Tipps & Tricks - für einen leichteren Umgang mit der SAS Software

Wolfgang Herff	Carina Ortseifen	Grischa Pfister	Heinrich Stürzl
RWTH Aachen	Universität Heidelberg	Systematics Integrations GmbH	Dade Behring Marburg
Institut für Statistik und Wirtschaftsmathematik	Universitäts-rechenzentrum	Info Ware	Statistical Systems & Data Management
herff@stochastik.rwth-aachen.de	Carina.Ortseifen@urz.uni-heidelberg.de	grischa.pfister@info-ware.de	Heinrich_Stuerzl@DadeBehring.com

Zusammenfassung

In Form von Kurzvorträgen werden nützliche Lösungen zu Problemen und Fragestellungen vorgestellt, die bei der täglichen Arbeit mit der SAS Software auftreten können. Im wesentlichen handelt es dabei um Base SAS und teilweise auch um SAS/GRAPH. Es werden dabei nicht unbedingt neue Prozeduren, Optionen oder Module vorgestellt. Stattdessen soll die effektive Anwendung vorhandener Anweisungen und Prozeduren an Beispielen aufgezeigt werden.

1. Makrovariablen mit PROC SQL erzeugen ..... Seite 2
2. Tückische Blanks im SAS Quellcode ..... Seite 7
3. Umgebungsvariablen (Windows) ..... Seite 8
4. Das Einleseformat w.d ist nicht immer die beste Wahl ..... Seite 11
5. Nochmals Einlesen - Gültigkeitsprüfung ..... Seite 13
6. Rundungsfehler sind nicht immer offensichtlich ..... Seite 14
7. Verknüpfen von zwei Tabellen – Match Merging ..... Seite 17
8. Sortieren mit deutschen Umlauten ..... Seite 20
9. Probleme im Zusammenhang mit der Einstellung DBCS ..... Seite 24
10. Reguläre Ausdrücke in SAS Base ..... Seite 27

Keywords:
Tipps, Tricks, SAS, Base, Version 8, V8, Makro, Macro, Makrovariable, Macrovariable, SQL, PROC SQL, SELECT, SELECT INTO, CALL SYMPUT, Liste, Werteliste, Leerzeichen, Blank, ERROR 180-322: Statement is not valid or it is used out of proper order, Umgebungsvariable, Environment Variable, SYSGET, %SYSGET, Einleseformat, Informat, Best., W.D, Gültigkeitsprüfung, Format Modifier, ?, ??, Rundungsfehler, Iteration, binäre Zahlendarstellung, ROUND(), Match-Merging, Sortieren, Umlaute, deutsche Umlaute, DBCS, double-byte character set, ODS HTML, Grafik, SAS/GRAPH, GRAPH, String-Verarbeitung, reguläre Ausdrücke, regular expression, RX.

1. Makrovariablen mit PROC SQL erzeugen

von Heinrich Stürzl

PROC SQL kann nicht nur Datenbanken abfragen, Views und Joins erstellen, sondern eignet sich auch hervorragend, um den Inhalt von Variablen einer SAS Tabelle (Data Set) in eine oder mehrere Makrovariablen zu schreiben. Im Gegensatz zur Standardmethode CALL SYMPUT() bietet PROC SQL dabei zusätzliche Möglichkeiten durch die Erstellung beliebiger **Wertelisten**. Die hierzu notwendige Syntax ist sehr einfach und setzt keinerlei Vorkenntnisse und Erfahrung mit PROC SQL voraus. PROC SQL gehört zum Basisumfang von SAS.

PROC SQL ist eine interaktive Prozedur, die solange aktiv bleibt, bis sie durch eine QUIT Anweisung (oder einen anderen Step) geschlossen wird. Die Variablen einer SAS Tabelle werden mit Hilfe der SELECT Anweisung ausgewählt und mit ihrer INTO Option in eine oder mehrere Makrovariablen geschrieben. Verschiedene Variablen werden in PROC SQL durch Komma (Blank reicht nicht) getrennt. Makrovariablen werden durch einen vorgestellten Doppelpunkt (:) gekennzeichnet. Optional können die Beobachtungen über eine WHERE Bedingung gefiltert oder über ORDER BY sortiert werden.
Syntax:

```
PROC SQL NOPRINT;
  SELECT <DISTINCT> var1<, var2> INTO :macrovar1<, :macrovar2>
    FROM libref.tabelle
    <WHERE bedingung>
    <ORDER BY variablenliste>
    ;
QUIT;
```

var1 steht dabei entweder für den Namen einer Variable der Tabelle oder einen beliebigen Funktionsausdruck, wobei (fast) **alle Data Step Funktionen** zulässig sind (Ausnahme: LAG(), DIF(), SOUND()).

Mit Hilfe der **DISTINCT** Option werden mehrfach enthaltene Ausprägungen (Dubletten) eliminiert, so dass jede vorkommende Ausprägung genau einmal (Unique Items) ausgegeben wird.

Bei der Erzeugung von Makrovariablen lassen sich 3 verschiedene Varianten unterscheiden.

1. Genau eine Ausprägung in einer Makrovariable speichern

2. Jede Ausprägung in einer anderen Makrovariable speichern

3. Mehrere Ausprägungen als Liste in einer Makrovariable speichern

Für die folgenden Beispiele wird die SAS Tabelle SASHELP.CLASS verwendet, welche fiktive Daten von Schülern enthält.

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

1.1 Genau eine Ausprägung pro Makrovariable speichern

Die Länge der erzeugten Makrovariablen entspricht der Länge der ausgelesenen Variable in der Tabelle.
Die Ausprägungen von Character-Variablen werden linksbündig, die numerischer Variablen rechtsbündig in die Makrovariable geschrieben.

Beispiel 1: Erste Beobachtung einer Tabelle in einer Makrovariable speichern
hier: Name des ersten Schülers in einer Makrovariable speichern

```
PROC SQL NOPRINT;
  SELECT name INTO :vorname
    FROM sashelp.class;
QUIT;
%PUT *&vorname*;
=>   *Alfred  *
```

Beispiel 2: Erste Beobachtung von 2 Variablen in 2 Makrovariablen speichern
hier: Name und Alter der jüngsten Schülerin in 2 Makrovariablen speichern

```
PROC SQL NOPRINT;
  SELECT name, age INTO :vorname, :alter
    FROM sashelp.class
    WHERE sex="F"
    ORDER BY age;
QUIT;

%PUT *&vorname* *&alter*;
=>   *Joyce   * *      11*
```

Durch eine Neudefinition der Makrovariablen mit %LET werden führende und nachfolgende Blanks eliminiert und die Länge der Makrovariable entsprechend reduziert.

```
%LET vorname=&vorname;
%LET alter=&alter;
%PUT *&vorname* *&alter*;
=>   *Joyce* *11*
```

Beispiel 3: Statistische Kenngrößen jeweils in einer Makrovariable formatiert speichern
hier: Mittleres Alter und Stichprobenumfang in 2 Makrovariablen speichern

```
PROC SQL NOPRINT;
  SELECT N(age), MEAN(age) FORMAT=6.2 INTO :Anzahl, :MeanAlter
    FROM sashelp.class;
QUIT;
%PUT *&Anzahl* *&MeanAlter*;
=> *      19* * 13.32*
```

Bei der Lösung derselben Aufgabe mit CALL SYMPUT() wäre ein Prozedur- und ein Datenschritt notwendig.

1.2 Jede Ausprägung in einer anderen Makrovariable speichern

Durch die Angabe einer Liste von n Makrovariablen werden die ersten n Beobachtungen in jeweils einer Makrovariable der Liste gespeichert. Interessanterweise enthalten die Makrovariablen hierbei keine zusätzlichen Blanks!
Die globale Makrovariable **SQLOBS** enthält die Anzahl der Listenelemente.

Beispiel 4: Die ersten 3 Beobachtungen in jeweils einer Makrovariable speichern
hier: Die Namen der 3 ältesten Schüler in 3 verschiedenen Makrovariablen speichern

```
PROC SQL NOPRINT;
  SELECT name INTO :name1-:name3
    FROM sashelp.class
    ORDER BY age DESC;
QUIT;

%PUT *&name1* *&name2* *&name3* *&SQLOBS*;
=> *Philip* *Ronald* *Mary* *3*
```

Im Log erscheint dabei folgende Note, die jedoch ignoriert werden kann.

```
NOTE: The query as specified involves ordering by an item that doesn't appear
  in its SELECT clause.
```

1.3 Mehrere Ausprägungen als Liste in einer Makrovariable speichern

Durch die Option **SEPARATED BY** *'Trennzeichen'* werden alle selektierten Beobachtungen als Liste in eine Makrovariable geschrieben. ***Trennzeichen*** steht für eine beliebige Zeichenkette, die jeweils zwischen die Ausprägungen gesetzt werden.
Die globale Makrovariable **SQLOBS** enthält die Anzahl der Listenelemente.

Beispiel 5: Die selektierten Beobachtungen einer Variable als Liste in einer Makrovariable speichern
hier: Die Namen aller Schülerinnen in einer Makrovariable speichern

```
PROC SQL NOPRINT;
  SELECT name INTO :namelist SEPARATED BY ' '
    FROM sashelp.class
    WHERE sex="F";
QUIT;

%PUT *&namelist*;
%PUT SQLOBS=&SQLOBS;
=> *Alice Barbara Carol Jane Janet Joyce Judy Louise Mary*
=> SQLOBS=9
```

Hinweis: Die Ausprägungen werden hierbei ohne überschüssige Blanks in die Makrovariable geschrieben!

