Tipps und Tricks
für den leichteren Umgang mit der SAS-Software

Hans-Peter Altenburg
Dade Behring Marburg GmbH, QAS/SV M546
Emil-von-Behring-Str. 76
35001 Marburg
HansPeter_Altenburg@dadebehring.com

Carina Ortseifen
Universitätsrechenzentrum Heidelberg
Im Neuenheimer Feld 293
69120 Heidelberg
carina.ortseifen@urz.uni-heidelberg.de

Tanja Petrowitsch
Bayer HealthCare AG
BHC-PH-GDC-GCD-GB-IA (Integrated Analyses)
42096 Wuppertal-Elberfeld
tanja.petrowitsch@bayerhealthcare.com

Grischa Pfister
iCASUS GmbH
Vangerowstraße
Heidelberg
g.pfister@icasus.de

Wilfried Schollenberger
WS Unternehmensberatung und Controlling-Systeme GmbH
Friedrich-Weinbrenner-Straße 20
69126 Heidelberg
wisch@ws-unternehmensberatung.de

Zusammenfassung

In Form von kurzen Beiträgen werden nützliche Lösungen zu Problemen und Fragestellungen vorgestellt, die bei der täglichen Arbeit mit der SAS Software auftreten können. Es werden dabei nicht unbedingt neue Prozeduren, Optionen oder Module vorgestellt. Stattdessen soll die effektive Anwendung vorhandener Anweisungen und Prozeduren an Beispielen aufgezeigt werden.
Themen:
1. Der Schlüsselworte _All_, _Character_ und _Numeric_
2. Dynamisch SAS-Code erzeugen und Ausführen mit Call Execute ()
3. Die Funktion Dcreate()
4. Neues von Proc Means und Proc Summary
5. Proc Means und der Umgang mit der _Type_-Variablen in der Output-Anweisung
6. Die Option Fuzz bei Proc Format
7. ODS Tagset ExcelXP
8. Einrichten einer Projekt-Umgebung
9. Auflösung der Publikumsfrage der 10. KSFE Hamburg

Schlüsselwörter: _ALL_, _CHARACTER_, _NUMERIC_, _TYPE_ Variable (PROC MEANS, OUTPUT), Autoexec.Sas, CALL EXECUTE(), CLASS Anweisung (PROC MEANS), Config.Sas, DCREATE(), ExcelXP Tagset, FORMAT Prozedur, FUZZ Option (PROC FORMAT), IDGROUP Option (PROC MEANS, OUTPUT), MEANS Prozedur, ODS Tagset ExcelXP, OUTPUT Anweisung (PROC MEANS), Programmierung, Prozedur FORMAT, Prozedur MEANS, Publikumsfrage, Systemoption, Tagset ExcelXP, Tipps.

1 Die Schlüsselworte _ALL_, _CHARACTER_ und _NUMERIC_

(Wilfried Schollenberger)

Dieser Tipp beschreibt keine neue Funktion in SAS, sondern ist mehr eine Erinnerung. Die Verwendung in der Array-Anweisung ist bereits für die Version 6.06 im Language Guide dokumentiert.

1.1 Beschreibung der Schlüsselworte

Die Schlüsselworte werden in den folgenden Beispielen in den Anweisungen Array, Format und By verwendet. Dabei haben die Schlüsselworte folgende Bedeutung:

- _All_ steht für die Liste aller Variablen
- _Character_ steht für die Liste aller alphanumerischen Variablen
- _Numeric_ steht für die Liste aller numerischen Variablen

Normalerweise sollten in produktiven Programmen möglichst alle Variablen explizit aufgeführt werden. Es gibt aber Fälle, insbesondere bei allgemein verwendbaren Makros und bei Ad-hoc-Auswertungen, in denen diese impliziten Listen sehr nützlich sind.

1.2 Beispiel: Alle alphanumerischen Variablen in Großbuchstaben umwandeln

Im ersten Beispiel werden alle alphanumerischen Variablen in Großbuchstaben umgewandelt:

```
Data work.gross;
     Set work.gemischt;
     Array _ac _CHARACTER_;
     do over _ac;
          _ac = upcase(_ac);
     end;
Run;
```

Wenn einzelne Variablen nicht umgewandelt werden sollen, kann man den Variablen-Namen mit der Funktion Vname() abfragen, wie in dem folgenden Beispiel:

```
Data work.gross;
     Set gemischt;
     Array _ac _CHARACTER_;
     do over _ac;
          if lowcase(vname(_ac)) NOT IN ("name", "vorname")
               then _ac = upcase(_ac);
     end;
Run;
```

1.3 Beispiel: Alle numerischen Variablen runden

Auf der letzten KSFE wurde gezeigt, dass es bei Dezimalzahlen zu Rundungs-Problemen kommen kann, weil SAS intern mit einer Gleitkomma-Darstellung rechnet. Deshalb ist es manchmal sinnvoll, alle numerischen Variablen auf eine definierte Anzahl von Nachkommastellen zu runden:

```
Data work.gerundet;
     Set original;
     Array _an _NUMERIC_;
     do over _an;
          _an = round(_an,0.01);
     end;
Run;
```

Natürlich können auch hier, wie oben beschrieben, einzelne Variablen vom Runden ausgenommen werden.

1.4 Beispiel: Einheitliche Formatierung von Variablen

Es kommt immer wieder vor, dass in SAS-Dateien die Format-Angaben fehlen. Bei Datenbank-Tabellen gibt es auch gar keine Möglichkeit, SAS-Formate fest zuzuordnen. Hier können dann einheitliche Formate beim Prozedur-Aufruf vergeben werden:

```
Proc fsview data=ksfe.bsp1;
     Format _NUMERIC_ commax12.2;
Run;
```

Danach kann diese „Voreinstellung“ für einzelne Variablen wieder überschrieben werden:

```
Proc fsview data=ksfe.bsp1;
     Format _NUMERIC_ commax12.2
          datum deudfdd10.;
Run;
```

1.5 Beispiel: Erstellen einer CSV-Datei

Auch beim „manuellen“ Erstellen einer CSV-Datei sind diese Schlüsselworte hilfreich. In diesem Beispiel werden zunächst alle alphanumerischen und dann alle numerischen Variablen ausgegeben.

```
Data _null_;
     Set ksfe.bsp1;
     Format _CHARACTER_ $quote200.
            _NUMERIC_ numx16.8
            datum deudfdd10.;
     Array _an _NUMERIC_;
     Array _ac _CHARACTER_;
     File "Test.csv" dlm=";";
     /* Spalten-Überschriften mit den Variablen-Namen */
```

```
        if _N_ EQ 1 then do;
             do over _ac;
                  _vn = vname(_ac);
                  Put _vn : $quote40. @;
             end;
             do over _an;
                  _vn = vname(_an);
                  Put _vn : $quote40. @;
             end;
             Put;
        end;

        /* Werte */
        do over _ac;
             Put _ac @;
        end;
        do over _an;
             Put _an @;
        end;
        Put;
   Run;
```

Auch hier wird das allgemeine Format Numx.32.16 für den Datumswert wieder überschrieben. Statt der Variablen-Namen könnte man natürlich auch Variablen-Labels (mit Vlabel()) verwenden.

