Richtlinien zur Programmierung von Standard SAS® Makros

Version 1.0

14. Februar 2010

Hanspeter Schnitzer

Inhalt

Zusammenfassung

SAS® bietet die Möglichkeit, wiederkehrende Programmsequenzen als Makro zu hinterlegen. Damit wird ein modulares Programmieren unterstützt, auch wenn es sich nicht um Unterprogramme im herkömmlichen Sinne handelt, sondern lediglich um Textersetzung während der Programmausführung. Viel Gehirnschmalz wird seitdem aufgebracht, um die kompliziertesten Anforderungen in SAS® Makros unterzubringen. Aber was sollte ich bei der Programmierung eines Makros beachten, ungeachtet der eigentlichen Aufgabe, für die ich es schreibe? Was sollte ich bedenken, wenn ich Makroparameter verwenden möchte? Wie verhindere ich ungewollte Wechselwirkungen meines Makros mit dem rufenden Programm oder mit anderen Makros? Wie erreiche ich Abwärtskompatibilität, wenn ich ein bestehendes Makro erweitern möchte?
In diesem Beitrag soll ein sicher nicht vollständiger Überblick gegeben werden, worauf bei der Makroprogrammierung geachtet werden soll, damit das entstehende Makro einfach in der Anwendung ist und ein robustes Verhalten an den Tag legt. Manches ist sicher auch Geschmacksache und jedem steht es frei, von dem hier beschriebenen abzuweichen. Hier könnten z. B. Performanceerwägungen eine Rolle spielen. Wichtig vor allem ist, dass sich jeder über die aufgeführten Aspekte seine eigenen Gedanken macht. Die hier geschilderten Gesichtspunkte der Makroprogrammierung sollten in erster Linie bei der Programmierung von Standard Makros beachtet werden, also Makros, die für den allgemeinen Gebrauch zur Verfügung gestellt werden sollen. Die verwendeten Beispiele sind in englischer Sprache verfasst (Kommentierung, Namensgebung etc.).

Ziele

Wenn ich ein Standard Makro entwickle, soll es in erster Linie eine zuvor definierte Aufgabe erfüllen. Dabei sollte ich mich aber nicht damit zufrieden geben, wenn das entstehende Makro tut, was es tun soll (und das natürlich fehlerfrei). Vielmehr sollte ich darüber hinaus dafür sorgen, dass mein Makro auch benutzt und erfolgreich eingesetzt wird. Das setzt voraus, dass die potentiellen Anwender Kenntnis erhalten und verstanden haben, was sie mit dem Makro machen können (Schulung, Manual) und über auftretende Fehler bzw. Neuigkeiten zeitnah informiert werden (Newsletter). Mögliche Bedienungsfehler sollen komfortabel an den Anwender zurückgemeldet werden. Wichtig ist auch, dass eventuell notwendige Fehlerkorrekturen bzw. funktionelle Erweiterungen ohne große Schwierigkeiten umgesetzt werden können.

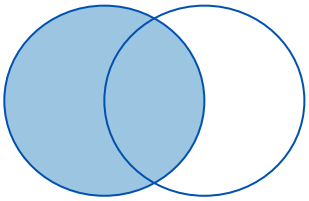
- Akzeptanz (gute Dokumentation, benutzerfreundlich)
- Qualität (fehlerfrei, sicher)
- Wartbarkeit (flexibel, verständliche Programmierung)

Beispielmakros

In den nachfolgenden Ausführungen wird auf die Makros %LeftJoin und %CreateLib Bezug genommen, die als Beispiele entwickelt wurden.

%LeftJoin

Das Makro %LeftJoin führt einen Merge bzw. eine Left Join durch, d.h. alle Records des linken Datasets werden übernommen sowie die in den Keyvariablen übereinstimmenden Records des rechten Datasets.



Beispielaufruf:

```
%LeftJoin(
  pDSIn1    = DM
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = DM2
);
```

Resultierender Datastep:

```
data DM2;
  merge DM(in = a) country;
  by studyid subjid;
  if a;
run;
```

%CreateLib

Das Makro %CreateLib erzeugt eine Library innerhalb des Verzeichnisses der Library WORK.

Beispielaufruf:

```
%let tNewLib = ;
%CreateLib(tNewLib);
%put &tNewLib.;
```

Im Anhang befindet sich ein vollständiges Listing beider Makros. Die Makros dürfen gerne verwendet werden, allerdings auf eigene Verantwortung, eine Garantie kann nicht übernommen werden.

