Paper 76484-2011

Creating Metadata-Driven Program Logic with the SAS Call Execute Routine - An Example with CTC Grading of Laboratory Results

Robert W. Graebner, Quintiles, Overland Park, KS, USA

Abstract

Certain programming tasks require the application of complex, data-dependent rules in the derivation of classification variables. One such example is assigning a CTC grade to laboratory results. Determination of the CTC grade depends on the laboratory parameter and its value in relation to either fixed values or multiples of the upper and lower limits of the normal range. This paper presents a solution in which a DATA step is used to access CTC grading logic metadata and the CALL EXECUTE routine is used to place the associated SAS® statements in the SAS command stack for execution after the DATA step has finished. This process creates a virtual DATA step that will assign the CTC grades to each observation in the laboratory results data set.

Introduction

Applications that utilize complex logic can require large, nested blocks of conditional statements. These statements may be based on the values of several variables, and the inclusion of some variables in the process may be conditional. CTC grading of lab results can require 9 to 11 logic statements per lab parameter. This typically results in large volumes of tedious source code that can be difficult to debug and maintain. If you add the possibility that the criteria for the decision rules may change over time, you quickly find yourself facing a maintenance nightmare if you adopt traditional techniques.

One approach to managing such a process is to build a “Logic Processing” procedure, that bases the logic rules that are applied to the lab data on another set of data that define the rules. The data that define the logic process to be performed can be referred to as metadata. Creating such a data set can be straightforward; however writing a program that can access this data and then form the necessary conditional statements to apply the rules can be difficult. Fortunately, the SAS CALL EXECUTE routine can greatly simplify the task.

Before getting into the details of how the CALL EXECUTE routine can be used for this task, let us take a look at a simple example to get a feel for how the basic process works. A common first task used when learning a new programming language is to print the text string “Hello World!”. Using this task as our starting point, we can create a short program that uses CALL EXECUTE in a simple, if indirect way of printing “Hello World!” in the SAS log. The program consists of a DATA step with three CALL EXECUTE commands. Each of these commands contains a single argument that is a text string containing one of the statements in a DATA step used to actually print “Hello World!” in the SAS log. The source code for the program is shown below.

```
DATA TEST;
   CALL EXECUTE("DATA _NULL_;");
   CALL EXECUTE("PUT 'HELLO WORLD!';");
   CALL EXECUTE("RUN;");
 RUN;
```

When we run the program, we get the following SAS log:

