Zwei Tipps und Tricks: Fallstricke bei Makrovariablen und SAS im Batchmodus

Daniel Schulte
viadee Unternehmensberatung GmbH
Anton-Bruchausen-Straße 8
48147 Münster
daniel.schulte@viadee.de

Zusammenfassung

Fallstricke bei Makrovariablen:
Bei der Entwicklung komplexer Abläufe in SAS kommt man in der Regel nicht um die Verwendung von Makrovariablen herum. Diese sind allerdings auch gerne Quelle von unerwarteten Ergebnissen und Abläufen. An Beispielen werden einige Fallstricke aufgezeigt.
SAS im Batchmodus:
Der ETL Prozess in DWH Projekten ist der entscheidende Schritt für die darauf aufbauenden Analysen. Diese Aufbereitung kann bei umfangreichen Daten zeitkritisch werden und nicht beliebig nur durch schnellere Hardware kompensiert werden. Eine effiziente Ausnutzung der vorhandenen Ressourcen und eine Optimierung für den Fehlerfall ermöglichen eine schnellere Verfügbarkeit und vereinfachten Betrieb.

Schlüsselwörter: Makro, Makrovariable, Batch, ETL

1 Fallstricke bei Makrovariablen

1.1 Ausgangssituation

Man nehme einen Datastep und fertig ist die erste Verarbeitung im SAS System. Mit der Zeit wachsen allerdings die Anforderungen an die Aufbereitung und die Anzahl der Steps wächst und wächst. Über Makrovariablen können auf einfache Art und Weise zwischen einzelnen Datasteps und Prozeduren Informationen ausgetauscht werden. Allerdings sind hier die Unterschiede zwischen SAS Base und SAS Makro in der Kompilations- und Ausführungsphase zu beachten.

1.2 Kompilations- und Ausführungsphase

Abgeschickter SAS Code durchläuft mehrere Phasen: die Kompilations- und die Ausführungsphase. Die Kompilierung ist weiter in einzelne Stufen zu unterteilt. Makrovariablen werden zunächst durch den Makro Prozessor verarbeitet und aufgelöst. Danach wird der Step kompiliert und ausgeführt. Gerade diese Trennung zwischen Makro Verarbeitung und Step Kompilierung wird dem Entwickler manchmal zum Verhängnis.

Code:	Code:

`%let date="24feb2011"d;`
 	
`data _null_;`
`  var=&date.;`
`  put var;`
`  put var ddmmyy.;`
`run;`
	`options symbolgen;`
	`%let date="24feb2011"d;`
	
	`data _null_;`
	`  var=&date.;`
	`  put var;`
	`  put var ddmmyy.;`
	`run;`

Log:
`1  %let date="24feb2011"d;`
`2  data _null_;`
`3  var=&date.;`
`4  put var;`
`5  put var ddmmyy.;`
`6  run;`
 	
`18682`
`24/02/11`
	LOG:
	`8  options symbolgen;`
	`9  %let date="24feb2011"d;`
	`10  data _null_;`
	`11  var=&date.;`
	`SYMBOLGEN:  Macro variable DATE resolves to "24feb2011"d`
	`12  put var;`
	`13  put var ddmmyy.;`
	`14  run;`
	
	`18682`
	`24/02/11`

Ablauf:
Der Makro Prozessor durchsucht den Code nach relevanten Schlüsselwörtern (%let…/ &) und löst diese auf. Die Makro Variable date wird im System als globale Variable angelegt und mit der Zeichenkette `"24feb2011"d` belegt. Im Source des Datasteps wird die Makrovariable gefunden und gegen die Zeichenkette `"24feb2011"d` ausgetauscht. Der Compiler bekommt also den folgenden Sourcecode zu „sehen“:

```
data _null_;
     var="24feb2011"d;
     put var;
     put var ddmmyy.;
run;
```

Bei der Zuweisung von Makrovariablen in einem Datastep per `call symput` wird genau dies zum Problem. Wird die Makrovariable erst in der Ausführungsphase zugewiesen, hat der Makro Prozessor davon noch nichts mitbekommen.

```
2    call symput ('date',"24feb2011"d);
3    var=&date.;
         -
         386
         200
WARNING: Apparent symbolic reference DATE not resolved.
ERROR 386-185: Expecting an arithmetic expression.

ERROR 200-322: The symbol is not recognized and will be
ignored.
```

`call symput` bringt also nur etwas für den nachfolgen Datastep, da hier der Makro Prozessor wieder aktiv werden kann. Symget kann hier Abhilfe schaffen. Aber Symget liefert immer einen Text. Dieser muss ggf. per input eingelesen werden.