Kommentierung

Kommentierung ist wichtig und hilfreich, damit sich der Entwickler bei Wartungsarbeiten besser und schneller im Programm zurechtfindet. Dies gilt sowohl für den Erstentwickler als auch für einen anderen Entwickler, der lediglich eine Erweiterung einbauen oder einen Fehler beheben soll.

Makro Header

Über den Inhalt eines Makro Headers ist bereits viel und kontrovers diskutiert worden, seine Erforderlichkeit hingegen ist sicher unstrittig. Jeder sollte für sich entscheiden, wie der Aufbau eines Makro Headers auszusehen hat. Deshalb an dieser Stelle lediglich ein unkommentierter Vorschlag.

Beispiel:

```
/************************************************************************************************
**  Macro Name:            %LeftJoin
**  Topic:                 Merge
**  Requirements:          ..\Macros\LeftJoin\LeftJoin_URS.doc
**  Source File:           ..\Macros\LeftJoin\LeftJoin.sas
**  Test Program:          ..\Macros\LeftJoin\LeftJoin_UAT.sas
**  Manual:                ..\Macros\LeftJoin\LeftJoin_UM.doc
**  Company/Department:    -
**  Software(Dev/Sys):     SAS Version 9.2 / Windows XP
**------------------------------------------------------------------------------------------------
**  Short Description:     Perform a left join.
**  Input:                 Two datasets for merge.
**  Output:                Result dataset.
**  Return Value:          -
**  Comments:              -
***************************** H I S T O R Y ***************************************************
**  Developer     Date     Remarks
**  ---------  ----------  ---------------------------------------------------------------------
**  HaS        02.08.2009  Start of development
**  HaS        08.08.2009  Implementation finished
************************************************************************************************/
```

Dateiende

Weitere Kommentierungen neben dem Makro Header sind sinnvoll und hilfreich.
Das Ende des Makros wird durch einen entsprechenden Kommentarblock markiert. Dabei kann der Name des Makros genannt werden, was jedoch eine beliebte Fehlerquelle ist (copy/paste).
Beispiel:

```
/************************************************************************************************
**  End of Macro
************************************************************************************************/
```

Programmblöcke

Jeder identifizierbare Programmblock wird durch einen Kommentarblock eröffnet. Dies verdeutlicht den Aufbau bzw. die innere Struktur des Makros.
Beispiel:

```
/************************************************************************************************
**  Parameter check
************************************************************************************************/
```

Makrovariablen

Verwendete Makrovariablen werden bei ihrer Definition mittels Kommentartext erläutert, um das Nachvollziehen der Programmzeilen zu vereinfachen.
Beispiel:

```
%local
  tDebug              /* Y/N - debug mode on/off */
```

Programmzeilen

Unklare oder komplizierte Programmzeilen werden kommentiert und damit die Verständlichkeit erhöht.
Beispiel:

```
  %do %while(&tVar. ne); /* loop over all key variables */
```

Namenskonventionen

Alle verwendeten Namen (Namen für Makros, Makroparameter, temporäre Makrovariablen, temporäre Datasets, temporäre Datasetvariablen etc.) sollten so weit wie möglich selbsterklärend sein. Sinnvolle Namenskonventionen sollten vereinbart und angewendet werden.
`%VarExist` soll offensichtlich herausbekommen, ob eine Variable existiert oder nicht. Die Frage ist nur, ob hierbei eine Datasetvariable oder eine Makrovariable gemeint ist. Besser ist es, hier `%DSVarExist` zu verwenden bzw. `%MVarExist`.
Weitere Beispiele:
`%CopyDir`, `%CreateDir`, `%Excel2SAS`, `%GetDSObsCnt`, `%GetLibPath`

Parameter

Als Schnittstelle zur Außenwelt dienen in aller Regel Makroparameter. Dabei ist zu beachten, dass es sich bei SAS® Parametern ausschließlich um eine Werteübergabe handelt. Die Verwendung von Adresszeigern (Pointern), die auf die Speicheradresse einer Makrovariablen verweisen, ist nicht möglich. Das bedeutet aber auch, dass die Makroparameter innerhalb des Makros ohne Auswirkung auf das rufende Programm verändert werden können. Eine Rückgabe von Werten mittels Makroparameter ist lediglich über einen Trick möglich (s.u.). Die Möglichkeit, ein Makro als Funktion zu schreiben und ein Funktionsergebnis zurückzugeben, existiert zwar, ist aber nur sehr eingeschränkt einsetzbar, da man lediglich Makrobefehle verwenden kann, aber z. B. keine Datasteps.
Wichtig ist, dass sämtliche innerhalb des Makros verwendeten Ressourcen, die von außen kommen, als Parameter definiert werden, also z.B.

