Daten vertikal transportieren im Data Step: Implizites und explizites Retain und seine Alternativen

Renate Scheiner-Sparna
iOMEDICO AG
Hanferstraße 28
79108 Freiburg i.Br.
renate.scheiner-sparna@iomedico.com

Jörg Sahlmann
iOMEDICO AG
Hanferstraße 28
79108 Freiburg i.Br.
joerg.sahlmann@iomedico.com

Zusammenfassung

Wer mit SAS programmiert, lernt als einen der ersten Grundsätze, dass der Data Step Daten immer zeilenweise verarbeitet. Das ist nur die halbe Wahrheit.
Jeder kennt Momente, in denen er diesen Grundsatz zu ernst genommen hat und durch ungewollte, fortgeschriebene Einträge in Variablen bestraft wurde. Die Verarbeitungsregeln, nach denen der Datenvektor durchlaufen wird, sehen in einigen Fällen implizit ein Retain vor, ohne dass der Befehl im Programmcode vorkommt. Andererseits gibt es immer wieder Bedarf, Information im Data Set von einer Zeile in folgende Zeilen fortzuschreiben. Hierzu stehen dem Programmierer verschiedene Werkzeuge zur Verfügung.
Dieser Beitrag stellt zusammenfassend dar, nach welchen Regeln der SAS Supervisor das Behalten bzw. Zurücksetzen von Werten im Data Step steuert. Es wird aufgezeigt, welche Möglichkeiten man hat, die Weitergabe von Information in folgende Zeilen zu kontrollieren. Implizites Retain, explizites Retain-Statement, Point Option und der sogenannte DoW Loop werden erläutert.

Schlüsselwörter: RETAIN, Accumulator, point, DoW Loop, SAS Supervisor, Data Step

1 Begriffserklärung

Die in Tabelle 1 zusammengestellten Definitionen werden in diesem Beitrag vorausgesetzt.

Tabelle 1: Definition von Begriffen

Retain	Eine Variable behält den Wert, den sie von der vorherigen Zeile mitbringt.
Explizites Retain	Retain wird als Anweisung im Data Step genannt.
Implizites Retain	Eine Variable behält den Wert der vorherigen Zeile, obwohl es keine Retain-Anweisung gibt.
DoW Loop	Do Loop nach Whitlock
SAS Supervisor	Kontrollinstanz beim Ablauf von SAS Programmen

2 Der SAS Supervisor im Data Step

Der SAS Supervisor ist ein komplexer Kontrollmechanismus, der die Abfolge von Teilschritten und deren Aufgaben beim Ablaufen von SAS Programmen steuert sowie temporäre und permanente Ergebnisse generiert [1, 2]. Es handelt sich um einen sehr umfangreichen Prozess. Im Rahmen dieses Beitrags werden nur die Teilprozesse des Data Step erklärt, die für das Verständnis des Behaltens bzw. Ersetzens von Werten in Variablen hilfreich sind.

Man unterteilt den Prozess in eine Komplierungs- und eine Ausführungsphase. In der Komplierungsphase wird zunächst in einem Syntax Scan der Programmcode auf offensichtliche Fehler überprüft. Im Anschluss wird die Datenstruktur des neuen Datensatzes generiert mit Spalten und deren definierten Eigenschaften und Ablageort. Die temporäre Zähl-Variable _n_ und weitere interne Flags, die den Prozess steuern, werden generiert, ebenso wie der Program Data Vector (PDV) und der Initialize To Missing Vector (ITMV).

Der PDV ist ein interner, temporärer Speicher aller Variablen eines Data Sets. Variablen werden in der Reihenfolge ihrer Initialisierung hinzugefügt. Die letzten Inhalte der Variablen werden hier abgelegt und bereitgehalten.

Der ITMV bekommt für jede Variable (SAS interne, temporäre Variablen wie auch Data Set Variablen) einmalig in der Kompilierungsphase einen Wert. Dieser Wert kann nicht mehr geändert werden, bis der Data Step abgeschlossen ist. Dabei ist der Zeitpunkt der ersten Nennung einer Data Set Variablen sehr wichtig. Werte in diesem Vektor sind Y, N und R. Diese drei Werte sind beim Befüllen der Variablen in jeder Zeile von entscheidender Bedeutung.

In der Ausführungsphase werden die Datenzeilen in die Datenstruktur eingetragen. Grundlage hierfür ist der PDV. Die erste Datenzeile des/eines Ursprungsdatensatzes wird gelesen und die Zellen für das resultierende Data Set im PDV befüllt. Für diese Datenzeile wird nacheinander der weitere SAS Code des Data Step abgearbeitet. Ist dies erledigt, geht die Ausführung in die nächste Datenzeile. Jetzt wird anhand der Werte im ITMV für jede Variable einzeln entschieden, ob vor dem Lesen der neuen Datenzeile aus dem/einem Ursprungsdatensatz der Wert auf Missing zurückgesetzt wird oder nicht.

