented by creating a macro variable that takes on the value of either '*' to turn off code in a program or ' ' to turn it on. This technique is especially useful for QC and debugging purposes. In the code below, the macro variable &CHECKIT could be either local within the program or global for a batch run of many programs.

```
%let checkit=*;

data vitals;
  set db.vitals;
  &checkit if patid=1 and visit=1 then wgt=162;
  &checkit if patid=9 and visit=3 then wgt=150;
run;
```

Entire blocks of code can be commented out one line at a time.

```
&checkit  proc print data=vitals;
&checkit    title "Data Fixes";
&checkit    where patid in (1, 9);
&checkit  run;
```

(ii) /* comment */ STYLE COMMENTS

This type of comment is useful for suppressing larger chunks of code. A disadvantage is the inability to embed comments of this type. Under some operating systems this style comment can be misinterpreted if the /* is located in the first two columns.

```
data vitals;
  set db.vitals;
  /* if patid=1 and visit=1 then wgt=162; */
  /* if patid=9 and visit=3 then wgt=150; */
run;
```

OR

```
/*
* Print the vitals data for
* patients 1 and 9;
proc print data=vitals;
  title "Data Fixes";
  where patid in (1, 9);
run;
*/
```

Again a macro variable such as &CHECKIT can be used to conditionally execute the section of code. In this case however, it takes on the value of '/' or ' '.

```
%let checkit=/;

&checkit*
* Start of debug section;
* Print the vitals data for
* patients 1 and 9;
proc print data=vitals;
  title "Data Fixes";
  where patid in (1, 9);
run;
*/
* End of debug section;
```

(iii) %*comment; MACRO COMMENTS

The macro comment statement behaves much like the asterisk style comment when used outside of a macro. Since macros are used to

generate code, the macro comment will not generate code while the others will. This is usually not an issue unless macro functions are being written. All three comment styles can be used within a macro. Which one is selected depends on the objective of the macro.

In the following example of open code, all three types of comments achieve the same results.

```
data vitals;
  set db.vitals;
  * if patid=1 and visit=1 then wgt=162;
  /* if patid=9 and visit=3 then wgt=150;*/
  %* if patid=10 and visit=1 then wgt=148;
run;
```

(iv) UNCALLED MACROS

One of the fastest and easiest ways of preventing execution of a block of code is by using uncalled macros (Grant, 1994). This method is especially useful when suppressing execution of code that contains /* comment */ style comments.

```
%macro dontdoit;
   /* The data set vitals is only printed
      when the macro is called.*/
   proc print data=vitals;
     title "Data Fixes";
     where patid in (1, 9);
   run;
%mend dontdoit;
```

Using a macro in this way can slightly increase the use of system resources because the macro must be compiled regardless of whether or not it is called.

II. DATE AND TIME CONTROL

There are a number of types of decisions that may need to be made based on various date, time, and datetime values. These can include decisions based on the data as well as decisions based on the actual time of execution.

(i) DATE BRANCHING

A date cutoff can be used to conditionally execute code. In a clinical trials setting, analyses of an ongoing study may require temporary hard-code fixes to the data. These may be corrections for outstanding queries which are not yet resolved and entered into the data base prior to the snapshot date for an interim analysis.

```
data vitals;
  set db.vitals;
  * For interim analysis, hard code values;
  * of known data corrections for;
  * outstanding queries;
  if today () <= "30JUN1999"d then do;
     if patid=1 and visit=1 then wgt=162;
     if patid=9 and visit=3 then wgt=150;
  end;
run;
```

After the snapshot date (30 June 1999), the values for WGT stored in DB.VITALS will be used. Since the study is ongoing, these queries will eventually be resolved and corrected in the data base and it would be inappropriate to continue to reassign these values programmatically. By conditionally coding in this manner, a documented trail of the hard-coded fixes that were applied for the interim analysis exists. This in turn may eliminate the need for a separate interim analysis program.