- Libraries
- Datasets
- Dataset Optionen
- Datasetvariablen
- By Variablen
- Formate

Zur besseren Übersicht sollte darauf geachtet werden, dass Makros nicht zu viele Parameter aufweisen, insbesondere Parameter, die in jedem Fall spezifiziert werden müssen, da für sie kein Defaultwert spezifiziert wurde.

Parameternamen

Es ist wichtig, dass Parameternamen möglichst selbsterklärend sind. Da man in aller Regel nicht nur ein Standard Makro entwickelt, sondern eine ganze Sammlung von Makros, sollen Parameternamen einheitlich vergeben werden und vordefinierten Konventionen (siehe oben) genügen. So sollte z.B. ein Inputdataset in allen Makros gleich heißen, also z.B. DSIN. Für eine bessere Lesbarkeit des Makrocodes kann man Parameternamen mit einem einheitlichen Buchstaben (z.B. p) beginnen. Zur Unterscheidung von Rückgabeparametern kann man auch i bzw. io verwenden.
Einige Beispiele:
LibIn, DSIn, DSOut, FileOut

Keyword Parameter

Es sollten in aller Regel Keyword Parameter und keine Positionsparameter verwendet werden. Dies trägt zur Lesbarkeit bei und ist Anwender freundlich sowie weniger fehleranfällig. Ausnahmen können z.B. Makros mit nur einem Parameter sein (siehe nächstes Beispiel) oder der bei der KSFE 2009 vorgestellte Help-Parameter (siehe weiter unten).

Beispiel: unser Beispielmakro %CreateLib mit einem Positionsparameter

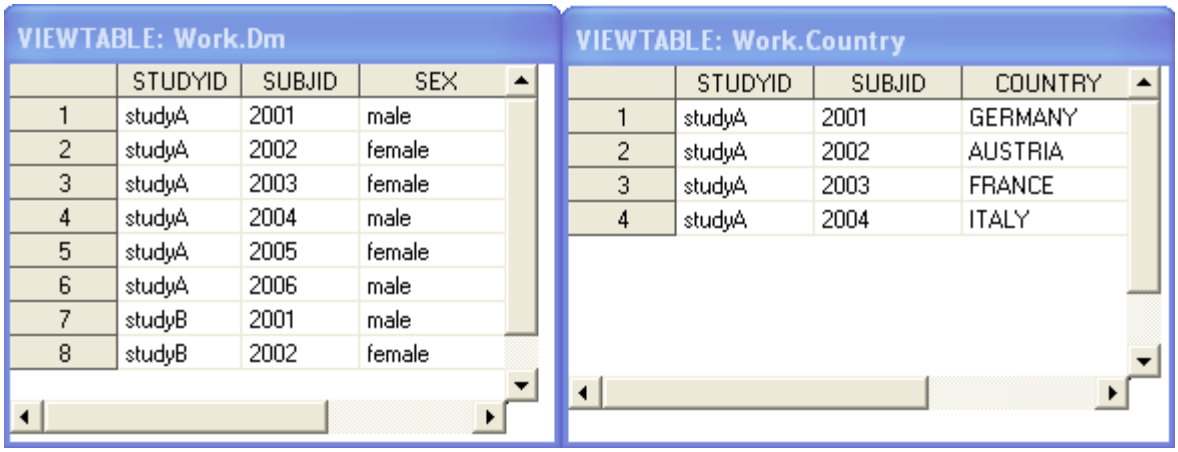
```
%macro CreateLib(
  pNewLib
);

/* ... more SAS code */

%mend;

%CreateLib(tNewLib);
```

Parameter Default Value

SAS® bietet die Möglichkeit, Keyword Parametern einen festen Wert zuzuordnen. Falls der Parameter beim Makroaufruf nicht spezifiziert wird, arbeitet das Makro mit diesem Defaultwert weiter. Dies ist sicherlich eine effiziente Möglichkeit, Makroaufrufe zu vereinfachen. Es sollte aber immer gut überlegt werden, ob nicht Situationen entstehen können, in denen der Anwender diesen

Defaultwert nicht wollte und dann Fehler auftreten, die bei der Ausführung des Makros eventuell nicht auffallen. Dies soll im folgenden Beispiel verdeutlicht werden.