Y – Wert wird auf Missing gesetzt

N – Wert wird im PDV belassen

R – Keine der beiden Aktionen vor dem Lesen der neuen Datenzeile

Diese Schritte sind von zentraler Bedeutung für das explizite und implizite Retain, wie in den Beispielen erläutert wird.

3 Explizites Retain

Das explizite Retain ist leicht zu erkennen. Es gibt im Data Step ein Retain-Statement, in dem die Variablen genannt sind, deren Werte in folgende Zeilen fortgeschrieben werden sollen [3]. Im ITMV steht für diese Variable N, der Wert im PDV wird zwischen den Datenzeilen nicht entfernt [1, 2]. Es können nur neue Variablen, die nicht bereits in einem der Ursprungs-Data Sets vorliegen explizit als Retain-Variablen definiert werden.

Ein Beispiel, das auf dem nach der Variablen DEATHCAUSE sortierten SASHELP.HEART beruht:

```
1    data heartg;
2         set heart (where=(deathcause ne ''));
3         by deathcause;
4         retain deathgroup;
5         length deathgroup $20;
6         if first.deathcause and deathcause='Coronary Heart Disease'
7              then deathgroup='Heart';
8         else if first.deathcause and deathcause ne 'Unknown'
9              then deathgroup='Modified Other';
10        else if first.deathcause
11             then deathgroup='Unknown';
12   run;
```

Es handelt sich um eine Gruppierung der Gründe für das Versterben. Die Retain-Variable wird in Zeile 4 initialisiert. Wichtig ist, dass das Length-Statement erst nach dem Retain-Statement angeordnet ist. Nur bei der ersten Nennung der Variablen kann die Retain-Eigenschaft zugewiesen werden.

Durch die By-Bedingung in Zeile 3 kann in Zeilen 6, 8 und 10 die SAS-interne FIRST-Eigenschaft verwendet werden, mit der der Wert in der Retain-Spalte immer an erster Stelle der neuen Gruppe geändert werden kann. Abb. 1 zeigt das Ergebnis.

VIEWTABLE: WORK.HEARTG

	DEATHCAUSE	DEATHGROUP
1	Coronary Heart Disease	Heart
2	Coronary Heart Disease	Heart
3	Coronary Heart Disease	Heart
4	Coronary Heart Disease	Heart
5	Coronary Heart Disease	Heart
...	...	...
604	Coronary Heart Disease	Heart
605	Coronary Heart Disease	Heart
606	Cancer	Modified Other
607	Cancer	Modified Other
608	Cancer	Modified Other
609	Cancer	Modified Other
610	Cancer	Modified Other
...	...	...
1143	Cancer	Modified Other
1144	Cancer	Modified Other
1145	Cerebral Vascular Disease	Modified Other
1146	Cerebral Vascular Disease	Modified Other
1147	Cerebral Vascular Disease	Modified Other
...	...	...
1521	Cerebral Vascular Disease	Modified Other
1522	Cerebral Vascular Disease	Modified Other
1523	Other	Modified Other
...	...	...
1878	Other	Modified Other
1879	Other	Modified Other
1880	Unknown	Unknown
1881	Unknown	Unknown
1882	Unknown	Unknown

Abbildung 1: Neugruppierung - Ergebnis

Mit explizitem Retain können auch Konstanten einem Data Set hinzugefügt werden [7], wie das zweite Beispiel zeigt.

```
1    data cars;
2         set sashelp.cars;
3         retain constantn 20 constantc „SASHELP.CARS";
4    run;
```

Das resultierende Data Set CARS enthält eine Spalte mit dem Wert 20 und eine mit dem String SASHELP.CARS.
Als letztes Beispiel zum expliziten Retain soll eine typische Anwendung der klinischen Forschung stehen, die Imputation fehlender Werte mittels Last Observation Carried Forward (LOCF).

```
1    data impute;
2         set sysBP;
3         by ptno;
4         retain newsbp;
5         if first.ptno then newsbp=.;
6         if not missing(sbp) then newsbp=sbp;
7    run;
```

In Zeile 4 wird die Retain-Variable NEWSBP definiert. In jeder ersten Patientenzeile (FIRST.PTNO) wird diese Variable zunächst auf Missing gesetzt (Zeile 5). Das ist sehr wichtig, um nicht Werte eines Patienten in Zeilen des nächsten Patienten mitzunehmen. Diese Anweisung wird für jede erste Datenzeile ausgeführt, bevor mit der Syntax von Zeile 6 vorhandene Werte aus der Variablen SBP in die Retain-Spalte übernommen werden.