1.6 Beispiel: Vergleich zweier Dateien

Im letzten Beispiel zu diesem Tipp geht es um den Vergleich zweier Dateien. Dafür gibt es die Prozedur Compare, welche die einzelnen Werte vergleicht. Wenn es aber nur wichtig ist, ob sich einzelne Zeilen einer SAS-Tabelle unterscheiden, und dann ein Programm verschiedene Aktionen ausführen soll, ist ein Daten-Schritt möglicherweise viel praktischer.
Zunächst werden Schlüssel-Variablen benötigt, die jede Zeile eindeutig identifizieren. Im Beispiel sind das die Variablen kunde, datum und produkt. Dann müssen beide Dateien nach diesen Variablen sortiert werden. In der By-Untergruppen-Verarbeitung wird dann hinter diesen Schlüssel-Variablen mit _All_ bewirkt, dass alle anderen Variablen verglichen werden.
Mit dem folgenden Daten-Schritt können dann alle Fälle unterschieden und verarbeitet werden:

```
   Data work.vergleich;
        Merge ksfe.bsp1 (in = _d1)
              ksfe.bsp2 (in = _d2) ;
        By kunde datum produkt _ALL_;
```

```
        Length Vergleich $11;

        * gleiche Daten (interessieren heute nicht);
        if _d1 AND _d2 then Delete;

        * neue und gelöschte Zeilen;
        else if first.produkt AND last.Produkt then do;
             if _d1 then vergleich = "gelöscht";
                    else vergleich = "neuer Satz";
        end;

        * geänderte Werte;
        else do;
             if _d1 then vergleich = "alter Wert";
                    else vergleich = "neuer Wert";
        end;
    Run;
```

Die Ergebnisdatei könnte dann z.B. so aussehen:

Kunde	Datum	Produkt	Vergleich	Name	Menge	Preis
A089bdC8	**03.03.2007**	**P4711abCVD**	**alter Wert**	Hans Müller	45,00	234,80
			neuer Wert	Hans Müller	65,00	434,80
b077bdC8	**01.03.2007**	**P4711ABCVx**	**gelöscht**	Kurt Meyer	34,00	234,68
b077bx99	**01.03.2007**	**P4711ABCVx**	**neuer Satz**	Harald Berger	77,00	77,00

Erzeugt mit:

```
ODS rtf file="~/k.rtf";
Proc print  data=vergleich;
     BY kunde datum produkt ;
     ID kunde datum produkt Vergleich;
     Format _numeric_ commax6.2 datum deudfdd10.;
Run;
ODS rtf close;
```

Literatur

[1] SAS Language Guide, alle Versionen ab 6.06, Beschreibung von Array

[2] Die Verwendung in den Format- und Informat-Anweisungen ist leider undokumentiert.

2 Dynamisch SAS-Code erzeugen und ausführen mit Call Execute()

(Wilfried Schollenberger)

2.1 Einführung

Normalerweise wird dynamisch erzeugter SAS-Code in eine temporäre Datei geschrieben und dann mit %Include eingelesen.
Beispiel:

```
Filename programm TEMP;
Data _null_;
     File programm;
     Put '%put Das bin ich;';
Run;
%include programm;
```

Dagegen ist normalerweise nichts einzuwenden, solange man die Umgebung, in der der Programmteil läuft, kennt. Anders ist das bei allgemein verwendeten Programm-Teilen und Autocall-Makros. Z.B. könnte der Benutzer vorher den Filename schon vergeben haben:

```
Filename programm '~/sas/meineProgramme';
```

Nachdem unser Programm-Teil ausgeführt wurde, wird er seine Programme nicht mehr finden, es sei denn, er setzt seine Filename-Anweisung erneut ab.
Dieses Problem lässt sich vermeiden, wenn auf die temporäre Datei verzichtet und der SAS-Code mit der Routine Call Execute() übergeben wird. Obiges Beispiel sähe dann so aus:

```
Data _null_;
     call execute('%Put Das bin ich;');
Run;
```

2.2 Beispiel 1: Bedingte Erstellung von Auswertungen

Zum Beispiel könnte man in einer SAS-Datei eine Liste von Auswertungs-Programmen speichern, und mit einem Kennzeichen steuern, ob die jeweilige Auswertung in einem Nacht-Lauf tatsächlich ausgeführt werden soll[1].

```
Data _NULL_;
     Set ksfe.auswertungen;
     if starten GT " " then
          call execute(auswertung);
Run;
```

Oder man macht die Auswertung davon abhängig, ob eine Datei geändert wurde:

[1] Da „freut“ sich dann der Operator, wenn er nicht weiß, wie lang der Batch-Job laufen wird, und deshalb nicht richtig planen kann. Andererseits ist es natürlich sinnvoll, aufwendige Auswertungen möglichst in die Nacht-Verarbeitung zu legen und nur bei Bedarf auszuführen.

```
Data ksfe.reports;
     Set ksfe.reports;

     dsid = open(datei,"I");

     if attrn(dsid,"modte")) GT letzter_Stand then do;
          call execute(auswertung);
          letzter_Stand = attrn(dsid,"modte"));
     end;

     dsid = close(dsid);
Run;
```

In der Variablen Auswertung wird vermutlich ein Makro-Aufruf stehen. Man kann mit Call Execute() aber auch normalen SAS-Code absetzen, wie in dem folgenden Beispiel.

2.3 Beispiel 2: Liste von Datenbank-Tabellen mit der Anzahl der Sätze

Hintergrund ist, dass die Dictionary-Tabellen von SAS bei Views und Datenbanken die Anzahl der Beobachtungen nicht enthalten. Wenn man nun eine Übersicht über alle Tabellen und ihre Größe benötigt, kann man diese mit dem folgenden Programm erzeugen:

```
/** Verzeichnis der Tabellen erstellen */
Proc sql;
     Create table work.dir as
          select trim(libname) !! "." !! memname as dsname,
                 memname
          from   dictionary.tables
          where libname EQ "DB2LIB"
          order by memname;
Quit;

/** Anzahl der Records in jeder Tabelle ermitteln */
Data _null_;
     Set work.dir end=finale;

     if _n_ EQ 1 then
          call execute('Proc Sql;'
               !!' create table work.dir_nobs as ');
     else call execute(' outer union corresponding ');

     call execute(' select "' !!memname
               !! '" as Name,'
               !! ' count(*) as records Format=commax12.0'
               !! ' from ' !! dsname);
```

```
     if finale;
     call execute(';Quit;');
Run;

/** Liste ausgeben */
Proc print data=work.dir_nobs label;
     Var memname records;
     Label memname = "Tabelle"
           records = "Anzahl Sätze"
     ;
Run;
```