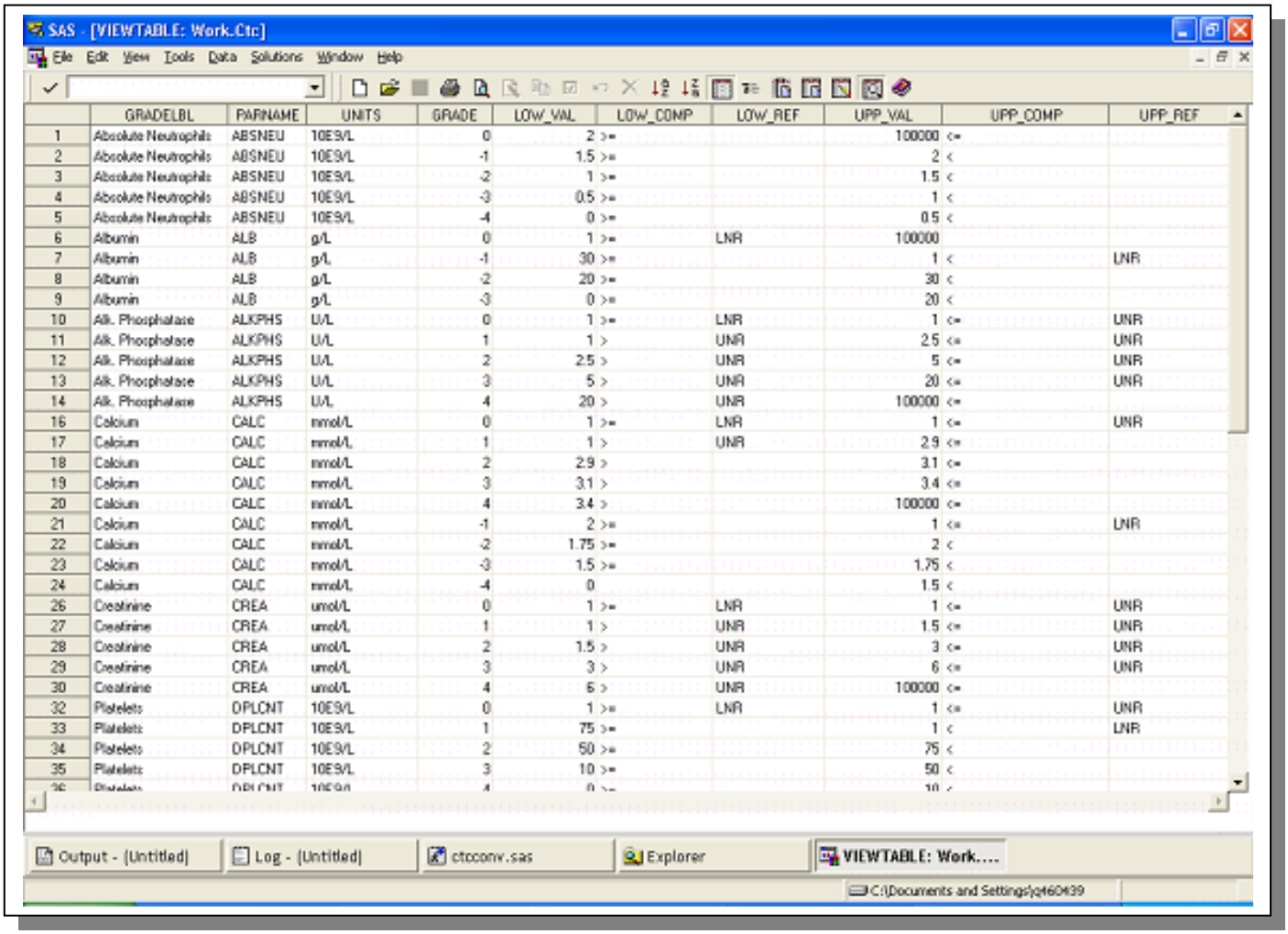
```
1     data test;
2       call execute("Data _null_;");
3       call execute("put 'Hello World!';");
4       call execute("run;");
5     run;

NOTE: The data set WORK.TEST has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.77 seconds
      cpu time            0.11 seconds


NOTE: CALL EXECUTE generated line.
1    + Data _null_;
2    + put 'Hello World!';
3    + run;

Hello World!
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

When the DATA step runs, each of the CALL EXECUTE statements sends the enclosed text string to the SAS macro processor and the resulting string is placed in a command stack. In this simple example, each argument consists entirely of verbatim text. The real power of this approach is achieved when fixed text is combined with the contents of DATA step or macro variables to form executable SAS statements. When the DATA step has completed, SAS begins processing the commands placed in the stack prior to processing any statements that follow the DATA step. The SAS log provides a note regarding the generated lines and below that we see the result of the PUT statement: “Hello World!”. This is a trivial example, but the basic process can be used to generate SAS code based on metadata and then run it immediately. In this paper, I use the process of CTC grading to illustrate a more complex example of the use of metadata in logic processing. Therefore, before proceeding to a more detailed look at the CALL EXECUTE routine, I will give a brief overview of the CTC grading process.

CTC Grading – An Overview

The National Cancer Institute Common Toxicity Criteria (CTC) was developed as a method of grading the severity of adverse events. CTC grading uses an ordinal scale ranging from 0 to 5, representing none or normal, mild, moderate, severe, life threatening or disabling, and fatal. While these general terms are associated with each grade, within individual categories, there are specific clinical criteria associated with each grade. The example in this paper is based on laboratory results within the Blood / Bone Marrow and Metabolic / Laboratory categories.

CTC grading of laboratory results indicates how far above or below normal a value lies, expressed in terms of the potential impact of such a value on the subject’s health and well being. For the example in this paper, CTC grades can range from –4 to 4 for each lab parameter result. A grade of 0 indicates that the result is within the normal range. Positive grades represent values above the normal range (hyper) and negative grades represent values below normal (hypo). In the formal definition of the CTC grades, the hypo- and hyper- conditions would be specified separately, with a range of 0 to 4 for each. Combining both conditions simplifies the program logic associated with assigning CTC grades. Each grade level has associated with it an upper and/or lower limit. These limits can be specified as either an absolute value or as a multiplier to be applied to the upper or lower normal range limit. The logical operator used to compare the laboratory result to each limit is also specified. An example of a SAS data set containing CTC grading metadata is given in Figure 1.



	GRADELBL	PARNAME	UNITS	GRADE	LOW_VAL	LOW_COMP	LOW_REF	UPP_VAL	UPP_COMP	UPP_REF
1	Absolute Neutrophils	ABSNEU	10E9/L	0	2	>=		100000	<=	
2	Absolute Neutrophils	ABSNEU	10E9/L	-1	1.5	>=		2	<	
3	Absolute Neutrophils	ABSNEU	10E9/L	-2	1	>=		1.5	<	
4	Absolute Neutrophils	ABSNEU	10E9/L	-3	0.5	>=		1	<	
