Tipps & Tricks - Nützliche Lösungen zu Problemen und Fragestellungen aus dem Praxisalltag von SAS Programmierern in Form von Kurzvorträgen

Carina Ortseifen[1], Grischa Pfister[2], Heinrich Stürzl[3]

1: Universität Heidelberg, eMail: Carina.Ortseifen@urz.uni-heidelberg.de
2: Info-Ware Heidelberg, eMail: Grischa_Pfister@Info-Ware.de
3: Dade Behring Marburg, eMail: Heinrich_Stuerzl@DadeBehring.com

Abstract

Die im folgenden Beitrag vorgestellten Tipps und Tricks stellen keine neuen Features des SAS Systems vor. Stattdessen zeigen sie den Einsatz von Prozeduren, Anweisungen, Optionen und Makros als Lösung von Problemen und Fragestellungen, wie sie im Alltag von SAS-Programmierern und Anwendern häufig auftreten. Die einzelnen Tipps reichen vom Datenmanagement über den Umgang mit Datumsvariablen bis hin zum Erzeugen von HTML-Seiten für Dokumentationen im Intra- und Internet.

Ausblick auf die nächste KSFE

Aufgrund der vielen positiven Rückmeldungen auf diese erstmalig im Rahmen einer KSFE angebotenen Tipps&Tricks erscheint es wünschenswert, diese Veranstaltung auch für die nächste KSFE vorzusehen.
Dabei wäre es vielleicht sinnvoll, verschiedene Themenblöcke zu bilden, z.B. zu SAS/Base, SAS/AF, Makroprogrammierung, SAS/STAT usw.

Kennen Sie einen interessanten Tipp, den Sie gerne vorstellen möchten?
So wenden Sie sich bitte an Carina Ortseifen.

Themenübersicht

1. Data Set Optionen
2. Data Set Management mit der PROC DATASET und Passwortschutz
3. Anzahl der Observations einer SAS Tabelle (Data Set)
4. Nationale Datumsformate, Altersberechnung
5. Aktueller Zeitstempel im Titel
6. Permanente Formate
7. Dictionary Tables
8. Special Merge (one-to-many)
9. Textvariable in numerische Variable umwandeln
10. WMF- und BMP-Grafiken im Batch-Mode mit True Type Fonts (PROC GETFNT)
11. Dubletten-Check
12. Export im HTML-Format aus dem Datenschritt

1. Data Set Optionen

Grischa Pfister

Aufgabenstellung

Sie arbeiten mit Massendaten aus verschiedenen Quellen. Sie wollen sich einen Überblick verschaffen, indem Sie Subsets bilden und Sie möchten Teilanalysen mit bestimmten Variablen durchführen, dabei greifen Sie auf eine Vielzahl von Datenschritten zurück. Leider nehmen die Berechnungen deshalb relativ viel Zeit in Anspruch.

Lösung

Insbesondere der Umgang mit Massendaten erfordert den sinnvollen Einsatz von Daten- und Prozedur-Schritten in Bezug auf den zu leistenden I/O. Nach meiner Erfahrung kann durch den Rückgriff auf Data Set Optionen oft eine Vielzahl von Data Steps, die Zwischenschritte bei der eigentlichen Operation bedeuten, eingespart werden.

Data Set Optionen

Hier wird nur eine Auswahl aller vorhandenen Data Set Optionen vorgestellt, nämlich diejenigen, die sich auf dem Ein-/Ausschluß von Variablen und Beobachtungen beziehen.
KEEP, DROP, RENAME und WHERE sind den meisten Nutzern als Statements des Datenschrittes vertraut, FIRSTOBS und OBS sind aus dem OPTIONS-Statement bekannt.
Der große Vorteil von Data Set Optionen aber ist, dass sie für Daten- und Prozedurschritte Gültigkeit besitzen. Im Datenschritt können sie sich auf den Data Set beziehen, der gerade erzeugt wird, oder auf die Datei(en), die benutzt wird (werden), um einen neuen Data Set zu erstellen, also im SET- oder MERGE-Statement. Im Prozedurschritt greifen sie ebenfalls an zwei Stellen, nämlich bei der Referenz der Quelldatei und bei der Angabe einer eventuellen Ausgabedatei.

Syntax

Die Data Set Optionen werden in Klammern an den Namen des Data Sets angehängt.

```
libref.dataset ( optionen )
```

Dabei weicht die Syntax leicht von der der entsprechenden Statements ab. Auf KEEP, DROP, RENAME und WHERE folgt jeweils ein “=”, RENAME und WHERE benötigen eine zusätzliche Klammerung.

```
KEEP   = var1 <var2 ...>
DROP   = var1 <var2 ...>
RENAME = (var_alt = var_neu ...)
WHERE  = (where-clause)
FIRSTOBS | OBS  = n
```

Beispiel 1 - Umbennen automatisch erzeugter Variablen

In diesem Beispiel werden die Optionen benutzt, um die von SAS automatisch generierten Variablennamen COUNT, PERCENT und PCT_ROW umzubenennen.. Das kann z.B. nötig sein, wenn die Weiterverarbeitung durch Makros oder compilierte Datenschritte bestimmte Variablennamen vorschreibt. Die Variable PCT_COL wird nicht in die Ergebnisdatei übernommen.

```
PROC FREQ DATA = sashelp.prdsale NOPRINT;
    TABLE year * country /
      OUTPCT
      OUT = work.test
            (RENAME = (COUNT    = umsatz
                       PERCENT = prozent
                       PCT_ROW = pctZeile)
             DROP    = PCT_COL
            );
    WEIGHT actual;
  RUN;
```

Beispiel 2 - Ausschluss von Variablen

In Version 6.12 unterstützen Prozeduren weder KEEP- noch DROP-Statement. Das ist bedauerlich, denn oft werden von großen Datenbanken für eine Berechnung nur wenige Variablen benötigt, und zumeist müssen die Ausgangsdaten auch entsprechend sortiert werden. In vielen Beispielen wird zunächst mit einem Datenschritt die Teilmenge erstellt und anschließend sortiert; beides kann aber auch in einem Arbeitsschritt geschehen, wie das Beispiel zeigt.

```
PROC SORT
    DATA = sashelp.prdsale
           (KEEP = year country actual predict)
    OUT  = work.test;
    WHERE actual > predict * 10;
    BY year country;
  RUN;
```

Beispiel 3 - Mehrere Data Sets in einem Schritt

Dieses Beispiel soll den Einsatz von Data Set Optionen im Datenschritt zeigen. Abhängig von den Werten der Variable SALES werden die eingelesenen Beobachtungen entweder in den Data Set WORK.GUT oder in WORK.SCHLECHT gespeichert.

```
DATA
  gut      (where=(sales >= 500 ))
  schlecht (where=(sales <  500 ));
  INFILE 'c:/KSFE2000/GP/test.dat';
  INPUT sales date year month day 8;
  RUN;
```

[Anmerkung: Hier könnte natürlich auch ein konditionelles OUTPUT-Statement Verwendung finden: `IF sales >= 500 then OUTPUT gut; ELSE OUTPUT schlecht;`]

Beispiel 4 - Einschränken der Datenmenge für Tests

Hier wird gezeigt, wie über Data Set Optionen die Datenmenge begrenzt wird, was z.B. für die Fehlerkontrolle in komplexeren Programmen sehr hilfreich sein kann. Bitte beachten Sie, dass FIRSTOBS die *erste* einzulesende Beobachtung benennt und OBS die *letzte* einzulesende Beobachtung. OBS muss deshalb immer größer sein als FIRSTOBS!
Der Vorteil der Data Set Optionen gegenüber dem OPTIONS-Statement liegt in diesem Falle darin, dass sich die Beschränkung nur auf diese eine Prozedur bezieht, die globalen Optionen aber unberührt bleiben. Es ist bestimmt jedem schon passiert, dass er sich den Kopf über ungewöhnliche Ergebnisse zerbrochen hat, die darauf beruhten, dass wegen der Einstellungen für FIRSTOBS und/oder OBS nur ein Teil der Daten in die Berechnung einbezogen wurde.

```
PROC PRINT
  DATA = SASHELP.PRDSALES
           (firstobs = 100 obs = 199);
  VAR year actual predict;
RUN;
```

Literatur

SAS Institute Inc. (1990): SAS Language: Reference, Version 6, First Edition, Cary NC.

2. Data Management und Password- Schutz

Carina Ortseifen

Aufgabenstellung

Sie möchten sich schnell einen Überblick über neue SAS Tabellen verschaffen, Variablennamen, Variablentypen u.a.m. ablesen, weil etwa der Chef/Vorgesetzte für die Sitzung am Nachmittag dringend aktuelle Zahlen und Grafiken benötigt und Ihnen dazu lediglich eine Diskette mit den SAS Tabellen übergeben wird.

Lösung

Metainformationen zu SAS Tabellen können sowohl interaktiv als auch prozedural gewonnen werden. Die beiden interaktiven Methoden, Libraries- und Access-Fenster, bieten neben der Metainformation den Vorteil, dass sie auch die Daten selbst anzeigen können - mit dem kleinen Nachteil, dass der Vorgang für jede SAS Tabelle erneut durchgeführt werden muss. Die Prozedur DATASETS kann dagegen die Metainformationen ganzer SAS Bibliotheken präsentieren.

1. Das Libraries-Fenster wird durch Anklicken des nebenstehenden Symbols der Tools-Leiste geöffnet (oder über das Menü mit *Globals* ➔ *Access* ➔ *Display Libraries*). Mit dem kontext-sensitiven Menü der rechten Maustaste kann das Variablen-Fenster und das Viewtable geöffnet werden.

2. Das Access-Fenster wird mit dem Kommando *access* (oder *Globals* ➔ *Access* ➔ *Access Database files*) aufgerufen und über Kommandos in der Selektionsspalte bedient. C liefert die Variablendeklaration, B zeigt den Tabelleninhalt im Browse-Modus an, E im Editiermodus. Weitere Kommandos findet man in der Online-Hilfe.
3. Neben dem Anzeigen der Metainforamtionen kann die Prozedur DATASETS noch zahlreiche weitere nützliche Dinge fürs Data Management tun.

