Der PDV
Dein Freund und Helfer

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

Zusammenfassung

Wenn in einem DataStep Daten verarbeitet werden, laufen sie durch den Programm Daten Vektor - kurz PDV. In diesem Speicherbereich geschehen alle Manipulationen der Verarbeitung. Aber auch häufig ist er auch der Grund, warum ein Step sich manchmal anders verhält, als es eigentlich gedacht war.
In diesem Beitrag soll ein Einblick in die Funktionsweise und mögliche Fallstricke im Zusammenhang mit dem PDV aufgezeigt werden. Des Weiteren werden Möglichkeiten zum Debugging von DataSteps vorgestellt.

Schlüsselwörter: PDV, Variablen, Initialisierung, Fallstricke, DataStep, Debug

1 Der PDV

Den PDV – kurz für den **P**rogramm **D**aten **V**ektor – haben die meisten SAS Entwickler im Rahmen ihrer ersten Base Schulung kennengelernt. Bei vielen gerät dieser allerdings leider auch schnell wieder in Vergessenheit.
Der PDV beschreibt einen Speicherbereich im Rahmen der SAS DataStep Verarbeitung, der alle Variablen beinhaltet und in dem auch die Manipulationen geschehen. Speicherbereiche von Variablen werden in der Kompilationsphase eines Steps aufgebaut und auch initialisiert. In den meisten Programmiersprachen müssen Variablen explizit definiert werden. Bei SAS werden Entwickler jedoch durch viele implizite Mechanismen unterstützt. Sollte also eine Variable nicht explizit definiert worden sein, versucht SAS hier einen Typ und Attribute zu vergeben. Diese Automatik basiert auf den Attributen der Variable in der ersten Verwendung. Aber auch bestimmte Schlüsselwörter werden in der Kompilationsphase dazu verwendet, um die Merkmale des Programm Daten Vektors zu beeinflussen.

Verfügbare Datentypen sind lediglich „Zeichenketten“ und „Numerisch“. Diese werden je mit einer Breite definiert, welche bei Zeichenketten die Anzahl an Zeichen bestimmt, welche aufgenommen werden können bzw. die Anzahl Byte, die für die Speicherung einer Zahl zur Verfügung stehen. Um sich anderer Datentypen wie z.B. Datum nutzbar zu machen, werden in SAS Formate verwendet. Datumswerte werden dann als Zahl zwischen dem 1.1.1960 und dem gewünschten Wert abgelegt.

Beispiel:

Datum	Wert
01.01.1960	0
05.01.1960	4
24.12.1959	-8
26.03.2015	20173

Um den Wert für uns leserlich auszugeben kann, z.B. mit dem Format ddmmyyp10. die Zahl 20173 als 26.03.2015 dargestellt werden. Da Formate für die Ausgabe gedacht sind, gibt es auch ein Konstrukt, welches das Einlesen von Werten vereinfachen soll: Informate. Diese finden z.B. häufig beim Einlesen sequenzieller Dateien Verwendung. Diese beschreibenden Daten – oder Metadaten – einer Variable stellen die Rahmenbedingung für die DataStep Verarbeitung dar. Im Step besteht die Möglichkeit hierauf Einfluss zu nehmen.

length

```
length Variable <$> 6;
```

Um eine Variable einem Typ und einer Länge zuzuweisen, kann das `length` Statement verwendet werden. Der notwendige Parameter `Variable` bestimmt den Namen, ein folgender Zahlenwert die Länge. Werden nur diese beiden Parameter verwendet, wird die Variable als numerische Variable angelegt. Um diese als Zeichenkette zu definieren ist noch das `$` Zeichen zwischen Variablenname und -länge notwendig.