5	Absolute Neutrophils	ABSNEU	10E9/L	-4	0	>=		0.5	<	
6	Albumin	ALB	g/L	0	1	>=	LNR	100000		
7	Albumin	ALB	g/L	-1	30	>=		1	<	LNR
8	Albumin	ALB	g/L	-2	20	>=		30	<	
9	Albumin	ALB	g/L	-3	0	>=		20	<	
10	Alk. Phosphatase	ALKPHS	U/L	0	1	>=	LNR	1	<=	LNR
11	Alk. Phosphatase	ALKPHS	U/L	1	1	>	LNR	2.5	<=	LNR
12	Alk. Phosphatase	ALKPHS	U/L	2	2.5	>	LNR	5	<=	LNR
13	Alk. Phosphatase	ALKPHS	U/L	3	5	>	LNR	20	<=	LNR
14	Alk. Phosphatase	ALKPHS	U/L	4	20	>	LNR	100000	<=	
16	Calcium	CALC	mmol/L	0	1	>=	LNR	1	<=	LNR
17	Calcium	CALC	mmol/L	1	1	>	LNR	2.9	<=	
18	Calcium	CALC	mmol/L	2	2.9	>		3.1	<=	
19	Calcium	CALC	mmol/L	3	3.1	>		3.4	<=	
20	Calcium	CALC	mmol/L	4	3.4	>		100000	<=	
21	Calcium	CALC	mmol/L	-1	2	>=		1	<=	LNR
22	Calcium	CALC	mmol/L	-2	1.75	>=		2	<	
23	Calcium	CALC	mmol/L	-3	1.5	>=		1.75	<	
24	Calcium	CALC	mmol/L	-4	0			1.5	<	
26	Creatinine	CREA	umol/L	0	1	>=	LNR	1	<=	LNR
27	Creatinine	CREA	umol/L	1	1	>	LNR	1.5	<=	LNR
28	Creatinine	CREA	umol/L	2	1.5	>	LNR	3	<=	LNR
29	Creatinine	CREA	umol/L	3	3	>	LNR	6	<=	LNR
30	Creatinine	CREA	umol/L	4	6	>	LNR	100000	<=	
32	Platelets	DPLCNT	10E9/L	0	1	>=	LNR	1	<=	LNR
33	Platelets	DPLCNT	10E9/L	1	75	>=		1	<	LNR
34	Platelets	DPLCNT	10E9/L	2	50	>=		75	<	
35	Platelets	DPLCNT	10E9/L	3	10	>=		50	<	

Figure 1. SAS data set containing CTC grading metadata

Assessing Conventional Approaches

Simpler processing tasks can be accomplished with more traditional approaches. For example, applying simple normal ranges to laboratory results can be accomplished by merging the two data sets and performing simple comparisons. More complex examples include applying normal ranges that are dependent on the subject's sex, age or weight or on the date of analysis. In this case, some of the identifier variables must match exactly, others (e.g. age or date) must fall within the range specified in the normal range file. This can be accomplished with a DATA step merge using the exact matching variables in the by statement, and the using a WHERE or IF statement to keep only those observations that meet the criteria. Another approach is to perform a fuzzy merge using a PROC SQL SELECT statement with appropriate GROUP BY and HAVING clauses.

With the example of determining CTC grading of laboratory results, the problem becomes more complex. The grading of each laboratory result is dependent on multiple rows in the CTC logic metadata. Not only are there multiple comparison ranges for each parameter, but some of the upper and lower limits are absolute values while others specify a multiple of the upper or lower normal range limit. To further complicate the process, the specific comparison operator (<, <=, >, >=) can vary as well. The grading process could be accomplished using a SELECT / WHEN block that contained a WHEN statement for each parameter and grade combination. However this approach would be difficult to maintain, requiring the program to be modified for any changes to the CTC grading criteria or for the addition of new laboratory parameters. Another possibility, thanks to the power and flexibility of the SAS CALL EXECUTE routine, is to read in logic metadata which specify the necessary comparison parameters for each CTC grade and build a DATA step that will apply them to the laboratory results.

The SAS CALL EXECUTE Routine