Die Option symbolgen ist hilfreich, um die effektiven Inhalte einer Makrovariable anzeigen zu lassen.

1.3 Anführungszeichen um Makrovariablen

Bei der Verwendung von Makrovariablen in einem Datastep führt die Einfassung in Anführungszeichen zu unterschiedlichen Ergebnissen.
Beispiel:

```
CODE:
%let mvar=Inhalt_der_Makrovariable;
data _null_;
 var="&mvar.";
 put var;
 var='&mvar.';
 put var;
 var=&mvar.;
 put var;
run;

LOG:
1    %let mvar=Inhalt_der_Makrovariable;
2    data _null_;
3     var="&mvar.";
4     put var;
5     var='&mvar.';
6     put var;
7     var=&mvar.;
8     put var;
9    run;

NOTE: Variable Inhalt_der_Makrovariable is uninitialized.
Inhalt_der_Makrovariable
&mvar.
```

Das Beispiel zeigt drei Varianten:

- Doppelte Anführungszeichen
  die Makrovariable wird in den Anführungszeichen aufgelöst und der Variable Var als Zeichenkette übergeben
- einfache Anführungszeichen
  die Makrovariable wird nicht aufgelöst und der Aufruf wird als Zeichenkette übergeben
- keine Anführungszeichen
  Die Makrovariable wird aufgelöst und entspricht nun im Datastep einer Zuweisung nach dem Muster VariableA=VariableB. Daher auch der Hinweis, dass die Variable Inhalt_der_Makrovariable nicht initialisiert ist.

Bei der Zuweisung wird alles nach dem = bis zum ; als Zeichenkette zugewiesen. Auch enthaltene Anführungszeichen werden übergeben. Sollen (Sonder)zeichen wie z.B. ein Semikolon in eine Makrovariable übergeben werden, muss mit der Funktion %quote dies übergeben werden.

1.4 Verschachtelte Makrovariablen

Bei der programmseitigen Erzeugung von Makrovariablen entsteht schnell die Anforderung nach verschachtelten Makrovariablen.
Beispiel:
In einem Programm soll für ein Datenfeld, welches z.B. Dateinamen enthält, für jeden Datensatz eine Makrovariable erzeugt werden. Da hier n Variablen entstehen, werden diese file1, file2, …. fileN benannt. Sollen diese Variablen nun per Programm weiterverwendet werden, kann bei der Erzeugung eine Hilfsvariable mit der Anzahl an Variablen definiert werden, die in einer Schleife die Obergrenze angibt.

Beispiel:

CODE:

```
%let file1=d:\ksfe2011\datei1.cport;
%let file2=d:\ksfe2011\datei2.cport;
%let file3=d:\ksfe2011\datei3.cport;
%let anzahl=3;
options macrogen symbolgen;
%macro import;
%do i=1 %to &anzahl.;
     proc cimport lib=sasuser file="&&file&i.";
     run;
%end;
%mend import;
%import;
```