Sollen die Ausprägungen von Character-Variablen jeweils in Anführungszeichen stehen, so kann man die Funktionen QUOTE() und TRIM() verwenden.

```
...
SELECT QUOTE(TRIM(name)) INTO :namelist SEPARATED BY ' '
...
=> *"Alice" "Barbara" "Carol" "Jane" "Janet" "Joyce" "Judy" "Louise" "Mary"*
```

Beispiel 6: Unique Items mit Hilfe der DISTINCT Option als Liste in einer Makrovariable speichern
hier: Die besetzten Altersklassen in einer Makrovariable speichern

```
PROC SQL NOPRINT;
  SELECT DISTINCT age INTO :altersklassen SEPARATED BY ', '
    FROM sashelp.class;
QUIT;

%PUT *&altersklassen*;
%PUT SQLOBS=&SQLOBS;
=> *11, 12, 13, 14, 15, 16 *
=> SQLOBS=6
```

Sinnvolle Anwendungen:
Wertelisten lassen sich auf diese Weise mit Hilfe von Makrovariablen transportieren und überall dort einsetzen, wo eine Liste von Elementen zulässig ist. Beispielsweise beim IN Operator einer Bedingung, bei der Übergabe von Initialwerten bei der Definition eines Array, für Referenzlinien in Grafiken (HREF bzw. VREF Option von PROC GPLOT), als Liste von Variablen, etc.

Beispiel 7: Bei allen formatierten Variablen einer Tabelle das Format entfernen, um sie z.B. unformatiert anzeigen zu lassen
hier: Ausgabe der Tabelle "sashelp.retail" mit PROC PRINT ohne Formate

```
* Die Namen aller formatierten Variablen in einer Makrovariablen speichern;
PROC SQL NOPRINT;
  SELECT name INTO :varlist SEPARATED BY ' '
    FROM DICTIONARY.COLUMNS
    WHERE LIBNAME="SASHELP" AND MEMNAME="RETAIL" AND FORMAT^="";
QUIT;
PROC PRINT DATA=sashelp.retail;
  FORMAT &varlist; * Variablen entformatieren;
RUN;
%PUT *&varlist*;
=> *SALES DATE*
```

Hinweis: Auf ähnliche Weise lassen sich alle Variablen mit bestimmten Eigenschaften wie Typ, Länge, Position, Label, Format, Informat, Index zu einer Liste zusammenfassen (Die Liste aller numerischen und alphanumerischen Variablen einer Tabelle sind als _NUMERIC_ und _CHARACTER_ bereits automatisch vordefiniert).

Beispiele für spezielle Variablenlisten:

- Die ersten 10 Variablen (VARNUM<=10)
- Alle Variablen mit einer Länge > 12 (LENGTH>12)
- Alle Variablen, deren Label den Text "[g/L]" enthält (INDEX(LABEL, "[g/L]")
- ...

Variablenlisten können an vielen verschiedenen Stellen verwendet werden, z.B. in den Anweisungen VAR, KEEP, DROP, FORMAT, RETAIN, ARRAY sowie bei vielen Funktionen.

Außerdem lassen sich diese Wertelisten zusammen mit der globalen Makrovariable SQLOBS und der Makrofunktion %SCAN für die Steuerung von Makroschleifen verwenden, wodurch eine Gruppenverarbeitung in Makros mit einfachen Mitteln möglich ist.

Beispiel 8: Gruppenverarbeitung in einem Makro
hier: Ausgabe mit PROC PRINT für jede Altersklasse

```
%MACRO gruppen;
  PROC SQL NOPRINT;
  SELECT DISTINCT age INTO :altersklassen SEPARATED BY ' '
    FROM sashelp.class;
  QUIT;

  %DO i=1 %TO &SQLOBS;
    %LET group=%SCAN(&altersklassen, &i);
    %PUT &i: *&group*;
    TITLE "Gruppe &i von &SQLOBS: Altersklasse=&group";
    FOOTNOTE;
    PROC PRINT DATA=sashelp.class;
      WHERE age=&group;
    RUN;
  %END;
%MEND;

%gruppen;
```

=> In diesem Beispiel werden die Daten der 6 Altersklassen 11, 12, 13, 14, 15, 16 in jeweils einem PROC PRINT angezeigt.

Hinweis: Bei geschachtelten Schleifen ist zu beachten, dass die globale Makrovariable SQLOBS mit jedem Aufruf von PROC SQL überschrieben wird!

Literatur

- SAS Institute Inc. (2000*). SAS Online Documentation, The SQL Procedure.*

2. Tückische Blanks im SAS Quellcode

von Heinrich Stürzl

Leerzeichen (Blanks) spielen im SAS Quellcode eine wichtige Rolle, da sie die verschiedenen Syntaxelemente voneinander trennen. Sobald im Quelltext anstelle eines Blanks ein anderes Zeichen steht, wird der Quellcode unbrauchbar. Dies gilt auch für Sonderzeichen, die vom Enhanced Editor wie ein Leerzeichen dargestellt werden z.B. Alt+255 (letztes darstellbares Zeichen im ASCII-Zeichensatz). Dadurch führt scheinbar richtiger Quelltext zu merkwürdigen Fehlermeldungen.

Dieses Phänomen kann bei bestimmten Emailprogrammen wie **Lotus Notes** auftreten, wenn SAS Quellcode in einer **Email im HTML-Format** verschickt wird! Durch die Codierung und Decodierung des Mail-Inhaltes können einfache Leerzeichen am Zeilenanfang durch Sonderzeichen ersetzt werden, die wie Blanks aussehen. Quelltext, der auf diese Weise "vorbehandelt" wurde, kann zu schwerem Kopfzerbrechen führen, wie das folgende, stark vereinfachte Beispiel zeigt.

```
DATA test;
   INPUT Num;
DATALINES;
1
2
3
RUN;
```

Dieser Quellcode führt zu folgenden Meldungen im Log:

```
NOTE: SCL source line.
35      INPUT Num;
     --
     180

ERROR 180-322: Statement is not valid or it is used out of proper order.

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.  When this step was
  stopped there were 0 observations and 0 variables.
```

Der entscheidende Hinweis zur Lösung des Problems liegt in den mit "-" markierten Stellen über der Fehlernummer 180. Die Markierungen weisen daraufhin, dass sich dort 2 unerwartete Zeichen befinden, die zwar im Log an dieser Stelle in der Zeile darüber angezeigt werden, aber auch dort leider unsichtbar sind. Die wahre Ursache des Problems erfährt man, wenn man den Quellcode in einen Hex-Editor wie UltraEdit-32 kopiert. Man findet so an dieser Stelle zweimal das Zeichen **"A0"x** gefolgt von einem echten Blank ("20"x).

Lösungen:
Zur Lösung des Problems braucht man also "nur" die unechten Blanks zu löschen und schon hat man wieder einwandfreien Quelltext. Bei längeren Quelltexten empfiehlt es sich, das falsche Blank zu kopieren und mit Edit/Replace automatisch ersetzen zu lassen. Noch einfacher geht es, wenn man den Quelltext **zuerst nach Word kopiert** und anschließend von dort mit einem erneuten Cut&Paste in den Enhanced Editor holt. Auf diese Weise werden die falschen Blanks automatisch durch richtige ersetzt.

Man könnte natürlich auch soweit gehen und fordern:
Wenn SAS Quelltext per Email verschickt wird, sollte die Email nicht HTML formatiert sein, sondern als reiner Text.

3. Umgebungsvariablen (Windows)

von Heinrich Stürzl

Unter Windows unterscheidet SAS zwischen **Windows** und **SAS Umgebungsvariablen** (environment variables). Dies sind Textvariablen, die innerhalb der jeweiligen Anwendung global gültig sind. Ein bekanntes Beispiel für eine Windows Umgebungsvariable ist TEMP bzw. TMP (Verzeichnis für temporäre Daten). Ein Beispiel für eine SAS Umgebungsvariable ist SASROOT (Hauptverzeichnis, in dem SAS installiert ist). Beide Arten von Umgebungsvariablen lassen sich mit Hilfe der Funktionen %SYSGET() und SYSGET() auslesen und damit ihre Informationen in SAS nutzen.