{in}format

```
{in}format Variable $6.;
```

Die Zuweisung von Formaten ermöglicht die Ausgabe von Werten in einer anderen Form. Ein Beispiel ist hier das Datumsformat ddmmyyp10. (siehe oben). Mit der Formatzuweisung werden allerdings auch implizit Länge und Typ einer Variablen definiert. Ein numerisches Format definiert eine Variable als eben solche.

attrib

```
attrib Variable length=$8 format=$6.;
```

Variablen mit mehreren Attributen explizit zu versorgen, ist die Aufgabe des attrib Statements. Das Beispiel erzeugt eine Zeichenkettenvariable der Länge 8 mit einem Ausgabeformat der ersten 6 Zeichen. Würde die Variable nur durch das Format $6. definiert werden, hätte sie die Länge 6!

retain

```
retain Variable <INITIALWERT>;
```

Soll eine Variable ihren Wert über die Iterationen eines DataSteps beibehalten und nicht bei jedem Durchlauf neu initialisiert werden, kann dies mit `retain` für einzelne Variablen erreicht werden. Das `retain` hat zunächst mal keinen Einfluss auf Typ, Format oder Länge. Dem Statement kann allerdings ein Initalwert mit übergeben werden, dann hat das Statement sehr wohl Einfluss auf Typ und Länge.

Neben den Statements an sich kommt es nun auch auf die Reihenfolge an. Das SAS System arbeitet sich von oben nach unten durch den Code des DataSteps und nutzt das erste Auftreten einer Variable bzw. der Definition von Metadaten zu einer Variablen für den PDV.
Es macht also einen Unterschied, ob man:

```
variable="Hallo";
format Variable $10.;
```

oder

```
format Variable $10.;
variable="Hallo";
```

schreibt. Im ersten Fall wird eine Variable vom Typ Zeichenkette mit der Länge 5 definiert – im zweiten Fall hat die Variable die Länge 10. Bei nachfolgend längeren Zeichenketten hat dies signifikante Auswirkungen – sie werden abgeschnitten!

Aber auch die Ableitung aus anderen Variablen hat Einfluss auf den PDV.

```
VarC = VarA !! VarB;
```

Bei der einfachen Verkettung von Zeichenketten bekommt die neue Variable eine Länge, die der Summe der Längen der beteiligten Variablen entspricht.

```
VarX = substr(VarB,1,5);
```

Auch bei der Anwendung von Funktionen wird die Länge und der Typ weiter gereicht. Hier allerdings immer im Kontext der Funktion. Die im Beispiel verwendete substr Funktion reicht hier die Länge von VarB weiter und nicht die Länge 5 mit der Zeichen extrahiert werden. Die `length` Funktion beispielsweise hingegen liefert einen numerischen Wert!

```
VarY = put(VarB,$6.);
```

Bei Verwendung von Formaten werden diese priorisiert angewandt. Im Beispiel bekommt VarY die Länge 6 obwohl VarB die Länge 10 hat.

```
set lib.tabellenname;
```

Mit dem `set` Statement werden auch die Metainformationen der einzulesenden Tabelle(n) dem PDV hinzugefügt. Sollte eine Spalte im Ausgabe DataSet mit anderen Attributen ausgegeben werden, ist es eine gute Praxis, die Spalte in der Tabelle des `set` Statements per `rename` umzubenennen, die neue Spalte explizit zu definieren und die alte Spalte per `drop` Statement aus der Ausgabe zu entfernen.

Neben den Variablen, die in der In- bzw. Ausgabetabelle enthalten sind, gibt es in jedem DataStep auch noch automatische Variablen, welche für die Steuerung verwendet werden können.

_N_
Da eine typische DataStep Verarbeitung eine Schleife darstellt, gibt es auch einen Schleifenzähler. In _N_ wird mit 1 beginnend die aktuelle Iteration des Steps gespeichert. Diese Information kann z.B. dazu verwendet werden, beim ersten Durchlauf eine bestimmte Initialverarbeitung durchzuführen.

ERROR
Treten Fehler auf, die syntaktisch falsch sind, brechen Programme ab. Fehler die aufgrund von Wertekonstellationen auftreten, werden ggf. nur als Note: im Log ausgewiesen und der Job läuft mit 0 durch. Eine Division durch Null erzeugt keinen Fehler. Allerdings wird dabei die Variable _ERROR_ verändert. Ist der Wert nicht 0 sind Fehler aufgetreten.