Beispiel:
Die Datasets Dm und Country haben folgenden Inhalt:



VIEWTABLE: Work.Dm

	STUDYID	SUBJID	SEX
1	studyA	2001	male
2	studyA	2002	female
3	studyA	2003	female
4	studyA	2004	male
5	studyA	2005	female
6	studyA	2006	male
7	studyB	2001	male
8	studyB	2002	female

VIEWTABLE: Work.Country

	STUDYID	SUBJID	COUNTRY
1	studyA	2001	GERMANY
2	studyA	2002	AUSTRIA
3	studyA	2003	FRANCE
4	studyA	2004	ITALY

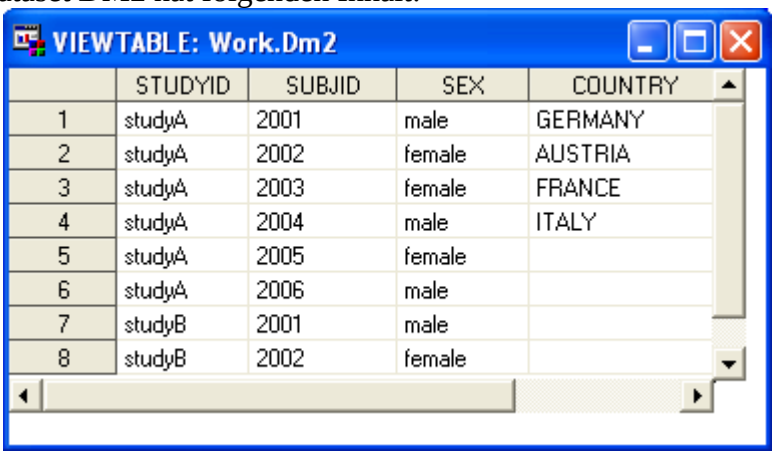
Zunächst hat der Makroparameter pKeys keinen Defaultwert.
Wir rufen das Makro %LeftJoin wie folgt auf:

```
%LeftJoin(
  pDSIn1    = DM
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = DM2
);
```

Es wird folgender Datastep ausgeführt:

```
data DM2;
  merge DM(in = a) country;
  by studyid subjid;
  if a;
run;
```

Der resultierende Dataset DM2 hat folgenden Inhalt:



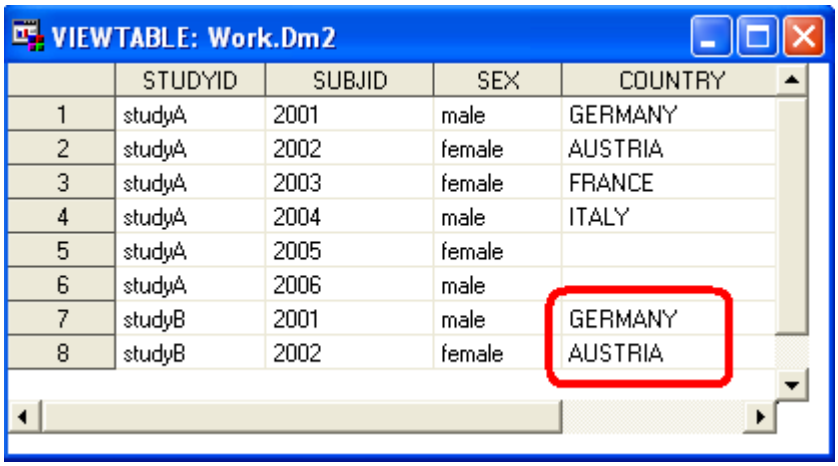
VIEWTABLE: Work.Dm2

	STUDYID	SUBJID	SEX	COUNTRY
1	studyA	2001	male	GERMANY
2	studyA	2002	female	AUSTRIA
3	studyA	2003	female	FRANCE
4	studyA	2004	male	ITALY
5	studyA	2005	female	
6	studyA	2006	male	
7	studyB	2001	male	
8	studyB	2002	female	

Haben wir aus Versehen den Parameter pKeys nicht spezifiziert, erhalten wir eine Fehlermeldung.

```
##### Begin of macro LEFTJOIN #####
ERROR: LEFTJOIN - Parameter pKeys is empty !
ERROR: LEFTJOIN - stop macro execution !
##### End of macro LEFTJOIN #####
```

Falls der Makroparameter pKeys jedoch den Defaultwert pKeys = SUBJID hat, erhalten wir keine Fehlermeldung. Stattdessen wird das Makro %LeftJoin ausgeführt und erzeugt folgenden Dataset.