3.1 Windows Umgebungsvariablen

Windows Umgebungsvariablen können mit Hilfe des DOS Kommandos **SET** sowohl definiert als auch angezeigt werden. Dazu muss man vorher ein DOS Fenster aufrufen, z.B. über Start Ausführen und CMD eingeben. Im DOS Fenster gibt man das Kommando SET ein und drückt die Enter Taste. Anschließend erscheint eine Liste aller Windows Umgebungsvariablen mit ihren aktuellen Definition, die bei Windows NT etwa so aussieht:

```
;Path=C:\WINNT\system32;C:\WINNT
COMPUTERNAME=ELDEMAWS079190
ComSpec=C:\WINNT\system32\cmd.exe
FULLNAME=Stuerzl, Heinrich
HOMEDRIVE=C:
HOMEPATH=\
LOGONSERVER=\\EMCHDUDC003
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Os2LibPath=C:\WINNT\system32\os2\dll;
Path=C:\DMINT40\WIN32\Bin;C:\WINNT\system32;C:\WINNT;;C:\PROGRA~1\EXTRA!
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 7 Stepping 3, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0703
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINNT
TEMP=C:\TEMP
TMP=C:\TEMP
USERDOMAIN=EMCHDU
USERNAME=StuerzlHeinrich
USERPROFILE=C:\WINNT\Profiles\StuerzlHeinrich
WIN32DMIPATH=C:\DMINT40\WIN32
windir=C:\WINNT
```

Syntax des DOS Kommandos SET

SET <*variable*<=*Zeichenfolge*>>|</?>

SET ohne Parameter zeigt die aktuellen Windows Umgebungsvariablen an
SET /? zeigt die Hilfeseiten zum SET Kommando an
SET *variable* ohne Gleichheitszeichen zeigt alle Umgebungsvariablen an, die mit *variable* beginnen
SET *variable=Zeichenfolge* weist der Umgebungsvariable *variable* den Wert *Zeichenfolge* zu

Unter Windows NT und 2000 kann man die Windows Umgebungsvariablen auch interaktiv anzeigen und setzen über **Einstellungen Systemsteuerung System Registerkarte "Umgebung" (NT) bzw. Registerkarte "Erweitert" Button "Umgebungsvariablen" (2000).**

3.2 SAS Umgebungsvariablen

SAS Umgebungsvariablen gelten nur innerhalb des SAS Systems und können mit Hilfe der SAS System Option **SET** definiert werden. Es wird empfohlen, dies in der SAS Konfigurationsdatei SASV8.CFG (bzw. CONFIG.SAS vor Version 8) zu tun. Sie stehen dann in jeder SAS Session für die gesamte Dauer zur Verfügung.

Syntax der SET System Option in der Konfigurationsdatei

```
-SET variable Zeichenfolge1|"Zeichenfolge1"|(Zeichenfolge1 <Zeichenfolge2>)
```

Beispiele

```
-SET sasroot    C:\Programme\SAS
-SET sasfolder "C:\Eigene Dateien\SAS"  (falls Leerzeichen im Pfad)
-SET SASAUTOS  ("!sasroot\core\sasmacro"
                "!sasext0\graph\sasmacro"
                "!sasext0\stat\sasmacro"
               )
```

3.3 Umgebungsvariablen auslesen

Um auf eine (zuvor definierte) SAS oder Windows Umgebungsvariable in der Konfigurationsdatei oder in einer LIBNAME Anweisung zu verweisen, muss man der Umgebungsvariable ein Ausrufezeichen voranstellen (`!Umgebungsvariable`).

Eine allgemeinere Methode zum Lesen einer SAS oder Windows Umgebungsvariable bieten die Makrofunktion **%SYSGET()** und die Data Step Funktion **SYSGET()**. Mit Hilfe der Makrofunktion %SYSGET() lassen sich die Inhalte von Umgebungsvariablen in Titeln oder Fußzeilen anzeigen, einer Makrovariablen zuweisen, in einer LIBNAME Anweisung verwenden, etc. Die Funktion SYSGET() erlaubt es, im Datenschritt einer Charactervariablen (Standardlänge 200) den Inhalt einer Umgebungsvariablen zuzuweisen.

Syntax:

```
SYSGET("Umgebungsvariable")
%SYSGET(Umgebungsvariable)
```

Beispiel 1: Das in der Umgebungsvariable TEMP bzw. SASFOLDER definierte Verzeichnis jeweils als Bibliothek verfügbar machen

```
LIBNAME work2  "!temp";
LIBNAME my     '!sasfolder';
```

oder

```
LIBNAME work2  "%SYSGET(temp)";
LIBNAME my     "%SYSGET(sasfolder)";
```

Anmerkung: Damit die Makrofunktion vom Makroprozessor ausgeführt wird, sind doppelte Anführungszeichen zwingend notwendig.

Beispiel 2: Benutzerspezifische Informationen im Log anzeigen hilfreich z.B. in der AUTOEXEC.SAS

```
%PUT User:             %SYSGET(username); * LoginID;
%PUT Workstation:      %SYSGET(computername); * Workstationname;
%PUT Operation System: %SYSGET(os); * Betriebsystem;
%PUT SAS-Folder:       %SYSGET(sasfolder); * Profil;
```

Beispiel 3: Umgebungsvariablen im Titel verwenden

```
TITLE "User %SYSGET(username) on %SYSGET(computername) under %SYSGET(os)";
```

Beispiel 4: Liste aller SAS Umgebungsvariablen im Log anzeigen

```
PROC OPTIONS OPTION=SET;
RUN;
```

Die Ausgabe im Log Fenster ist nicht besonders übersichtlich, obwohl klar strukturiert in der folgenden Form:

```
SET=
  [Umgebungsvar1 = "Zeichenfolge1"|("Zeichenfolge1" "Zeichenfolge1" ...)]
  [Umgebungsvar2 = "Zeichenfolge1"|("Zeichenfolge1" "Zeichenfolge1" ...)]
```

Beispiel:

```
SET=[FT15F001 = 'FT15F001.DAT'] [sasext0 = "\\eldemads000007\program\sas-dmbs\SAS.V82\SAS"] [sasroot =
"\\eldemads000007\program\sas-dmbs\SAS.V82\SAS"] [SASFOLDER = "G:\SASDM\Staff\Stuerzl\saswin.v82"]
[onldoc = "\\eldemads000007\program\sas-dmbs\SASOnlineDocV8"] [MYSASFILES = !sasfolder\sasuser\] [SASCFG =
!sasroot\..\ClientFiles] [SASSAML = !sasext0\share\sasmacro\] [SASAUTOS = (                  "!sasroot\core\sasmacro"
    "!sasext0\graph\sasmacro"                 "!sasext0\intrnet\sasmacro"                "!sasext0\qc\sasmacro"
 "!sasext0\share\sasmacro"                "!sasext0\stat\sasmacro"               )] [SAMPSIO = (
"!sasroot\core\sample"                "!sasext0\base\sample"                "!sasext0\connect\sample"
"!sasext0\graph\sample"                "!sasext0\inttech\sample"                "!sasext0\intrnet\sample"
"!sasext0\qc\sample"                "!sasext0\share\sample"                "!sasext0\stat\sample"                )]
[SAMPSRC= (                "!sasroot\core\sample"                "!sasext0\base\sample"
"!sasext0\connect\sample"                "!sasext0\graph\sample"                "!sasext0\inttech\sample"
"!sasext0\intrnet\sample"                "!sasext0\qc\sample"                "!sasext0\share\sample"
"!sasext0\stat\sample"                )]
[INSTALL = (                "!sasroot\core\sastest"                "!sasext0\base\sastest"
"!sasext0\qc\sastest"                "!sasext0\stat\sastest"                )]
                Defines an environment variable
```

Anmerkung: Auch andere Betriebssysteme wie OS/2, Unix, OpenVMS verwenden Umgebungsvariablen. Diese können in ähnlicher Weise in SAS genutzt werden. Siehe hierzu die SAS Online Dokumentation des jeweiligen Betriebssystems.

Literatur

- SAS Institute Inc. (2000*). SAS Online Documentation.*
  In Base SAS nach dem Satz (the phrase) "environment variable" suchen

4. Das Einleseformat w.d ist nicht immer die beste Wahl

von Carina Ortseifen

Ein Fall aus der Beratungspraxis, Hinweis dazu kam von Frau Hildegard Jakobi, Kinderklinik am Universitätsklinikum Heidelberg.

Problemstellung

Die folgenden drei Werte sollen in eine numerische Variable einer SAS Tabelle eingetragen werden:

12.3
6
4.21

Welches Einleseformat (Informat) wählt man dafür? Spielt die Wahl überhaupt eine Rolle, worauf muss man achten?

Lösung 1

Der klassische Ansatz sieht dazu einen Datenschritt mit INPUT und DATALINES (bzw. LINES oder CARDS) Anweisung vor. (Die Anweisung PUT dient lediglich dazu, den eingelesenen Wert im Log-Fenster auszugeben.)

```
DATA test;
  INPUT zahl;
  PUT 'Zahl: ' zahl;
  DATALINES;
12.3
6
4.21
RUN;
```

Im Log-Fenster erscheint nach dem Ausführen des Programms:

```
Zahl: 12.3
Zahl: 6
Zahl: 4.21
```

Ohne Angabe eines speziellen Einleseformats erscheinen die Zahlenwerte korrekt.

Lösung 2

Was aber nun, wenn man ein Einleseformat angeben will, etwa um zwei Nachkommastellen festzulegen? Das geänderte Programm hat die Form:

```
DATA test;
  INPUT zahl 6.2;
  PUT 'Zahl: ' zahl;
  DATALINES;
12.3
6
4.21
RUN;
```

Im Log-Fenster erscheint nun nach dem Ausführen des Programms:

```
Zahl: 12.3
Zahl: 0.06
Zahl: 4.21
```

Während die Werte 12.3 und 4.21 korrekt eingelesen worden sind, erkennt SAS den ganzzahligen Wert 6 nicht.