Die Prozedur DATASETS

Für die folgenden Beispiele wird vorausgesetzt, dass in der Bibliothek `sasuser` mehrere SAS Tabellen gespeichert sind, unter anderen die Tabellen `iris` und `fisher` mit den Variablen `kb`, `kl`, `bb`, `bl` sowie die Tabelle `rauch`.

```
PROC DATASETS LIB=sasuser;
QUIT;
```

erzeugt eine Liste aller SAS Tabellen und Kataloge in der Bibliothek `sasuser`. Der Prozedurschritt wird mit der Anweisung QUIT; beendet. RUN; wie in den folgenden Beispielen schließt lediglich eine Reihe von Anweisungen ab, ohne die Prozedur gänzlich zu beenden.

```
PROC DATASETS LIB=sasuser;
   CONTENTS DATA=iris;
   RUN;
   CONTENTS DATA=iris OUT=work.c_iris;
   RUN;
```

Die Anweisung CONTENTS zeigt die Variablendeklaration der SAS Tabelle `iris` an. Mit der zusätzlichen Option OUT= wird diese Metainformation in die Tabelle `work.c_iris` überführt.

```
PROC DATASETS LIB=sasuser NOLIST;
   CONTENTS DATA=iris OUT=work.c_iris NOPRINT;
   RUN;
```

Die Optionen NOLIST und NOPRINT unterdrücken die Listen, wenn etwa nur die neue Tabelle `work.c_iris` erzeugt werden soll.

```
   CHANGE iris=blume;
   RUN;
```

Die Anweisung CHANGE benennt die Tabelle `iris` in `blume` um. Alternativ kann die SAS Tabelle auch über das Betriebssystem umbenannt werden (Windows Explorer, Kommandos copy oder rename), was in der alten Version 6.04 nicht ging.

```
   EXCHANGE fisher=rauch;
   RUN;
```

tauscht die Namen der beiden Tabellen `fisher` und `rauch`, d.h. die Tabelle, die ursprünglich `fisher` hieß, heißt jetzt `rauch` und umgekehrt.

Mit der Anweisung MODIFY können Variableneigenschaften geändert werden.

```
   MODIFY fisher;
      LABEL kb="Blütenkelch-Breite"
            kl="Blütenkelch-Länge";
      RENAME bb=blbreite;
   RUN;
```

Die Änderungen betreffen die Tabelle `fisher`. Bei zwei Variablen wird das Label geändert bzw. gesetzt. Die Variable `bb` wird umbenannt in `blbreite`. Analoge Anweisungen liegen vor, um Format und Informat zu ändern.

Die Anweisungen LABEL, RENAME usw. könnten mit der gleichen Syntax auch in einem Datenschritt verwendet werden. Die gleichen Veränderungen können auch interaktiv mit dem Libraries-Fenster durchgeführt werden (Var-Fenster und Rename über das rechte Mausmenü). Über das Access-Fenster können dagegen nur die Formate und Informate verändert werden.

```
   DELETE rauch;
   RUN;
```

löscht die SAS Tabelle `rauch` aus der Bibliothek. Um alle SAS Tabellen und Kataloge einer Bibliothek zu entfernen, verwendet man die Option KILL. (Der Ordner bleibt dabei erhalten.)

```
PROC DATASETS LIB=sasuser KILL;
QUIT;
```

Verknüpfen von SAS Tabellen

Zum Thema "Verknüpfen von SAS Tabellen" zeige ich in meinen Kursen aus didaktischen Gründen zunächst die beiden Varianten "untereinander" und "nebeneinander" anhand eines Datenschritts mit SET und MERGE, um die Unterschiede im Ergebnis und die Parallelität der Syntax deutlich zu machen. Die Variante mit SET hat dann beispielsweise folgende Form:

```
DATA gesamt;
   SET tab1 tab2;
RUN;
```

Möchte man nun aber wenige Beobachtungen an einen großen Datenbestand anhängen, etwa Umsatzzahlen des vergangenen Monat März an die Jahresliste, dann ist der folgende Datenschritt

```
DATA jahr;
   SET jahr monat3;
RUN;
```

sehr ineffizient, da jede Beobachtung der beiden Ausgangstabellen `jahr` und `monat3` neu eingelesen werden muss. Als bessere, weil schnellere Variante kann die Prozedur DATASETS mit der Anweisung APPEND (oder alternativ die Prozedur APPEND) eingesetzt werden.

```
PROC DATASETS LIB=work;
   APPEND BASE=jahr DATA=monat3;
RUN;
```

Hier ein kleiner Vergleich der erforderlichen CPU-Zeiten:

SAS-Tabelle mit	Datenschritt mit SET	PROC DATASETS
2 Variablen	0.99 Sek.	0.34 Sek.
4 Variablen	2.42 Sek.	0.35 Sek.

Voraussetzung für die Anweisung APPEND ist allerdings, dass in beiden Tabellen identische Variablenstrukturen vorliegen. Ist dies nicht der Fall, wird der Prozedurschritt mit einer ERROR-Meldung abgebrochen.

```
PROC DATASETS LIB=work FORCE;
   APPEND BASE=jahr DATA=monat3;
RUN;
```

Die Option FORCE kann dabei soweit Abhilfe schaffen, dass die Struktur der Basistabelle als Referenz angesehen wird und nur die passenden Variablen der hinzuzufügenden Tabelle übernommen werden. Eine sorgfältige Kontrolle des Ergebnisses ist dabei anzuraten.

SAS Tabellen kopieren

Zur Erzeugung von Transportfiles mit der Engine XPORT oder zum Kopieren von SAS Tabellen in andere Bibliotheken wird die Prozedur DATASETS mit der Anweisung COPY (oder die Prozedur COPY) verwendet.

```
PROC DATASETS LIB=work;
   COPY OUT=sasuser MT=DATA;
      SELECT fisher;
QUIT;
```

Die zugehörige Syntax ist leider nicht sehr intuitiv und, der weit größere Nachteil: Man kann keine Kopie einer SAS Tabelle in der gleichen Bibliothek anlegen, u.a. weil man keine Möglichkeit hat, einen anderen Tabellennamen zu wählen. Folgende Auswege bieten sich an:

1. Man kopiert über das Betriebssystem (Windows-Explorer, Kommando copy), oder man verwendet die
2. Anweisung APPEND (und nutzt die Tatsache aus, dass die Basistabelle neu angelegt wird, wenn sie nicht existiert):

```
PROC DATASETS LIB=work;
   APPEND BASE=neu DATA=fisher;
QUIT;
```

   Oder, die vielleicht naheliegendste, aber nicht sonderlich effiziente Lösung:
3. Der Datenschritt:

```
DATA neu;
   SET fisher;
RUN;
```

Passwort-Schutz

Eine SAS Tabelle enthält die Daten einer umfangreichen Befragung. Innerhalb eines größeren Projekts müssen mehrere Gruppen mit dieser Tabelle arbeiten, die auf Grund ihrer Größe (mehrere 100 MB) zentral auf einem Server abgelegt wird. Da noch nicht alle Mitarbeiter sehr fit im Umgang mit SAS sind, soll die Tabelle vor dem versehentlichen Löschen oder Ändern geschützt werden.
Eine geschickte Lösung des Problems mit SAS Hilfsmitteln besteht in der Vergabe eines Passwortes für die SAS Tabelle.

Das SAS System kennt drei Arten von Passworten:

READ=	für den Lesezugriff
WRITE=	für den Schreibzugriff (beinhaltet Ändern)
ALTER=	für das Löschen und Umbenennen
PW=	schließt alle drei Rechte ein

Eine Möglichkeit, diese Passworte zu definieren, ist – wie sollte es anders sein – die Anweisung MODIFY der Prozedur DATASETS.

```
PROC DATASETS LIB=work;
   MODIFY fisher (WRITE=stop ALTER=stop);
QUIT;
```

Der Tabelle `fisher` wird ein Schreib- und Lösch-Passwort zugewiesen. Das bedeutet, dass der Anwender die Tabelle z.B. mit der Prozedur PRINT ausdrucken kann, aber sobald er versucht, die Tabelle zu ändern, fordert das SAS System das betreffende Kennwort. (Eine andere Möglichkeit für die Zuweisung von Passworten ist der Datenschritt mit den entsprechenden Data Set Optionen.)

Literatur:

SAS Procedures Language Reference, Version 6 Third Edition, S. 247 ff.
SAS Technical Report P-222: Changes and Enhancements to Base SAS Software, Release 6.07, S: 183ff

3. Wie viele Observations hat eine SAS Tabelle (Data Set)

Heinrich Stürzl

Aufgabenstellung

Die Anzahl der Beobachtungen soll als Zahl allgemein verfügbar sein z.B.
im Titel, im Data Step, für %IF-Abfragen, in %DO-Schleifen.

Lösung:

Mit Hilfe von %SYSFUNC lassen sich über die SCL-Funktion ATTRN aus dem Header der SAS Tabelle verschiedene Informationen auslesen, ohne die Datei selbst zu lesen, z.B.

- Anzahl aller Beobachtungen ('NOBS') inkl. der zum Löschen markierten Beobachtungen
- Anzahl der logischen Beobachtungen ('NLOBS') ohne die zum Löschen Markierten
- Anzahl der Variablen ('NVARS')

Nachfolgend der Quelltext einer Makrofunktion für die Rückgabe von NOBS.

```
%MACRO nobs (ds);
  %LOCAL dsid rc;

  %if %SYSFUNC(EXIST(&ds)) %THEN %DO;
    %LET dsid =%SYSFUNC(OPEN(&ds));
    %SYSFUNC(ATTRN(&dsid, NOBS))
    %LET rc=%SYSFUNC(CLOSE(&dsid));
  %END;
  %ELSE -1;

%MEND nobs;
```

Eigenschaften:

- Gibt die Anzahl aller Observations eines Data Set zurück (Ganzzahl ≥ 0)
- Als Text linksbündig formatiert
- "-1", falls kein Zugriff auf Datei möglich
- Makro ist sehr schnell und unabhängig von der Dateigröße
- Voraussetzung: SAS 6.11 TS020 oder höher (wegen %sysfunc)

Anwendungsbeispiele:

```
TITLE "n=%nobs(work.xyz)";

DATA new;
    n=%nobs(work.xyz)*2;
    ARRAY x [%nobs(work.xyz)];
RUN;

%DO i=1 %TO %nobs(work.xyz);
  ...
%END;
```

Literatur:

SAS Screen Control Language: Reference 6, Second Edition, S. 238 ff

4. Nationale Datumsformate und Altersberechnung

Carina Ortseifen

Aufgabenstellung

In einem Report erscheinen u.a. die Geburtstage der Teilnehmer. Diese Datumsangaben sollen in der für uns in Deutschland üblichen Form 01.10.1980 erscheinen und nicht als 01OCT1980 oder 01/10/1980. Weiterhin soll das Alter der Teilnehmer in Jahren zum Zeitpunkt der Befragung berechnet und ebenfalls dargestellt werden.