VIEWTABLE: Work.Dm2

	STUDYID	SUBJID	SEX	COUNTRY
1	studyA	2001	male	GERMANY
2	studyA	2002	female	AUSTRIA
3	studyA	2003	female	FRANCE
4	studyA	2004	male	ITALY
5	studyA	2005	female	
6	studyA	2006	male	
7	studyB	2001	male	GERMANY
8	studyB	2002	female	AUSTRIA

Help Parameter

Ein weiteres Beispiel eines Positionsparameters ist der Help Parameter (siehe KSFE 2009). Dieser äußerst hilfreiche Parameter verschafft dem Anwender während der Programmierung einen schnellen Überblick über die Funktionalität eines Makros und dient als Ergänzung zum Manual des Makros.

Beispiel:

```
%macro LeftJoin(
  pHelp
 ,pLibIn1   = work
 ,pDSIn1    =
 ,pDSOpt1   =

/* ... more SAS code */

%if (&pHelp. eq ?) %then %do;
  %put %nrstr(%LeftJoin);
  %put %nrstr(Perform a left join.);

/* ... more SAS code */
```

Beispielaufruf:

```
%LeftJoin(?);
```

Dataset Parameter

Oft ist es notwendig, einem Makro einen oder mehrere Datasets zu übergeben. Zusätzlich zu einem Dataset werden oft auch noch die Library oder gar Datasetoptionen übergeben. Dabei können zwei verschiedene Strategien verwendet werden, getrennte Parameter oder ein Parameter für alles.

Beispiel:

Getrennte Parameter

```
%LeftJoin(
  pLibIn1   = lib1
 ,pDSIn1    = dm
 ,pDSOpt1   = where = (sex eq 'male')
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = dm
);
```

Kombinierte Parameter

```
%LeftJoin(
  pDSIn1    = lib1.dm(where = (sex eq 'male'))
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = dm
);
```

Beide Versionen haben ihre Vorteile. Bei der Verwendung von getrennten Parametern haben wir innerhalb des Makros Library, Dataset und Option einzeln verfügbar und müssen sie im Bedarfsfall nicht separieren. Dafür ist der Aufruf durch die größere Anzahl von Parametern aufwendiger.

SAS® Code Parameter

Um die Flexibilität von Makros zu erhöhen, kann es hilfreich sein, durch einen entsprechenden Parameter zusätzlichen SAS® Code innerhalb des Makros auszuführen.

Beispiel:

Makroaufruf	Resultierender Datastep

`%LeftJoin(`
`  pDSIn1  = dm`
`,pDSIn2  = country`
`,pKeys  = studyid subjid`
`,pDSOut  = dm2`
`,pDSCode  = if missing(country) then country = '*'`
`);`
	`data dm2;`
	`  merge dm(in = a) country;`
	`  by studyid subjid;`
	`  if a;`
	`  if missing(country) then country = '*';`
	`run;`

Im Handbuch des Makros sollte aber genau beschrieben werden, wie sich der SAS® Code auswirkt.

Debug Parameter

Falls bei der Entwicklung, beim Test oder bei der Anwendung eines Makros Probleme auftreten, ist es hilfreich, möglichst viele Informationen im Logfenster zu erhalten, die den Ablauf des Makros aufzeigen oder auch den aktuellen Wert von Makrovariablen. Beim normalen Betrieb soll das Log jedoch nicht mit unnötigen Meldungen überfrachtet werden. Hierzu dient der Debug Parameter, mit dem man die zusätzlichen Meldungen ins Log aktivieren kann. Auch kann verhindert werden, dass die temporären Datasets am Ende des Makros gelöscht werden. Das ist zwar bei der Programmierung des Makros ein Mehraufwand, der sich aber beim späteren Betrieb auszahlen wird.