VIEWTABLE: WORK.IMPUTE

	PTNO	SBP	NEWSBP
1	1	120	120
2	1	160	160
3	1		160
4	1	140	140
5	2	100	100
6	2		100
7	2	115	115
8	2	108	108
9	3		
10	3	130	130
11	3	90	90
12	3	135	135
13	3		135

Abbildung 2: LOCF - Ergebnis

4 Implizites Retain

Das implizite Retain ist viel komplexer und von verschiedenen Bedingungen innerhalb des Data Step abhängig [1, 2]. Die Werte folgender Variablen werden beibehalten:

- Accumulator-Variablen, das sind Variablen links von einem + Zeichen
- Variablen, die nur in einem von mehreren Data Sets vorliegen und ein eigenes Set-Statement bekommen
- SAS-interne Variablen (z.B. _n_, _ERROR_)
- Bisher leere Variablen(-bereiche), die
    1) aus einem anderen als dem aktuell gelesenen Data Set stammen
    2) neu befüllt werden
    3) nicht für alle Zeilen angesprochen werden

4.1 Accumulator-Variable

In dem folgenden Beispiel erscheint in Zeile 3 erstmals der Accumulator COUNT. Bei dieser Art der Initialisierung braucht man kein Gleichheitszeichen, +1 reicht aus, um sowohl die Retain-Eigenschaft, als auch die Rechenoperation von einer Zeile zur nächsten zu definieren [4].

```
1    data implicit;
2         set SASHELP.CLASS;
3         count +1;
4         by age;
5         if first.age then count = 1;
6         if last.age then N_agegroup=count;
7    run;
```

Das Data Set ist nach der Variablen AGE sortiert. In Zeile 5 wird die Variable COUNT wieder auf 1 gesetzt, wenn ein neuer Jahrgang beginnt. In jeder letzten Zeile eines Jahrgangs wird der Wert von COUNT in eine neue Variable N_AGEGROUP geschrieben. Abb. 3 zeigt das Ergebnis.

VIEWTABLE: WORK.IMPLICIT

	NAME	SEX	AGE	COUNT	N_AGEGROUP
1	Joyce	F	11	1	.
2	Thomas	M	11	2	2
3	Louise	F	12	1	.
4	James	M	12	2	.
5	John	M	12	3	.
6	Jane	F	12	4	.
7	Robert	M	12	5	5
8	Alice	F	13	1	.
9	Jeffrey	M	13	2	.
10	Barbara	F	13	3	3
11	Carol	F	14	1	.
12	Henry	M	14	2	.
13	Judy	F	14	3	.
14	Alfred	M	14	4	4
15	Janet	F	15	1	.
16	Mary	F	15	2	.
17	William	M	15	3	.
18	Ronald	M	15	4	4
19	Philip	M	16	1	1

Abbildung 3: Accumulator – Ergebnis

4.2 Variable aus bedingtem Set

Variablen aus Data Sets, die allein in einem bedingten Set-Statement verwendet werden, sind ebenfalls implizit Retain-Variablen. Das kann man nutzen, um eine Konstante an alle Zeilen eines Data Sets anzufügen und weiter zu verwenden [5]. Im folgenden Beispiel wird zunächst eine Summe mit PROC MEANS berechnet und in ein Data Set OVERALL geschrieben. Im folgenden Data Step wird diese Summe als Spalte an das ursprüngliche Data Set angefügt.

```
1    proc means data=SASHELP.PRDSALE noprint;
2         var actual;
3         output out=OVERALL sum=ALLSALES;
4    run;
5
6    data percents;
7         set SASHELP.PRDSALE;
8         if _n_=1 then set OVERALL;
9         percent=100*actual/allsales;
10   run;
```

Auf diese Weise kann man nur genau ein Data Set mit IF-Bedingung anfügen. Abb. 5 zeigt das Ergebnis.
Es könnten mehrere Konstanten zuvor in einem Data Set zusammengefasst werden, das dann mit der IF-Bedingung wie in Zeile 8 einzusetzen wäre.

VIEWTABLE: WORK.OVERALL

	TYPE	_FREQ_	ALLSALES
1	0	1440	$730,337.00

Abbildung 4: Proc Means – Ergebnis

VIEWTABLE: WORK.PERCENTS

	ACTUAL	ALLSALES	PERCENT
1437	$251.00	$730,337.00	0.034367696
1438	$526.00	$730,337.00	0.0720215462
1439	$652.00	$730,337.00	0.0892738558
1440	$573.00	$730,337.00	0.0784569315

Abbildung 5: Bedingtes Set – Ergebnis

Lässt man die IF-Bedingung weg und ersetzt Zeile 7 durch
`set SASHELP.PRDSALE OVERALL;`
erscheint das Ursprungs-Data Set unverändert und darunter die eine Zeile aus OVERALL mit der Variablen
ALLSALES. (Abb. 6).