Before delving into the specifics of generating a DATA step, it is important to have a basic understanding of the SAS CALL EXECUTE routine. The SAS CALL EXECUTE routine has a single argument that may contain literal text strings, DATA step variable names or SAS macro statements. When the CALL EXECUTE routine is executed, the processing of the current DATA step is temporarily stopped and the text string provided as the argument is sent to the macro processor to be resolved. Any SAS macro variable references are resolved and then any macro statements, functions or calls are processed. The text string produced by the macro processor is then placed in the command stack. When the current DATA step has completed, SAS begins processing the lines of text in the command stack and attempts to interpret them as valid SAS program statements. When all of the lines in the command stack have been processed, SAS begins executing any statements that follow the DATA step.

When macro statements are included in the argument to CALL EXECUTE, it is crucial to consider the order of events and the flow of control between the SAS macro processor and the DATA step. It is also important to remember that the macro processor will attempt to resolve text contained within double quotes, while text in single quotes will be treated as literal text and will be simply passed through the macro processor. One example where double quotes are necessary is when you include a macro variable reference in a text string and you want the macro variable to be resolved as part of creating the SAS statement to be sent to the command stack. Single quotes are necessary if you want the macro variable reference to be resolved when the SAS statement you create is actually executed.

Generating a DATA Step from Metadata

A key challenge associated with basing DATA step programming logic on metadata stored in a data set is timing. We need to access variables in the logic metadata to build the necessary program statements and then have those statements available to process laboratory data from another data set. This process requires the use of two DATA steps; one to loop through the logic metadata and generate valid SAS statements, and a second to process the laboratory results data set using the logic statements generated by the first. This may seem like a daunting task, however it can be accomplished using the SAS CALL EXECUTE routine. An overview of the process is illustrated below.

By sending a series of commands that form a DATA step to the command stack, we can use one DATA step to create another. Since the second DATA step is created by the first, they in essence form an intertwined DATA step pair, consisting of a real, constructor DATA step, and a virtual, logic processing DATA step. In the CTC grading example, a virtual SAS DATA step with a SELECT / WHEN block to assign a CTC grade to each observation of laboratory data is constructed line by line and placed in the SAS command stack. The second DATA step is completely dependent on the first for its properties, but completely independent as far as processing goes. This basic process is illustrated in Figure 2.

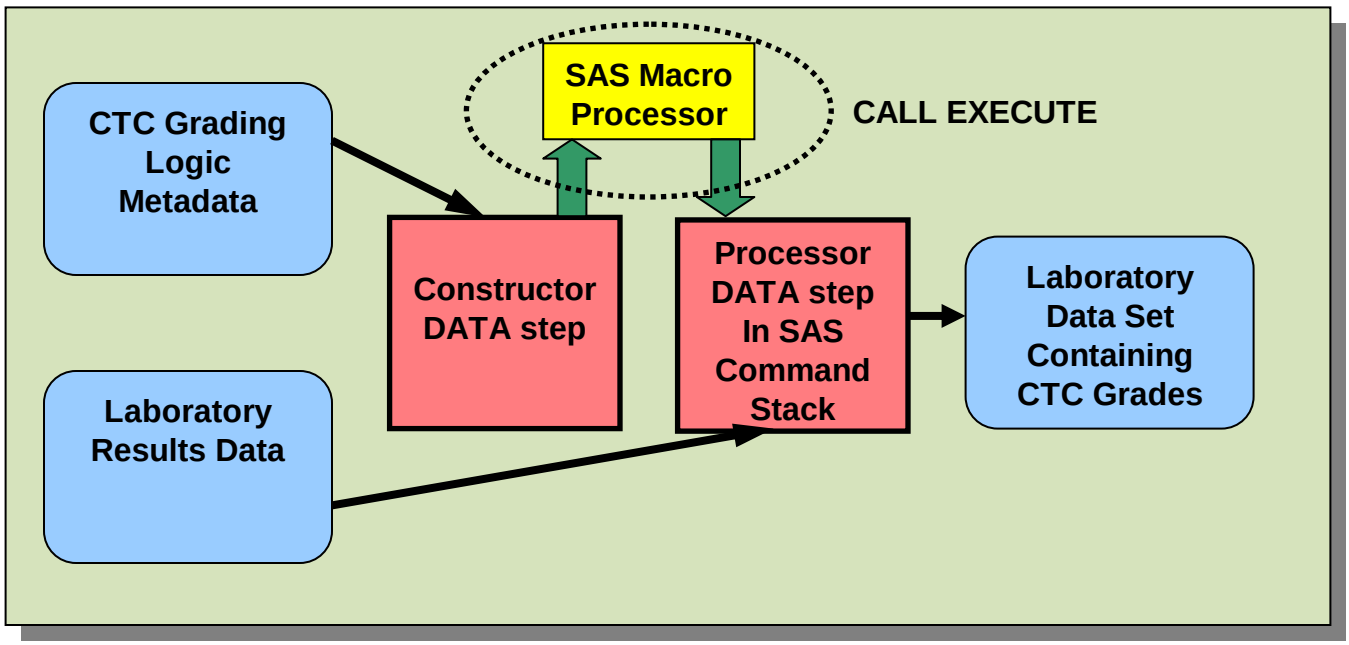


Figure 2. The process of creating a CTC grading DATA step from metadata

Anatomy of a Constructor DATA Step

The following example illustrates the basic structure of a constructor DATA step. First, the CTC grading logic metadata data set and end-of-file flag are specified in a SET statement. Then the text variable that will be use to construct lines of SAS source code is defined. In this example, a DATA NULL step is used; however a temporary or permanent SAS data set name can be specified in order to save the source code lines you generate. This can be particularly helpful during debugging.