2.3 Besonderheit bei Makros

Zu beachten ist, dass Makros sofort, noch während des Laufs des Daten-Schritts ausgeführt werden. In dem folgenden Programm wird dieser Effekt demonstriert. Das Makro zeigt immer den zur Laufzeit aktuellen Inhalt der Makro-Variablen 'Name':

```
%macro zeigeName;
     %put &name;
%mend;
Data _null_;
     Set ksfe.bsp1;
     call symput('Name', name);
     call execute('%zeigeName; ');
Run;
```

Wenn man das vermeiden will, muss die Makro-Funktion %Nrstr() verwendet werden:

```
Data _null_;
     Set ksfe.bsp1;
     call symput('Name',name);
     call execute('%nrstr(%zeigeName); ');
Run;
```

Jetzt wird immer der zuletzt in die Makro-Variable gestellte Wert ausgegeben. Das ist sehr wichtig, wenn in einem Makro SAS-Anweisungen und Makro-Verarbeitung verknüpft werden. Z.B. funktioniert das folgende Makro nur in Verbindung mit %Nrstr():

```
%macro testefile(n);
     Filename ao "&n";
     %if %sysfunc(fexist(ao))
          %then %put Die Datei &n existiert;
          %else %put Die Datei &n existiert nicht;
%mend;
```

Die Filename-Anweisung kann erst nach dem Daten-Schritt ausgeführt werden. Also darf auch der Aufruf der Funktion Fexist() erst nach dem Datenschritt ausgeführt werden.

Literatur

[1] SAS Language Guide, ab Version 8, Beschreibung von call execute()

3 Die Funktion Dcreate()

(Carina Ortseifen)

Als gute Alternative zum X-Kommando `x "md dirname"` kann ab Version 9 im SAS System im Datenschritt die Funktion Dcreate() verwendet werden, um einen neuen Ordner anzulegen.
Vorteil: Die Funktion gibt über einen Rückgabewert Auskunft, ob der Ordner angelegt werden konnte oder ob dabei ein Fehler aufgetreten ist.

Die **Syntax** der Funktion lautet:

```
rc=DCREATE(dirname, path);
```

wobei

`dirname`	der Name des neuen Ordners (ohne Pfadangabe) enthält
`path`	den Namen des Überordners, unter dem dirname angelegt wird
`rc`	Rückgabewert oder Returncode Konnte der Ordner angelegt werden, enthält dieser den Namen des Ordners, ansonsten ist es ein leerer Textstring.

Beispiel 1: Ordner test im Hauptordner von Laufwerk C. anlegen

```
DATA _NULL_;
   rc=DCREATE("test","c:\");
   PUT rc=;
RUN;
```

Im Log-Fenster erscheint daraufhin

`rc=c:\test`	wenn der Ordner angelegt wurde bzw.
`rc=`	wenn c:\test bereits existiert oder Laufwerk c: nicht existiert:

Beispiel 2: Ordner test unter dem “Work”-Ordner anlegen

```
DATA _NULL_;
   rc=DCREATE("test",PATHNAME("work");
```

```
   PUT rc=;
RUN;
```

Und im Log-Fenster erscheint:

```
rc=C:\DOKUME~1\x16.AD\LOKALE~1\Temp\SAS Temporary Files\_TD3356\test
```

4 Neues von Proc Means

(Grischa Pfister)

Seit Version 8 unterstützt Proc Means verschiedene neue Anweisungen, die den Umgang mit der Prozedur wesentlich bequemer machen. Für die Klassenvariablen können verschiedene Sortierungen angegeben werden, außerdem können – wie im Beitrag von Herr Altenburg beschrieben – mit Hilfe der Type- und der Ways-Anweisung bestimmte Kombinationen der Klassenvariablen gezielt ausgewählt werden.
Hier ein Beispiel für die Verwendung mehrerer Class-Anweisungen:

```
Proc Means data=Sashelp.Prdsale;
  Class country ;
  Class region / descending;
  Class prodtype / descending;
  Class product;
  Var actual;
Run;
```

Die verschiedenen Class-Anweisungen sorgen dafür, dass die Ausprägungen der Variablen auf- bzw. absteigend sortiert werden.

Die Output-Anweisung unterstützt jetzt das Erstellen sogenannter „Top/Bottom N"-Berichte, also die Selektion der besten oder schlechtesten N Beobachtungen aus einer Tabelle. Dabei dienen bestimmte Variablen als Selektionskriterium. Ausgegeben werden die gleichen oder andere Spalten. So kann z.B. die Tabelle SASHELP.CLASS herangezogen werden, um die drei schwersten Schüler bzw. Schülerinnen zu ermitteln, deren Namen und Geschlecht dann in eine Ergebnistabelle gespeichert werden. Das Programm sieht dann so aus:

```
Proc Summary data=Sashelp.Class;
  Output out=Work.Test
    idgroup( max(weight) out[3] (name sex)= );
  ;
Run;
```

Die Idgrop-Option der Output-Anweisung steuert die Gesamtausgabe, `max(weight)` enthält das Selektionskriterium, `out[3]` gibt an, dass die ersten drei Werte gespeichert werden sollen und `(name sex)=` zeigt an, dass die Spalten Name und Sex in die Ergeb-

nistabelle übernommen werden sollen. Die Ergebnis-Tabelle enthält eine Beobachtung und die Spalten Name_1, Name_2, Name_3 sowie Sex_1, Sex_2 und Sex_3. Die Rangnummer wird dabei an den Spaltenbezeichner (= Idgroup) angehängt.
Wird eine Class-Anweisung dazu genommen, enthält die Ergebnistabelle je Ausprägung der Class-Variable(n) eine Beobachtung. Mit Ways oder Types können wieder bestimmte Kombinationen selektiert werden.

```
Proc Summary data=Sashelp.Class;
  Output out=Work.Test
    idgroup( max(weight) out[3] (name sex)= );
  ;
  Class sex;
Ways 1;
Run;
```

Neu in Version 8 ist auch die Option Chartype im Prozeduraufruf.Sie sorgt dafür, dass die _Type_ Variable als alphanumerischer String ausgegeben wird statt als numerische Spalte. Die genaue Bedeutung der Spalte erläutert Herr Altenburg im nächsten Beitrag.