Fazit
Das **Einleseformat w.d** ist für Zahlen mit Dezimalpunkt- bzw. wissenschaftlicher Notation angelegt. w legt die Breite des Einlesefeldes fest, d die Anzahl der Nachkommastellen. Enthält der einzulesende Wert einen Dezimalpunkt, wird der Wert d ignoriert. Enthält der einzulesende Wert dagegen keinen Dezimalpunkt, wie unsere 6, dann wird die Zahl durch 10^d dividiert. In unserem Beispiel, d=2, $6/10^d=0.06$.

Richtig wäre in diesem Fall, wenn man denn ein Einleseformat angeben will (oder muss), den Wert d, d.h. die Anzahl der Dezimalstellen, nicht zu spezifizieren.

```
DATA test;
  INPUT zahl 6.;
  PUT 'Zahl: ' zahl;
  DATALINES;
12.3
6
4.21
RUN;
```

```
Zahl: 12.3
Zahl: 6
Zahl: 4.21
```

Eine Alternative zu dem Einleseformat w. mit der festen Breite der Werte stellt BEST. dar, um dem SAS System die Wahl der Breite zu überlassen.

Literatur

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

5. Nochmals Einlesen - Gültigkeitsprüfung

von Carina Ortseifen

Hier wird eine eMail von Christian Höns aus der SAS-EDU Diskussionsliste vom 01.10.2002 aufgegriffen.

Problemstellung

Der Inhalt einer Textdatei soll zeilenweise in eine SAS Variable eingelesen werden und, falls es sich dabei um eine Zahl handelt, in eine numerische Variable übertragen werden. Die Textdatei hat folgende Form:

```
12
012
123..
123.4
123,45
-110+110
11E+1
POLIZEI: 110, schnell
```

Liest man die Textdatei direkt in eine numerische Variable ein, erhält man für jede Einlesezeile, die keinen gültigen Zahlenwert enthält, Fehlermeldungen der folgenden Form:

```
DATA test;
   INPUT num;
   DATALINES;
123..
RUN;

NOTE: Invalid data for num in line 4 1-5.
RULE:      ----+----1----+----2----+----3----+----4----+----
4          123..
num=. _ERROR_=1 _N_=1
```

Bei großen Textdateien ist dann in der Regel nach 50 Fehlermeldungen Schluss und der Datenschritt wird abgebrochen.

Lösung

Mit einem bzw. zwei Fragezeichen vor dem Einleseformat in der Anweisung INPUT (als **Format Modifier**) ändert man die Anzeige der Fehlermeldungen. Ein Fragezeichen (?) unterdrückt die Note „Invalid data for …“, zwei Fragezeichen (??) unterdrücken alle Notes und das Verändern der internen Variable _ERROR_.

```
DATA test;
   INPUT num ??;
   DATALINES;
123..
RUN;
```

Hinweis

Die Format Modifier können auch mit der Funktion INPUT verwendet werden: `num=INPUT(a, ?? best.);`

6. Rundungsfehler sind nicht immer offensichtlich

von Carina Ortseifen

Ein Fall aus der Beratungspraxis, Hinweis dazu kam von Frau Susanne Broll, Tierärztliche Hochschule Hannover. Der Weg zur Lösung von Grischa Pfister, Info Ware Heidelberg.

Problemstellung

In einem ersten Datenschritt werden mit Hilfe einer DO/END Schleife und OUTPUT Anweisung 10 Beobachtungen erzeugt. Die Variable i stellt den Iterator dar, der die Werte 0 bis 1 in 0.1er Schritten annimmt.

```
DATA test;
   DO i=0.1 TO 1 BY 0.1;
      OUTPUT;
   END;
RUN;
PROC PRINT;
RUN;
```

```
Obs     i

  1    0.1
  2    0.2
  3    0.3
  4    0.4
  5    0.5
  6    0.6
  7    0.7
  8    0.8
  9    0.9
 10    1.0
```

Im anschließenden Prozedurschritt, hier exemplarisch PROC PRINT, soll nun mit einer WHERE Anweisung auf eine einzelne Beobachtung gezielt zugegriffen werden:

```
PROC PRINT DATA=test(WHERE=(i=0.8));
RUN;
```

Anstelle der entsprechenden Zeile aus obiger Liste erscheinen im Log-Fenster folgende beide Meldungen:

```
NOTE: No observations were selected from data set WORK.TEST.
NOTE: There were 0 observations read from the data set WORK.TEST.
      WHERE i=0.8;
```

Ursachenforschung

Obige Meldung bereitet einige Kopfzerbrechen. Denn hätte man anstelle von 0.8 mit 0.5 geprüft, wäre das Problem nicht aufgetreten.

```
PROC PRINT DATA=test(WHERE=(i=0.5));
Run;
```

```
Obs     i

  5    0.5
```

Und prüft man nicht exakt ab, sondern berücksichtigt einen Toleranzbereich:

```
PROC PRINT DATA=test(WHERE=(i>0.799 and i<0.8));
Run;

Obs     i

  8    0.8
```

so kommt man auf die Idee, dass das Problem mit der Zahlendarstellung zu tun haben könnte und SAS ein Rundungsproblem hat. Doch wie weißt man so was nach? Das numerische Format lässt keine entsprechende Genauigkeit zu.

Um zu erkennen, welche tatsächlichen Zahlen durch den Iterator (0.1) erzeugt wurden, muss man ein binäres Format verwenden. In einem zweiten Datenschritt werden die Werte 0.1 bis 1 ‚von Hand' eingetragen. Danach verknüpft man die beiden Tabellen und gibt die Werte i bzw. j mittels PUT und einem binären Format aus.

```
DATA vergleich;
  INPUT j @@;
  DATALINES;
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
RUN;

DATA gesamt;
  MERGE test vergleich;
  PUT i=@15 i=:binary64. /
      j=@15 j=:binary64. ;
RUN;
```

```
i=0.1    i=0011111110111001100110011001100110011001100110011001100110011010
j=0.1    j=0011111110111001100110011001100110011001100110011001100110011010
i=0.2    i=0011111111001001100110011001100110011001100110011001100110011010
j=0.2    j=0011111111001001100110011001100110011001100110011001100110011010
i=0.3    i=0011111111010011001100110011001100110011001100110011001100110100
j=0.3    j=0011111111010011001100110011001100110011001100110011001100110011
i=0.4    i=0011111111011001100110011001100110011001100110011001100110011010
j=0.4    j=0011111111011001100110011001100110011001100110011001100110011010
i=0.5    i=0011111111100000000000000000000000000000000000000000000000000000
j=0.5    j=0011111111100000000000000000000000000000000000000000000000000000
i=0.6    i=0011111111100011001100110011001100110011001100110011001100110011
j=0.6    j=0011111111100011001100110011001100110011001100110011001100110011
i=0.7    i=0011111111100110011001100110011001100110011001100110011001100110
j=0.7    j=0011111111100110011001100110011001100110011001100110011001100110
i=0.8    i=0011111111101001100110011001100110011001100110011001100110011001
j=0.8    j=0011111111101001100110011001100110011001100110011001100110011010
i=0.9    i=0011111111101100110011001100110011001100110011001100110011001100
j=0.9    j=0011111111101100110011001100110011001100110011001100110011001101
i=1      i=0011111111101111111111111111111111111111111111111111111111111111
j=1      j=0011111111110000000000000000000000000000000000000000000000000000
```

Man erkennt, dass die Zahlen tatsächlich voneinander abweichen. Und warum das ganze bei i=0.5 funktioniert, wird auch deutlich.

Lösungsmöglichkeiten

Eine Möglichkeit besteht darin, dass man hinfort nur noch **ganzzahlige Iteratoren** verwendet und ggfs. im Anschluss an die Iteration durch 10 teilt.

Eine allgemeingültigere Möglichkeit, dieses „Feature“ zu umgehen, besteht in der Verwendung der Funktion ROUND(). Mit dieser Funktion kann man explizit auf eine bestimmte Nachkommastellenzahl runden.

```
DATA test;
  DO i=0.1 TO 1 BY 0.1;
     i=ROUND(i,.1);
     OUTPUT;
  END;
RUN;
PROC PRINT DATA=test(WHERE=(i=0.8));
RUN;
```

Für ältere Versionen von SAS gibt es zwar auch Probleme mit der Funktion ROUND(), diese können aber mittels eines Patchs behoben werden. (Für SAS 8.1 s.u. http://support.sas.com/techsup/unotes/SN/003/003510.html)

7. Verknüpfen von zwei Tabellen – Match Merging

von Carina Ortseifen

Auf dieses Problem hat mich während der letzten KSFE in Dortmund Herr S. Steinberg aufmerksam gemacht.

Problemstellung

Die folgenden beiden Tabellen dateiA und dateiB

```
DATA dateiA;
   INPUT id $ x;
   DATALINES;
A 1
RUN;
```

```
DATA dateiB;
   INPUT id $ y;
   DATALINES;
A 20
A  5
A 15
RUN;
```

sollen hinsichtlich der Werte der Variable id verknüpft („gemergt“) werden. Gleichzeitig soll der Wert der Variable x in Abhängigkeit von den Werten der Variable y geändert werden: Wenn y<10 gilt, erhält x den Wert 0.