Bevor die Lösungen vorgestellt werden, erst ein paar

Hintergrundinformationen

Datumsangaben werden intern im SAS System als die Anzahl der Tage seit dem 1. Januar 1960 gespeichert.
Das hat den *Vorteil*, dass Datumsdifferenzen, wie z.B. die Dauer des Aufenthalts im Krankenhaus oder die Zeit bis zum Auftreten eines Ereignisses einfach durch Differenzbildung des Ende- und Anfangsdatums berechnet werden können. Die Funktionen für Datumsvariablen wie MDY und INTCK bieten dem Anwender weitere Unterstützung.
Diesen Vorteil erkauft man sich mit dem *Nachteil*, dass beim Einlesen oder Importieren die Datumsdarstellung in die Zahl der Tage verwandelt werden und für jede Darstellung wieder das Datum mit Tag, Monat und Jahrangabe umgerechnet werden muss. Diese Umrechnungen nehmen dem Anwender die Informate und Formate ab.

Im folgenden Beispiel werden die Geburtsdaten von vier Personen in `gebdat` erfasst:

```
DATA datum;
   INPUT gebdat DDMMYY10.;
   tagx="02MAR2000"D;
   tagy=MDY(3,2,2000);
   LINES;
01.01.1960
02.01.1960
01.01.1961
31.12.1959
RUN;
PROC PRINT DATA=datum;
   FORMAT gebdat tagx tagy DDMMYY10.;
RUN;
```

Das Einleseformat DDMMYY10. wird in der Anweisung INPUT definiert. Gleichzeitig werden zwei weitere Datumsvariablen `tagx` und `tagy` definiert, einmal durch das SAS Format DATE9, gekennzeichnet mit D am Ende und ein weiteres Mal mit der Funktion MDY. Im Prozedurschritt wird den Variablen das Format für die korrekte Darstellung zugewiesen.

Lösung

Durch Verwendung nationaler Datumsformate erhält man nun "schönere" Datumsangaben.

```
PROC PRINT DATA=datum;
   FORMAT gebdat tagx tagy DEUDFDD10.;
RUN;
```

```
OBS        GEBDAT          TAGX          TAGY

 1      01.01.1960    02.03.2000    02.03.2000
 2      02.01.1960    02.03.2000    02.03.2000
 3      01.01.1961    02.03.2000    02.03.2000
 4      31.12.1959    02.03.2000    02.03.2000
```

Die ersten drei Zeichen des Formatnamens bezeichnen die Nation: DEU für Deutsch, dann folgt DF für Datumsformat und schließlich DD für das gewöhnliche Datumsformat. Folgende Tabelle gibt eine Auswahl aus den vorhandenen nationalen Datumsformaten.

SAS Datumsformat	Nationales Datumsformat	Beispiel für die Darstellung
DATE.	DEUDFDE.	31Dez59
DATETIME.	DEUDFDT.	31Dez59:23:59:59
DDMMYY.	DEUDFDD.	31.12.59
MONYY.	DEUDFMY.	Dez59
WEEKDAY.	DEUDFDN.	4
DOWNAME.	DEUDFDWN.	Donnerstag
MONNAME.	DEUDFMN.	Dezember
WORDATX.	DEUDFWDX.	31. Dezember 1959
WEEKDATX.	DEUDFWKX.	Donnerstag, 31. Dezember 1959

Analog zu diesen deutschen Datumsformaten gibt es französische, italienische, spanische etc. Soll eine Publikation für eine spanische Zeitschrift vorbereitet werden, verwendet man anstelle von DEUDFDD10. einfach ESPDFDD10..

Muss die Sprache allerdings häufiger gewechselt werden, etwa weil ein Report mithilfe einer AF-Anwendung geschrieben wird, die einen Umschalter für Sprachen hat, dann verwendet man besser die allgemeineren Datumsformate EURDF... und greift zusätzlich auf die Systemoption DFLANG= zurück, die für die entsprechende Sprache sorgt.

```
OPTIONS DFLANG=english;
PROC PRINT DATA=datum;
   FORMAT gebdat tagx tagy EURDFDD10.;
RUN;
```

Literatur

Online Hilfe über Kommando help format

Berechnung des Alters

In engem Zusammenhang mit den Datumsangaben steht die Berechnung des Alters zu einem Stichtag in Jahren.

Eine einfache, aber nicht immer ganz genaue Schätzung liefert die Berechnung der Differenz des Geburtsdatums zum Stichtag geteilt durch 365 (Tage) oder auch 365.25, wenn man die Schaltjahre mitberücksichtigen möchte. Aber in bestimmten Situationen stimmt dies halt nicht genau.

Was braucht man zur Bestimmung des Alters? Zunächst die Differenz der Jahre. Liegt das Geburtsdatum nun vor dem Stichtag, hatte die Person im betreffenden Jahr also schon Geburtstag, dann stimmt die Differenz in Jahren. Liegt der Geburtstag aber nach dem Stichtag, dann ist die Person zum Stichtag ein Jahr jünger, d.h. man muss von der Differenz in Jahren ein Jahr abziehen.

Da es im SAS System keine Funktion gibt, um aus einer Datumsangabe die Anzahl der Tage zu bestimmen, die im Jahr verstrichen sind, sondern nur die Anzahl der Tage im jeweiligen Monat, Funktion DAY, müssen für die Altersberechnung die Anzahl der zwischen den beiden Datumsangaben liegenden Monate berechnet werden. Das macht die Funktion INTCK (Interval checking).

```
INTCK('MONTH',gebdat,tagx);
```

Für die beiden Datumsangaben 31. Januar 2000 und 28. Februar 2000 liefert die Funktion das gleiche Resultat wie für 31. Januar 2000 und 1. Februar 2000, nämlich 1, obwohl im zweiten Fall nur ein Tag verstrichen ist. Die Funktion geht stets vom 1. des Monats aus.

Falls der Tag des Geburtstags vor dem Tag des Stichtags, bezogen auf den Monat, liegt, muss man wieder einen Monat abziehen.

```
INTCK('MONTH',gebdat,tagx) - (DAY(tagx)<DAY(gebdat));
```

Der logische Ausdruck in der Klammer ist wahr, d.h. sein Resultat ist 1, wenn die Funktion INTCK einen Monat zu viel gezählt hat. Er ist falsch, d.h. resultiert im Wert 0, wenn die Monate korrekt sind.

Um die Anzahl der Monate nun in Jahre umzurechnen, muss der Wert schließlich noch durch 12 dividiert werden und mit der Funktion FLOOR auf die nächste ganze Zahl abgerundet werden:

```
FLOOR(
   (INTCK('MONTH',gebdat,tagx) - (DAY(tagx)<DAY(gebdat)))
   /12 );
```

Diese Berechnung gilt für alle Datumsangaben nach dem 4.10.1582 (= Einführung des Gregorianischen Kalenders), wobei zu berücksichtigen ist, dass manche Länder erst später auf diese Zeitrechnung umgestellt haben. Eine Besonderheit sind die Geburtstage an Schalttagen (29. Februar). In Jahren ohne Schalttag werden diese dem 1. März gleichgestellt.

Literatur:

SAS Communications, 4 Q 1998, S. 38-39.

5. Aktueller Zeitstempel im Titel

Heinrich Stürzl

Aufgabenstellung

Der aktuelle Zeitpunkt der Erstellung eines Outputs soll im Titel oder der Fußzeile erscheinen.

Über die Optionen DATE und TIME sowie die automatischen Macrovariablen SYSDATE, **SYSDATE9** (ab 6.12 TS060), SYSDAY, SYSTIME wird der Startzeitpunkt von SAS festgehalten. Dies ist also während einer SAS Session eine Konstante, die auch das falsche Datum tragen kann, wenn der Rechner über Nacht läuft.

Lösung mit %SYSFUNC:

%SYSFUNC erlaubt das Ausführen der meisten Data-Step- und SCL-Funktionen an beliebiger Stelle, d. h. auch im Open Code und in der Macro Umgebung.
Syntax:

```
%SYSFUNC( funktion(argument(e) <, format> ) )
```

Dies hat auch den Vorteil, dass landesspezifische oder selbstdefinierte Formate verwendet werden können.

Beispiel 1:

```
TITLE1 "%SYSFUNC(DATE(), DEUDFWKX.)";
```

z.B. Donnerstag, 9. März 2000

Beispiel 2:

```
TITLE1 "%SYSFUNC(DATE(), DEUDFWKX.), %SYSFUNC(TIME(), TIME8.)";
```

z.B. Montag, 28. Februar 2000, 23:59:59

Hinweis:

Um jeden Output mit dem aktuellen Zeitstempel zu versehen, muß die entsprechende TITLE Anweisung vor jeder outputerzeugenden Prozedur wiederholt werden.

Literatur:

SAS Macro Language: Reference, First Edition, S. 240 ff
Index der Online Hilfe unter %sysfunc

6. Permanente Formate

Carina Ortseifen

Aufgabenstellung

Formate sollen permanent abgespeichert werden, damit zu einem späteren Zeitpunkt ohne erneuten Aufruf der Prozedur FORMAT auf sie zugegriffen werden kann. Für die Dokumentation sollen die Formate ausgedruckt werden.