VIEWTABLE: WORK.PERCENTS1

	ACTUAL	ALLSALES	PERCENT
1437	$251.00	.	.
1438	$526.00	.	.
1439	$652.00	.	.
1440	$573.00	.	.
1441	.	$730,337.00	.

Abbildung 6: Normales, kombiniertes Set – Ergebnis

Setzt man noch ein weiteres Data Set mit einer weiteren Konstante hinter OVERALL in Zeile 8, bleibt diese Spalte leer. Jedes weitere Data Set würde eine neue Zeile mit IF-Bedingung brauchen.

4.3 Variable aus bedingtem Set mit By-Gruppierung

Analog dem Beispiel aus Abschnitt 3.2 ist es auch möglich, Konstanten pro By-Group anzufügen. Hierzu ein komplexeres Beispiel aus der Onkologie. Einer Reihe von Untersuchungsbefunden (Target Lesions) sollen pro Patient drei konstante Variablen aus einem separaten Data Set (io.IG_RS_TUMRESPDONEBASE) angefügt werden. Ein By-Statement (Zeile 9) erlaubt das Ansprechen der jeweils ersten Zeile (Zeile 11) der By-Gruppe und bietet die Möglichkeit, den Variableninhalt aus dem PDV zu löschen und neu zu befüllen.

```
1 data lesions;
2   set
3     io.IG_TargetLesion1 (in = in1)
4     io.IG_TargetLesion2 (in = in2)
5     io.IG_TargetLesion3 (in = in3)
6     io.IG_TargetLesion4 (in = in4)
7     io.IG_TargetLesion5 (in = in5)
8   ;
9   by USUBJID;
10   * add constants per patient with conditional set;
11   if first.USUBJID then set io.IG_RS_TUMRESPDONEBASE
12    (keep=USUBJID ID_RS_RSSTATBONE ID_RS_RSSTAT 13 ID_RS_RSDTC);
14   if in1 then Lesion = "TL1";
15   if in2 then Lesion = "TL2";
16   if in3 then Lesion = "TL3";
17   if in4 then Lesion = "TL4";
18   if in5 then Lesion = "TL5";
19 run;
```

Abb. 7 zeigt einen Ausschnitt aus dem Ergebnis.

VIEWTABLE: WORK.LESIONS

	USUBJID	LESION	ID_RS_RSSTATBONE	ID_RS_RSSTAT	ID_RS_RSDTC
1	P0318000001-001	TL1		y	2016-10-20
2	P0318000001-001	TL2		y	2016-10-20
3	P0318000001-001	TL3		y	2016-10-20
4	P0318000001-001	TL4		y	2016-10-20
5	P0318000001-001	TL5		y	2016-10-20
6	P0318000001-002	TL1		y	2016-10-11
7	P0318000001-002	TL2		y	2016-10-11
8	P0318000001-002	TL3		y	2016-10-11
9	P0318000001-002	TL4		y	2016-10-11
10	P0318000001-002	TL5		y	2016-10-11

Abbildung 7: Bedingtes Set mit By-Gruppierung - Ergebnis

4.4 Variablen aus einem von mehreren Data Sets

Für das folgende Beispiel ist es notwendig, die Arbeitsweise des SAS Supervisor bei der Anwendung des ITMV und der Abfolge von Teilschritten im Detail zu verstehen [1,2].

Es liegen zwei Data Sets vor, die zum Teil unterschiedliche Spaltennamen haben:

VIEWTABLE: WORK.ECOGN

	VARIABLE	RSTGE	RSTGN	RSTFF	RSTG
1	0	57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)
2	1	102 (51,5%)	172 (51,7%)	94 (49,0%)	429 (52,4%)
3	2	39 (19,7%)	34 (10,2%)	8 (4,2%)	88 (10,7%)

VIEWTABLE: WORK.FECOGN

	VARIABLE	COLGE	COLGN	COLFF	COLG
1	ECOG (n)	198	333	192	819

Abbildung 8: Bereits vorhandene Data Set Variablen befüllen – Ausgangslage

Die in Abb. 8 gezeigten Data Sets werden mit einem Set-Statement zusammengefügt. Es entsteht ein Data Set mit allen Spalten und vielen leeren Zellen (Abb. 9).