Abhängig von verwendeten Statements gibt es noch weitere Variablen. Wird mittels BY Statement eine Gruppenwechselverarbeitung durchgeführt, gibt es für jede BY Variable ein first.<by-variable> bzw. last.<by-variable>. Diese wird bei Erreichen des ersten bzw. letzten Elements der Gruppe mit 1 belegt.
Wird eine Datei per INFILE verarbeitet, steht hier die Variable _INFILE_ zu Verfügung, welche den kompletten InputBuffer beinhaltet und kann für Problemanalysen genutzt werden.

Automatische Variablen gelangen allerdings nicht in die Ausgabe eines DataSteps. Um die Werte zu übernehmen, müssen die Werte erst in eine „normale“ Variable übernommen werden.

2 Fallstricke

Wie im ersten Teil ersichtlich wird, stellt der PDV das zentrale Element der DataStep Verarbeitung dar. Treten Probleme auf, sind die Ursachen dafür sehr häufig in der Initialisierung zu finden. Leider führen viele Fehlersituationen nicht zu einem Abbruch der Verarbeitung sondern zu falschen Werten. Daher ist bei der Entwicklung nicht nur die syntaktische sondern auch die inhaltliche Prüfung wichtig.
Manchmal sind es nicht mal die Anpassungen an dem eigenen Programm, sondern eine Änderung der Rahmenbedingungen, die dazu führen, das ein DataStep nicht mehr korrekt arbeitet.

Nachfolgend werden einige Fallstricke aufgezeigt, die so oder ähnlich schon oft für Fehlersituationen gesorgt haben.

2.1 Neuer Verarbeitungsstrang

Im Laufe der Zeit werden immer weitere Anforderungen an die Datenverarbeitung gestellt und die Sourcen werden um diese erweitert. In dem (hier vereinfachten) Beispiel

```
length Nachname $ 10
       Vorname  $ 10
        ;
Nachname = "Meier";
Vorname  = "Susanne";
Name     = Nachname !! ", " !! Vorname;
```

wurde bisher der Name aus Nach- und Vorname gebildet. Name bekommt die Attribute aus der Verkettung und wird damit insgesamt 22 Zeichen breit.

Column Name	Type	Length
Aa Nachname	Text	10
Aa Vorname	Text	10
Aa Name	Text	22

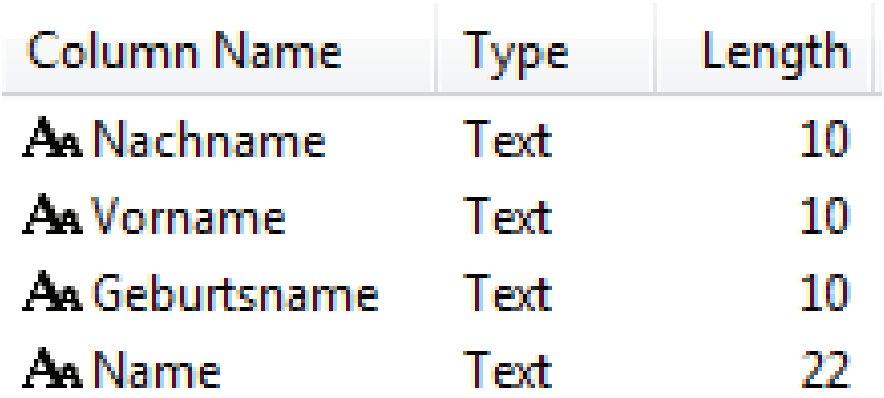
Abbildung 1: Attribute einer Tabelle

Zukünftig soll noch der Geburtsname mit in den Namen aufgenommen werden. Aber eben nur, wenn dieser auch belegt ist.