```
DATA _NULL_;
  SET CTC_GRAD END = EOF;
  LENGTH CODELINE $200;
```

Next, since the processor DATA step statements in the command stack will be executed immediately after the constructor DATA step is finished, it is important to test the values in each record of metadata to ensure that they will create a valid line of SAS code. In the example below, only observations with the minimum sufficient subset of metadata parameters are retained. A subsetting IF statement will allow the current iteration of the DATA step to continue if the parameter name, grade, upper value and lower value are present. Otherwise a warning is printed in the SAS log and the DATA step skips to the next observation in the metadata data set.

```
IF PARNAME ^='' AND GRADE ^= . AND (LOW_VAL ^= . OR UPP_VAL ^= .);
ELSE PUT '**** WARNING: INCOMPLETE METADATA OBSERVATION ****';
```

Prior to generating SAS code based on the first observation of metadata, the opening statements of the processor DATA step and the beginning of the SELECT block are placed in the command stack by using CALL EXECUTE with literal text arguments.

```
IF _N_ = 1 THEN
  DO;
    CALL EXECUTE('DATA LABCTC;');
    CALL EXECUTE('  SET LAB;');
    CALL EXECUTE('  SELECT;');
  END;
```

The following SAS code is used to construct the individual WHEN statements within the SELECT / WHEN block. Each valid observation in the CTC process logic data set will result in a WHEN statement that will perform the logic test for a single parameter and grade combination.

In the block of SAS code below, a character variable which contains source code for the lower limit of the CTC grade (LOW_LIM) is created, based on the lower limits of the current parameter and grade (LOW_VAL) and the value of the limit reference variable (LOW_REF). If the reference variable is missing, then the grade limit variable's value is used as an absolute value. If the reference variable is equal to 'LNR' or 'UNR', then the grade limit value is multiplied by the lower (LABLLN) or upper (LABULN) normal range value to determine the limits to be used for the grade. A similar block of code is use to create the variable UPP_LIM for the upper limit, based on UPP_REF and UPP_VAL.