Lösung

Formate lassen sich in permanenten SAS Katalogen speichern, wenn im Aufruf der Prozedur FORMAT die Option LIB= verwendet wird. Im folgenden Beispiel wird dieser Katalog in der SAS Bibliothek `libref` abgelegt. Standardmässig heißt der Katalog FORMATS.SC2. Grundsätzlich können Format-Einträge in jedem Katalog gespeichert werden. Für die Suche muß dann aber die Option FMTSEARCH verwendet werden.

```
LIBNAME libref 'C:\studien';
PROC FORMAT LIB=libref;
   VALUE janein 1=ja
                0=nein
                other=" ";
RUN;
```

Auf diese permanenten Formate kann später auf zwei Arten zugegriffen werden: Automatischer Suchpfad LIBRARY oder Option FMTSEARCH.

1. Existiert eine Bibliothek namens LIBRARY, dann sucht das SAS System darin automatisch nach den ausgewiesenen Formaten.

   ```
   LIBNAME LIBRARY 'C:\studien';
   PROC PRINT DATA=irgend.was;
      FORMAT f1-f10 janein.;
   RUN;
   ```

2. Die Option FMTSEARCH erlaubt, Formate in verschiedenen Katalogen in mehreren Ordnern zu suchen.
   Standardmässig gilt folgende Suchreihenfolge:
   ```
   OPTIONS FMTSEARCH=(work, library);
   ```

   Wenn nur ein Bibliothekname (libref) angegeben wird, wird automatisch als Katalogname `formats` angenommen, d. h. soweit nichts anderes spezifiert wird, sucht SAS zuerst nach einem Katalog `work.formats`, und falls das gesuchte Format darin nicht gefunden wird, nach einem Katalog `library.formats`.
   Über FMTSEARCH können aber auch andere Kataloge angegeben werden, in denen nach den entsprechenden Format-Einträgen gesucht wird.

   ```
   LIBNAME fmt 'C:\studien';
   OPTIONS FMTSEARCH=(work, fmt, irgend.katalog);
   PROC PRINT DATA=irgend.was;
      FORMAT f1-f10 janein.;
   RUN;
   ```

Für die Ausführung der Prozedur PRINT sucht das SAS System nach dem Format `janein` nun zunächst in der Bibliothek `work` im Katalog `formats`, dann im Katalog `fmt.formats` und schließlich dann im Katalog `irgend.katalog`, d.h. die Suchreihenfolge geht von links nach rechts, wobei das zuerst gefundene Format verwendet wird.

Was tun, wenn permanente Formate nicht greifbar sind?

Wurde einer Variablen ein permanentes eigenes Format zugewiesen und kann man auf dieses Format aus irgendeinem Grund nicht zugreifen, erscheint folgende Fehlermeldung:

```
1     PROC PRINT DATA=irgend.was;
2     RUN;
ERROR: Format JANEIN not found or couldn't be loaded for variable F1-F10.
NOTE: The SAS System stopped processing this step because of errors.
```

Hier kann man für die Prozedur PRINT das Format für die Variablen f1 bis f10 entfernen:

```
PROC PRINT DATA=irgend.was;
   FORMAT f1-f10 ;
RUN;
```

Bei vielen Variablen mit fehlenden Formaten kann dieses Verfahren sehr aufwendig sein. Und mit FORMAT _ALL_; werden u.U. auch SAS-eigene Formate, z.B. für Datumsvariablen, entfernt. Eine Alternative bietet die Option NOFMTERR.

```
OPTIONS NOFMTERR;
PROC PRINT DATA=irgend.was;
RUN;
```

Formate dokumentieren

Die beiden Optionen der Prozedur FORMAT, FMTLIB und CNTLOUT=, helfen bei der Dokumentation der Formatdefinitionen. FMTLIB erzeugt eine Tabelle der Definitionen im Ausgabefenster.

```
PROC FORMAT FMTLIB;
RUN;
```

FORMAT NAME: JANEIN LENGTH: 4 NUMBER OF VALUES: 3
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 4 FUZZ: STD

START	END	LABEL (VER. 6.12 26MAR00:15:05:06)
0	0	nein
1	1	ja
OTHER	**OTHER**	

Die Option CNTLOUT= legt eine SAS Tabelle, welche die gleichen Informationen enthält.

Literatur:

SAS Procedures Guide, Version 6, Third Edition, S. 275 ff.
Online Hilfe zur Option FMTSEARCH

7. Dictionary Tables

Grischa Pfister

Aufgabenstellung

Metadaten spielen eine Schlüsselrolle bei der Konzipierung und Implementierung größerer Projekte. SAS stellt in den Dictionary Tables sämtliche verfügbaren Metainformationen über alle Members aller allokierten Libraries bereit. Insbesondere AF-Entwickler besitzen damit ein wertvolles Werkzeug, aber auch SAS/BASE Anwender können die Dictionary Tables sinnvoll einsetzen.

Welche Tables gibt es?

Die eigentlichen Dictionary Tables sind Tabellen der Library DICTIONARY. Diese ist nur mit PROC SQL ansprechbar. Es existiert jedoch in der Library SASHELP eine Reihe von VIEWs auf diese Tabellen. Sie können sich schnell einen Überblick verschaffen, wenn Sie das Kommando “DIR sashelp.view” in der Kommandozeile absetzen.

```
SASHELP.VCATALG = select * from dictionary.catalogs
```
Alle Einträge in den Katalogen der allokierten Libraries mit Typ, Beschreibung und Datum der letzten Änderung.

```
SASHELP.VCOLUMN = select * from dictionary.columns
```
Alle Variablen in allen Views und Data Sets der allokierten Libraries mit Name, Typ, Länge, Label, Format, Informat und Indextyp.

```
SASHELP.VMACRO  = select * from dictionary.macros
```
Automatische und globale Makrovariablen mit Name und Wert.

```
SASHELP.VOPTION = select * from dictionary.options
```
Alle Systemoptionen mit Name, Einstellung und Beschreibung.

Daneben gibt es noch einige gefilterte VIEWs, die eine geringere Informationsdichte enthalten und deren Namen mit dem Kürzel “VS” (view short) beginnen:

```
SASHELP.VSCATLG
SASHELP.VSLIB
```

Diese Kurzversionen sind nötig, weil der Zugriff auf die VIEWs mittels Daten- oder Prozedurschritt deutlich langsamer ist als mit Hilfe der Prozedur SQL.

Anwendungsbeispiel 1 - Konsistenzchecks

Stellen Sie sich ein Projekt vor, in dem Daten aus unterschiedlichen Quellen zusammengeführt werden sollen. Eine der wichtigsten Fragen ist, ob inhaltlich gleiche Variablen auch gleich definiert wurden. Im folgenden Beispiel werden drei SAS-Tabellen mit einer Variablen SEX erzeugt, die jeweils unterschiedlich definiert ist. PROC PRINT und PROC SQL liefern das gleiche Ergebnis, wobei der SQL-Schritt eindeutig performanter ist.

```
*drei Data Sets mit unterschiedlichen Definitionen der Variable SEX;

DATA test01;
  LENGTH sex $ 10;
  INPUT sex @@;
  CARDS;
männlich weiblich männlich weiblich
RUN;

DATA test02;
  LENGTH sex 3;
  INPUT sex @@;
  CARDS;
1 2 1 2
RUN;

DATA test03;
  LENGTH sex $ 1;
  INPUT sex@@;
  CARDS;
m w m w
RUN;

* Überprüfung mit Hilfe des Views *;
PROC PRINT
  DATA=sashelp.vcolumn(WHERE=( libname='WORK' AND memtype='DATA' AND
  name='SEX') );
  VAR libname memname name type length;
run;

* Überprüfung mit Hilfe des Dictionary Tables *;
PROC SQL;
  CREATE VIEW tmp AS
    SELECT libname, memname, name, type, length
      FROM dictionary.columns
      WHERE libname = 'WORK' AND memtype='DATA' AND name='SEX';
QUIT;

PROC PRINT data=tmp;
RUN;
```

Ausgabe im OUTPUT-Fenster (SQL-View)

OBS	LIBNAME	MEMNAME	NAME	TYPE	LENGTH
1	WORK	TEST01	SEX	char	10
2	WORK	TEST02	SEX	num	3
3	WORK	TEST03	SEX	char	1

Anwendungsbeispiel 2 – AF Auswahlfenster für SAS-Tabelle

In diesem Beispiel werden zwei Dictionary Tables für die interaktive Auswahl einer SAS-Tabelle benutzt, die dann in einem FSVIEW-Fenster angezeigt wird, und zwar entweder im BROWSE- oder im EDIT-Modus.
Die linke Listbox zeigt alle allokierten Libraries an, während sich der Inhalt der rechten Listbox abhängig von der Auswahl in der linken ändert. Über eine Radiobox kann zwischen BROWSE- und EDIT-Modus umgeschaltet werden. Mit Hilfe des Pushbuttons “OK” verlässt der Benutzer den Schirm und das FSVIEW-Fenster wird aufgerufen.

Eigenschaften der linken Listbox

Name: LB01
Title: Libraries
Fill type: SAS data set...: SASHELP.VSLIB
Variable in the data set: LIBNAME

Eigenschaften der rechten Listbox

Name: LB02
Title: Data Sets
Fill type: SAS data set id: DSID
Variable in the data set: MEMNAME