LOG:

```
1    %let file1=d:\ksfe2011\datei1.cport;
2    %let file2=d:\ksfe2011\datei2.cport;
3    %let file3=d:\ksfe2011\datei3.cport;
4    %let anzahl=3;
5    options macrogen symbolgen;
6    %macro import;
7        %do i=1 %to &anzahl.;
8            proc cimport lib=sasuser file="&&file&i.";
9            run;
10       %end;
11   %mend import;
12   %import;
SYMBOLGEN:  Macro variable ANZAHL resolves to 3
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 1
```

```
SYMBOLGEN:  Macro variable FILE1 resolves to
d:\ksfe2011\datei1.cport
MACROGEN(IMPORT):   proc cimport lib=sasuser
file="d:\ksfe2011\datei1.cport";
MACROGEN(IMPORT):   run;

NOTE: Proc CIMPORT begins to create/update data set
SASUSER.TEST1
NOTE: Data set contains 1 variables and 1 observations.
      Logical record length is 8

NOTE: PROCEDURE CIMPORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 2
SYMBOLGEN:  Macro variable FILE2 resolves to
d:\ksfe2011\datei2.cport
MACROGEN(IMPORT):   proc cimport lib=sasuser
file="d:\ksfe2011\datei2.cport";
MACROGEN(IMPORT):   run;

NOTE: Proc CIMPORT begins to create/update data set
SASUSER.TEST2
NOTE: Data set contains 1 variables and 1 observations.
      Logical record length is 8

NOTE: PROCEDURE CIMPORT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 3
SYMBOLGEN:  Macro variable FILE3 resolves to
d:\ksfe2011\datei3.cport
MACROGEN(IMPORT):   proc cimport lib=sasuser
file="d:\ksfe2011\datei3.cport";
MACROGEN(IMPORT):   run;

NOTE: Proc CIMPORT begins to create/update data set
SASUSER.TEST3
NOTE: Data set contains 1 variables and 1 observations.
      Logical record length is 8

NOTE: PROCEDURE CIMPORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

2 SAS im Batchmodus

2.1 Ausgangsituation

Sollen Daten regelmäßig verarbeitet werden, ist es sinnvoll, dies nicht manuell zu starten, sondern als zeit- oder eventgesteuerte Aufgabe einzuplanen. Hierbei sind aber auch einige Dinge zu beachten, die den Unterschied zwischen einer erfolgreichen Batchverarbeitung und einer mühevollen Bereinigung von Datenbeständen ausmachen. Auch die verfügbare Zeit ist meist ein kritischer Faktor.

2.2 SAS als Batch aufrufen

Die übliche Oberfläche ist für einen Batchaufruf des SAS Systems natürlich nicht notwendig. Dies kann über einige Aufrufparameter gesteuert werden.

sas.exe	Aufruf des SAS System
-sysin pfad\dateiname	Programm, welches ausgeführt werden soll
-autoexec pfad\dateiname	Initialisierung der Umgebung
-log pfad\dateiname	Ausgabe des Logs
-print pfad\dateiname	Ausgabe des Outputs
-sysparm parameter	Übergabe von einem Wert an die Makrovariable SYSPARM

Zwingend notwendig ist der Einsatz von SYSIN, die anderen Parameter sind zwar optional aber dennoch sehr wichtig.

AUTOEXEC
Über diesen Parameter kann jedem Job eine passende Grundlage geboten werden. In einem Projekt sollten Pfade und Bibliotheken einheitlich abgelegt sein. Mit einer projektweit verwendeten autoexec und den darin enthaltenen Parametern und Bibliothekszuweisungen lässt sich schnell Ordnung schaffen.

LOG
Wird dieser Parameter nicht gesetzt, wird automatisch im Arbeitsverzeichnis ein Logfile mit dem Namen des SAS Programms angelegt. Mit diesem Parameter sollte das Log in einem definierten Order unter einem passenden Namen abgelegt werden. Dieser kann z.B. im Aufruf durch Datum und Uhrzeit gekennzeichnet und so historisiert werden.

PRINT
Alles was sonst, gewollt oder auch ungewollt, im Output Fenster einer interaktiven SAS Sitzung landet, würde bei nicht Verwendung automatisch im Arbeitsverzeichnis eine Ausgabedatei mit dem Namen des SAS Programms anlegen.
Hier kann man nun zwei Ansätze verfolgen. Die einfache Variante wäre, den Parameter PRINT im Aufruf zu setzen und alle Outputs in eine definierte Datei zu schreiben. Nachvollziehbarer ist es jedoch, bei allen Output erzeugenden Programmen die Ausgabe per ODS in die gewünschten Pfade und Dateien zu lenken.

```
ods listing file="d:\ksfe2011\Ausgabe1.txt";
```

SYSPARM
Um dem aufzurufenden Job noch externe Steuerinformationen mitzugeben ist eine Möglichkeit, diese per SASPARM direkt in den Aufruf einzubauen. Sollen allerdings umfangreichere Informationen mitgeliefert werden, bietet sich eine Steuerdatei an, die z.B. per SYSPARM selektiert wird.

2.3 Parallelisierbarkeit

SAS Programme laufen standardmäßig rein sequenziell ab. Datastep für Datastep und Prozedur für Prozedur werden nacheinander abgearbeitet. Einige Schritte sind zwar mittlerweile Multithreading tauglich, hierbei wird jedoch nur ein Schritt auf mehrere Threads verteilt und man erreicht so ein schnelleres Ergebnis – aber auch nur für diesen Schritt.
Ein Beispielszenario wäre etwa, dass 5 Dateien eingelesen und dann weiterverarbeitet werden sollen. Statt nun 5 Datasteps mit den Einleseroutinen hintereinander ablaufen zu lassen, werden diese nun in je einen eigenständigen Job abgelegt. Diese können durch ein Aufrufscript, unter Windows z.B. eine .cmd Datei, gestartet werden, wobei der letzte mit einem Wartehinweis versehen wird.