(ii) SLEEP FUNCTION

The sleep function suspends execution of a DATA step for a specified number of seconds. The maximum sleep time is slightly over 46 hours. This example suspends execution for 9 hours and 27 minutes.

```
data _null_;
  slept=sleep((60*60*9)+(60*27));
run;
```

The DATA _NULL_ step can be eliminated by using macro statements including the %SYSFUNC macro function. The data step becomes:

```
%let slept=
sysfunc(sleep(%eval((60*60*9)+(60*27))));
```

When the system is to wake up at a specific time, the number of sleep seconds needs to be calculated. The following example will wake up at 28 minutes and 35 seconds after 5 P.M. on 30th of October 1998.

```
data _null_;
  now=datetime();
  startat='30OCT1998:17:28:35'dt;
  sleep=startat-now;
  slept=sleep(sleep);
run;
```

III. STEP AND JOB TERMINATION

It is common to want to control the flow of the individual steps of the program. This may include the termination of a step or even the job itself. There are several exit strategies depending on the objective.

(i) ENDSAS

The ENDSAS statement immediately terminates the SAS job or session. An interactive program that writes output to the OUTPUT window should not use this statement because the contents of the OUTPUT window (as well as the LOG & Program Editor windows) will be lost. However when output is directed to files (as in a batch program or when using PROC PRINTTO) or when just data sets are being created, the ENDSAS statement can be used to control job or session termination.

The entire job is terminated in the following data step if the variable X is ever less than or equal to zero.

```
data new;
  set old;
  lnx=log(x);
  if _error_ then endsas;
run;
```

The ENDSAS statement can also be used between steps when execution of only a portion of a program is desired.

```
…some SAS statements…
run;
endsas;
…unexecuted SAS statements…
```

(ii) STOP

The STOP statement is most often used to prevent infinite loops in DATA steps that contain a SET statement with a POINT= option. This statement can also be used to terminate any DATA step.

In the following example, the data set NEW will contain all observations up to but not including the observation containing the error (X <= 0). STOP does not set the system error conditions and the job continues executing with the next step.

```
data new;
  set old;
  lnx=log(x);
  if _error_ then stop;
run;
```

(iii) ABORT

The ABORT statement is usually used when error conditions are identified. Like the STOP statement, ABORT also stops the execution of the current data step. The behavior of this statement depends to some extent on the operating system and whether the job is being run under batch or interactive modes.

ABORT writes an error message to the log and sets the system error conditions. In batch mode, ABORT sets OBS=0 and continues limited processing. In interactive mode, ABORT acts like STOP. The ABORT statement can be used with two options, ABEND and RETURN. With the ABEND option, ABORT writes an error message to the log and returns control to the host system as an abnormal termination. The RETURN option is the same as ABEND, but exits from SAS as a normal termination. For operating systems that support it, a return code can also be specified with the ABEND and RETURN options.

In the following DATA step, a bad value of X terminates the job with an error condition.

```
data new;
  set old;
  if x le 0 then abort abend 123;
  lnx=log(x);
run;
```

(iv) RUN CANCEL

The CANCEL option on the RUN statement prevents the current step from being executed. This is a quick way to prevent code from executing as opposed to using comments.

In the following example, the PROC PRINT is only executed when the macro variable &CANCEL is set to a null value.

```
* set CANCEL to null to
* print data fixes;
%let cancel=cancel;

proc print data=vitals;
  title "Data Fixes";
  where patid in (1, 9);
run &cancel;
```

Although the PROC PRINT is not executed when the CANCEL option is used in the RUN statement, the TITLE statement will be executed and the title will be reset since the TITLE statement is a global statement.

(v) ERRORABEND

Like the ABORT and ENDSAS statements, the system option ERRORABEND terminates the SAS session or job. Since this is an option, no IF-THEN-ELSE processing is required. As soon as the first error is encountered the job terminates. This option does not terminate for data errors, but is primarily directed to syntax errors and other errors that would otherwise cause the system to go into syntax check mode.

When ERRORABEND is used in an interactive session, termination may also eliminate the LOG. Thus any clues as to the reason for the termination will also be eliminated.