Eigenschaften der Radiobox

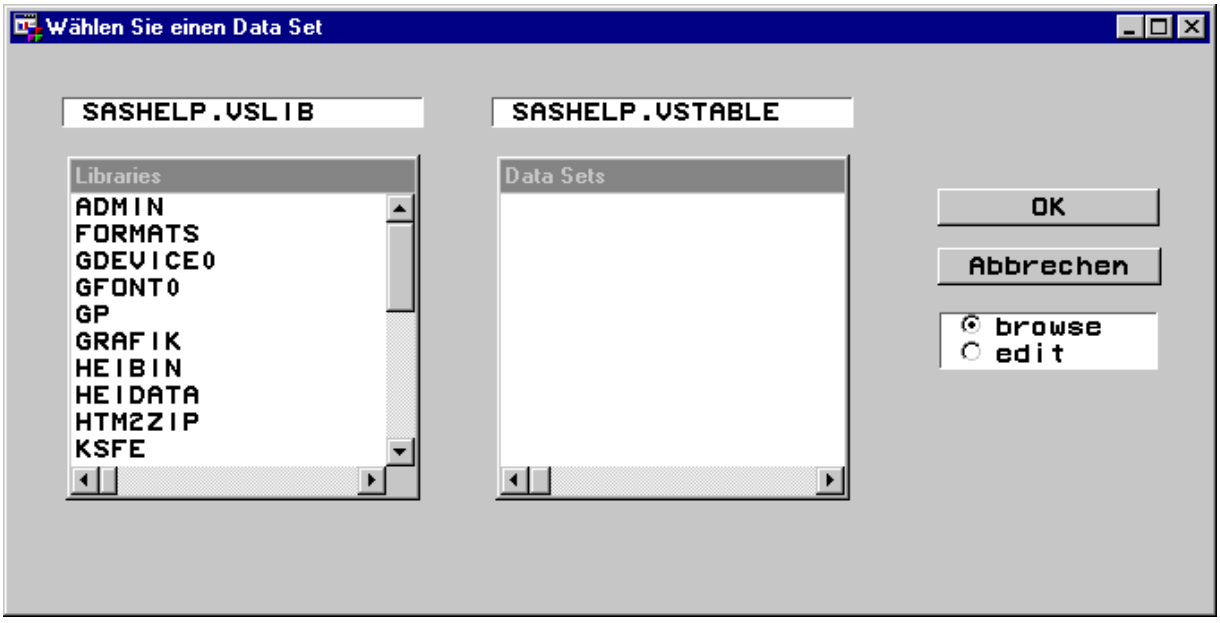
Name: RBMODE
Fill type: Enter Values: browse, edit

Eigenschaften des Pushbuttons „OK“ [command pushbutton]

Name: PBOK
Label: OK

Eigenschaften des Pushbuttons „Abbrechen“ [command pushbutton]

Name: PBCANCEL
Label: Abbrechen
Command: CANCEL



Im zugehörigen SCL-Programm wird im INIT-Label der VIEW SASHELP.VSTABLE geöffnet, die WHERE-Bedingung sorgt dafür, dass zunächst keine Data Sets angezeigt werden, denn in der linken Listbox ist noch keine Auswahl erfolgt.
Unter dem Label LB01 wird die Benutzerauswahl abgefragt und eine entsprechende WHERE-Bedingung auf den geöffneten View gesetzt. Damit werden in der rechten Listbox nur die SAS-Tabellen der gerade in der linken Listbox ausgewählten Library angezeigt.
Wählt der Benutzer den Pushbutton PBOK an, wird mit CALL FSVIEW die SAS-Tabelle im gewählten Modus (rbMode) geöffnet.

```
LENGTH
  nRow             8
  nIsSelected      8
  dsId             8
  cCurLib      $   8
  cCurData     $   8
  rc               8
;

INIT:

  dsId = open('SASHELP.VSTABLE');
  rc   = where(dsId,'libname = ""');

return;


LB01:

  call notify('lb01','_get_last_sel_',nRow,nIsSelected,cCurLib);

  if (nIsSelected) then do;
    rc = where(dsid,"libname = '"||cCurLib||"'");
    call notify('lb02','_update_');
  end;

return;


PBOK:

  call notify('lb02','_get_last_sel_',nRow,nIsSelected,cCurData);
  if (cCurLib = _blank_) or (cCurData = _blank_) then RETURN;
  call notify('pbok','_set_cmd_','END');
  call fsedit(cCurLib||'.'||cCurData,'',rbMode);

return;


TERM:

    if (dsId) then dsId = close(dsId);

return;
```

Anwendungsbeispiel 3 – Dokumentation

Die in den Dictionary Tables enthaltenen Metainformationen bieten sich zur automatisierten Erstellung von Dokumentationen an, insbesondere wenn die SAS-Tabellen ordentlich aufgebaut, d.h. Label und Formate vergeben wurden. Eine solche Dokumentation kann z.B. in einem Intranet bereitgestellt werden, wenn mehrere Diplomanden/Mitarbeiter auf die gleichen Datenbestände zurückgreifen (vergl. 12.).

z.B. Listing der Variablendefinition

Die gleiche Information, die PROC CONTENTS über Variablen liefert, kann mit Hilfe von PROC SQL abgefragt werden; hier kann allerdings direkt nach Name oder Typ sortiert werden.

```
PROC SQL;
  CREATE VIEW tmp AS
    SELECT name, type, length, label, format
      FROM dictionary.columns
      WHERE libname="SASHELP" AND memname="PRDSALE"
      ORDER BY name;
QUIT;

PROC PRINT DATA=work.tmp LABEL;
RUN;
```

Literatur

SAS Institute Inc. (1991): SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, S. 286-291.
SAS Institute Inc. (1994): SAS Screen Control Language: Reference, Version 6, Second Edition.
Weinberger, Volker (1996): Dictionary Tables, SAS-Benutzertreff am URZ Heidelberg 2.08.1996.

8. Special Merge (one-to-many)

Heinrich Stürzl

Aufgabenstellung

Eine einzelne Beobachtung aus einer SAS Tabelle (Data set) soll mit allen Beobachtungen einer anderen SAS Tabelle verknüpft (gemergt) werden.

Beispiel:

Den Median eines Merkmals in allen Beobachtungen als Variable zur Verfügung stellen, um die Abweichung davon für jede Beobachtung berechnen zu können.

OBS	X	MEDIAN
1	10	55
2	20	55
3	30	55
4	40	55
5	50	55
6	60	55
7	70	55
8	80	55
9	90	55
10	100	55

Lösung 1:

Hilfsvariable mit identischer Ausprägung in allen Beobachtungen bilden und als By-Variable zum Mergen verwenden. Dies ist umständlich, vor allem wenn die By-Variable extra erzeugt werden muß, da meist mindestens ein zusätzlicher Data Step erforderlich ist.

Lösung 2:

Mergen mit bedingter SET Anweisung(en), als Spezialfall des “one-to-one reading”.
Dies hat den Vorteil, daß keine By-Variable nötig ist und kann auch beim Verknüpfen mehrerer Dateien verwendet werden.

```
DATA gesamt;
  DO X=10 TO 100 BY 10; OUTPUT; END;

PROC UNIVARIATE DATA=gesamt NOPRINT;
  VAR x;
  OUTPUT OUT=work.median MEDIAN=median;

DATA neu;
  SET work.gesamt;
  IF _N_=1 THEN SET work.median;
RUN;
```

Die Datei `work.neu` enthält über ein automatisches Retain in allen Beobachtungen von `work.gesamt` zusätzlich alle Variablen der Datei `work.median` mit den Ausprägungen der 1. Beobachtung.

Literatur:

SAS Language Reference, Version 6 First Edition, S. 487

9. Textvariable in numerische Variable umwandeln

Carina Ortseifen

Aufgabenstellung

Eine Textvariable muss in eine numerische Variable verwandelt, damit zwei SAS Tabellen korrekt miteinander verknüpft werden können.

Lösung

Verwendung der Funktion INPUT (in Analogie zur Anweisung INPUT, die beim Einlesen neuer Variablen den Typ festlegt. Im folgenden Beispiel ist die Variable x die Textvariable, die in die numerische Variable y transformiert werden soll.

```
DATA test;
   X="12.000,25";
   Y=INPUT(x,COMMA9.2);
RUN;
```

Die Funktion INPUT hat zwei Argumente: Die zu transformierende Variable und das Einleseformat. Dieses Informat bestimmt den Datentyp der Variable y. Das Resultat der Funktion INPUT kann daher numerisch oder Text sein.

Soll nun eine numerische Variable in eine Textvariable verwandelt werden, kann auch die Funktion PUT verwendet werden, deren Ergebnis – analog zur Anweisung PUT – immer ein Text ist.

```
DATA test;
   X=12000.25;
   Y=PUT(x,COMMAx9.2);
RUN;
```

Das Format, das als zweites Argument übergeben wird, muss genau dem Typ der Variable x entsprechen.

Alternativ dazu können auch die SCL-Funktionen PUTN für numerische Variablen und PUTC für Textvariablen verwendet werden.

```
DATA test;
   X=12000.25;
   Y=PUTN(x,"COMMAx9.2");
RUN;
```

Das Format muss dabei in Anführungszeichen eingeschlossen werden. Das wirkt auf den ersten Blick umständlich, doch erlaubt es, dass Formate in (Makro-)Variablen gespeichert werden können:

```
DATA test;
   X=12000.25;
   Y=PUTN(x,xfmt);
RUN;
```

10. WMF- und BMP-Grafiken im Batch-Mode mit True Type Fonts

Grischa Pfister

Aufgabenstellung

SAS/GRAPH entwickelt seine Stärke insbesondere dann, wenn eine Vielzahl von Grafiken automatisch erzeugt werden soll. Leider ließen die Export-Schnittstellen in Punkto Schriftarten einige Wünsche offen, auch für SAS-Anwender, die sich schon geraume Zeit mit SAS/GRAPH beschäftigen ein Problem.

Devices in SAS/GRAPH

Die Ausgabe von Grafiken wird in SAS/GRAPH über sog. Devices gesteuert, das sind unterschiedliche Treiber für die Ausgabe auf den Bildschirm, auf einen Drucker oder für den Export in ein bestimmtes Format. Dabei kennt jeder dieser Treiber nur bestimmte "hardware fonts" (eigene Schriftarten), die mit einer besonderen Syntax angesprochen werden müssen.
Neben einer Vielzahl spezifischer Treiber wurde in Version 6.08 auch ein Packet speziell für Windows integriert mit dem Vorteil, dass die in diesem Packet enthaltenen Treiber die Windows-Schriftarten unterstützen und einheitlich benennen, d.h. Programme können ohne Modifikation sowohl für die Bildschirm- als auch die Druckerausgabe verwendet werden.
Zu diesem Packet gehören die Treiber WIN (für den Bildschirm), WINPRTx (benutzt den Windows-Standarddrucker; für "x" kann ein "C" für Farbausgabe, ein "G" für Grauraster oder "M" für monochrom angegeben werden) und WINPLOT (für Farbplotter). Diese Treiber werden über die sog. "SAS/GRAPH Font Management Utility" (Help -> Utility Application) installiert.
Nur für den Export gab es keine adäquaten Treiber, obwohl z.B. der WMF-Export aus dem GRAPH-Fenster möglich ist und hier auch True Type Fonts exportiert werden.