Literatur

[1] SAS Procedure Guide, ab Version 8, Beschreibung von Proc Means

5 Proc Means und der Umgang mit der _Type_-Variablen in der Output-Anweisung

(Hans-Peter Altenburg)

Die Speicherung von Kennzahlen in einer SAS-Datei unter Verwendung der Output-Anweisung in der Prozedur Means ist allgemein üblich. Vorsicht ist geboten, wenn an Stelle der By-Anweisung die Class-Anweisung verwendet wird. Verwendet man die By-Anweisung, so enthält die Output-Datei die gewünschten Kennzahlen für jede By-Kombination sowie die beiden Variablen _Freq_ (für die Anzahl der jeweiligen Kombination) und _Type_ mit dem Wert Null. Der Nachteil ist, dass die By-Anweisung eine sortierte Datei erwartet, was bei größeren Dateien für jede By-Kombination einen Aufruf zum Sortieren und einen Aufruf der Prozedur Means erfordert. Dagegen entfällt bei Verwendung der Class-Anweisung das vorherige Sortieren, und man braucht auch nur einmal die Prozedur Means aufzurufen um alle „Unter-„Kombinationen der Class-Liste zu erhalten. Die Output-Datei enthält jetzt die gewünschten Kennzahlen für alle möglichen Kombinationen der Class-Liste sowie die Class-Variablen, die Variable _Freq_ für die Anzahl dieser Kombination sowie die _Type_-Variable, die in binär verschlüsselter Form den Subset-Typ der Class-Liste enthält.

Die Ausgabe-Datei enthält eine „vollständige Analyse“ (d.h. alle „Kombinationen“ bzgl. der Class-Variablenliste) der numerischen Analysevariablen. Bei zwei Class-Variablen kann die Variable _Type_ etwa folgende Ausprägungen annehmen:

Type =0: Kennzahlen ohne Class-Variablen-Stratifizierung
Type =1: Kennzahlen stratifiziert nach der ersten Class-Variablen
Type =2: Kennzahlen stratifiziert nach der zweiten Class-Variablen
Type =3: Kennzahlen stratifiziert nach allen Class-Variablen

Bei einer größeren Anzahl von Variablen in der Class-Liste gibt es entsprechend mehr Ausprägungen für die _Type_-Variable. Der Vorteil liegt darin, dass bei großen Datenmengen die Prozedur Means nicht mehrmals aufgeführt werden muss, wenn bestimmte Unterkombinationen auch gesucht sind. Bei mehreren Class-Variablen kann die gewünschte Kombination über eine binäre Abfrage der _Type_-Variablen ausgewählt werden.

Beispiel mit vier Stratifizierungsvariablen Eins, Zwei, Drei, Vier

	TYPE-Variable	
Kombination:	**Binär**	**Dezimal**
0 (keine)	0000	0
Nur Eins	1000	8
Nur Vier	0001	1
Eins und Drei	1010	10 (=8+2)
Zwei und Vier	0101	5(=5+1)
…		
Alle	1111	15 (=8+4+2+1)

Mit Hilfe der Formatanweisung

```
Format _Type_ Binary4. ;
```

lassen sich die oben genannten Binär-Kombinationen darstellen. Den gleichen Effekt erreicht man, wenn man in der Proc Means-Anweisung die Option Chartype verwendet. Damit kann man auch in einer Where-Anweisung eine bestimmte Binär-Kombination abfragen.

Alternativen

Bestimmte Kombinationen der Class-Variablen auswählen kann man auch mit Hilfe der Types-Anweisung.

```
PROC MEANS DATA= ... ;
   CLASS eins zwei drei vier ;
   TYPES eins zwei drei vier ;
...
```

liefert die Kennzahlen für die einzelnen vier Statifizierungsvariablen. Types erlaubt auch andere Schreibweisen wie

```
TYPES eins*( zwei drei vier) ;
```

anstelle

```
TYPES eins*zwei eins*drei eins*vier ;
```

Eine weitere Alternative ist die Ways-Anweisung, um bestimmte Kombinationstiefen auszuwählen. So liefert z.B.

```
WAYS 2 ;
```

alle zweier Kombinationen,

```
TYPES eins*zwei eins*drei eins*vier
      zwei*drei zwei*vier
      drei*vier ;
```

oder

```
WAYS 3 ;
```

liefert alle Dreierkombinationen:

```
TYPES eins*zwei*drei eins*zwei*vier eins*drei*vier;
```

6 Die Option Fuzz bei Proc Format oder Vorsicht bei der Gruppierung von (ungerundeten) Werten mit Proc Format

(Tanja Petrowitsch)

Eine oft gestellte Frage ist die nach der Häufigkeit einzelner Werte gruppiert nach bestimmten Kategorien. Die Werte werden meist aus vorhandenen Variablen berechnet, etwa durch Differenzbildung zweier Variablen. Die Kategorien werden durch eine Formatzuweisung gebildet.

Bei Dezimalzahlen führt dieses Vorgehen jedoch zu ‚unerwarteten' Ergebnissen. Diese sind auf die unpräzise Darstellung von Dezimalwerten in SAS zurückzuführen. Z. B. ist die Differenz von 2.3-1.8 SAS-intern nicht 0.5, sondern 0.49999999.

Endet ein Formatintervall dann bei <0.5 und beginnt das nächste bei >= 0.5,

```
Proc Format;
   Value fabs 0.4 - < 0.5 = ' >=0.4 - <0.5 '
              0.5 - < 0.6 = ' >=0.5 - <0.6 ';
Run;
```

wird der Wert nicht wie erwartet zur 2. Kategorie gezählt, sondern zur ersten. Um die Abweichungen vom erwarteten Wert zu sehen, reicht bei diesem Beispiel die Darstellung mit dem Format Best32. aus. (Beispiele siehe Tabelle 1).

Tabelle 1: Kategorisierte Differenz der Variablen X und Y

X (Best32.)	Y (Best32.)	ABSDIFF	ABSDIFF (Best32.)	Put(absdiff, fabs.)	
1.7	1.21	0.49	0.49	>=0.4 - <0.5	✓
0.84	1.33	0.49	0.49	>=0.4 - <0.5	✓
1.82091	1.32327	0.49764	0.49764	>=0.4 - <0.5	✓
2.3	1.8	0.5	0.4999	>=0.4 - <0.5	☝
0.9	1.4	0.5	0.5	>=0.4 - <0.5	?
1.6	1.1	0.5	0.5	>=0.5 - <0.6	✓
1.1	0.6	0.5	0.5	>=0.4 - <0.5	?
2.2	1.7	0.5	0.5	>=0.4 - <0.5	?
0.5	0	0.5	0.5	>=0.5 - <0.6	✓
1.23	1.74	0.51	0.51	>=0.5 - <0.6	✓

Bei anderen Werten erhält man erst einen Hinweis, wenn man vom Ergebnis den erwarteten Wert subtrahiert (siehe Tabelle2).