Eine mögliche Implementierung könnte wie folgt aussehen:

```
length Nachname $ 10
       Vorname  $ 10
       Geburtsname $ 10
           ;
Nachname    = "Meier";
Geburtsname = "Müller";
Vorname     = "Susanne";
if Geburtsname = " " then
   Name = Nachname !! ", " !! Vorname;
else
   Name = Nachname !! " (geb. "!! Geburtsname!! "), " !! Vorname;
```

Da hier der Fall, dass der Geburtsname leer ist als erstes aufgeführt wird, zieht bei der impliziten Definition von der Variable Name `Nachname !! ", " !! Vorname`. Hier bleibt es bei 22 Zeichen und der `else` Zweig wird sehr wahrscheinlich Daten abschneiden.



Column Name	Type	Length
Nachname	Text	10
Vorname	Text	10
Geburtsname	Text	10
Name	Text	22

Abbildung 2: Attribute einer Tabelle

2.2 Zahlen werden zu groß

Neben der Länge von Zeichenketten lässt sich auch die Länge von numerischen Variablen festlegen. Hierbei sind allerdings auch plattformabhängige Unterschiede zu berücksichtigen. Zwischen zOS und Windows / Unix ist ein unterschiedlicher Zahlenraum zu berücksichtigen. Dies wird bei der Migration von SAS Programmen schnell zum Verhängnis.

Tabelle 1: Zahlenbereich von Integerwerten je Plattform

Länge der Variable	Integer z/OS	Integer Windows/UNIX
2	256	nicht gültig
3	65.536	8.192
4	16.777.216	2.097.152
5	4.294.967.296	536.870.912
6	1.099.511.627.776	137.438.953.472
7	281.474.946.710.656	35.184.372.088.832
8 (default)	72.057.594.037.927.936	9.007.199.254.740.992

Bei Nachkommazahlen sollten immer mit der Länge 8 definiert werden, da sonst starke Abweichungen auftreten.

Beim Austausch von Daten über PROC CPORT / CIMPORT werden numerische Werte mit einer Länge unter 8 Byte beim Import um ein Byte verbreitert. Dies kann dann allerdings bei der Verknüpfung von Daten aufgrund der unterschiedlichen Längen zu Abbrüchen führen. Die Verbreiterung der Variablen kann im PROC CIMPORT kann durch die Option extendsn=no|yes beeinflusst werden.

2.3 Ungenauigkeiten durch Nachkommazahlen

Beim Rechnen mit Nachkommazahlen kommt es aufgrund der internen Darstellung der Zahlen zu Ungenauigkeiten, welche zu abweichendem Programmverhalten führen können.

```
data _null_;
  VarA= 0.3;
  VarB= 3 * 0.1;
  Diff= VarA - VarB;
  if Diff eq 0 then
     put "Keine Differenz";
  else
     put 'Differenz: ' Diff;
run;
```

Da 0.1 als endliche Zahl im Dezimalsystem für uns Menschen kein Problem darstellt ist die Erwartung oft auch an die EDV, das dies ebenfalls kein Problem ist. In dem obigen Beispiel wird eine Differenz von -5.55112E-17 ausgewiesen. Das ist zwar nicht viel, aber eben nicht gleich 0. Die `if` Abfrage läuft daher in den `else` Zweig.
Da diese Abweichung allerdings (in diesem Fall) vernachlässigbar ist, sollte mittels der `round` Funktion die gewünschte Genauigkeit selbst bestimmt werden:

```
if round(Diff,.000001) eq 0 then
```

So werden dann erst Abweichungen ab der sechsten Nachkommastelle als relevant ausgewiesen.