PROC GETFNT

Seit Version 6.12 ist jetzt auch in der Online-Dokumentation die Prozedur GETFNT beschrieben, deren Aufgabe es ist, die Windows True Type Fonts für die Treiber der Windows-Familie verfügbar zu machen. Diese Prozedur kann auch für die WMF und BMP Treiber verwendet werden.

Vorgehensweise

Libraries einrichten

- GDEVICE0
- GFONT0

SAS/GRAPH benötigt zwei Libraries, um die Informationen über "hardware fonts" zu speichern. Die Library GDEVICE0 enthält einen Katalog namens DEVICES, er nimmt die benutzerdefinierten Treiberinformationen auf. Die Namenskonventionen für Libref und Katalognamen müssen unbedingt eingehalten werden, weil sie in einer SAS-internen Suchreihenfolge Verwendung finden. Die Library GFONT0 beinhaltet den Katalog FONTS, er enthält für jede eingelesene Schriftart einen Entry vom Typen FONT, der Entry-Name entspricht dem Namen, unter dem die Schriftart für SAS/GRAPH verfügbar ist, die Beschreibung enthält den Schriftartnamen im Klartext.

Wer die Font Management Utility bereits für einen der Treiber WIN/WINPRTx aufgerufen hatte, verfügt bereits über die beiden Libraries, denn in diesem Falle wird die CONFIG.SAS um zwei Einträge erweitert. Normalerweise zeigen beide Librefs auf das gleiche Verzeichnis wie SASUSER. Wenn Sie die Schriftarten ohne Font Management Utility einrichten, sollten Sie Ihre autoexec.sas um (z.B.) folgende Einträge erweitern:

```
libname gdevice0 '!sasroot\sasuser';
libname gfont0 '!sasroot\sasuser';
```

Treiber kopieren

- PROC GDEVICE

SAS/GRAPH besitzt die Prozedur GDEVICE, die zur Verwaltung von Treiberinformationen benutzt wird. Mit ihrer Hilfe können Treiber, die nichts anderes sind als Katalogeinträge vom Typ "DEV", erstellt, kopiert, modifiziert oder gelöscht werden. Die Prozedur kann im interaktiven oder nicht-interaktiven Modus benutzt werden, die von SAS mitgelieferten Standardtreiber befinden sich im Katalog SASHELP.DEVICES.

```
PROC GDEVICE
  CAT=gdevice0.devices NOFS;
    COPY wmf FROM=sashelp.devices
    NEWNAME=gpWmf;
  QUIT;
```

In diesem Beispiel wird die Prozedur im nicht-interaktiven Modus aufgerufen (NOFS=no full screen) und der Treiber WMF aus SASHELP.DEVICES nach GDEVICE0.DEVICES kopiert, wobei der Treiber zudem umbenannt wird in GPWMF.

Schriftarten einlesen

- PROC GETFNT

Anschließend wird die Prozedur GETFNT aufgerufen, die die Schriftartenliste des Treibers um die unter Windows verfügbaren Schriftarten (FONTSYS=win) erweitern wird. Mit der Option DELIMITER wird angegeben, welches Zeichen als Trennungssymbol bei der Beschreibung der Schriftartennamen verwendet wird, dies wird im nächsten Punkt deutlich.

```
PROC GETFNT
  FONTSYS=win
  DELIMITER='/'
  DEVICE=gpWmf;
RUN;
```

PROC GETFNT erzeugt eine Reihe von Fehlermeldungen, die getrost ignoriert werden können. Die ebenfalls generierten Warnings stammen von PROC GFONT, das von GETFNT aufgerufen wird, auch sie sind vernachlässigbar.

Schriftarten überprüfen

- PROC GDEVICE

Im nächsten Schritt wird überprüft, ob die Windows-Schriftarten wirklich installiert wurden, hierfür wird wieder die Prozedur GDEVICE herangezogen.

```
PROC GDEVICE CAT=gdevice0.devices NOFS;
  LIST gpWmf;
QUIT;
```

Im OUTPUT-Fenster wird ein Listing mit allen Treiberinformationen ausgegeben, in diesem Falle ist der Abschnitt "CHARTYPE RECORDS" von Interesse, er zeigt eine Liste der für das Device verfügbaren Schriftarten an.

```
Chartype Rows  Cols                 Font Name                 Scalable
     0      1     1   DMS Font                                    N
     1      1     1   Courier New                                 Y
     2      1     1   Courier New/me/it                           Y
     3      1     1   Courier New/bo/it                           Y
     4      1     1   Courier New/bo/ro                           Y
     5      1     1   Courier New/me/ro                           Y
     6      1     1   Lucida Console                              Y
     7      1     1   SAS Monospace                               Y
     8      1     1   SAS Monospace/me/ro                         Y
     9      1     1   SAS Monospace Bold                          Y
...
```

Die wichtigsten Spalten sind "Chartype" und "Font Name". Letzteres zeigt den eigentlichen Namen der Schriftart an, ersteres ist ein Zähler, der benutzt wird, um die Schriftart in SAS/GRAPH ansprechen zu können.
Zunächst zu "Font Name": Für jede Schriftart werden mehrere Einträge benötigt, weil SAS/GRAPH keine andere Möglichkeit kennt, weitere Layout-Optionen (fett, kursiv) anzugeben. Diese Informationen werden an den Schriftartnamen angehängt und durch das in PROC GETFNT unter "DELIMITER=" vereinbarte Zeichen abgetrennt. Die verwendeten Kürzel stehen für:

- me = medium
- it = italic
- bo = bold
- ro = roman

Ansprache der True Type Fonts mit SAS/GRAPH

In Version 6 gelten für die Namen von Schriftarten die gleichen Restriktionen wie z.B. für Variablennamen (maximal acht Zeichen etc.). Wie oben gezeigt, enthalten die "Font names" sogar Sonderzeichen als Delimiter für Layout-Optionen, sie können also nicht benutzt werden. Deshalb wird für die oben gezeigte Liste ein Referenzmechanismus benutzt, der auf den laufenden Index (=Chartype) verweist.
Alle Schriftarten des Devices werden mit dem Referenznamen HWDMX*nnn* angesprochen (für hardware device microsoft), "*nnn*" entspricht dem dreistellig anzugebenden "Chartype" der Schriftart, wie er in der obigen Liste enthalten ist. Die Schriftart "Courier, fett und kursiv" hat für SAS/GRAPH dementsprechend den Namen HWDMX003.

Unterstützte Treiber

Die folgenden Treiber unterstützen das Fontsys "win", können also mit PROC GETFNT eingerichtet werden:

- WIN (Bildschirmdarstellung)
- WINPRTx (Windows-Systemdrucker)
- BMP,BMP20 (Pixelgrafik)
- WMF (Vektorgrafik)

Sie sollten am besten die vier wichtigsten Treiber (WIN, WINPRTC, BMP und WMF) in einem Schritt installieren, damit die Chartype Records auch wirklich identisch sind.

Anmerkung

Beim Export von WMF-Grafiken nach Winword kann es zu dem Phänomen kommen, dass nur ein Teil der Grafik angezeigt wird. Das liegt daran, dass Winword nicht automatisch erkennt, wie groß die Grafik ist. Wenn Sie auf die Grafik doppelklicken und in den Grafik-Bearbeiten-Modus wechseln, wird die Größe aber erkannt und die Grafik korrekt dargestellt, wenn Sie den Bearbeiten-Modus wieder verlassen. Sie können dieses Problem umgehen, indem Sie folgende GOPTIONS verwenden, das empfiehlt sich jedoch nur für kleinere Grafiken.

```
GOPTIONS hsize=6.0 in vsize = 5.0 in nopiefill nocirclearc;
```

[vergl. TS352C]

Literatur

Help -> SAS Online Documentation -> What's New in Release 6.12 -> What's New in SAS/GRAPH -> The GETFNT Procedure

TS352C-Beta-Release: Changes and Enhancements for Release 6.11. Imaging Systems and Export Drivers.

SAS Institute Inc. (1990): SAS/GRAPH Software: Reference, Version 6, First Edition, Volume 1 & 2, Cary, NC.

11. Dubletten Check

Heinrich Stürzl, Martina Rossi

Dieser Beitrag bezieht sich auf den von Martina Rossi auf der 3. KSFE in Heidelberg vorgestellten Beitrag mit dem Thema "Makro zum Auffinden doppelt vergebener Schlüsselnummern". Das dort vorgestellte Makro wurde in seiner Funktionsweise dahingehend verändert, dass keine Beobachtungen mehr automatisch gelöscht werden. Stattdessen erfolgt eine detaillierte Anzeige der Dubletten im Log Fenster. Gleichzeitig wurde das Makro verallgemeinert (Anzahl und Typ der By-Variablen).
Sie finden diesen Beitrag mit Quellcode des hier vorgestellten Makros MULTKEY zum Herunterladen im SAS-Anwenderhandbuch (SAS-Ah) unter dem Titel "Erkennung mehrfach vorhandener Schlüssel - SAS Makro MULTKEY als Alternative zur NODUPKEY Option von PROC SORT"
http://www.urz.uni-heidelberg.de/statistik/sas-ah/

Einleitung

Wer häufig Datendateien aus unterschiedlichen Quellen erhält und diese zusammenführt, kennt das Problem von mehrfach vorhandenen und damit nicht eindeutigen Schlüsseln. Bleiben diese unbemerkt, kann das zu falschen Ergebnissen führen.
PROC SORT bietet mit den Optionen NODUP bzw. NODUPKEY vermeintlich bequeme Lösungsmöglichkeiten.
Die Option NODUP löscht ausschließlich echte Dubletten, d. h. Beobachtungen, deren Ausprägungen in den Schlüssel(By)-Variablen **und** in allen übrigen Variablen übereinstimmen. Allerdings werden echte Dubletten nur dann erkannt und beseitigt, wenn sie in der Tabelle unmittelbar aufeinander folgen!
Die Option NODUPKEY löscht die zweite und jede weitere Beobachtung mit demselben Schlüssel. Im Log Fenster erscheint dabei lediglich die Anzahl der insgesamt gelöschten Beobachtungen. Welche Beobachtungen aus der Datei entfernt worden sind, ist nicht mehr direkt erkennbar und hängt maßgeblich von der Sortierung der Datei ab.