Tabelle2: Kategorisierte Differenz der Variablen X und Y ohne und mit Fuzz-Faktor

Y	X	ABSDIFF – 0.5 (Best32.)	Put (absdiff, fabs_fu.)		Put (absdiff, fabs_fx.)	
2.3	1.8	-2.2204460492503E-16	>=0.4 - <0.5	☝	>=0.5 - <0.6	✓
0.9	1.4	-1.1102230246251E-16	>=0.4 - <0.5	☝	>=0.5 - <0.6	✓
1.6	1.1	0	>=0.5 - <0.6	✓	>=0.5 - <0.6	✓
1.1	0.6	1.1102230246251E-16	>=0.5 - <0.6	✓	>=0.5 - <0.6	✓
2.2	1.7	2.2204460492503E-16	>=0.5 - <0.6	✓	>=0.5 - <0.6	✓
0.5	0	0	>=0.5 - <0.6	✓	>=0.5 - <0.6	✓

mit

```
Proc Format;
   Value fabs_fu (Fuzz=0) 0.4 - < 0.5  = ' >=0.4 - <0.5 '
                          0.5 - < 0.6  = ' >=0.5 - <0.6 ';
   Value fabs_fx (Fuzz=0.099999)
                          0.4 - < 0.5  = ' >=0.4 - <0.5 '
                          0.5 - < 0.6  = ' >=0.5 - <0.6 ';
Run;
```

Abhilfe für die falsche Kategorisierung schafft hier die Option Fuzz der Prozedur Format. Der Faktor gibt an, um welchen Wert das Ende des Wertebereichs erweitert wird, mit dem der Wert noch eine Zuweisung findet. Der Default ist 1E-12 für numerische und 0 für Textformate.

Wenn ein Wert sowohl in einen Eintrag (Wert oder Range) eines Formats ohne als auch mit Fuzz-Faktor fällt, so wird der Wert dem Format-Eintrag ohne Fuzz-Faktor zugewiesen. (Weitere Details sind in der SAS-Hilfe zu finden). Um eine korrekte Zählung der Werte zu erreichen, muss daher der Fuzz-Faktor entsprechend den Formatgrenzen angepasst werden.
Da aber die interne Darstellung von Dezimalstellen nicht gerade sehr übersichtlich ist, kann es durchaus aufwendig sein, den geeigneten Faktor zu finden.
Auch die Einteilung der Werte in Kategorien mittels Datenschritt schützt nicht vor der Fehlzuteilung dieser unpräzisen Werte. Abhilfe schafft da entweder die Fuzz-Funktion oder (besser) das Runden vor der Kategorisierung. Und sollte hier mal ein ‚echter' Wert durch das Runden in der falschen Kategorie landen, dann weiß man wenigstens warum.

7 ODS Tagset ExcelXP

(Grischa Pfister)

Mit dem Tagset ExcelXP stellt SAS eine weitere Schnittstelle in die Excel-Welt bereit, diesmal schreibt SAS direkt das von Microsoft vorgegebene Excel-spezifische XML-Format. Dabei bietet der Tagset folgende Besonderheiten:

- Benennung der Tabellenblätter
- Je SAS-Tabelle ein Blatt in Excel bzw. je By-Gruppe, Prozedur,…
- Überschriften (Spalten und Zeilen) fixieren
- Automatische Spaltenfilter
- Formatierung mit Excel-Formaten
- Verwendung von Excel-Formeln
- Und viele mehr…

Der Aufruf des Tagsets folgt den normalen Konventionen des ODS, um den eigentlichen Code wird eine Klammer aus zwei ODS Anweisungen gesetzt, die erste öffnet den ODS Kanal, die zweite schließt ihn wieder.

```
Ods Tagsets.ExcelXP file=„path" options(optionen);
…
Ods Tagsets.ExcelXP Close;
```

In den Tagset ist die vollständige Dokumentation gleich mit eingearbeitet, durch den Aufruf des folgenden Programms wird sie in das LOG ausgegeben.

```
Ods Tagsets.ExcelXP file="ExcelXP001.xls" options(doc="help");
Ods Tagsets.ExcelXP close;
```

Der Tagset wird über Optionen gesteuert, die per ODS Anweisung gesetzt werden, zum einen beim Öffnen des Kanals, aber auch vor einer individuellen Tabelle:

```
Ods Tagsets.ExcelXP file=„path“ options(optionen);
…
Ods Tagsets.ExcelXP options(optionen);
…
Ods Tagsets.ExcelXP Close;
```

Einfache Optionen sind z.B. die Seitenausrichtung (portrait=Hochformat oder landscape=Querformat) und der Zoom-Faktor für die Ansicht.

```
Ods Tagsets.ExcelXP file="&root/excelXP002.xls"
  options(orientation="portrait" zoom="100")
;

Proc Print data = Sashelp.Prdsale(obs=50);
Run;

Ods Tagsets.ExcelXP options(orientation="landscape" zoom="200");
Proc Print data = Sashelp.class;
Run;

Ods Tagsets.ExcelXP Close;
```

Im Beispiel werden zwei Tabellen ausgegeben, die erste mit dem Zoom-Faktor 100 in Hochformat, die zweite mit dem Zoom-Faktor 200 in Querformat.

Die Optionen Frozen_Headers und Frozen_RowHeaders sorgen dafür, dass Spalten- und Zeilenüberschriften fixiert werden, d.h. wenn der Benutzer im Excel-Blatt scrollt, werden die Überschriften weiter angezeigt. Das macht für größere Ausgaben von Proc Print oder Proc Tabulate Sinn.
Excel unterstützt die Vorgabe von automatischen Spaltenfiltern, dabei wird zu der Spaltenüberschrift ein Kontextmenü hinzugefügt, welches das Sortieren und Filtern der Spaltenwerte erlaubt. Mit Hilfe der Option Autofilter kann dies aus SAS heraus gesteuert werden, allerdings nur für eine oder mehrere aufeinander folgende Spalten.

```
Ods Tagsets.ExcelXP file="&root/excelXP003.xls"
  options(frozen_headers="yes" frozen_rowHeaders="yes")
;

Proc Tabulate data = Sashelp.Prdsale;
  Class country region product;
  Var actual;
  Table country*Region, product*actual*sum;
Run;
```

```
Ods Tagsets.ExcelXP
  options(frozen_headers="no" frozen_rowHeaders="no"
          autoFilter="1-3")
;
Proc Print data = Sashelp.class noobs ;
  Var sex age name height weight;
Run;

Ods Tagsets.ExcelXP Close;
```

Das Verwenden von Excel-Formaten gestaltet sich etwas komplexer, da es nicht über Optionen des Tagsets geschaltet werden kann, sondern die Format-Information spaltenbezogen mit Hilfe eines Style-Attributes an den Tagset übergeben wird. Die Style-Anweisung hat die Form Tagattr="*Excel-Format*". Dabei kann das Excel-Format eine Anweisung sein, wie eine Ziffer darzustellen ist (0 als Platzhalter für signifikante Zahlen, # als Platzhalter für nichtsignifikante Zahlen), genauso gut können aber auch Farbkodierungen angegeben werden (in diesem Falle Format in Excel erstellen und dann im XML-Format speichern – daraus lässt sich ablesen, wie das Format in SAS angegeben werden muss. Hier ein paar Beispiele:

- `tagattr="format:000.00"`
- `tagattr="###.###.###.0#"`
- `tagattr="Red][<=100];[Blue][>100]"`

Die Angabe von `"format:"` vor dem eigentlichen Excel-Format ist nicht zwingend. Das letzte Beispiel enthält eine bedingte Formatierung, Werte kleiner/gleich 100 werden in roter Farbe dargestellt, solche größer 100 blau. Ein vollständiges Programm sieht dann so aus:

```
Ods Tagsets.ExcelXP file="&root.\ExcelXP004.xls";

Proc Print data = Sashelp.Class Noobs;
  Var name sex age;
  Var weight height / style(data)={tagattr="0000.00"};
Run;

Proc Print data = Sashelp.Class Noobs;
  Var name sex age;
  Var weight height/
    Style(data)={tagattr='format:[Red][&lt;=100];[Blue][&gt;100]'};
Run;

Ods Tagsets.ExcelXP Close;
```

Im ersten Falle erfolgt die Darstellung der Werte für Weight und Height mit führenden Nullen und zwei Nachkommastellen, bei dem zweiten Beispiel werden die Werte farblich unterschiedlich hervorgehoben.