Verknüpfen im Datenschritt unter Berücksichtigung einer ID-Variable, sogenanntes Match Merging, führt man üblicherweise mittels der Anweisungen MERGE und BY durch, die bedingte Wert-Zuweisung mit IF/THEN.

```
DATA gesamt;
   MERGE dateiA dateiB;
   BY id;
   IF y<10 THEN x=0;
RUN;
```

Betrachten wir uns aber das Ergebnis des Datenschritts, so müssen wir feststellen, dass etwas schief gelaufen ist und das Resultat nicht unseren Erwartungen entspricht:

```
PROC PRINT;
RUN;
```

Obs	id	x	y
1	A	1	20
2	A	0	5
3	A	**0**	15

Die Variable x sollte in der 3. Beobachtung den Wert 1 enthalten, nicht 0.

Ursachenforschung

Um heraus zu finden, welche Anweisung das Fehlverhalten verursacht, wird der Datenschritt in zwei Datenschritte aufgeteilt: Zunächst werden die beiden Tabellen verknüpft, in einem zweiten Datenschritt wird die bedingte Wertzuweisung durchgeführt:

```
DATA gesamt;
   MERGE dateiA dateiB;
   BY id;
RUN;

PROC PRINT;
RUN;
```

Obs	id	x	y
1	A	1	20
2	A	1	5
3	A	1	15

```
DATA gesamt;
   SET gesamt;
   IF y<10 THEN x=0;
RUN;

PROC PRINT;
RUN;
```

Obs	id	x	y
1	A	1	20
2	A	0	5
3	A	**1**	15

Sowohl die Verknüpfung als auch die bedingte Zuweisung für sich funktionieren richtig, wenn man sie getrennt ausführt. Die Ursache des Problems scheint daher in der Kombination von beiden in einem einzigen Datenschritt zu liegen.

Manche SAS-Anwender mögen sich mit obiger Variante zufrieden geben. Sie haben einen Weg gefunden, wie sie das Problem umschiffen können. Vielleicht nicht sehr elegant, aber Hauptsache, die Tabellen werden richtig verknüpft und die Zuweisung ist korrekt.

Doch wenn man es genau wissen möchte, um beim nächsten Mal nicht noch mal in diese Falle zu tappen, wirft man einen Blick in die Dokumentation, Handbuch (SAS Language Reference. S. 151ff) oder Help > Books and Training > OnlineDoc (> Base SAS Software > SAS Language Reference > Data Step Concepts > Reading, Combining and Modifying SAS Data Sets), und liest dort:

“SAS reads the descriptor information of each data set that is named in the MERGE statement and then creates a program data vector that contains all the variables from all data sets as well as variables created by the DATA step. …
…
SAS retains the values of all variables in the program data vector except those variables that were created by the DATA step ….”

Auf unseren Fall übertragen bedeutet dies, dass alle Variablen, insbesondere auch die Variable X ein automatisches Retain erhalten, weil sie mit der Merge Anweisung gelesen werden. Erst durch diese Eigenschaft wird der Wert von X (die 1), der ja nur aus einer einzigen Beobachtung gelesen wird, sooft vervielfältigt wie es Beobachtungen in der

zweiten Tabelle gibt. Falls nun der Wert von X wie hier durch die Abfrage in der 2. Beobachtung geändert wird (auf 0), so wird von nun an dieser Wert beibehalten, bis er geändert wird. (Oder, was für unser Beispiel aber keine Rolle spielt, die By-Gruppe wechselt. Der Wert wird nur innerhalb der By-Gruppe beibehalten und beim Beginn der nächsten By-Gruppe auf missing value zurückgesetzt.)

Lösungsmöglichkeiten

Da das Beibehalten von Werten (retaining) eine der wichtigen Eigenschaften des Match-Merging bei SAS ist, gibt es nur zwei Möglichkeiten, wie das Problem behoben werden kann:

1. Der Datenschritt wird, wie bereits gezeigt, in zwei Datenschritte aufgeteilt. So werden das Verknüpfen und das bedingte Zuweisen in unterschiedliche Schritte verlegt und Konflikte vermieden.

```
DATA gesamt;
   MERGE dateiA dateiB;
   BY id;
RUN;
DATA gesamt;
   SET gesamt;
   IF y<10 THEN x=0;
RUN;
```

2. Wenn die bedingte Zuweisung, aus welchen Gründen auch immer, im gleichen Datenschritt erfolgen soll, muss die Anweisung IF um ein ELSE ergänzt werden, damit die Zuweisung unabhängig von bestehenden Werten erfolgen kann.

```
DATA gesamt;
   MERGE dateiA dateiB;
   IF y<10 THEN x=0;
   ELSE x=1;
   BY id;
RUN;
```

3. Eine allgemeingültige Lösung (Lösung 2 gilt nur für unser konkretes Ausgangsbeispiel) erhält man, wenn man eine neue Variable z anlegt, deren Werte nicht behalten werden und der man entweder die Werte von x oder 0 zuweist:

```
DATA gesamt;
   MERGE dateiA dateiB;
   IF y<10 THEN z=0;
   ELSE z=x;
   BY id;
RUN;
```

Literatur

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

8. Sortieren mit deutschen Umlauten

von Carina Ortseifen

Problemstellung

Die Verarbeitung und Darstellung deutscher Umlaute im SAS System waren schon immer problematisch und auch mit der Version 8.2 hat sich das nicht geändert, wie der Beitrag „Probleme im Zusammenhang mit der Einstellung DBCS“ von Wolfgang Herff und der folgende aufzeigen.

Die Prozedur SORT wird gewöhnlich verwendet, um eine Tabelle auf- oder absteigend nach einer oder mehreren Variablen zu sortieren. Bei numerischen Variablen werden die Werte bei aufsteigender Sortierreihenfolge von klein nach groß angeordnet, bei Textvariablen von A nach Z. Umlaute treten, weil spezifisch deutsch, in den Originalbeispielen von SAS nicht auf. Möchte man folgende Stichwortliste

```
abc
Abc
ABC
Ahne
ähnlich
Ähnlichkeit
Ahnung
trüb
Trugbild
Überlebenszeitanalyse
Univariate Statistik
UNIVARIATE
UPLOAD
```

aufsteigend sortieren, erhält man, nachdem die Stichworte in eine SAS Tabelle test mit Variable text eingelesen wurden, als Ergebnis der Sortierung mit der Prozedur SORT ohne zusätzliche Optionen,

```
PROC SORT DATA=test;
   BY text;
RUN;
```

die folgende Reihenfolge (hier mit PROC PRINT visualisiert):

Obs	text
1	ABC
2	Abc
3	Ahne
4	Ahnung
5	Trugbild
6	UNIVARIATE
7	UPLOAD
8	Univariate
9	**abc**
10	**trüb**
11	**Ähnlichkeit**
12	**Überlebenszeitanalyse**
13	**ähnlich**

Folgende Prinzipien sind bei der Sortierung zu erkennen:

1. Großbuchstaben werden vor den Kleinbuchstaben angeordnet („ABC“ erscheint vor „abc“, „Überlebenszeitanalyse“ vor „ähnlich“. Das gilt nicht nur für das erste, sondern auch für die weiteren Zeichen („UPLOAD“ vor „Univariate“, aber nach „UNIVARIATE“).

2. Die Umlaute Ä, Ö, Ü usw. werden nach den ‚normalen’ Buchstaben A, B … Z einsortiert. („Ähnlichkeit“, „Überlebenszeitanalyse“ und „ähnlich“ erscheinen am Ende der Liste.)

Diese Sortierweise entspricht nicht den etwa von Wörterbüchern her gewohnten Vorstellungen. Vielmehr führt diese Sortiermethode bei längeren Listen oder Stichwortsammlungen, in denen Groß- und Kleinschreibung für gleiche Begriffe vorkommen kann, zu unübersichtlichen und verwirrenden Resultaten.

Lösungsmöglichkeiten

Um die Unterscheidung zwischen Groß- und Kleinbuchstaben aufzuheben (Prinzip 1), kann man alle Zeichen in Kleinbuchstaben (oder alternativ Großbuchstaben) verwandeln und den Sortiervorgang daran anschließen. Dies kann auf zwei Arten realisiert werden:

a) Man erzeugt in einem (weiteren) Datenschritt eine neue Textvariable und sortiert die Tabelle anschließend nach der neuen Variable:

```
DATA test2;
   SET test;
   textL=LOWCASE(text);
RUN;
PROC SORT DATA=test2;
   BY textL;
RUN;
```

b) Man verwendet die Prozedur SQL mit einem ORDER BY Ausdruck:

```
PROC SQL;
   CREATE TABLE test AS
   SELECT * FROM test
     ORDER BY LOWCASE(text);
QUIT;
```

(Mit CREATE TABLE test wird eine SAS Tabelle test angelegt. SELECT * wählt alle Variablen aus der nach FROM genannten Tabelle test. Im Log-Fenster erscheint nach Ausführen des Prozedurschritts die Note „The query as specified involves ordering by an item that doesn't appear in its SELECT clause.”, was allerdings keine negativen Auswirkungen auf die Bildung der Tabelle test hat.).