Beispiel:

```
%macro LeftJoin(

/* ... more SAS code */

,pDebug    = no     /* = (yes, y) - debug mode on */
                    /* = else     - debug mode off */
);

/* ... more SAS code */

  %if (%upcase(&pDebug.) eq YES) or (%upcase(&pDebug.) eq Y) %then %do;
    %let tDebug = Y;
  %end;
  %else %do;
    %let tDebug = N;
  %end;

/* ... more SAS code */

  %if &tDebug. eq N %then %do; /* debug mode off */
    proc datasets lib=&tNewLib. nolist kill memtype=all;
    quit;
    libname &tNewLib. clear;
  %end;
```

Parameterrückgabe

Wie oben bereits erwähnt können SAS® Makroparameter lediglich als Wert übergeben werden, nicht aber als Pointer. Will man einen Wert zurückgeben, kann man dies über eine Hilfskonstruktion erreichen. Dazu wird im rufenden Programm eine Makrovariable angelegt. Diese

Makrovariable wird dem Makro als Parameter übergeben. Im Makro wird dann der entsprechende Makroparameter mit dem Ergebniswert gesetzt.

Beispiel:

SAS Programm	Log Fenster

`%macro CreateLib(`
`  pNewLib`
 	
`/* ... more SAS code */`
 	
`  %let &pNewLib. = &tLibName.;`
`%mend;`	

`%let tNewLib = vorher;` `%put vorher  - &tNewLib.;`	`vorher  - vorher`
`%CreateLib(tNewLib);` `%put nachher - &tNewLib.;`	`nachher - MLIB0001`

Erweiterung des Makros durch neue Parameter

Oft soll ein Makro nach Fertigstellung um eine Funktionalität erweitert werden. Diese neue Funktion spiegelt sich in einem oder gar mehreren zusätzlichen Makroparametern wieder. Damit bestehende Programme, die das Makro bereits verwenden, auch mit der neuen Version des Makros genauso funktionieren wie vor der Änderung (Abwärtskompatibilität), ist es oft sinnvoll, den oder die neuen Parameter mittels Defaultwert so vorzukonfigurieren, dass das Makro das gleiche Verhalten hat, wie vor der Änderung.
Beispiel:
Das Makro %LeftJoin sucht die Input Datasets in der Library WORK und schreibt den Ergebnisdataset ebenfalls in die Library WORK. Nun soll es dahingehend erweitert werden, dass man die Libraries mittels Parameter angeben kann. Damit das Verhalten unverändert bleibt zur vorigen Version, setzt man die neuen Parameter auf WORK.

Beispiel:

```
%macro LeftJoin(
  pHelp              /* = ? - print help text to log window and exit macro */
 ,pLibIn1   = work   /* library with input data set 1 */
 ,pDSIn1    =        /* input data set 1 */
 ,pDSOpt1   =        /* options of data set 1 */
 ,pLibIn2   = work   /* library with input data set 2 */
 ,pDSIn2    =        /* input data set 2 */
 ,pDSOpt2   =        /* options of data set 2 */
 ,pKeys     =        /* key variables for by statement */
 ,pLibOut   = work   /* library for output data set */
 ,pDSOut    =        /* output data set */

/* ... more SAS code */
```

Parameter Check

Werden Makros nicht korrekt angewendet, kann es zu einem Fehlverhalten kommen. Nun könnte man darauf vertrauen, dass SAS schon irgendwelche Fehlermeldungen erzeugen und dem Anwender damit den Fehler melden wird. Dieser passive Ansatz ist aber äußerst gefährlich, da man nicht immer sicher sein kann, dass die jeweilige Fehlbedienung wirklich bemerkt wird. Auch ist es oft nur sehr schwierig, die Fehlermeldung zu verstehen und die Ursache der Fehlermeldung herauszufinden, wenn man das Makro nicht kennt. Besser ist es deswegen, im Vorfeld ein mögliches Fehlverhalten zu unterbinden. Hierzu dient ein ausführlicher Parameter Check, bei dem der übergebene Wert jedes Makroparameters zu Beginn eines Makros überprüft wird, also bevor der eigentliche Prozess beginnt. Dadurch kann man dem Anwender eine detaillierte Meldung geben, falls etwas nicht in Ordnung ist. Und es wird sichergestellt, dass keine Ergebnisse basierend auf inkorrekten Parametern erzeugt werden, ohne dass eine Fehlermeldung erscheint. Das Erscheinungsbild der Fehlermeldungen sollte konsistent sein.