```
IF LOW_VAL ^= . THEN
  DO;
    SELECT( UPCASE( LOW_REF ));
      WHEN('LNR') LOW_LIM = PUT(LOW_VAL, BEST12.) || '* LABLLN';
      WHEN('UNR') LOW_LIM = PUT(LOW_VAL, BEST12.) || '* LABULN';
      WHEN('')    LOW_LIM = PUT(LOW_VAL, BEST12.);
      OTHERWISE LOW_LIM = '';
    END;
  END;
ELSE LOW_LIM = '';
```

The SAS code in the second block below is used to construct a WHEN statement that handles the logic for the laboratory parameter and CTC grade specified in the current observation of the CTC grading metadata. The WHEN statement is created by combining literal text strings and the contents of metadata variables into a text variable called CODELINE. When the statement is complete, the CALL EXECUTE routine is used to place the contents of CODELINE in the command stack.

```
IF UPP_LIM ^= '' OR LOW_LIM ^= '' THEN
  DO;

    CODELINE = '    WHEN(PARNAME="'||STRIP(PARNAME)||'" AND UNITS="'||
               STRIP(UNITS)||'" AND ';

    IF LOW_VAL ^= . AND LOW_COMP ^= '' THEN
       CODELINE = STRIP(CODELINE) || ' LABRES ' || STRIP(LOW_COMP) ||
STRIP(LOW_LIM));

    IF LOW_VAL ^= . AND LOW_COMP ^= '' AND UPP_VAL ^= . AND UPP_COMP ^= '' THEN
       CODELINE = STRIP(CODELINE) ||' AND ';

    IF UPP_VAL ^= . AND UPP_COMP ^= '' THEN
       CODELINE = STRIP(CODELINE) || ' LABRES ' || STRIP(UPP_COMP) ||
STRIP(UPP_LIM);

    IF LOW_REF = 'LNR' OR UPP_REF = 'LNR' THEN
       CODELINE = STRIP(CODELINE) || ' AND LABLLN ^= . ';

    IF LOW_REF = 'UNR' OR UPP_REF = 'UNR' THEN
       CODELINE = STRIP(CODELINE) || ' AND LABULN ^= . ';

    CODELINE = STRIP(CODELINE) || ' AND LABRES ^= . ) CTC_1C = ' ||
               STRIP( PUT( GRADE, BEST12. )) || ';';

    CALL EXECUTE( CODELINE );
```

When the last line of SAS code that is dependent on metadata has been generated, the final lines of the processor DATA step are added to the command stack.

```
  IF EOF THEN
    DO;
      CALL EXECUTE('    OTHERWISE;');
      CALL EXECUTE('  END;');
      CALL EXECUTE('RUN;');
    END;
RUN;
```

Anatomy of a Processor DATA Step

The processor DATA step is relatively simple, but with many parts. The DATA step contains a single SELECT / WHEN block with a WHEN statement for each CTC grade that can be associated with each laboratory parameter. The good news is that you did not have to type it all in, or find the correct location to make changes if the criteria change. A small subsection of a processor DATA step is shown below.

```
DATA LRS5CTC;
  SET LRS5;
  SELECT;
    WHEN(PARNAME="CALC" AND LABRES >2.4 AND LABRES <=2.6) CTCGRD = 0;
    WHEN(PARNAME="CALC" AND LABRES >2.6 AND LABRES <=2.9) CTCGRD = 1;
    WHEN(PARNAME="CALC" AND LABRES >2.9 AND LABRES <=3.1) CTCGRD = 2;
    WHEN(PARNAME="CALC" AND LABRES >3.1 AND LABRES <=3.4) CTCGRD = 3;
    WHEN(PARNAME="CALC" AND LABRES >3.4) CTCGRD = 4;
    …
    …
    …
    OTHERWISE;
  END;
RUN;
```

Conclusion

The example in this paper illustrates how a complex set of logic rules can be defined in a data set and then applied in a relatively simple program. This approach can significantly reduce programming development and maintenance costs, and it can facilitate the maintenance of logic rule sets by nonprogrammers. Once the basics of this method are understood, programming complex logic rules can be a simple task. If SAS macro statements or variables are used in the argument for CALL EXECUTE, consideration of process timing issues is critical. One potential change to increase the efficiency of the processor DATA step would be to generate nested SELECT / WHEN blocks, using an outer set of WHEN statements to test for laboratory parameter, and inner SELECT / WHEN blocks to test for value ranges. This could be accomplished by specifying the laboratory parameter as a BY-variable in the constructor DATA step and then using logic statements based on FIRST.PARNAME and LAST.PARNAME.

Contact Information

Robert Graebner
16107 Russell Rd.
Stilwell, KS 66085

Email: bobgraebner@kc.rr.com