(vi) MACRO %IF-%THEN-%ELSE

Within a macro, complete steps or even groups of steps can be conditionally executed based on values stored in macro variables. %IF-%THEN-%ELSE processing in the macro language is similar to conditional processing within the DATA step. The notable exception, of course, is that the macro language is not limited to the DATA step.

In this example, the macro %OBSCNT (Carpenter, 1998) returns the number of observations in the data set of interest, and the appropriate data display depends on this number. Since macro %IF-%THEN-%ELSE processing cannot be used in open code, it must be used within a macro.

```
%macro showit(dsn);
   %if %obscnt(&dsn) >50 %then %do;
      proc summary data=&dsn;
      …sas statements not shown…
   %end;
   %else %do;
      proc print data=&dsn;
      …sas statements not shown…
   %end;
%mend showit;
```

(vii) &SYSERR

After each DATA step and most PROC steps execute, the automatic macro variable &SYSERR is loaded with the error status of that step. Steps which complete successfully will result in &SYSERR having a value of zero. Other values of &SYSERR may depend on the step and the type of error within the step.

In the following example, the copy will be unsuccessful if the user cannot get read access to the incoming data sets. Since this is critical to the process, the job is terminated with an ABORT.

```
%macro chkcopy;
   * Copy the current version of
   * the COMBINE files
   * to COMBTEMP;
   proc datasets memtype=data;
     copy in=combine out=combtemp;
   quit;
   %if &syserr ge 5 %then %do;
      data _null_;
        put '*** combine copy failure ***';
        put 'One of the data sets';
        put 'may be in use.';
        abort abend;
      run;
   %end;
%mend chkcopy;
```

CONCLUSION

There are a number of ways to control how and when your SAS program executes. You can control the process both within and across steps. The major groupings of the topics of interest include using comments, execution based on date and time, and step and job termination.

Several levels of control are available. These include DATA step specific statements, global statements, system options, automatic macro variables, and macro statements

Comments can be used both within and across steps. Macros that are never called can act as comments. Date and time values can be used for conditional decision making within a DATA step as well as to determine when a job is to start or stop. There are a variety of statements and options that can be used to conditionally terminate a specific step or even the whole job. These determinations can be based on the data or on error conditions found within the program. The power and flexibility of the SAS System allow you, the

programmer, to take control of the flow of the entire program and conditional execution will no longer imply DATA step IF-THEN-ELSE processing.

REFERENCES

Carpenter, Arthur L., *Carpenter's Complete Guide to the SAS Macro Language*, Cary, NC: SAS Institute Inc., 1998, 242 pp.

Grant, Paul, 1994, "The 'SKIP' Statement", *Proceedings of the Second Annual Conference of the Western Users of SAS Software*, Cary, NC: SAS Institute Inc., pp. 87-88.

TRADEMARK INFORMATION

CONTACT INFORMATION

Justina M. Flavin
Pfizer Global Research & Development, La Jolla Laboratories
11085 Torreyana Road
San Diego, CA 92121
(858) 622-7376
justina.flavin@agouron.com

Justina M. Flavin is a Senior Clinical Programmer/Analyst at Pfizer Global Research & Development, La Jolla Laboratories in San Diego, California and served as Conference Chair of Pharma*SUG* '99. She has been employed as a SAS programmer in the pharmaceutical industry for nine years. Other SAS experience includes developing SPC charts and graphs for manufacturing processes in the aerospace industry, and performing data analyses for research projects in the medical, psychological, and political fields. She has a B.A. in Applied Mathematics from the University of California, San Diego.

Art Carpenter
California Occidental Consultants
PO Box 6199
Oceanside, CA 92058-6199
(760) 945-0613
art@caloxy.com
www.caloxy.com

Art Carpenter's publications list includes two chapters in *Reporting from the Field*, three books and over three dozen papers and posters presented at SUGI, WUSS, and PharmaSUG. Art has been using SAS since 1976 and has served in leadership roles in various local, regional, and national user groups.

Art is a SAS Certified Professional™. Through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.