```
start sas -autoexec autoexec.sas -sysin einlesen1.sas -log job1.log
start sas -autoexec autoexec.sas -sysin einlesen2.sas -log job2.log
start sas -autoexec autoexec.sas -sysin einlesen3.sas -log job3.log
start sas -autoexec autoexec.sas -sysin einlesen4.sas -log job4.log
start/w sas -autoexec autoexec.sas -sysin einlesen5.sas –log
job5.log
start sas -autoexec autoexec.sas -sysin verarbeiten.sas -log etl.log
```

Nun werden alles 5 Einlese-Jobs parallel gestartet. Das Script wartet allerdings auf das Ende des 5. Jobs und fährt erst nach dessen Ende weiter fort. Hierbei ergibt sich allerdings das Problem, dass nur auf den 5. Job gewartet wird. Ist dieser nun schneller als die ersten vier, würde die Verarbeitung fälschlicherweise weiterlaufen.

Um eine genauere Kontrolle über die einzelnen Jobs zu erlangen, besteht die Möglichkeit, dies direkt über das SAS System zu steuern.

```
%macro JOB;
systask command "sas -sysin D:\KSFE2011\einlesen1.sas -log job1.log"
mname=job1 status=RC_job1;
systask command "sas -sysin D:\KSFE2011\einlesen2.sas -log job2.log"
mname=job2 status=RC_job2;
systask command "sas -sysin D:\KSFE2011\einlesen3.sas -log job3.log"
mname=job3 status=RC_job3;
systask command "sas -sysin D:\KSFE2011\einlesen4.sas -log job4.log"
mname=job4 status=RC_job4;
systask command "sas -sysin D:\KSFE2011\einlesen5.sas -log job5.log"
mname=job5 status=RC_job5;

waitfor _all_ &job1. &job2. &job3. &job4. &job5.;
```

```
%if %sysevalf(&RC_job1. + &RC_job2. + &RC_job3. + &RC_job4. +
&RC_job5.) eq 0 %then;
   %include "D:\KSFE2011\verarbeiten.sas";

%mend JOB;
%JOB;
```

Wie im vorherigen Beispiel werden die einzelnen Programmteile parallel abgesetzt. Der Autoexec Parameter wurde zur besseren Lesbarkeit weggelassen. Das Waitfor Kommando wartet nun auf alle 5 Teile, bis diese abgeschlossen wurden. Um nun sicher zu gehen, dass keine Fehler beim Einlesen aufgetreten sind, wird noch die Summe der Rückgabewerte der Jobs addiert. Wenn diese erfolgreich waren, liefert jeder Job eine 0 und die Summe daraus ist ebenfalls 0. Ist genau diese Bedingung erfüllt, wird der verarbeitende Teil des Jobs per `%include` eingebunden.

2.4 Verhalten bei Fehlersituationen

Sollte es bei der Batchverarbeitung zu Problemen kommen, empfehlt es sich, das Programm hart oder kontrolliert abzubrechen. Die harte Methode wird über die Option errorabend erreicht. Kommt es zum Fehler, wird die Verarbeitung sofort gestoppt und die SAS Sitzung beendet. Zur Fehlerdiagnose kann dann zwar das Log genutzt werden, aber die Daten in WORK sind mit dem Abbruch verloren. Wird statt WORK nun eine definierte Bibliothek verwendet, die nur bei erfolgreichem Programmabschluss wieder gelöscht wird, stehen auch Daten für eine Fehleranalyse zur Verfügung. Eine kontrollierte Steuerung ist beispielsweise über die Makrovariablen SYSERR und SYSWARNINGTEXT durchführbar. Nach jedem Step kann z.B. geprüft werden, ob Fehler oder Warnungen aufgetreten sind und abhängig von Fehlerart und –position kann reagiert werden.

2.5 Abhängigkeiten

Egal ob die Jobs per Betriebssystem oder SAS gesteuert parallel laufen sollen, in beiden Fällen muss vorher sichergestellt werden, dass keine Konkurrenzsituationen entstehen. Diese sind z.B. gleichzeitiger schreibender Zugriff auf Ausgabe- oder Steuertabellen. Gerade bei Steuertabellen, die häufiger im Zugriff sind, sollen daher nicht nativ in SAS sondern in einem DBMS wie DB2, Oracle, MySQL o.ä. abgelegt werden. Hierfür sind allerdings im Lizenzumfang ACCESS Module notwendig.

Literatur

[1] SAS 9.2 Documentation
http://support.sas.com/documentation/cdl_main/