Es gibt zwei Varianten, Excel-Formeln aus SAS heraus zu steuern. Zum einen gibt es die Möglichkeit, alphanumerische Spalten zu erzeugen, die Excel-Formeln enthalten. Dabei kann mit relativen Bezügen auf die Spalten und Zeilen des Excel-Arbeitsblattes verwiesen werden, die Syntax ist R[+/-n]C[+/-n]. Die Koordinate RC[-1] bezieht sich auf den Wert in der ersten Spalte nach links, RC[-2] auf den Wert in der zweiten Spalte nach links, beides in der gleichen Zeile. R[-1]C[-1] bezieht sich auf den Wert der eine Zeile weiter oben in der direkt links gelegenen Spalte enthalten ist.
Beispiel:

```
differenz = "=RC[-2] - RC[-1]";
```

Außerdem können Excel-Funktionen verwendet werden wie z.B. Abs(), Sum() etc., allerdings dürfen nur die englischsprachigen Funktions-Namen verwendet werden. Im Internet finden sich Übersetzungshilfen (z.B. unter http://www.wi.euv-frankfurt-o.de/lehre/wi/materialien/Deutsch-Englisch-Funktionen.xls), ein Speichern des Arbeitsblattes in XML-Format liefert ebenfalls die Übersetzung.

```
Summe = "=sum(RC[-2] : RC[-1])";
```

Das funktioniert aber nur mit denjenigen Funktionen, die keine XML-spezifischen Sonderzeichen wie doppelte Anführungszeichen, >, <, o.ä. enthalten. Wenn Formeln verwendet werden sollen, die solche Funktionen enthalten, muss die zweite Variante verwendet werden, die wieder das Style-Attribut Tagattr benutzt. Hier kann nämlich neben dem Format auch eine Formel hinterlegt werden. Aus der Excel-Formel

```
=WENN(E2 > 100; "gut";"schlecht")
```

wird im SAS-Code

```
tagattr='formula:=IF(RC[-1] &gt; 100,
         &quot;gut&quot;,&quot;schlecht&quot;)'};
```

Auch hier bietet es sich an, zunächst in einem Excel-Arbeitsblatt die Formel zu erstellen und dann das ganze in XML-Format zu speichern. Die resultierende XML-formatierte Formel kann dann in das Style-Angabe übernommen werden.
Das vollständige Beispiel sieht dann so aus:

```
Proc Sql;
  Create View Work.Differenz As
    Select country, product, actual, predict,
      . As Summe, . As Urteil
      From Sashelp.Prdsale(obs=10)
```

```
  ;
Quit;
```

Im ersten Schritt werden zwei leere Spalten hinzugefügt, für die später mit Hilfe des Style-Attributes eine Formel hinterlegt wird.

```
Ods Listing Close;
Ods Tagsets.ExcelXP
  file="&root.\ExcelXP006.xls"
;

Proc Print data = Work.Differenz Noobs;
  Var country product;
  Var actual predict / style = {cellwidth=3 cm};
  Var summe / style = {cellwidth=3 cm
                          tagattr='formula:sum(RC[-2],RC[-1])'};
  Var urteil / style = {cellwidth=3 cm
                        tagattr='formula:=IF(RC[-1] &gt; 1200,
                        &quot;gut&quot;,&quot;schlecht&quot;)'};
Run;

Ods Tagsets.ExcelXP Close;
```

Die Formel für die Spalte Summe definiert eine Summierung der beiden linken Spalten, die Spalte Urteil wertet ihrerseits die Summe aus und zeigt den Text „gut“ an, wenn der Wert größer 1200 ist, ansonsten den Text „schlecht“.

Literatur

[1] Dokumentation des Tagset ExcelXP

[2] Base-Community auf support.sas.com
http://support.sas.com/rnd/base/topics/odsmarkup/

8 Einrichten einer Projektumgebung

(Grischa Pfister)

Hintergrund dieses Tipps ist die allgegenwärtige Fragestellung, wie SAS-Sitzungen individuell angepasst werden, und wie dafür gesorgt wird, dass bestimmte Ressourcen (Bibliotheken, externe Dateien etc.) von Anfang an bereit stehen.
Im professionellen Einsatz von SAS wird dies z.B. für SAS/AF Anwendungen benötigt, die zwar ein „normales“ SAS als Basis haben, wo der Benutzer aber nicht mit den klassischen Fenstern PGM, LOG, OUTPUT arbeiten soll.

Grundsätzlich wird SAS über eine Vielzahl von (System) Optionen konfiguriert, ein Teil davon kann nur während der Startphase gesetzt werden – die eigentlichen System-

Optionen – ein anderer Teil wird in der laufenden SAS-Sitzung nach dem Schalter-Prinzip bzw. durch Wertzuweisung gesetzt. Eine Übersicht aller Optionen findet sich in der Online-Hilfe. Hier ist der schnellste Weg der Index, der nach Eingabe von „system options“ den Unterpunkt „by category“ anbietet.
Hilfe zu einer konkreten Option oder einer Gruppe von Optionen bietet auch die Prozedur Proc Options, die eine Definition ausgeben kann:

```
Proc Options option=mprint define;
Run;
Proc Options group=macro define;
Run;
```

System-Optionen werden auf zwei verschiedene Arten gesetzt. Zum einen können sie an den Aufruf von SAS angehängt werden, unter Windows sieht der typische Aufruf von SAS 9 z.B. so aus:

```
D:\sas9\sas.exe -CONFIG d:\Programme\SAS\sas9\nls\en\SASV9.CFG
```

Zum anderen – und das ist der bessere Weg, wenn mehrere Optionen gesetzt werden – kann eine benutzerspezifische CONFIG-Datei erstellt werden, in der die Optionen aufgeführt sind und auch dokumentiert werden können (mit der Kommentarform /*…*/).
Bis Version 8 musste dazu die vollständige Original CONFIG-Datei in Kopie übernommen werden, seit V9 ist auch das Ineinander-Schachteln verschiedener CONFIG-Dateien möglich. So kann das Original referenziert werden und anschließend folgen nur die Optionen, die vom Standard abweichen sollen.

