Dynamische Programmcode-Erzeugung mit CALL EXECUTE Möglichkeiten, Grenzen, Alternativen

Martin Hensel
Roche Products Pty Ltd
Biometrics
Dee Why, Australien
Martin.Hensel@Roche.com

Zusammenfassung

Die Call-Routine EXECUTE bildet eine Schnittstelle zwischen Data Step und Makroprozessor. Mit ihrer Hilfe kann sehr flexibel aus dem Data Step heraus Programmcode erzeugt werden. Nützlich ist vor allem die Möglichkeit, Werte von Variablen als Parameter an ein Makro zu übergeben und auf diese Weise Programmverzweigungen von der aktuellen Datenlage steuern zu lassen.

Dabei ist jedoch zu beachten, dass CALL EXECUTE und CALL SYMPUT nicht ohne Weiteres aufeinander treffen dürfen. Falls es gilt, ein Makro aufzurufen, das Referenzen auf Makrovariablen enthält, die innerhalb dieses Makros mittels CALL SYMPUT erzeugt werden, kann CALL EXECUTE nicht verwendet werden. Als alternative Programmiertechniken für diesen Fall eignen sich u. a. Makroschleifen und Skriptfiles.

Keywords: CALL EXECUTE, CALL SYMPUT, Data Step, globale Symboltabelle, lokale Symboltabelle, Laufzeitproblem, Makroprozessor, Makrovariable, PROC SQL, Programmverzweigung, Schleifenprogrammierung, Skriptfile, SYMDEL, SYSSCP, SYSSCPL.

1 Einleitung

CALL EXECUTE bietet die Möglichkeit, aus dem Data Step heraus Programmcode zu erzeugen. Genauer wird mit CALL EXECUTE Text generiert, der vom Word Scanner interpretiert wird. Dabei ist zu beachten, dass auf diese Weise generierte Elemente der Makrosprache sofort ausgeführt werden – also noch während der aktuelle Data Step läuft. Dagegen wird Programmcode, der nicht in den Zuständigkeitsbereich des Makroprozessors fällt, erst im Anschluss an den laufenden Data Step ausgeführt. Durch dieses Laufzeitverhalten unterliegt CALL EXECUTE gewissen Beschränkungen, die hier näher beschrieben werden sollen.

Ich persönlich habe CALL EXECUTE kennen und schätzen gelernt, als es darum ging, Speicherplatzprobleme bei der Ausgabe einer sehr großen SAS-Tabelle mit BY-Gruppen in PROC REPORT zu umgehen. Ohne viel Aufhebens ließ sich das bestehende Programm so modifizieren, dass die Hardware-Anforderungen des Programms den Gegebenheiten angepasst werden konnten. Lange Zeit war ich ein Fan von CALL EXECUTE. Zwar kamen gelegentlich im Gespräch mit Kollegen diffuse ungute Gefühle gegenüber CALL EXECUTE zur Sprache. Aber konkrete Anhaltspunkte für eine bestehende Problematik sind mir lange nicht begegnet. Erst als ein SAS-Programm scheinbar unerklärbare falsche Werte ausgab, kam ich den genauen Hintergründen auf die Spur.

Für die Beispiele wird die Tabelle SASHELP.CLASS verwendet:

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

2 Möglichkeiten

Mit CALL EXECUTE kann man aus einem Data Step heraus SAS-Code erzeugen. Sinnvoll ist das natürlich nur, wenn der Data Step dabei ein Rolle spielt bzw. vielmehr, wenn Werte oder Attribute der Variablen, die die bearbeitete Tabelle enthält, eine Rolle spielen.

Im folgenden Beispiel werden Werte einer Variablen als Parameter an ein Makro übergeben. Darüber hinaus wird das Makro nur dann aufgerufen, wenn diese Variable einen bestimmten Wert annimmt.

Den Sachverhalt dieses Beispiels würde man in der Praxis sicher mit einfacheren Methoden angehen. Hier soll lediglich das Prinzip von CALL EXECUTE verdeutlicht werden.

```
%macro agegroup ( class = ) ;
  title "Altersklasse der &class.-jährigen" ;
  proc print
    data = sashelp.class ( where = ( age = &class. ) ) ;
  run ;
%mend  agegroup ;

proc sort
  data = sashelp.class
  out  = class ;
    by age ;
run ;

data _null_ ;
  set class ;
    by age ;
  if first.age and age = 15 then
    do ;
      call execute ( '%agegroup(class=' || put(age,5.) || ');' ) ;
    end ;
run ;
```

Das Ergebnis ist folgendes Output:

```
             Altersklasse der 15-jährigen

              Obs    Name       Sex    Age    Height    Weight

                8    Janet       F      15     62.5      112.5
               14    Mary        F      15     66.5      112.0
               17    Ronald      M      15     67.0      133.0
               19    William     M      15     66.5      112.0
```