Das nachfolgend beschriebene Makro MULTKEY löscht keine Beobachtung, sondern zeigt mehrfach vorkommende Schlüssel im Log Fenster. Außerdem wird eine neue Variable _MULTI_ gebildet, in der mehrfach auftretende Schlüssel numeriert werden. Damit ist es möglich, diese Beobachtungen einfach zu finden und in einem nachfolgenden Daten- oder Prozedurschritt gezielt zu löschen oder zu korrigieren.

Beschreibung und Anwendung des Makros MULTKEY

Das Makro MULTKEY überprüft eine SAS Tabelle (data set) auf Eindeutigkeit aller angegebenen Schlüssel- bzw. By-Variablen. Es zeigt alle Beobachtungen mit nicht-eindeutigem das heißt identischem Schlüssel im Log Fenster als Warnung an. Diese werden dabei innerhalb jedes mehrfach vorkommenden Schlüssels fortlaufend nummeriert. Es werden keine Beobachtungen gelöscht!

Das Makro MULTKEY erwartet eine SAS Tabelle und eine beliebige Anzahl darin enthaltener Schlüssel(By)-Variablen, deren Ausprägungen (Schlüssel) auf Eindeutigkeit überprüft werden sollen.

Ähnlich wie bei PROC SORT kann optional eine Ausgabedatei angegeben werden. Andernfalls wird die Datei durch eine nach den Schlüssel-Variablen sortierte Kopie, die zusätzlich die numerische Variable _MULTI_ (3 Byte) enthält, überschrieben. Die Variable _MULTI_ enthält im Fall eines eindeutigen Schlüssels den Wert 0, andernfalls bei wiederholt vorkommendem Schlüssel eine fortlaufende Nummer beginnend mit dem Wert 1.

Anwendungsbeispiel:

Die nachfolgende Datei enthält zwei Schlüsselvariablen GROUP und IDNO, wobei zwei Schlüssel mehrfach vorkommen (group=two idno=100 und group=two idno=200).

```
DATA check;
INFILE DATALINES;
INPUT group $ idno x;
DATALINES;
one 100 1
two 100 2
two 100 2
two 200 3.1
two 200 3.2
two 200 3.3
two 300 4
;
RUN;
```

Nach dem Aufruf des Makros MULTKEY

```
%multkey (data=check, key=group idno);
```

erscheint im Log Fenster folgende Warnung:

```
WARNING: Multiple key in GROUP IDNO: two 100
_N_=2 two IDNO=100 _MULTI_=1
_N_=3 two IDNO=100 _MULTI_=2

WARNING: Multiple key in GROUP IDNO: two 200
_N_=4 two IDNO=200 _MULTI_=1
_N_=5 two IDNO=200 _MULTI_=2
_N_=6 two IDNO=200 _MULTI_=3
```

Die Beobachtungen mit mehrfach vorkommendem Schlüssel lassen sich z. B. mit folgender WHERE Anweisung mit PROC PRINT vollständig ausgeben:

```
PROC PRINT DATA=check;
  WHERE _multi_>0;
RUN;
```

Hinweis:
Die untersuchte Datei `work.check` ist durch den Aufruf des Makros neu sortiert und überschrieben worden. Falls dies nicht gewünscht wird, kann wie im folgenden Beispiel eine andere Ausgabedatei angegeben werden.

```
%multkey (data=check, key=group idno, out=check2);
```

Makro Quelltext

```
/*+------------+
  !MULTKEY .sas!
  +------------+

created by: Heinrich Stürzl on: 28DEC1999
  (Heinrich_Stuerzl@DadeBehring.com)

-------------------------------------------------------------------
DESCRIPTION:

Macro to check a given data set for multiple keys i.e.
observations with repeated (non-unique) values of key(by)-variables.

The NODUPKEY Option of PROC SORT deletes multiple keys automatically
but you do not know which observations have been deleted.

This Macro generates a warning in the log for each multiple key
found and lists them successively. Therefore it adds a new numeric
variable _MULTI_ to the data set, which contains the value 0 in case
of an unique key and consecutive numbers for repeated keys.

With this variable it's easy to identify non-unique (multiples) keys
and to correct or delete them in a further step.

The macro is similar to PROC SORT and overwrites the original data
set! Otherwise you can use the OUT parameter to store the sorted
copy to a new data set.

-------------------------------------------------------------------
MACRO PARAMETER:

DATA   Name of input data set
       Input data set remains unchanged if the OUT parameter is used

KEY    Key(By)-Variable(s) in the input data set, which should be
       checked, whether their data values occur uniquely or not.
       By-variables can be character or numeric.

OUT    (optional) Name of an output data set.
       If empty the name of the input data set is used which
       overwrites the input data set.
       The output data set is a copy of the input data set, sorted by
       the key(by)-variables and with the variable _MULTI_ added
       (dropped before if it exists already in this data set)
       The variable _MULTI_ contains the value 0 for unique keys and
       consecutive numbers beginning with 1 for repeated keys. I.e.
       observations with identical key values are numbered within
       each key.
```

```
EXAMPLE Data Set:
                GROUP     IDNO      _MULTI_

                  A       1001          0
                  A       1002          1
                  A       1002          2
                  A       1002          3
                  A       1003          1
                  A       1003          2

-----------------------------------------------------------------------
EXAMPLES:

DATA check;
  INPUT group $ idno x;
  DATALINES;
one 100 1
two 100 2
two 100 2
two 200 3.1
two 200 3.2
two 200 3.3
two 300 4
; RUN;

%multkey (data=check, key=group idno);

PROC PRINT DATA=check;
  where _multi_>0;
RUN;

%multkey (data=check, key=group);
  * overwrites existing Variable _multi_;

%multkey (data=check, key=group, out=new);
  * creating a new output file;
-----------------------------------------------------------------------*/


%MACRO multkey (data=, key=, out=) / DES='find multiple keys';

%LOCAL error lastkey drop;

%*********************************************************************
check input
**********************************************************************;

%IF "&data" = "" %THEN %DO;
  %PUT >>>> ERROR: No input data set <<<<;
  %GOTO ende;
%END;

%IF "&key" = "" %THEN %DO;
  %PUT >>>> ERROR: No key(by)-varible <<<<;
  %GOTO ende;
%END;
```

```
DATA _null_;
  IF NOT (EXIST("&data")) THEN DO;
    _msg_=SYSMSG();
    PUT "ERROR: Input data set not accessible ";
    PUT _MSG_;
    CALL SYMPUT ('error', "1");
  END;
  ELSE DO;
%* get the innerst By-Variable;
    CALL SYMPUT('lastkey', REVERSE(SCAN(REVERSE("&key"),1)));
%* Check if variable _MULTI_ already exists in the input data set
   If yes, than drop while reading;
    dsid=OPEN("&data", "I");
    no=VARNUM(dsid, "_multi_");
    IF no>0 THEN CALL SYMPUT('drop', '1');
    sysrc=CLOSE(dsid);
  END;
RUN;
%IF &error = 1 %THEN %GOTO ende;

%IF "&out" = "" %THEN %let out=&data;

%*********************************************************************
check for multiple keys
**********************************************************************;

PROC SORT DATA=&data out=&out;
  BY &key;
RUN;
%IF &SYSERR ^= 0 %THEN %GOTO ende;

DATA &out;
  LENGTH _multi_ 3;
  SET &out  %IF "&drop" = "1" %THEN (DROP=_multi_) ; ;
  BY &key;

  IF FIRST.&lastkey                          THEN _multi_=0;
  %* initializing;
  IF NOT (FIRST.&lastkey AND LAST.&lastkey) THEN _multi_+1;
  %* count multiples;

  IF _multi_=1 THEN PUT / "WARNING: Multiple key in %upcase(&key): "
  &key;
  IF _multi_>0 THEN PUT _N_= &key=  _multi_=;
RUN;

%ende:
%MEND multkey;
```

12. HTML aus dem Data Step

Grischa Pfister

Aufgabenstellung

SAS bietet verschiedene Möglichkeiten an, den Output von Prozeduren oder den Inhalt von Data Sets in HTML-Format auszugeben. Für Version 6 gibt es eine Makrobibliothek, in Version 8 wird das Output Delivery System diese Aufgabe übernehmen.
Beides sind jedoch für den Benutzer zunächst einmal "black boxes", die Anpassung an eigene Darstellungswünsche erfordert eine intensive Einarbeitung. Deshalb soll hier eine andere Möglichkeit dargestellt werden, die im Verhältnis zu den anderen beiden als unkompliziert, transparent und leicht anpassbar gelten kann.
Es soll ein HTML-Dokument erstellt werden, das den Inhalt eines Data Sets in einer Tabelle anzeigt.
Bei dem Export aus dem Datenschritt wird auf Syntax zurückgegriffen, die den meisten SAS-Anwendern vertraut, bzw. schnell nachvollziehbar ist, und auch die nötigen HTML-Kenntnisse sind schnell erlernt.

Benötigte SAS-Statements

- DATA _NULL_;
  Mit diesem Statement wird ein Datenschritt eingeleitet, der keinen neuen SAS-Data Set erzeugt, denn die Ausgabe erfolgt in ein ASCII-File.
- FILE 'pfad/name.htm(l)';
  Das FILE-Statement referenziert die Zieldatei, die sinnvollerweise die Erweiterung HTM bzw. HTML hat.
- PUT variable <'string'>;
  Mit PUT werden Werte oder Zeichenketten in das ASCII-File geschrieben

Besonderheiten von HTML

Die HyperText Markup Language (HTML) arbeitet mit speziell formatierten Zeichenkombinationen die als Layoutinformation dienen. Diese werden als Tags bezeichnet und vom normalen Text dadurch unterschieden, dass sie zwischen spitze Klammern gesetzt werden "<tag>". Für viele Tags wird das Klammerprinzip benutzt, d.h. dass ein Tag als Startschalter fungiert, ein zweites als Endschalter. Als Beispiel dient der Grundaufbau eines HTML-Dokumentes:

```
<HTML>
  <HEAD> ... </HEAD>
  <BODY> ... </BODY>
</HTML>
```

Das Grundgerüst besteht aus drei Klammerungen, "<HTML>" bis "</HTML>" umschließt das gesamte Dokument und gibt dem Browser die Information, dass er den Inhalt als HTML-Dokument interpretieren soll. In der "<HEAD> ... </HEAD>"-Sektion werden Metadaten gespeichert, hier wird aber auch der Seitentitel festgelegt (s.u.). Der Abschnitt "<BODY>" bis "</BODY>" enthält den eigentlichen Inhalt der Seite.