Allokationsanweisungen für Bibliotheken und externe Dateien (sowie beliebiger anderer SAS Code) können in einer sog. AUTOEXEC.SAS zusammengefasst werden. Dieses Programm wird beim Starten von SAS nach abgeschlossener Initialisierung automatisch abgearbeitet. Dazu muss es sich entweder im Arbeitsverzeichnis von SAS befinden (Windows) oder per Option beim Aufruf bzw. in der CONFIG-Datei referenziert worden sein (System-Option Autoexec).

```
D:\sas9\sas.exe -CONFIG „…/config.cfg“ –AUTOEXEC „…/autoexec.sas“
```

Es folgt eine subjektive Auswahl verschiedener System-Optionen, die für die Einrichtung einer Projektumgebung interessant sein können:
(In der CONFIG-Datei werden die Optionen mit *–Option* angegeben)

- -Awstitle „text“
  gibt einen Text an, der als Titel des Hauptfensters verwendet wird
- -Splashloc „…/*.bmp“ referenziert eine Grafik die beim Start anstelle des SAS-Logos angezeigt wird
- -Awsdef llx lly urx ury
  gibt Größe und Position des SAS-Fensters bei Start in % des Bildschirms an
- -Nodsmexp
  Schaltet den Explorer und das Resultsfenster aus

- -Work
  Setzt den Pfad für die WORK-Bibliothek
- -Sasuser
  Setzt den Pfad für die SASUSER-Bibliothek
- -Rsasuser
  Erlaubt nur lesenden Zugriff auf die SASUSER-Bibliothek (readonly)

Es gibt verschiedene Möglichkeiten, Bibliotheken zu allokieren. Klassischerweise wird eine Libname-Anweisung verwendet, die z.B. in der AUTOEXEC.SAS enthalten ist.

```
Libname libref „Verzeichnis";
```

Genauso gut kann die Bibliothek aber bereits in der CONFIG-Datei definiert werden, hier ist die Syntax anders:

```
- -SET LIBREF „Verzeichnis"
```

Bei dem ersten Zugriff auf die Bibliothek *LIBREF* wird sie im Hintergrund automatisch allokiert, dabei entscheidet SAS selbständig, welcher Engine verwendet wird.

Eine wesentliche Rolle beim Aufsetzen von Projekten spielen die verschiedenen Projektumgebungen. In der Regel gibt es eine Entwicklungs-, eine Test- und eine Produktiv-Umgebung. Natürlich sollen diese Umgebungen verschiedene Ressourcen verwenden, andererseits soll der administrative Aufwand zum Anpassen so gering wie möglich gehalten werden. Die Verzeichnisstruktur der verschiedenen Umgebungen ist meistens identisch, d.h. die Bibliotheken beziehen sich auf gleich benannte Unterverzeichnisse, nur am Anfang des Pfades taucht die Unterscheidung in DEV, TEST, PROD auf. Mit Hilfe von relativen Bezügen oder Platzhaltern lässt sich der Pflegeaufwand minimieren.

Mit „-Sasinitialfolder" wird das Arbeitsverzeichnis einer SAS-Sitzung festgelegt. Relative Pfadangaben – also solche ohne „C:\..." oder „/…" am Anfang – beziehen sich immer auf diesen absoluten Pfad.

Das Einstellen von

```
-SASINITIALFOLDER "C:\projekt\daten"
```

in der CONFIG-Datei, gefolgt von diesen Libname-Anweisungen in der AUTOEXEC.SAS

```
Libname t1 "Oracle";
Libname t2 "Db2";
```

sorgt dafür, dass die Library T1 auf den Pfad „C:\projekt\daten\Oracle" zeigt, die Library T2 auf den Pfad „C:\projekt\daten\Db2".

Es gibt noch eine zweite Methode, relative Bezüge zu verwenden, dabei wird in der CONFIG-Datei mit „-SET *PlatzhalterName PlatzhalterWert*" ein Platzhalter definiert,

auf den dann in weiteren Anweisungen mit „!*PlatzhalterName*“ referenziert wird (in der CONFIG, in der AUTOEXEC, oder in beliebigen anderen SAS-Programmen). Das oben bereits verwendete Beispiel sieht dann so aus:
In der CONFIG-Datei:

```
-SET root "C:\projekt\daten"
```

In der AUTOEXEC.SAS

```
Libname t1 "!root\Oracle";
Libname t2 "!root\Db2"
```

In der AUTOEXEC.SAS können natürlich auch Makro-Variablen definiert werden, die Pfade oder Teilpfade enthalten und dann in entsprechenden Libname-Anweisungen Verwendung finden.

Hier noch einmal ein durchgehendes Beispiel für eine eigene CONFIG-Datei:

```
/* GP angepasste Config */

/* GP vorgelagerte Config einlesen */
-config "D:\Programme\SAS\SAS9\sasv9.cfg"

/* GP Arbeitsverzeichnis festlegen */
-SASINITIALFOLDER "D:\Eigene\iCASUS\02 Konferenzen\02 KSFE\Ksfe
2007\02 Tipps und Tricks\02 SAS-Umgebungen\GP"

/* GP Library Sasuser relativ zu SASINITIALFOLDER */
-SASUSER "sasuser"

/* GP Titel fuer Session vergeben */
-Awstitle "SAS Entwicklungsumgebung [GP]"

/* GP Explorer und Results-Fenster abschalten */
-NoDmsexp

/* GP Fenster auf maximale Groesse setzen */
-AWSDEF 0 0 100 100

/* GP Platzhalter mit Hauptpfad fuer Projekt */
-SET root "D:\Eigene\iCASUS\02 Konferenzen\02 KSFE\Ksfe 2007\02
Tipps und Tricks\02 SAS-Umgebungen\GP"

/* GP Startbild */
```

```
-splashloc "!root\SAS_Start.bmp"

/* GP Libraries allokieren */
-Set TempPath "d:/temp"
-Set daten1 "!root\daten"
```

Und hier die dazugehörende AUTOEXEC.SAS

```
/* GP angepasste Autoexec */

* GP Daten allokieren *;
Libname daten2 "daten";

Libname daten3 "../AndereDaten";

* GP Explorer einschalten
     --> Nur Results-Fenster bleibt geschlossen *;
DM "explorer";

Libname _all_ list;
```