3 Grenzen

Um das Beispiel etwas realistischer zu machen, bestimmen wir zunächst für die zu druckende Altersgruppe den Median des Gewichts und berechnen für jedes Kind die Abweichung davon. Weiter drucken wir den errechneten Median im Titel mit aus.

```
%macro agegroup ( class = ) ;

  %local median ;

  proc univariate noprint
    data = sashelp.class ( where = ( age = &class. ) ) ;
```

```
    var weight ;
    output out = stats
      median = median ;
  run ;

  data _null_ ;
    set stats ;
    call symput ( 'median' , put ( median , 8.2 ) ) ;
  run ;

  data _temp ;
    set sashelp.class ( where = ( age = &class. ) ) ;
    DeltaW = weight - &median. ;
  run ;

  title "Altersklasse der &class.-jährigen – "
        "Median Gewicht ist %trim ( %left ( &median. ) )" ;
  proc print
    data = _temp ;
  run ;

%mend  agegroup ;
```

Direkt aufgerufen, "`%agegroup ( class = 11 ) ;` ", liefert dieses Programm folgendes einwandfreie Ergebnis:

```
      Altersklasse der 11-jährigen - Median Gewicht ist 67.75

      Obs     Name     Sex    Age    Height    Weight    DeltaW

       1     Joyce      F      11     51.3      50.5      -17.25
       2     Thomas     M      11     57.5      85.0       17.25
```

Wegen der Benutzung von CALL SYMPUT und der nachfolgenden Referenzierung der so erzeugten Makrovariablen MEDIAN führt ein Aufruf mittels CALL EXECUTE aber zu einer Fehlermeldung, weil in der Zeile "`DeltaW = weight - &median ;`" der Ausdruck "`&median.`" nicht aufgelöst wird. Da dies im Data Step passiert, kann das Problem noch behoben werden, indem man die direkte Referenz "`&median.`" durch "`symget("median")`" oder "`%nrstr(resolve("&median."))`" oder auch "`resolve('&median.')`" ersetzt. Bei letzterer Variante achte man auf die Verwendung der einfachen (!) Anführungsstriche. Ebenso wie durch die Verwendung von `%nrstr` wird auf diese Weise der Makrokompiler daran gehindert, während der Kompilierung schon `&median.` zu ersetzen. Stattdessen erfolgt die Auflösung der Makrovariablen erst zur Laufzeit des entsprechenden Data Steps.
Diese Möglichkeiten stehen allerdings nur innerhalb eines Data Steps zur Verfügung. Es gibt kein Mittel, eine direkte Referenz ausserhalb eines Data Steps zur passenden Laufzeit aufzulösen.
Die SAS-online-Dokumentation sagt zu dem Thema folgendes.
"Note: Because macro references execute immediately and SAS statements do not execute until after a step boundary, you cannot use CALL EXECUTE to

invoke a macro that contains references for macro variables that are created by CALL SYMPUT in that macro."

Aber was, wenn man es doch tut? Immerhin ist dieser Hinweis in der SAS Dokumentation relativ leicht zu übersehen. Hier muss gesagt werden, dass unter Umständen gar keine Referenzierungsprobleme auftreten, nämlich dann, wenn man die `%local`-Anweisung weglässt und in der nächsthöheren nichtleeren Symboltabelle keine Makrovariable des gleichen Namens existiert. Ganz und gar problematisch wird es aber, wenn es in der nächsthöheren Symboltabelle doch eine Variable gleichen Namens gibt. Dann rechnet das Makro mit dem Wert, den diese Makrovariable dort zur Zeit der Kompilierung hatte und gibt also vermutlich ein falsches Ergebnis aus. Obendrein weist unser Makro dieser Variablen in der nächsthöheren Symboltabelle zur Laufzeit des Data Steps den Wert zu, den wir eigentlich zum Zeitpunkt der Kompilierung gebraucht hätten. Das kommt für unseren Data Step zu spät und kann anderswo schlimme Folgen haben, wenn die Makrovariable der höheren Symboltabelle das nächste Mal verwendet wird. Insgesamt hat man es mit einem reichlich unübersichtlichen und fehlerträchtigen Sachverhalt zu tun. Das Ergebnis unseres Programmes wird gewissermaßen zufällig und hängt von Faktoren ab, die mit dem Makro selber nichts zu tun haben. Sicherlich tut man gut daran, sich an obigen Hinweis zu halten und CALL EXECUTE nicht in Verbindung mit CALL SYMPUT zu verwenden.