2.4 Subsetting IF / implizites & explizites Output

Im DataStep wird beim Erreichen des Endes eines Step implizit der PDV in die Ausgabetabelle(n) geschrieben. Wenn mittels Subsetting `if` (z.B. `if age > 14;`) die Werte Menge eingeschränkt werden soll, wird bei „wahr“ lediglich ein Flag für das implizite Output gesetzt. Liefert die Abfrage allerdings „falsch“ wird der Rest des Codes nicht ausgeführt und die nächste Iteration des Steps beginnt.
Wird statt des Subsetting IF ein explizites OUTPUT genutzt, wird die Observation genau zu diesem Zeitpunkt geschrieben. Alle nachfolgenden Statements werden zwar durchlaufen, haben aber keinen Einfluss auf den soeben geschrieben Satz, da durch ein explizites OUTPUT das implizite deaktiviert wird.

2.5 DB (Unicode)

Bei der Speicherung von Daten in einer DB2 sind Unterschiede in der Spaltendefinition zu berücksichtigen. SAS definiert die Breite als Anzahl Zeichen, DB2 in Anzahl Byte. In Unicode Systemen ist das allerdings nicht immer das gleiche. Umlaute wie ü, ä oder ö benötigen zwei Byte und können so zu Fehlern führen. Um die Spalte auf DB2 Seite explizit zu definieren, können in den DataSet Optionen die gewünschten Datentypen übergeben werden:

```
data dblib.tabelle(dbtype=(var='char(10)'));
```

So können Spalten mit genügend Puffer versehen werden und die Beladung kann erfolgreich abgeschlossen werden.

3 DataStep Debugging

Ist in einem DataStep eine Ausgabe produziert worden, die nicht zum anvisierten Ergebnis passt, wünscht man sich oft, bei der Verarbeitung nicht nur auf das Endprodukt sondern auch auf einzelne Zwischenschritte schauen zu können.

Oftmals werden für diese Zwischeninformationen im Programm die Variablen in das Log ausgegeben und damit in den Quellcode eingegriffen.

```
data Ausgabe;
set sashelp.cars;
if TYPE="SUV" then
    PUTLOG Make;
run;
```

Die Fragestellung, warum in bestimmten Konstellationen eine Schleife oder `if` Abfrage nicht durchlaufen wird oder wie zu diesem Zeitpunkt Variablen aussehen, kann im Debugging Modus des DataSteps leichter beantwortet werden.
Der interkative Debugger steht leider nur im klassischen Display Manger zur Verfügung. Im Batch oder in anderen Frontends wie dem Enterprise Guide kann leider nicht auf den interaktiven Debugger zugegriffen werden.

Um bei der Ausführung in den Debugger Modus zu wechseln muss lediglich die Option /debug dem `data` Statement hinzugefügt werden.

```
data Ausgabe /debug;
set sashelp.cars;
run;
```

Damit werden dann in der kompletten Programmausführung bei den so gekennzeichneten Steps das Debugger Log und der Debugger Source geöffnet. Das Log dient dabei als Ausgabe aller Ergebnisse und der Protokollierung der Aktionen und das Source Fenster dient der Steuerung.