VIEWTABLE: WORK.REP_FINAL

	VARIABLE	COLGE	COLGN	COLFF	COLG	RSTGE	RSTGN	RSTFF	RSTG
1	Anzahl Patienten (N)	198	333	192	819				
2	ECOG (n)	198	333	192	819				
3	0					57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)
4	1					102 (51,5%)	172 (51,7%)	94 (49,0%)	429 (52,4%)
5	2					39 (19,7%)	34 (10,2%)	8 (4,2%)	88 (10,7%)

Abbildung 9: Bereits vorhandene Data Set Variablen befüllen – einfaches Set

Der folgende Data Step wurde erstellt, um die leeren Zellen der mit COL beginnenden Spalten mit den Werten aus den mit RST beginnenden Spalten zu füllen. Ein weiteres Data Set FAGE70 wird zusätzlich zu den beiden in Abb. 8 gezeigten verwendet, es hat die selbe Struktur wie ECOGN. FFULL enthält die spätere Zeile 1 in REP_FINAL.

```
1    data rep_final;
2      set ffull fecogn ecogn fage70;
3         if missing(colge) then colge=rstge;
4         if missing(colgn) then colgn=rstgn;
5         if missing(colff) then colff=rstff;
6         if missing(colg)  then colg =rstg;
7    run;
```

Der SAS Supervisor schreibt für jede der Variablen den Wert R (read) in den ITMV. In der Ausführungsphase wird jede Zeile einzeln bearbeitet.

- Es wird zuerst entschieden, welches Data Set gerade die Daten liefert. Ist es ein anderes als das vorher bearbeitete Data Set, wird der PDV (initiate to missing). Ist es das bisher gelesene Data Set, bleiben die Werte aus der Zeile vorher im PDV erhalten.
- Die Werte aus dem gewählten Data Set gelesen und in den PDV eingefügt.
- Die weiteren Data Step Code-Zeilen werden abgearbeitet.
- Die Ergebniszeile wird behalten.

Das Ergebnis des genannten Data Steps erfüllt nicht seinen Zweck. In Abb. 10 ist es gezeigt.

VIEWTABLE: WORK.REP_FINAL

	VARIABLE	COLGE	COLGN	COLFF	COLG	RSTGE	RSTGN	RSTFF	RSTG
1	Anzahl Patienten (N)	198	333	192	819				
2	ECOG (n)	198	333	192	819				
3	0	57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)	57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)
4	1	57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)	102 (51,5%)	172 (51,7%)	94 (49,0%)	429 (52,4%)
5	2	57 (28,8%)	127 (38,1%)	90 (46,9%)	302 (36,9%)	39 (19,7%)	34 (10,2%)	8 (4,2%)	88 (10,7%)
6	Age >= 70	167 (84,3%)	186 (55,9%)	31 (16,1%)	436 (53,2%)	167 (84,3%)	186 (55,9%)	31 (16,1%)	436 (53,2%)

Abbildung 10: Bereits vorhandene Data Set Variablen befüllen – implizites Retain

Es werden alle leeren Zellen mit dem ersten Satz Werten befüllt, es liegt also ein implizites Retain vor. Die Erklärung liegt im Unterschied zwischen den Data Set Zeilen 3 und 4. In Zeile 3 werden Werte aus einem neuen Data Set (FECOG) gelesen der PDV wird vorab geleert. Die Abfrage `if missing(col...)` findet tatsächlich Missing im PDV vor und trägt die RST... Werte für Zeile 3 ein. In Zeile 4 handelt es sich immer noch um FECOG, daher wird der PDV an dieser Stelle nicht geleert. Die Abfrage `if missing(col...)` findet keine Missings und trägt daher keine neuen Werte ein. Genauso wird Zeile 5 erzeugt. Data Set Zeile 6 stammt aus einem neuen Ursprungs-Data Set, FAGE70. Der PDV wird daher vor Bearbeitung dieser Zeile geleert und die Missing-Bedingung vor Befüllung mit den Werten aus den RST... Variablen funktioniert.
An dieser Stelle wird klar, dass es sich bei dem impliziten Retain nicht um einen Fehler handelt, sondern ein Denkfehler des Programmierers vorliegt. Das Problem lässt sich mit einem in-Operator lösen:

```
1    data rep_final;
2      set ffull fecogn fecog (in=fill) fage70 (in=fill);
3         if fill then do;
4              colge=rstge;
5              colgn=rstgn;
6              colff=rstff;
7              colg =rstg;
8         end;
9    run;
```