Literatur

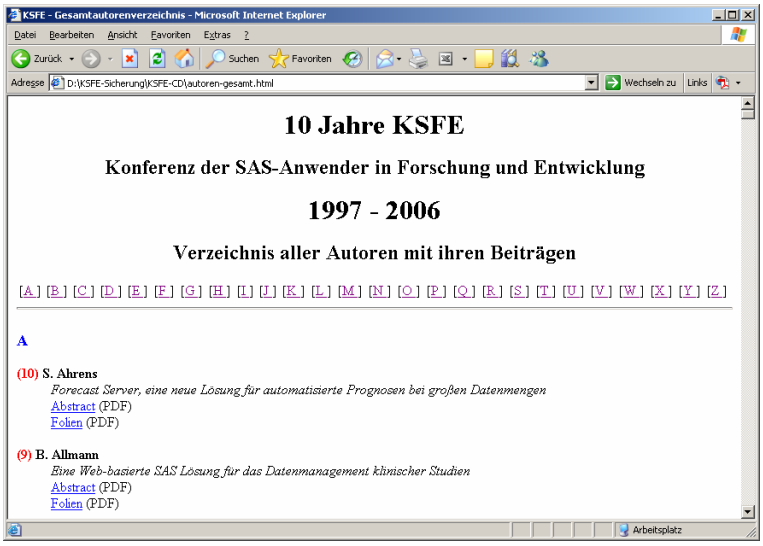
[1] SAS Online Hilfe zu (System-) Optionen

9 Auflösung der Publikumsfrage der 10. KSFE 2006 Hamburg

(Carina Ortseifen)

9.1 Fragestellung

Die Publikumsfrage entstand im Zusammenhang mit der Jubiläums-CD, die anlässlich der 10. KSFE 2006 in Hamburg verteilt wurde. Diese CD enthält alle Proceedings-beiträge der bisherigen 10 Konferenzen und ein Gesamtautorenverzeichnis, das mittels Sprungmarken zu den einzelnen Abschnitten verlinkt. Buchstaben wie X oder Y ohne entsprechende Autoren sollen dabei – im Unterschied zum nachfolgenden Screenshot - ausgeblendet werden.



Konkret wurde die folgende Aufgabe gestellt: Gegeben ist eine SAS-Tabelle mit einer Variablen tier, die 10 Beobachtungen mit den Werten Ziege, Affe, Gorilla, Schimpanse, Giraffe, Elefant, Seeelefant, Pinguin, Pony, Esel enthält.

Gesucht wird dazu eine Makrovariable, welche die

- Anfangsbuchstaben der Tiere
- ohne doppelte Buchstaben
- in alphabetischer Reihenfolge
- mit Komma getrennt
- in Anführungszeichen

enthält, also folgende Form hat: 'A','E','G','P','S','Z'

9.2 Lösungen

Innerhalb der gesetzten Frist reichten 19 SAS-Anwender zum Teil mehrere alternative Lösungsvarianten ein, wobei sehr unterschiedliche Methoden aus dem Spektrum der Möglichkeiten, die die SAS Software bietet, zum Einsatz kamen.

Rund 2/3 aller Lösungen basierten auf dem Einsatz der Prozedur SQL:

```
Proc Sql Noprint;
   Select Distinct "," || Substr(tier,1,1) || ","
   Into :_mvar Separated By ','
   From tiere;
Quit;
%Put &_mvar;
```

Aus der SAS Tabelle tiere werden mittels des Select-Befehls und der Option Distinct die unterschiedlichen Anfangsbuchstaben (Substr(tier, 1, 1) entnommen, in Anfüh-

rungszeichen gesetzt und mittels Into in eine Makrovariable, getrennt durch Komma gepackt.
Anstelle von Distinct hätte auch der Befehl Unique erscheinen können. Der zusätzliche Befehl Order, den manche noch einsetzten, war nicht notwendig.

Die übrigen Lösungsvarianten zeigten die vielfältigen Möglichkeiten der SAS Software auf:

- Ein oder mehrere Datenschritte mit einem Proc Sort-Schritt kombiniert
- Ein Datenschritt kombiniert mit einem Proc Sort-Schritt und einer Makroschleife
- Eine reine Makrolösung mit Verwendung der Funktionen Open, Close und Attrn
- Mehrere Datenschritte und Einsatz von Hash-Tabellen
- Ein einziger Datenschritt unter Zuhilfenahme des Formats Pib1.

Und weil mir persönlich die letzte Lösung besonders gut gefallen hat (vielleicht weil mir erst dadurch klar wurde, wozu das Format Pib1. gut sein kann), soll sie hier dargestellt werden. Alle anderen Lösungen finden Sie auf der Begleit-CD zum Tagungsband.

```
data _null_;
  set Tiere end=fertig;

  /* Arrays zur Aufnahme der Anfangsbuchstaben und Anzahl */
  array _Feld(255) _temporary_;
  array _Wert(255) $ _temporary_;

  format Ausgabe $200.;

  /* Initialisieren der Arrays */
  if _n_ eq 1 then do;
    do ik=1 to 255;
      _Feld(ik) = 0;
      _Wert(ik) = ' ';
    end;
  end;

  /* Das erste Zeichen (als Großbuchstabe) als Zahl übergeben */
  Pos = input(upcase(substr(Tier,1,1)),PIB1.);

put tier= pos=;
  /* In das entsprechende Array den Wert übertragen */
  _Feld(Pos) = sum(_Feld(Pos),1);
  _Wert(Pos) = upcase(substr(Tier,1,1));

  /* Nach dem letzten Satz Ausgabe in eine Makrovariable */
  if fertig then do;
```

```
    /* Durch alle Felder des Arrays laufen */
    do ik=1 to 255;
      /* Wenn etwas ungleich 0 dann wird das Zeichen übertragen */
      if _Feld(ik) then do;
        Ausgabe = compress(Ausgabe) !! "'"
                  !! compress(_Wert(ik)) !! "',";
      end;
    end;
    Ausgabe = substr(Ausgabe,1,length(Ausgabe) - 1);
    /* Makrovariable erzeugen */
    call symput("Ausgabe",Ausgabe);
  end;
run;

/* Zur Kontrolle die Makrovariable ausgeben */
%put;
%put &Ausgabe;
%put;
```

Allen Teilnehmer sei an dieser Stelle für ihre Beteiligung recht herzlich gedankt.
Und damit es auch im nächsten Jahr weitergeht, folgt hier die

9.3 Publikumsfrage der 11. KSFE 2007 Ulm

In einem Programmschritt (oder auch Makro) muss die SAS-Datei lib.tab erzeugt werden, etwa mit `DATA DATA=lib.tab;`

Damit dies ohne Komplikationen ablaufen kann, soll vorneweg geprüft werden, ob

a) der Bibliotheksname lib existiert und auf den richtigen Ordner zeigt und
b) die SAS-Datei tab in dieser Bibliothek noch nicht existiert.

Der richtige Ordnername für die Bibliothek steht dabei in der Makrovariable libpfad. Und sollte die SAS-Datei lib.tab bereits existieren, wird diese vor Neuanlegen umbenannt in lib.__tab.

Ihre Lösung(en) für diese Überprüfung können Sie bis zum 31. Dezember 2007 per Mail senden an: carina.ortseifen@urz.uni-heidelberg.de.

Auf der 12. KSFE 2008 in Aachen werden die eingereichten Lösungswege wie bisher auch im Rahmen der Tipps und Tricks-Sitzung vorgestellt.