Das Ergebnis der beiden Varianten hat im Prinzip folgende Form (Die Variable textL wurde hier weggelassen und die Reihenfolge der Worte, die sich nur durch ihre Groß- und Kleinschreibung unterscheiden, wie ABC, Abc und abc, kann leicht variieren.):

```
Obs    text

  1    abc
  2    Abc
  3    ABC
  4    Ahne
  5    Ahnung
  6    Trugbild
  7    trüb
  8    UNIVARIATE
  9    Univariate
 10    UPLOAD
 11    ähnlich
 12    Ähnlichkeit
 13    Überlebenszeitanalyse
```

Das Problem mit der Groß- und Kleinschreibung ist damit gelöst, die gleich lautenden „Abc“s stehen beieinander, ebenso „UNIVARIATE“ und „univariate“. Verbleiben die deutschen Umlaute.

Ein möglicher Schritt zur Lösung des Umlaute-Problems (Prinzip 2) besteht im Zugriff auf geeignete Umwandlungstabellen (Katalogeinträge vom Typ TRANTAB). In der Version 8 werden diese während der Installation automatisch angelegt und im Katalog SASHELP.LOCALE gespeichert. Unter der Version 6 gibt es dazu in der Sample-Library ein Programm namens Trabase.sas, das die gewünschten Umwandlungstabellen mit Hilfe der Prozedur TRANTAB erstellt. Die Umwandlungstabellen für die deutschen Umlaute haben die Namen DESOLAT1 (German sort for ISO 8859/1-latin1) bzw. DESOWLT1 (German sort for Win cp1252-latin1). Man greift auf diese Tabellen sowohl bei der Prozedur SORT als auch bei SQL mit der Option SORTSEQ= zu:

```
PROC SORT DATA=test2 SORTSEQ=DESOLAT1;
   BY textL;
RUN;

PROC SQL SORTSEQ=DESOLAT1;
   CREATE TABLE test AS
   SELECT * FROM test
     ORDER BY LOWCASE(text);
QUIT;
```

```
Obs    text

  1    ABC
  2    Abc
  3    abc
  4    Ahne
  5    Ahnung
  6    ähnlich
  7    Ähnlichkeit
  8    Trugbild
  9    trüb
 10    Univariate
 11    UNIVARIATE
 12    UPLOAD
 13    Überlebenszeitanalyse
```

Die Worte, die mit einem Umlaut beginnen, erscheinen nun zwar nicht mehr am Ende der Liste, aber die mit Ä beginnenden Worte erscheinen nach den mit A beginnenden Worten („ähnlich“ nach „Ahnung“, „Überlebenszeitanalyse“ nach „UPLOAD“). Nach dem gleichen Muster werden Umlaute innerhalb der Worte behandelt („trüb“ nach „Trugbild“). D.h. mit den Übersetzungstabellen lassen sich nur einzelne Buchstaben richtig sortieren, nicht jedoch ganze Wörter wie in unserem Beispiel.

Abhilfe schafft hier die TRANSLATE Funktion. Im Datenschritt oder direkt im PROC SQL Schritt – nur letzterer wird hier ausgeführt – werden die Umlaute ääöü und ß in aou und s verwandelt und die Tabelle entsprechend sortiert.

```
PROC SQL;
   CREATE TABLE test AS
   SELECT * FROM test
     ORDER BY TRANSLATE(LOWCASE(text),'aous','ääöüß');
QUIT;
```

Obs	text
1	abc
2	Abc
3	ABC
4	Ahne
5	ähnlich
6	Ähnlichkeit
7	Ahnung
8	trüb
9	Trugbild
10	Überlebenszeitanalyse
11	UNIVARIATE
12	Univariate
13	UPLOAD

Nun endlich erhalten wir ein zufrieden stellendes Ergebnis, das „ähnlich“ vor „Ahnung“ und „trüb“ vor „Trugbild“ anordnet.

Hinweis

Eine kleine Einschränkung gilt leider auch hier: Da mit der TRANSLATE Funktion lediglich ß in s verwandelt wird, könnten in diesem Fall noch kleinere Schönheitsfehler auftreten. Eine Umwandlung in ss kann dagegen weit größere Probleme mit der Variablenlänge hervorrufen.

9. Probleme im Zusammenhang mit der Einstellung DBCS

von Wolfgang Herff

Einleitung

Ab Version 8 bietet SAS mit der Einstellung DBCS die Unterstützung sogenannter *double-byte character sets*, die z.B. zur Darstellung asiatischer Schriftzeichen benötigt werden. Diese Einstellung findet zwar im europäischen Sprachraum vermutlich seltener Verwendung, kann aber andererseits während der SAS-Installation – zumindest ab Version 8.2 - leicht versehentlich oder automatisch aktiviert werden. Die sich hieraus ergebenden Probleme werden u.U. erst viel später bemerkt, ohne dass der Zusammenhang mit der beim Setup gewählten Option DBCS direkt erkennbar wird. Diese Probleme betreffen die Darstellung von deutschen Umlauten in Grafiken sowie ODS HTML und sollen im folgenden näher erläutert werden. Außerdem wird eine Möglichkeit zur nachträglichen Abhilfe angegeben, ohne SAS neu installieren zu müssen.

Problembeschreibung

Bislang sind die folgenden beiden Probleme bekannt, die im Zusammenhang mit der DBCS-Unterstützung entstehen.

9.1 Fehlerhafte Darstellung deutscher Umlaute in Grafiken

Bekanntlich kann man in SAS unter Windows deutsche Umlaute in Grafiken mittels der beiden Grafikoptionen `DEVMAP=WINANSI` und `KEYMAP=WINANSI` in einer vorangehenden GOPTIONS-Anweisung korrekt darstellen. Das funktioniert – zumindest für die SAS-Schriftarten (Simplex, Swiss, etc.) - nicht mehr, wenn während der Installation die Einstellung DBCS gewählt wurde. In Titeln, Fußnoten, etc. werden dann statt der deutschen Umlaute gewisse Sonderzeichen (bzw. Leerzeichen) ausgegeben.

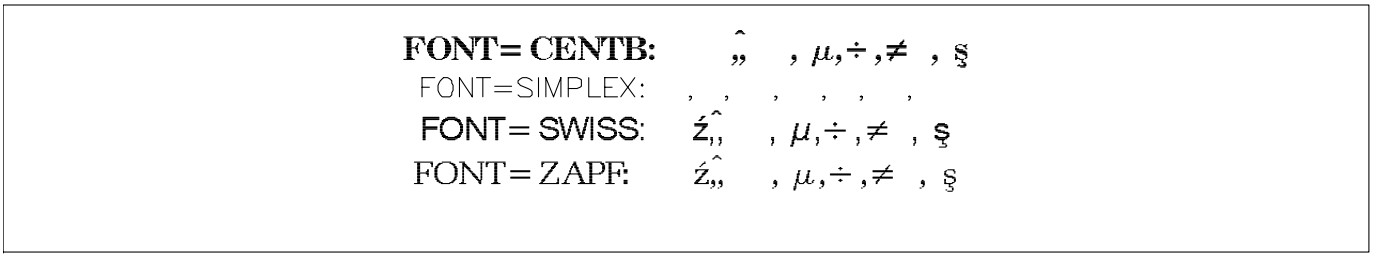
Da sich in Abhängigkeit des gewählten Fonts unterschiedliche fehlerhafte Darstellungen der deutschen Umlaute ergeben, werden in dem folgenden Beispielprogramm verschiedene (typische) SAS-Fonts verwendet (man vergleiche hierzu *SAS/Graph Software: Reference, Version 8*, p. 132/133).

Beispiel 1: Grafik-Ausgabe deutscher Umlaute für verschiedene SAS-Fonts

```
GOPTIONS RESET=ALL DEVMAP=WINANSI KEYMAP=WINANSI HTEXT=1.4 VSIZE=5cm;
TITLE1 F=CENTB   'FONT=CENTB:   Ä,Ö,Ü , ä,ö,ü , ß';
TITLE2 F=SIMPLEX 'FONT=SIMPLEX: Ä,Ö,Ü , ä,ö,ü , ß';
TITLE3 F=SWISS   'FONT=SWISS:   Ä,Ö,Ü , ä,ö,ü , ß';
TITLE4 F=ZAPF    'FONT=ZAPF:    Ä,Ö,Ü , ä,ö,ü , ß';

PROC GSLIDE BORDER;
RUN; QUIT;
```

Ohne DBCS-Unterstützung werden die deutschen Umlaute jeweils fehlerfrei dargestellt, mit aktivierter DBCS-Einstellung liefert dieses Programm jedoch die folgende Grafik:



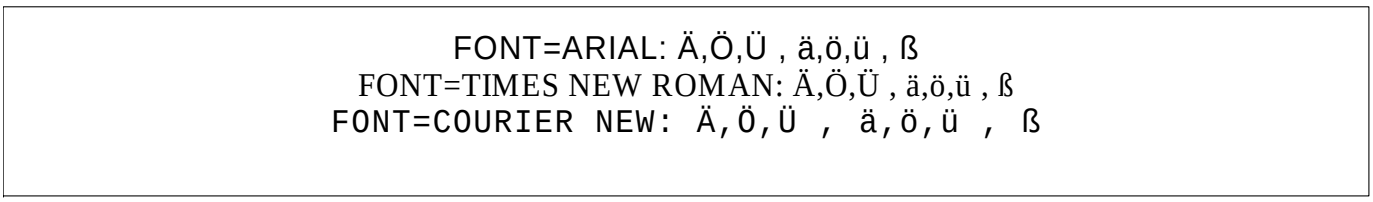
Bestimmte Grafik-Ausgabeformate, wie etwa das Format EMF zur Einbindung von SAS-Grafiken in MS-Office-Anwendungen, erlauben die Verwendung von TrueType-Fonts (man vergleiche hierzu Stürzl (2001)). Wie das folgende Beispiel zeigt, wird die Darstellung deutscher Umlaute in Grafiken mittels dieser TrueType-Fonts durch die DBCS-Einstellung offenbar nicht beeinflusst.