4.5 Leere Spalten aus einem Set oder Merge

Für die Erstellung eines standardisierten Datensatzes kann dessen Struktur mit allen Spalten und deren Eigenschaften als erster Schritt erstellt und dann nach und nach mit Daten befüllt werden. Das hat den Vorteil, dass die Spalten tatsächlich immer dieselben Eigenschaften haben. Im nächsten Schritt wird diese Datenstruktur mit einer Datenquelle zusammengeführt und die leeren Spalten nach und nach befüllt.
Der folgende Ausschnitt aus einem SAS Programm veranschaulicht diesen Prozess in der SDTM-Programmierung.

```
data template;
  attrib
    STUDYID label = "Study Identifier"                    length = $8
    DOMAIN  label = "Domain Abbreviation"                 length = $2
    USUBJID label = "Unique Subject Identifier"           length = $20
    SUBJID  label = "Subject Identifier for the Study"    length = $20
    RFSTDTC label = "Subject Reference Start Date/Time"   length = $10
    AGE1    label = "Age at start of therapy line 1"      format = 8.
    ...
  ;
  retain _numeric_ . _character_ ' ';
  stop;
run;

data dm1 (keep = ... );
  set template source1 (keep = ...);
  SUBJID  = USUBJID;
  STUDYID = "ABC";
  DOMAIN  = "DM";
  COUNTRY = "DEU";

     <Ableitungen>;
run;
```

Bei diesem Data Step greifen die selben Schritte des SAS Supervisors wie beim Beispiel in Abschnitt 3.4. Während Daten aus SOURCE1 gelesen werden, werden die Inhalte der Variablen aus TEMPLATE nicht auf Missing zurückgesetzt, bevor SAS in eine neue Datenzeile geht. So bleiben Werte, die in vorherigen Zeilen generiert wurden, erhalten und werden in folgende Zeilen übertragen.

5 DoW Loop

Der Begriff DoW Loop setzt sich zusammen aus Do und dem W von einem der ersten Autoren, die diese Art des Datenlesens im Data Step publiziert und weiter entwickelt haben, Ian Whitlock [6]. Bei der Technik handelt es sich um eine Do-Schleife, innerhalb derer ein Set-Statement ausgeführt wird. Mit Kenntnis der vorher beschriebenen Funktionsweise des SAS Supervisors ist gut zu verstehen, was hier abläuft. Als Beispiel wird die LOCF Imputation aus Abschnitt 2 hier mit einem DoW Loop durchgeführt:

```
1    data imputeDoW;
2         do until (last.PTNO);
3              set sysBP;
4              by PTNO;
5              if not missing(SBP) then NEWSBP=SBP;
6              output;
7         end;
8    run;
```

Innerhalb der Data Step Grenzen (Zeilen 1, 8) ist eine Do Until-Schleife platziert (Zeilen 2, 7). Diese Schleife wird für jede Zeile mit Identifier PTNO durchlaufen. Das impliziert, dass jede Patientenzeile ein eigenes Set-Statement mit By-Bedingung hat. Wir erinnern uns an die Auswirkung für die Regeln des SAS Supervisor: Der PDV wird zu Beginn jedes neuen Patienten (PTNO) zunächst geleert. Daher ist dies eine sehr sichere Methode, die Datenübertragung in folgende Patienten zu verhindern. Der ITMV enthält R (read) für alle Variablen des Ursprungs-Data Sets. Hier werden also alle Werte stets neu ausgelesen, auch Missing-Werte. Die neue Variable NEWSBP wird neu befüllt, wenn in SBP ein Wert steht. Steht dort keine Wert, wird durch implizites Retain für neue Variablen der vorherige Wert innerhalb der By-Gruppe beibehalten, das LOCF ist somit sauber und sicher ausgeführt.
Die Schleife endet nach einer expliziten Ausgabe per Output-Statement (Zeile 6). Dieses ist notwendig, da erst das RUN die automatische Ausgabe triggern würde, dies wird jedoch nicht erreicht, bevor alle Zeilen eines Patienten durchlaufen sind.
Hier wurde ein sehr einfaches Beispiel des DoW Loop beschrieben. Die Komplexität und der Nutzen lassen sich steigern. So ist auch ein Double DoW Loop beschrieben [] und die Kombination mit vielen weiteren Set Parametern ist möglich. Der versierte Programmierer findet hier eine Fülle von neuen Möglichkeiten, die jedoch den Rahmen dieses Beitrag sprengen.

6 POINT= Data Set Option

Die POINT= Data Set Option birgt ergänzende Möglichkeiten, Datenzeilen in einem Data Set gezielt anzusprechen und Informationen aus verschiedenen Zeilen zu vereinen.
Es wird eine temporäre Variable erzeugt, die nicht im Ausgabe-Data Set behalten wird.
Informationen können um eine beliebige Zeilenzahl weit versetzt werden.
Die POINT= Option hat die Einschränkung, dass sie nicht mit END=, KEY= oder By zusammen verwendet werden kann.
Besonders hervorzuheben ist, dass mit POINT= ein Versatz nicht nur abwärts (von einer Zeile in folgende Zeilen), sondern auch aufwärts (von einer Zeile in vorherige Zeilen) möglich ist. Dieses aufwärts Versetzen wird auch Lead genannt [6].