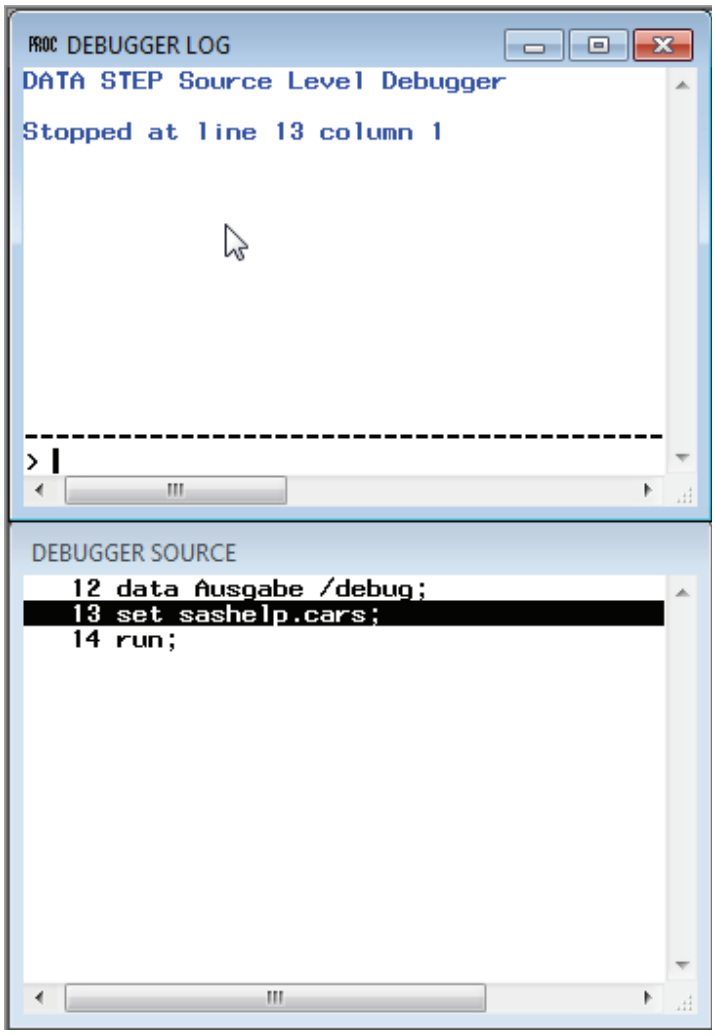


Abbildung 3: Debugger Fenster im Display Manager

Nun kann man sich Schrittweise durch den DataStep bewegen und zu jedem Zeitpunkt Werte abfragen oder sogar verändern. Für einzelne Schritte genügt die Eingabetaste oder das Kommando STEP (kurz ST). Bei komplexeren Steps kann es sinnvoll sein, nicht in Einzelschritten sondern in größeren Sprüngen durch das Programm zu wandern. STEP kann hierfür mit einer Anzahl ergänzt werden mit der dann z.B. 5 Schritte durchgeführt werden.
Um neuralgische Stellen zu kennzeichnen, können Unterbrechungspunkte definiert werden, an denen ein Step automatisch anhält. Mittels BREAK und einer Zeilennummer aus dem Debugger Source können solche Punkte definiert werden. Im Source werden diese Zeilen dann mit einem „!“ (Ausrufezeichen) gekennzeichnet. Mit GO wird dann von Unterbrechungspunkt zu Unterbrechungspunkt gesprungen. Um noch spezifischer zu springen, kann ein Unterbrechungspunkt mit einer Bedingung gekoppelt werden.

```
BREAK <ZEILE> WHEN VARIABEL=WERT
```

Um bei Werteveränderungen reagieren zu können, gibt es neben Unterbrechungspunkten auch Wertänderungsindikatoren. Mit WATCH VARIABLE wird bei jeder Wertveränderung die Ausführung gestoppt. Hierbei ist allerdings zu beachten, dass Variablen

bei einer neuen Iteration eines DataSteps auf missing reinitialisiert werden und der Indikator ebenfalls reagiert.
Bei der Ausführung ist darauf zu achten, dass zum Fortsetzen das Kommando GO verwendet wird. Die sonst zum Start verfügbaren Tasten F3 bzw. F8 stoppen die Verarbeitung sofort und führt dazu, dass unvollständige Daten verarbeitet werden!
Um Unterbrechungspunkte bzw. Wertänderungsindikatoren zu löschen, kann mit DELTE B <Zeile> bzw. DELTE W VARIABLE der Eintrag entfernt werden.

An einem relevanten Punkt im Debugging angekommen, können die Variablen mit EVALUATE (kurz E) dargestellt werden. Statt einzelner Variablennamen kann auch _ALL_ genutzt werden um den kompletten PDV auszugeben. Die Metadaten (Typ, Länge, Formate) können mit DESCRIBE (kurz DESC) ausgegeben werden. Auch hier können einzelne Variablennamen oder _ALL_ abgefragt werden.