Beispiel 2: Grafik-Ausgabe deutscher Umlaute für verschiedene TrueType-Fonts

```
GOPTIONS RESET=ALL DEVMAP=WINANSI KEYMAP=WINANSI HTEXT=0.8cm VSIZE=4cm;
TITLE1 F='ARIAL' 'FONT=ARIAL: Ä,Ö,Ü , ä,ö,ü , ß';
TITLE2 F='TIMES NEW ROMAN' 'FONT=TIMES NEW ROMAN: Ä,Ö,Ü , ä,ö,ü , ß';
TITLE3 F='COURIER NEW' 'FONT=COURIER NEW: Ä,Ö,Ü , ä,ö,ü , ß';

PROC GSLIDE BORDER;
RUN; QUIT;
```

Dieses Programm erzeugt – unabhängig von der DBCS-Einstellung – die folgende Grafik:



9.2 HTML-Ausgaben mittels ODS können nicht erzeugt werden

HTML-Ausgaben mittels ODS (*Output Delivery System*) können nicht erzeugt werden, falls die DBCS-Unterstützung gewählt wurde. Statt dessen erhält man bestimmte Fehlermeldungen.

Beispiel 3: HTML-Ausgabe der Prozedur UNIVARIATE für die Variable `weight` des Datensatzes SASHELP.CLASS

```
ODS HTML file='d:\class.htm';
PROC UNIVARIATE DATA=sashelp.class;
  VAR weight;
RUN;
ODS HTML CLOSE;
```

Während ohne DBCS-Unterstützung die HTML-Datei einwandfrei angelegt wird, werden zu diesem Programm mit aktivierter DBCS-Einstellung die folgenden Fehlermeldungen im Log-Window ausgegeben.

```
ERROR:  Read Access Violation In Task ( Submit ]
Exception occurred at (63CC5C0D)
Task Traceback

1    ODS HTML file='d:\class.htm';
NOTE: Writing HTML Body file: d:\class.htm
ERROR: Generic critical error.
```

Im Unterschied zu HTML-Ausgaben konnte ich PRINTER-Ausgaben (PS bzw. PDF) sowie OUTPUT-Ausgaben mittels ODS auch mit aktivierter DBCS-Unterstützung erstellen.

Problemlösung

Man kann anhand der SAS-Systemoptionen überprüfen, ob die DBCS-Unterstützung aktiviert ist. Hierzu muß man im Menü `Tools` nacheinander die Menüpunkte `Options` und `System` aufrufen. Im zugehörigen Hilfsfenster wählt man dann die Menüpunkte `Environment control` und `Language control`. Die aktuelle DBCS-Einstellung wird durch den Wert 1 (DBCS aktiv) bzw. 0 (DBCS inaktiv) angezeigt.

Im Unterschied zu vielen anderen Systemoptionen, deren Einstellungen man (mittels rechter Maustaste) modifizieren kann, lässt sich der aktuelle DBCS-Wert **nicht** verändern. Wenn man eine Neuinstallation vermeiden will, kann man die DBCS-Unterstützung wieder deaktivieren, indem man die Konfigurationsdatei SASV8.CFG wie nachfolgend beschrieben modifiziert. (Man beachte hierzu auch die SAS-Notes SN-004927, SN-004943 und SN-005161.)

Während der Installation werden innerhalb der Datei SASV8.CFG automatische Eintragungen vorgenommen. Bei aktivierter DBCS-Einstellung werden diese Eintragungen um einige Zeilen, welche die DBCS-Unterstützung betreffen, ergänzt. Diese Zeilen müssen aus der Konfigurationsdatei entfernt werden, um die DBCS-Einstellung wieder zu deaktivieren.

Man sollte sich nicht davon abschrecken lassen, dass vor dem Editieren der automatisch erzeugten Eintragungen in der Datei SASV8.CFG ausdrücklich gewarnt wird, denn auch in den angegebenen SAS-Notes wird dieser Lösungsweg vorgeschlagen. Es empfiehlt sich allerdings, zuvor eine Sicherheitskopie der Konfigurationsdatei anzulegen.

Anhang

In diesem Anhang sind die für die DBCS-Unterstützung relevanten Blöcke innerhalb der Datei SASV8.CFG (zu SAS-Version 8.2, TS2M0) auszugsweise angegeben. Die zu löschenden Zeilen sind fett hervorgehoben und zusätzlich grau unterlegt.

```
/* Default resources location */
-RESOURCESLOC !sasroot\core\resource
-DBCS
-DBCSTYPE PCMS

/* Setup the SAS System help directory definition */
-SASHELP       (
                "!SASCFG\SASCFG"
                "!sasext0\dbcs\sashelp"
                "!sasroot\core\sashelp"

                        …

/* Setup the SAS System load image search paths definition */
-PATH          (
                "!sasext0\dbcs\sasexe"
                "!sasroot\core\sasexe"

                        …
```

Literatur

1. SAS-Notes SN-004927, SN-004943, SN-005161 (http://support.sas.com/techsup/unotes/SN/).
2. SAS Institute Inc. (1999): *SAS/Graph Software: Reference, Version 8.* Cary, NC, SAS Institute.
3. Stürzl, H. (2001): *Export von SAS Grafiken nach Microsoft Office 97 mit Version 8. Ein Vergleich von EMF und CGM.* Proceedings der 5. KSFE, S. 430-434.

10. Reguläre Ausdrücke in SAS Base

von Grischa Pfister

Reguläre Ausdrücke (regular expressions, RX) stellen Methoden zur Verfügung, die komplexe Analysen von alphanumerischen Variablen erlauben. Der hier zur Verfügung gestellte „Werkzeugkasten“ reicht deutlich über die bekannten Funktionen SCAN(), SUBSTR(), TRANSLATE() etc. hinaus.

Anwendungsmöglichkeiten für Reguläre Ausdrücke

Im Wesentlichen lassen sich drei Einsatzbereiche finden:

1. Durchsuchen eines Strings nach definierten Mustern
2. Gewichtung eines Strings abhängig vom Enthaltensein verschiedener Muster
3. Ersetzen von Teilen eines Strings, bzw. Umgruppierung einzelner Bestandteile

SAS Funktionen und Call Routinen für Reguläre Ausdrücke

Der „Werkzeugkasten“ besteht aus verschiedenen Funktionen und Call Routinen, die in einer bestimmten Reihenfolge aufgerufen werden müssen:

- **rx = rxparse(*pattern*);**
- **rc = rxmatch(rx, *charVar*);**
- **call rxsubstr(rx, *charVar*, *position* <, *length* <, *score*>>);**
- **call rxchange(rx, *times*, *charVar* <, *charVarNew*>);**
- **call rxfree(rx);**

Mit RXPARSE() wird ein Muster definiert. Die Funktion kompiliert die Suchanfrage und gibt eine Referenz zurück. Diese Referenz wird für die anschließenden Aufrufe benötigt, es können also auch mehrere Muster gleichzeitig definiert sein.
Die Funktion RXMATCH() testet, ob der Inhalt einer Variablen dem Suchpattern entspricht, und gibt den Index des ersten Treffers zurück. Mehr Informationen liefert die Call Routine RXSUBSTR(), *position* ist der Index des ersten Treffers, *length* ist die Gesamtlänge des passenden Teilstring und *score* ist die Summe einer im Suchpattern definierten Gewichtung nach enthaltenen Teil-Strings.
Mit Hilfe von RXCHANGE() können Ersetzungen vorgenommen werden, das Ergebnis wird entweder in der Ausgangsvariable (*charVar*) oder in einer neuen Variable (*newCharVar*) gespeichert. Dabei muss unbedingt darauf geachtet werden, dass die entsprechende Variable auch lang genug definiert ist. Die Variable *times* gibt an, wie viele Treffer innerhalb der Variable ersetzt werden sollen.
Die Call Routine RXFREE() muss aufgerufen werden, wenn ein Muster nicht mehr benötigt wird, denn sie ist dafür zuständig, den belegten Speicher wieder freizugeben.

Dieser „Werkzeugkasten“ ist wohl ursprünglich für den Einsatz in der SCL entwickelt worden, deshalb ergeben sich für die Nutzung im Data-Step folgende Besonderheiten:

1. Das Muster sollte bei der ersten Ausführung des Data-Steps kompiliert werden.
2. Mit Retain muss eine numerische Variable definiert werden, die die Referenz auf die Suchanfrage über die Data-Step Schleife erhält.
3. Am Ende des Data-Steps sollte der Speicher wieder freigegeben werden.