6.1 Versetzen mit POINT= nach oben

Der wohl häufigste Fall ist ein Lead von +1. Ein Beispiel zur Veranschaulichung, dessen Sinn hier keine Rolle spielt. Es soll nur um das Versetzen von Inhalten einer Variable (hier: SepalLength) um eine Zeile nach oben gehen.

```
1 data iris1;
2    set SASHELP.IRIS nObs=_nObs;
3    lead_SepalLength=.;
4    _point=_N_ +1;
5
6    if _point<=_nObs then do;
7         set SASHELP.IRIS
```

```
8            (keep=SepalLength
9             rename=(SepalLength=lead_SepalLength))
10            point=_point;
11   end;
12 run;
```

In Zeile 2 wird die SAS-interne, temporäre Variable NOBS angesprochen und so zur weiteren Verwendung zugänglich gemacht. Die Lead Variable wird initialisiert (Zeile 3) und zunächst für jede neu gelesene Zeile auf Missing gesetzt. Das ist wichtig, da für neue Variablen ein implizites Retain gilt. In Zeile 4 wird mit der Pointer Variable _POINT der Versatz festgelegt. Dann wird eine Schleife initialisiert, die nur dann ausgeführt werden soll, wenn noch eine weitere Zeile folgt (Zeile 6). Innerhalb dieser Schleife erfolgt ein zweites Set-Statement (Zeile 7) mit den Optionen KEEP, RENAME und POINT. In Zeile 8 wird die Variable SepalLength behalten aus dem zweiten Set behalten und in Zeile 9 gleich umbenannt, da das Ausgabe-Data Set bereits eine Variable mit diesem Namen aus dem ersten Set-Statement enthält. Zeile 10 enthält die Verknüpfung von NOBS aus dem ersten Set von SASHELP.IRIS zu POINT aus dem zweiten Set dieses Data Sets.
Zusammengefasst bedeutet dies: Jede Observation aus dem Ursprungs-Data Set wird einzeln gelesen und mit einer Variablen ergänzt, die jeweils die Werte aus der Zeile darunter enthält. Das gekürzte Ergebnis zeigt Abb. 11.

VIEWTABLE: iris1

	Species	SepalLength	lead_SepalLength
1	Setosa	50	46
2	Setosa	46	46
3	Setosa	46	51
4	Setosa	51	55
5	Setosa	55	48
			...
146	Virginica	64	68
147	Virginica	68	57
148	Virginica	57	58
149	Virginica	58	63
150	Virginica	63	.

Abbildung 11: Ergebnis des Versetzens um +1 mit Data Set Option POINT

Der Missing-Wert in Zeile 150 des erzeugten Data Sets wird generiert aus zwei Bedingungen:

- die Variable lead_SepalLength wird jedes Mal auf Missing gesetzt (Programmcode Zeile 3)
- es folgt keine Datenzeile mehr im Data Set SASHELP.IRIS, aus der hier ein Wert ausgelesen und hineingeschrieben werden könnte (Programmcode Zeile 6).

Der Verzicht auf Zeile 3 des Programmcodes würde zu einem impliziten Retain führen und den Wert 63 in diese Zelle fortschreiben. Analog zum Vorgehen in diesem Beispiel kann auch ein Versetzen um eine andere Zeilenanzahl realisiert werden. Ein Lead von 3 wird erreicht mit

```
4     _point=_N_ +3;
5
6     if _point<=_nObs then do;
```

Man erhält folgendes Ergebnis:

VIEWTABLE: WORK.IRIS3

	Species	SepalLength	lead_SepalLength
1	Setosa	50	51
2	Setosa	46	55
3	Setosa	46	48
4	Setosa	51	52
5	Setosa	55	49
6	Setosa	48	44
7	Setosa	52	50

Abbildung 12: Ergebnis des Versetzens um +3 mit Data Set Option POINT

Auch ein Versetzen in die andere Richtung ist möglich. Die Bedingung der Schleife muss dann entsprechend angepasst werden.

```
4     _point=_N_ -3;
5
6     if _point > 0 then do;
```

Das Ergebnis zeigt Abb. 13.