Mögliche Testmöglichkeiten:

- Inhalt des Makroparameters ist leer
- Library/Dataset/Datasetvariable/Format/… existiert nicht
- Verzeichnis/Datei/Makrovariable… existiert nicht
- Dataset ist leer
- Übergabewert ist ungültig
  - Typ einer Datasetvariablen ist ungültig
  - Library/Dataset/Format Name ist kein gültiger SAS® Name
- Ungültige Kombination von Parametern
- Ungültiger Wertebereich

Beispiel: Mit Parametercheck

SAS Programm	Log Fenster

`%macro LeftJoin(`
 	
`/* ... more SAS code */`
 	
`  %if %quote(&pKeys.) eq %then %do;`
`  %put %str(ERR)%str(OR: &tMacName. –`
`  Parameter pKeys is empty !);`
`  %let tErr = 1;`
`  %end;`	

`%LeftJoin(`
`  pDSIn1  = dm`
`  ,pDSIn2  = country`
`  ,pDSOut  = dm2`
`);`
	`##### Begin of macro LEFTJOIN #####`
	`ERROR:  LEFTJOIN - Parameter pKeys is empty !`
	`##### End of macro LEFTJOIN #####`

Beispiel: Ohne Parametercheck

SAS Programm	Log Fenster

`%LeftJoin(`
`  pDSIn1  = dm`
`  ,pDSIn2  = country`
`  ,pDSOut  = dm2`
`);`
	`##### Begin of macro LEFTJOIN #####`
	`...`
	
	`ERROR: No BY statement used or no BY variables specified. A BY statement must be used with variable names to sort on.`
	`NOTE: The SAS System stopped processing this step because of errors.`
	`WARNING: The data set LEFTJOIN.TDSIN1 may be incomplete.  When this step was stopped there were`
	`  0 observations and 0 variables.`

Im Nachfolgenden sind Beispiele für einige der oben aufgezählten Testfälle aufgeführt.

Inhalt des Makroparameters ist leer

```
%if %quote(&pParam.) eq %then %do;
  %put %str(ERR)%str(OR: &tMacName. - Parameter pParam is empty !);
  /* ... more SAS code */
%end;
```

Library existiert nicht

```
%if %sysfunc(libref(&pParam.)) ne 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. - Library &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

Dataset existiert nicht

```
%if %sysfunc(exist(&pParam.)) eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. - Dataset &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

Datasetvariable existiert nicht

```
proc sql noprint;
  select left(put(count(*), best.)) into :tRes
  from dictionary.columns
  where upcase(libname) eq "WORK" and
        upcase(memname) eq "TEST" and
        upcase(name)    eq "%upcase(&pParam.)";
quit;
%if &tRes. eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. – Dataset variable &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

oder

```
%let tDSID   = %sysfunc(open(WORK.TEST, i));
%let tVarNum = %sysfunc(varnum(&tDSID., &pParam.));
%let tRC     = %sysfunc(close(&tDSID.));
%if &tVarNum. le 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. – Dataset variable &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

Typ der Datasetvariablen ist ungültig

```
proc sql noprint;
  select left(type) into :tRes
  from dictionary.columns
  where upcase(libname) eq "WORK" and
        upcase(memname) eq "TEST" and
        upcase(name)    eq "%upcase(&pParam.)";
quit;
%if %upcase(&tRes.) ne NUM %then %do;   /* use CHAR for character variables */
  %put %str(ERR)%str(OR: &tMacName. – Dataset variable &pParam. isn%'t from type numeric !);
  /* ... more SAS code */
%end;
```

Falls auf Character getestet werden soll, muss CHAR an Stelle von NUM verwendet werden.