Allgemein formuliert sieht ein typischer Data-Step also so aus:

```
Data (drop = rx);
  Retain rx;
  Set lib.table End = eof;
  If ( _n_ = 1 ) Then rx = rxparse(ausdruck);
  ...
  If ( eof ) Then Call rxfree(rx);
Run;
```

Aufbau von Suchausdrücken

Suchausdrücke können sich aus verschiedenen Teil-Mustern zusammensetzen. Neben einer Reihe vordefinierter Muster können auch eigene Patterns definiert werden. Vordefiniert sind:

- $a oder$A Letter in Klein- oder Großschreibung ($'a-zA-Z')
- $d oder $D Ziffer ($'0-9')
- $l oder $L Letter in Kleinschreibung ($'a-z')
- $u oder $U Letter in Großschreibung ($'A-Z')
- $w oder$W Whitespace, also Leerzeichen, carriage return etc.
- und einige mehr

Eigene Patterns werden als Zeichenkette in Anführungszeichen mit dem Präfix „$" definiert, wie die Beispiele in der obigen Liste zeigen also z.B. „$'ABC'" oder „$'abc'". Es können auch Wertebereiche angegeben werden.
Zusätzlich gibt es eine Reihe von Modifizierern: Das Suffix „*" bedeutet, dass ein Muster null mal oder beliebig oft vorkommen soll, „+" dagegen, dass es mindestens einmal oder beliebig oft vorkommen kann. Das Zeichen „|" wird als „oder" verwendet, während „||" die klassische Konkatenierung meint. Durch Klammerung kann das Muster entsprechend gegliedert werden. Mit „@n" kann die genaue Spaltenposition, an der ein Muster enthalten sein soll, vorgegeben werden.

Daneben kann natürlich auch nach konstanten Strings gesucht werden wird der String dabei in Anführungszeichen gesetzt, wird die exakte Groß- Kleinschreibung beachtet. Außerdem sind die SAS-typischen Platzhalter „?" für ein beliebiges und „:" für mehrere beliebige Zeichen erlaubt.
Dies ist natürlich nur eine sehr unvollständige Darstellung der Syntax, die Online-Doc bietet hier genauere Auskunft.

Reguläre Ausdrücke im Einsatz

Es folgt eine Reihe von Beispielen, die die Einsatzmöglichkeiten von Regulären Ausdrücken grob skizzieren sollen.

Suchen nach einem konstanten String

```
Data _null_;
  rx = rxparse(" Test ");
  * rx = rxparse(" test ");
  * rx = rxparse(" 'Test' ");
  * rx = rxparse(" 'test' ");
  string = "dies ist ein Test";
  rc = rxmatch(rx,string);
  Put rc = string =;
  Call rxfree(rx);
Run;
```

Die Funktion RXMATCH() gibt für die ersten drei Ausdrücke 14, für das vierte Muster 0 zurück, da der String „test" nicht enthalten ist.

Suchen nach dem ersten Großbuchstaben

```
Data _null_;
  If ( _n_ = 1 ) Then rx=rxparse(" $U ");
  string = "dies ist ein Test";
  rc = rxmatch(rx,string);
  Put rc = string =;
  Call rxfree(rx);
Run;
```

Der Rückgabewert ist wiederum 14.

Suche nach einem Muster an einer bestimmten Stelle

```
Data _null_;
  If ( _n_ = 1 ) Then rx = rxparse(" @1 Test ");
  *If ( _n_ = 1 ) Then rx = rxparse(" @-8 'ein' ");
  string = "dies ist ein Test";
  rc = rxmatch(rx,string);
  Put rc = string =;
  Call rxfree(rx);
Run;
```

Das erste Muster ergibt eine 0, denn der String beginnt nicht mit dem Wort „Test“. Das zweite Muster platziert den Einlese-Zeiger auf das achte Zeichen vom Ende des Strings aus gesehen und überprüft, ob hier das Wort „ein“ folgt.
Ergebnis:

```
rc=0 string=dies ist ein Test
```

bzw.

```
rc=10 string=dies ist ein Test
```

Gewichten nach Vorkommen bestimmter Teil-Strings

```
Data Test;
  Length string $30;
  Input string & @@;
  cards;
  dies ist ein Test  auf der KSFE  dies ist ein Test auf der KSFE
Run;

Data _null_;
  Set Test End = eof;
  Length rx position length score 8;
  Retain rx;
  If ( _n_ = 1 ) Then rx = rxparse(" 'Test' #1 |'KSFE' #2 | 'Test' : 'KSFE' #3");
  Call rxsubstr(rx, string, position, length, score);
  Put position = length = score = string = ;
  If ( eof ) Then Call rxfree(rx);
Run;
```

In dieser Suchanfrage werden drei verschiedene Teilmuster definiert: Damit die Suchfunktionen einen Treffer melden, reicht es, wenn eines der Teilmuster gefunden wird („|“ bedeutet „oder“). Das erste Suchmuster erkennt also alle Strings, die das Wort „Test“ enthalten, das zweite nur „KSFE“, das dritte alle Strings, die „Test“ gefolgt von beliebigen Zeichen und „KSFE“ enthalten. Mit „#n“ wird eine Gewichtung zurückgegeben.
Ergebnis:

```
position=14 length=4 score=1 string=dies ist ein Test
position=9 length=4 score=2 string=auf der KSFE
position=14 length=17 score=3 string=dies ist ein Test auf der KSFE
```

Einfaches Ersetzen von Teilstrings

```
Data _null_;
  Set Test End = eof;
  Length rx rc 8 newString $ 40;
  Retain rx;
  If ( _n_ = 1 ) Then rx = rxparse(" 'Test' To 'Versuch', 'KSFE' To 'KSFE 2003'");
  rc = rxmatch(rx,string);
  Put rc=;
  If ( rc ) Then cCll rxchange(rx,5,string,newString);
  Put string=;
  Put newString= ;
  If ( eof ) Then Call rxfree(rx);
Run;
```

In den eingelesenen Strings werden “Test” mit “Versuch” und “KSFE” mit “KSFE 2003” ersetzt.

Vertauschen der Reihenfolge von Teilstrings

```
Data _null_;
  Length newString $30;
  rx = rxparse("<:> ' ' <$a*> To =2 ', ' =1");
  string = "Gaius Julius Caesar";
  rc = rxmatch(rx,string);
  If ( rc ) Then Call rxchange(rx,9,string,newString);
  Put rc= string= newString=;
  Call rxfree(rx);
Run;
```

Ein oft vorkommendes Problem sind Namensfelder, die einen oder mehrere Vornamen, sowie den Nachnamen enthalten. Auch hier können Reguläre Ausdrücke verwendet werden, um die Reihenfolge der Teil-Strings zu verändern. Das Muster zerlegt den String in eine beliebige Zeichenfolge, gefolgt von einem Leerzeichen und einer weiteren Zeichenkette (die allerdings keine Leerzeichen mehr enthalten darf). Um bei der Ersetzung auf die einzelnen Bestandteile des Strings zuzugreifen, werden spitze Klammern verwendet. Sie zerlegen das gefundene Muster in tokens, die dann bei der Ersetzung einzeln angesprochen werden. Im Beispiel wird der erste token aus einer Kombination beliebiger Zeichen gebildet – „<:>“. Ein Leerzeichen, gefolgt von einer Kombination von Buchstaben , trennt den zweiten Token ab (, ’ <$a*>). Damit wird immer das letzte, durch ein Leerzeichen getrennte Wort als Nachname erkannt und in den zweiten token gespeichert, während der erste Teil des eingelesen Strings im ersten Token gespeichert werden. Bei der Ersetzung wird mit „=2“ auf den zweiten token referenziert und er wird vorangestellt, „=1“ hängt das erste Teilmuster an.

Dateiliste einlesen

```
%Let dir = c:\*.txt;
Filename in Pipe "dir &dir";
Data _null_;
  Length file $256;
  Retain rx;
  If ( _n_ = 1 ) Then rx = rxparse(" (<$d*.$d*.$d*> ' '*
                                     <$d* ':' $d*> ' '*
                                     <$d* '.'* $d* '.'* $d*> ' '
                                     <$a*>) TO =4 ");
  Infile in End = eof;
  Input;
  If ( rxmatch(rx,_infile_) ) Then Do;
     Call rxchange(rx,1,_infile_,file);
     Put file=;
  End;
  If ( eof ) Then Call rxfree(rx);
run;
Filename in Clear;
```

Dieses zweite Beispiel zeigt nochmals den Einsatz der vordefinierten Patterns. Ziel ist es, eine Liste aller TXT-Dateien im Verzeichnis „C:\“ zu erstellen. Dafür wird ein Filename vom Typ Pipe auf das entsprechende DOS-Kommando gesetzt und das Ergebnis in einem Data-Step eingelesen. Mithilfe eines regulären Ausdruckes werden die Zeilen selektiert, die Dateinamen enthalten (nur diese Zeilen beginnen mit einem Datum, gefolgt von einer Uhrzeit und der Dateigröße), bei der anschließenden Ersetzung werden diese zusätzlichen Informationen entfernt, so dass nur der Dateiname übrig bleibt.

Zusammenfassung

Mit den regulären Ausdrücken ist ein mächtiges Werkzeug zur Verarbeitung von Strings hinzugekommen. Wie üblich lässt die Dokumentation zu wünschen übrig, aber immerhin steht die gesamte Syntax mit wenigen Beispielen in der Online-Hilfe unter der Funktion RXPARSE() zur Verfügung.

Literatur

- SAS Institute Inc. (1999*): SAS Online Doc*. Cary, NC: SAS Institute.
  Im globalen Index nach der Funktion rxparse() suchen.