Wichtige HTML-Tags

- Titel

```
<H1(-9)> Text </H1(-9)>
```

In HTML können Titel über neun Hierarchiestufen untergliedert werden.

- Tabelle

```
<TABLE optionen >
<TR> <TD>Item01 <TD> Item02 ...
...
</TABLE>
```

Tabellen werden durch die Klammer “<TABLE>” bis “</TABLE>” festgelegt, innerhalb der Tabelle kann jedoch auf die Klammerung verzichtet werden. Die Darstellung der Tabelle erfolgt zeilenweise, jede Zeile beginnt mit dem Tag “<TR>” (table row), der Beginn einer Zelle wird durch das Tag “<TD>” (table data) angezeigt.

Implementierung einer Start- und Endsequenz im Datenschritt

Um aus dem Datenschritt ein HTML-Dokument zu erzeugen, muss der normale Ablauf des Datenschrittes unterbrochen werden. Denn der Datenschritt ist eine Schleife, die für jede einzulesende Beobachtung durchlaufen wird, das HTML-Dokument besteht aber aus einem Kopfteil, einem Datenkörper und einem Schlußteil. Der Datenkörper soll die Daten eines Data Sets aufnehmen, ergänzt um die Layoutanweisungen für eine HTML-Tabelle. Er kann in der regulären Datenschritt-Schleife geschrieben werden. Davor muss jedoch die Information für den HTML-Kopfteil generiert werden, und wenn alle Daten exportiert sind, muss das HTML-Dokument abgeschlossen werden. Zu diesem Zweck werden zwei weitere SAS-Features benötigt.

Startsequenz

- automatische Zählervariable _N_

```
DATA _NULL_;
IF _N_ = 1 THEN DO;
  Startsequenz
END;
```

Der Datenschritt verfügt über die automatische Zählervariable _N_, die nur im Datenschritt selbst existent ist. Sie wird benutzt, um die Startsequenz abzusetzen, bevor zum ersten Mal Daten geschrieben werden.

Endsequenz

- „END = Variable“ Option des SET-Statements

```
DATA _NULL_;
SET libref.dataset END = eof;
...
IF eof THEN DO;
  Endsequenz
END;
run;
```

Das SET-Statement bietet die Möglichkeit, eine ebenfalls nur im Datenschritt verfügbare Variable zu definieren, die anzeigt, ob die letzte Beobachtung erreicht wurde. Sie ist numerisch mit dem Wert 0 und wird am Ende auf 1 gesetzt.

Beispiel 1

Das Beispiel greift auf die bereits vorgestellten Dictionary Tables zurück (vergl. 7.). Für den Data Set SASHELP.PRDSALE wird ein HTML-Dokument erstellt, das alle enthaltenen Variablen mit Name, Typ, Länge, Label und Format beschreibt.

```
* Datengrundlage *;
PROC SQL;
  CREATE VIEW tmp AS
    SELECT name, type, length, label, format
      FROM dictionary.columns
      WHERE libname="SASHELP" AND memname="PRDSALE"
      ORDER BY name;
QUIT;

* HTML-Export *;
DATA _null_;
  SET tmp END=eof;
  FILE 'c:/test.html';

  * Startsequenz *;
  IF _N_=1 THEN DO;
    PUT '<HTML>';
    PUT '  <HEAD> <TITLE> HTML aus dem Data Step </TITLE> </HEAD>';
    PUT '  <BODY>';
    PUT '    <H1> Definition der Variablen in SASHELP.PRDSALE
  </H1>';
    PUT '    <HR>';
    PUT '    <TABLE border=1>';
    PUT @7 '<TR>' '<TD> Name <TD> Typ <TD> L&auml;nge <TD>
  Beschreibung <TD> Format';
  END;

  * Schreiben des Tabelleninhaltes *;
  PUT @7 '<TR>' '<TD>' name '<TD>' type '<TD>' length '<TD>' label
  '<TD>' format;

  * Endsequenz *;
  IF eof THEN DO;
    PUT '    </TABLE>';
    PUT '  <HR>';
    PUT '  <I><B> HTML aus dem Data Step <B></I>';
    PUT '  </BODY>';
    PUT '</HTML>';
  END;
RUN;
```

Beispiel 2

Hier wird das obige Beispiel etwas modifiziert: statt nur für SASHELP.PRDSALE eine Tabelle auszugeben, wird jetzt für jeden Data Set in SASHELP eine Tabelle erzeugt. Dafür muss lediglich eine weitere Subschleife erzeugt werden, wofür das folgende SAS-Konstrukt benutzt wird: Wird im SET-Statement ein sortierter Data Set angegeben, kann ein BY-Statement verwendet werden.
Im Datenschritt stehen dann pro By-Variable zwei weitere numerische Variablen zur Verfügung, die anzeigen ob gerade die erste oder die letzte Variable einer BY-Gruppe eingelesen wurde (first.*variable* bzw. last.*variable*). Diese Information wird wieder genutzt, um die entsprechenden HTML-Tags zum Öffnen bzw. Schließen von Tabellen zu schreiben.

```
* Datengrundlage *;

PROC SQL;
  CREATE VIEW tmp AS
    SELECT memname, name, type, length, label, format
      FROM dictionary.columns
      WHERE libname="SASHELP" AND memtype="DATA"
      ORDER BY memname, name;
QUIT;

* HTML-Export *;

DATA _null_;
  SET tmp END=eof;
  BY memname;
  FILE 'c:/test.html';

  * Startsequenz *;
  IF _N_=1 THEN DO;
    PUT '<HTML>';
    PUT '  <HEAD> <TITLE> KSFE 2000 Giessen </TITLE> </HEAD>';
    PUT '  <BODY>';
  END;

  IF first.memname THEN DO;
    PUT '    <HR>';
    PUT '    <H1> Definition der Variablen in '
        'SASHELP.'memname'</H1>';
    PUT '    <TABLE border=1>';
    PUT @7 '<TR> <TH width=10%> Name <TH width= 5%> Typ '
           ' <TH width= 5%> L&auml;nge <TH width=40%> Beschreibung '
           ' <TH width=10%> Format';
  END;

  * Schreiben des Tabelleninhaltes *;
  if label = '' then label = '-';
  if format = '' then format = '-';
  PUT @7 '<TR><TD>' NAME' <TD align=center>' TYPE
              '<TD align=right>' LENGTH '<TD>' LABEL '<TD>' FORMAT;

  * Endsequenz *;
  IF last.memname THEN DO;
    PUT '    </TABLE>';
  END;
```

```
  IF eof THEN DO;
    PUT '  <HR>';
    PUT '  <I><B> HTML aus dem Data Step <B></I>';
    PUT '  </BODY>';
    PUT '</HTML>';
  end;

run;
```

Das Beispiel zeigt außerdem weitere HTML-Tags bzw. Optionen. So werden die Tabellenüberschriften diesmal über das “<TH>”-Tag (table header) referenziert und verschiedene Optionen für Spaltenbreiten und Ausrichtung gezeigt.

Problem nationale Umlaute

Nationale Umlaute können ein Problem darstellen, wenn die Spracheinstellung des Browsers sie nicht unterstützt. Deshalb bietet HTML spezielle Zeichenkombinationen an, die sie ersetzen. Wie in der Diskussion angesprochen, wird es problematisch, wenn eine Variable 200 Zeichen hat und Umlaute enthält, die ersetzt werden sollen, denn die HTML-Strings sind länger als die Sonderzeichen.
Andererseits ist mir kein Wort mit 200 Zeichen Länge bekannt, deshalb schlage ich hier folgende Lösung vor: Die HMTL-Datei wird, nachdem sie erzeugt wurde, noch einmal eingelesen und dabei werden die Sonderzeichen ersetzt.
Im folgenden Code werden zunächst zwei Arrays angelegt, FROM enthält die Sonderzeichen, TO das HTML-Äquivalent. Mit INFILE und INPUT wird der Text aus dem HTML-File eingelesen, und zwar jedes Wort einzeln. Dann wird in der DO WHILE-Schleife geprüft, ob der Text Sonderzeichen enthält, wenn ja, werden sie ersetzt. Am Ende wird der korrigierte Text mit FILE und PUT in die neue HTML-Datei geschrieben. Damit Text der in der Eingabedatei in einer Zeile steht auch in der Ausgabedatei wieder in einer Zeile steht wird die Option LINE des INFILE-Statements genutzt. Das Statement “IF line THEN PUT;” schließt die mit “@@” offengehaltene Ausgabezeile und positioniert den Schreibcursor in eine neue Zeile.

```
DATA _null_;
  ARRAY from {7} $1 ('Ä','Ö','Ü','ä','ö','ü','ß');
  ARRAY to {7}  $8
                    ('&Auml;','&Ouml;','&Uuml;','&auml;',
                     '&ouml;','&uuml;', '&szlig;');
  LENGTH text txt1 txt2 $200 replace $8;

  INFILE 'c:/test.html' line=line;
  INPUT text @@;

  DO WHILE (INDEXC(text,'ÄÖÜäöüß') > 0);
    index=INDEXC(text,'ÄÖÜäöüß');
    IF index > 1 THEN txt1 = SUBSTR(text,1,index-1);
    ELSE txt1 = '';
    replace =SUBSTR(text,index,1);
    IF index < LENGTH(text) then txt2 = SUBSTR(text,index+1);
    ELSE txt2 = '';

    DO i=1 TO DIM(from);
      replace = tranwrd(replace,from(i),TRIM(to(i)));
    END;
```

```
    text = TRIM(LEFT (txt1))||TRIM(LEFT (replace))||
                TRIM(LEFT (txt2));
  END;

  FILE 'c:/test2.html';
  IF line THEN PUT;
  PUT text @@;

RUN;
```

Literatur

Münz, Stefan und Nefzger, Wolfgang (1997): HTML Handbuch, Franzis-Verlag, Feldkirchen.
http://www.w3.org/TR/html4/