Ein wichtiger Hinweis ist vor allem in gemeinsam genutzten Umgebungen interessant: Die DataSets sind, wie bei der normalen Ausführung auch, durch das SAS System gesperrt. Wenn nun ein komplexer DataStep mit vielen beteiligten Datasets analysiert wird, stehen die Tabellen für den gesamten Zeitraum keinem anderen zur Verfügung. In der Praxis haben sich hier eigene Testdatenbestände bewährt um den regulären Betrieb nicht durch Analysen zu blockieren.

Wie schon weiter oben erwähnt, ist der interaktive Debugger nur dem klassischen Display Manager vorbehalten. Um im Batch oder EG auf Debuginformationen zugreifen zu können, steht noch die Option /ldebug zur Verfügung. Dieser Batchmodus kann allerdings nicht mit viel Komfort aufwarten. In diesem Debugger Modus müssen dem DataSteps die Kommandos folgen, welche man sonst interaktiv eingegeben hätte. Besonders erschwerend ist die Identifizierung der Zeilennummer, in der ein Breakpoint gesetzt werden soll. Wichtig auch, dass zum Ende die Breakpoint zu entfernen und mit go das Programm weiter ausführen zu lassen. Wird das vergessen, entstehen unvollständige DataSets.

Ein Beispiel:

```
data BMI_OK BMI_low /ldebug;
  set sashelp.class;
  *if age > 14;*  then output;
  BMI = ( Weight / ( Height * Height ) ) * 703;
  if BMI < 15 then output BMI_Low;
  else output BMI_OK;
run;
break 5 when BMI<15
go
e _all_
delete b 5
go
quit
```

Zu beachten ist auch, dass direkt nach dem `run;` mit den Debugger Kommandos begonnen wird. Eine Leerzeile zur besseren Lesbarkeit wird bereits als EINGABETASTE interpretiert und springt damit schon in die nächste Zeile! Im Log sind dann die Informationen der DataStep ausführung und des Debuggers zu finden.

```
1    data BMI_OK BMI_low /ldebug;
2      set sashelp.class;
3      *if age > 14;*  then output;
4      BMI = ( Weight / ( Height * Height ) ) * 703;
5      if BMI < 15 then output BMI_Low;
6      else output BMI_OK;
7    run;

Stopped at line 2 column 3
  2   set sashelp.class;
DEBUG>
Breakpoint 1 set at line 5
DEBUG>
Break at line 5 column 3
  5   if BMI < 15 then output BMI_Low;
DEBUG>
Name = Joyce
Sex = F
Age = 11
Height = 51.3
Weight = 50.5
BMI = 13.4900007219695
_ERROR_ = 0
_N_ = 11
DEBUG>
Breakpoint 1 cleared at line 5
DEBUG>
The DATA STEP program has completed execution
  2   set sashelp.class;
DEBUG>
NOTE:  There  were  19  observations  read  from  the  data  set
SASHELP.CLASS.
NOTE: The data set WORK.BMI_OK has 18 observations and 6 variables.
NOTE: The data set WORK.BMI_LOW has 1 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.11 seconds
      cpu time            0.09 seconds
```

4 Fazit

Der Debugger, vor allem im Display Manager, ist ein sehr hilfreiches Werkzeug, um Probleme identifizieren zu könne. In erster Linie ist es aber das Verständnis über den PDV, welches viele Fallstricke vermeiden lässt und so Programme stabiler machen kann.

Literatur

[1] Numerical Accuracy in SAS Software
http://support.sas.com/documentation/cdl/en/lrcon/67885/HTML/default/viewer.htm#p0ji1unv6thm0dn1gp4t01a1u0g6.htm
[2] Data Set Options for DB2 Under UNIX and PC Hosts
http://support.sas.com/documentation/cdl/en/acreldb/63647/HTML/default/viewer.htm#a001348729.htm
[3] Dictionary of DATA Step Debugger Commands:
http://support.sas.com/documentation/cdl/en/lebaseutilref/63492/HTML/default/viewer.htm#n1ulcr5494fc5on1gg1cdzufez76.htm