VIEWTABLE: WORK.IRIS_3

	Species	SepalLength	min3_SepalLength
1	Setosa	50	.
2	Setosa	46	.
3	Setosa	46	.
4	Setosa	51	50
5	Setosa	55	46
6	Setosa	48	46
7	Setosa	52	51
8	Setosa	49	55

Abbildung 13: Ergebnis des Versetzens um -3 mit Data Set Option POINT

6.2 Versetzen mit POINT= mit Beachtung einer Gruppierung

Im Alltag hat man oft Gruppierungen, die bei der Datenzuordnung beachtet werden sollen. Es ist viel Sorgfalt darauf zu verwenden, dass z.B. keine Daten von einem Patienten versehentlich in Zeilen eines anderen Patienten übertragen werden. Für den häufigsten Fall des Versetzen um +1 oder -1 Zeile bietet es sich an, ein By-Statement beim ersten Set und die daraus resultierenden internen Variablen FIRST. oder LAST. zu verwenden.

```
1 data iris1;
2    set SASHELP.IRIS nObs=_nObs;
3     by species;
4         lead_SepalLength=.;
5         _point=_N_ +1;
6
7    if not last.species and _point<=_nObs then do;
8         set SASHELP.IRIS
9           (keep=SepalLength
10              rename=(SepalLength=lead_SepalLength))
11              point=_point;
12   end;
13 run;
```

Mit Zeile 3 wird die Gruppierung definiert. Im zweiten Set-Statement (Zeile 8) ist kein By möglich. Durch diese Gruppierung wird die Abfrage auf LAST.SPECIES in Zeile 7 möglich. Da das implizite Retain für neue Variablen in Zeile 4 umgangen wird, erhält man das in Abb. 14 gezeigt Ergebnis.

VIEWTABLE: WORK.IRIS1

	Species	SepalLength	lead_SepalLength
47	Setosa	54	51
48	Setosa	51	52
49	Setosa	52	53
50	Setosa	53	.
51	Versicolor	65	62
52	Versicolor	62	59
53	Versicolor	59	61

Abbildung 14: Ergebnis des Versetzens um +1 mit Beachtung der Gruppierung

Möchte man um mehr als eine Zeile versetzen und die Gruppierung beachten, kann man den Umweg gehen, zuerst alle Zeilen zu befüllen und anschließend aus den nicht gewünschten Zellen die Werte wieder zu löschen.
Hier kommt der Data Step an seine Grenzen. Eine gute Alternative ist hier PROC SQL. Ein Einstieg in diese Prozedur sprengt jedoch die Grenzen dieses Beitrags.

7 Zusammenfassung

Für die Übergabe von Information von einer zur folgenden Zeilen im Data Step stehen dem Programmierer verschiedene Optionen und Anweisungen zur Verfügung. Diese zu verstehen und gezielt einzusetzen bewahrt vor Fehlern und gibt Sicherheit in der täglichen Arbeit. Hat man die Abläufe und Regeln des SAS Supervisor verstanden, kann man die vielfältigen Möglichkeiten auch ausschöpfen.
Explizites Retain für neu einzuführende Variablen verwenden; implizites Retain aufgrund der ITMV verstehen und gezielt einsetzen; DoW Loops entwickeln, z.B. mit bedingtem Set zum Fortschreiben von Werten; die Point= Option zum Versetzen von Daten um eine festzulegende Anzahl von Zeilen – der Data Step bietet einen Fundus an Werkzeugen, Werte in folgende Zeilen zu übergeben. Nicht behandelt wurde im Rahmen dieses Beitrags die Lag-Funktion [6]. Mir ist kein Beispiel eingefallen, das mit der Lag-Funktion besser hätte ausgeführt werden können als mit einer bereits beschriebenen Methode dieses Beitrags.
Wo der Data Step an seine Grenzen stößt, kann sich der Programmierer den Einsatz von PROC SQL überlegen, hier finden sich viele weitere Möglichkeiten, auch für einen zukünftigen KSFE-Beitrag.

Literatur

[1] http://www.sascommunity.org/wiki/The_SAS_Supervisor

[2] https://www.lexjansen.com/nesug/nesug88/SAS_supervisor.pdf

[3] Cody, R: Longitudinal Data and SAS: A Programmer's Guide. Cary, NC: SAS Institute Inc. 2001.
https://www.sas.com/storefront/aux/en/splongitudinal/58176_excerpt.pdf

[4] https://communities.sas.com/t5/SAS-Procedures/How-do-I-create-a-cumulative-total-column-subset-by-another/td-p/91118 (Accumulator variable)

[5] McDonald Paul D: If _n_=1 then SET. SPIKEWARE, Inc., Schaumburg, IL 2000.
http://www2.sas.com/proceedings/sugi25/25/po/25p222.pdf

[6] Dunn, T., Chung, C.Y.: Retaining, Lagging, Leading, and Interleaving Data.
http://changchung.com/download/retainLagLeadInterleave_draft.pdf

[7] Zhao, Y.: Effective Use of RETAIN Statement in SAS Programming. Merck&Co. Inc., Upper Gwynedd, Pennsylvania 2009.
http://mwsug.org/proceedings/2009/stats/MWSUG-2009-D14.pdf

[8] Chakravarthy, V., Arbor, A: The DOW (not that DOW!!!) and the LOCF in Clinical Trials. SUGI 28, 2003. http://www2.sas.com/proceedings/sugi28/099-28.pdf