Zur Erläuterung, warum die nächsthöhere Symboltabelle hier überhaupt zum Tragen kommt, noch ein Zitat aus gedruckter sowie online SAS-Dokumentation: "CALL SYMPUT creates the macro variable in the current symbol table (...), provided that symbol table is not empty. If it is empty, usually CALL SYMPUT creates the variable in the closest nonempty symbol table. (...) You may find it helpful to use the %PUT statement with the _USER_ option to determine what symbol table the CALL SYMPUT routine has created the variable in." Sorgen bereitet hier die Verwendung des Wortes "usually". Überraschend ist aber der Rat, vorsichtshalber nachzusehen, welcher Symboltabelle die erzeugte Makrovariable denn nun angehört. Dieses Zitat zu CALL SYMPUT steht im übrigen in keinem Zusammenhang zu der Problematik mit CALL EXECUTE.

Besondere Vorsicht im Umgang mit CALL EXECUTE ist geboten, wenn man in einer Umgebung mit umfangreicher autocall-Makrobibliothek programmiert. Im Grunde muss vollständig von der Verwendung von CALL EXECUTE abgeraten werden.

4 Alternativen

An dieser Stelle sollen die zwei gängigsten Alternativen zur Verwendung von CALL EXECUTE kurz skizziert werden. Sicher gibt es weitere Ansätze, und je nach Programmierziel kann u. U. ein gänzlich überarbeiteter Programmaufbau eine effiziente Lösung darstellen.

4.1 Schleifen:

In vielen Fällen sind einfache Schleifendurchläufe eine gute Technik, beispielsweise wenn ein Makro für jeden in den Daten vorkommenden Wert einer Variablen aufgerufen werden soll.
Mit PROC SQL lässt sich leichter als per Data Step eine Liste der vorkommenden Werte herstellen. Zudem erzeugt PROC SQL auch gleich automatisch die Makrovariable SQLOBS, in der die Anzahl der Listenelemente gespeichert ist. Der anschließende Schleifendurchlauf findet in einem kurzen Makro statt.
Obiges Programmierproblem für den Fall der mindestens 14jährigen sähe etwa so aus:

```
proc sql noprint ;
  select distinct age into :agelist separated by " "
  from sashelp.class
    <where = …>
    <order by …> ;
quit ;

%macro runmacro ;
  %local m agevar ;
  %do m = 1 %to &sqlobs. ;
    %let agevar = %scan ( &agelist. , &m. ) ;
    %if %sysevalf ( &agevar. >= 14 ) %then
      %do ;
        %agegroup ( class = &agevar. ) ;
      %end ;
  %end ;
%mend  runmacro ;
%runmacro ;
```

4.2 Skriptfiles:

Die grundsätzliche Idee beim Erzeugen von Skriptfiles ähnelt dem Ansatz mit CALL EXECUTE sehr. Das ist ein großer Vorteil vor allem dann, wenn man mit CALL EXECUTE begonnen hat und unerwartet auf Probleme gestoßen ist. Das Programm benötigt nicht viele Änderungen. Ein kleiner Nachteil ist allerdings, dass die Definition von Pfadangaben plattformabhängig ist. Für den Fall, dass ein Programm auf verschiedenen Plattformen laufen soll, kann jedoch leicht mit Hilfe der automatischen Makrovariablen SYSSCP oder SYSSCPL die aktuelle Plattform abgefragt und ein Pfad entsprechend definiert werden.
Unser Programm für die mindestens 14jährigen sieht dann ungefähr so aus:

```
proc sort
  data = sashelp.class
  out  = class ;
    by age ;
run ;

%macro deffile ;
  %if &sysscp. = WIN %then
    %do ;
      filename script "c:\temp\sasskript.sas" ;
    %end ;
  %else %if &sysscp. = UNIX %then
    %do ;
      filename script "~/temp/sasskript.sas" ;
    %end ;
  %else %if &sysscp. = VMS %then
    %do ;
      filename script "work::[username.temp]sasskript.sas;1"
    %end ;
  %else
    %do ;
      %put **** Cannot define filename for this platform.
    %end ;
mend deffile ;
%deffile ;

data _null_ ;
  file script ;
  set class ;
    by age ;
  if first.age and age >= 14 then
    put '%agegroup ( class = ' age ' ) ; ' ;
run ;
%include script ;
```

Für Fragen oder Kommentare stehe ich jederzeit zur Verfügung. Meine Emailadresse steht am Anfang dieses Aufsatzes.

Literatur

1. SAS online Dokumentation (V8)
2. SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, 1991, SAS Institute Inc., Cary , NC, USA
3. SAS Macro Language, Reference, First Edition (1997), SAS Institute Inc., Cary, NC, USA
4. SAS Macro Language, Reference, Second Edition (1999), SAS Institute Inc., Cary, NC, USA