Format existiert nicht

```
proc sql noprint;
  select count(*) into :tRes
  from dictionary.catalogs
  where objtype contains 'FORMAT' and
        objname eq "%upcase(&pParam.)";
quit;
```

```
%if &tRes. eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. – Format &pParam. not found !);
  /* ... more SAS code */
%end;
```

Dataset ist leer

```
proc sql noprint;
  select left(put(nobs, 8.)) into :tRes
  from dictionary.tables
  where upcase(libname) eq "WORK" and
        upcase(memname) eq "%upcase(&pParam.)";
quit;
%if &tRes. eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. – Dataset &pParam. is empty !);
  /* ... more SAS code */
%end;
```

Verzeichnis existiert nicht

```
%let tFileRef = _tmpf;
%let tResult = %sysfunc(filename(tFileRef, &pParam.));
%let tRes = %sysfunc(dopen(_tmpf));
%if &tRes. eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. - Directory &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
%else %do;
  %let tResult = %sysfunc(dclose(&tRes.));
%end;
%let tResult = %sysfunc(filename(tFileRef));
```

Datei existiert nicht

```
%if %sysfunc(fileexist(&pParam.)) ne 1 %then %do;
  %put %str(ERR)%str(OR: &tMacName. - File &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

Makrovariable existiert nicht

```
%if %symexist(&pParam.) eq 0 %then %do;
  %put %str(ERR)%str(OR: &tMacName. – Macro variable &pParam. doesn%'t exist !);
  /* ... more SAS code */
%end;
```

Kapselung

Ein Makro sollte keine ungewollten Wechselwirkungen mit anderen Makros oder Programmen haben. Temporäre Makrovariablen innerhalb eines Makros sollten keine Makrovariablen außerhalb des Makros verändern. Im Makro verwendete Datasets sollen existierende Datasets nicht beeinflussen. Auch sollten keine Rückstände (Datasets etc.) zurückbleiben, wenn das Makro beendet ist. SAS® stellt nur teilweise Hilfsmittel zur Verfügung, um dieses Ziel zu erreichen. Aber mit ein paar Tricks erreichen wir eine fast vollständige Kapselung.

Temporäre Makrovariablen

Damit im Makro verwendete Makrovariablen nicht Makrovariablen beeinflussen, die z.B. vom rufenden Programm benutzt werden, werden sämtliche im Makro verwendeten Makrovariablen zu Beginn des Makros mit `%local` definiert und kommentiert. Dazu gehören auch Makrovariablen, die mit `call symput` oder `proc sql` angelegt werden genauso wie Indexvariablen in Schleifen. Die dabei verwendeten Namen sollten einheitlichen Konventionen genügen. Makrovariablen, die dynamisch erzeugt und gesetzt werden, werden dabei zusätzlich mit `%local` definiert.

Temporäre Datasets

Falls innerhalb des Makros temporäre Datasets angelegt werden müssen (üblicherweise in der Library WORK), ist es wichtig, dass man beim Anlegen der temporären Datasets nicht bereits vorhandene Datasets überschreibt. Leider gibt es kein `%local` für Datasets. Stattdessen müssen wir uns mit Hilfskonstruktionen behelfen, wobei wir diverse Möglichkeiten haben. Z. B. kann man sämtliche temporären Datasets mit einem einheitlichen Namensteil beginnen lässt, der sonst nicht verwendet werden darf (z.B. ein ‚_' oder der Name des Makros). Zusätzlich kann man vorher überprüfen, ob der Dataset bereits existiert. Weiterhin sollten temporäre Datasets am Ende des Makros gelöscht werden, damit sie beim nächsten Aufruf des Makros z. B. im Fehlerfall nicht fälschlicher Weise verwendet werden können. Ein einheitlicher Namensteil würde auch hier die Sache vereinfachen.
Um eine eventuelle Wechselwirkung mit rufenden Programmen zu verhindern, kann man auch beim Start von SAS® eine Library (z. B. MacWork) anlegen, die ausschließlich von Makros verwendet wird. Dann muss lediglich noch sichergestellt werden, dass sich die Makros an die Konventionen halten und untereinander nicht beeinflussen.
Eine weitere Möglichkeit ist es, zu Beginn des Makros eine temporäre Library anzulegen (vorzugsweise im Verzeichnis von WORK) und am Ende zu löschen. Auch wenn dies die aufwändigste Variante ist, wird dadurch jegliche Wechselwirkung mit anderen Makros oder Programmen ausgeschlossen.

Beispiel:
Um zunächst lediglich die Erzeugung einer Library zu verdeutlichen, nennen wir die temporäre Library TempLib. Da wir die temporäre Library in einem Unterverzeichnis der ja immer vorhandenen Library WORK anlegen wollen, müssen wir dieses Unterverzeichnis zunächst anlegen.

```
%let tLibPath = %sysfunc(pathname(WORK))\TempLib; /* path for temporary library */
%if %sysfunc(fileexist(&tLibPath.)) ne 1 %then %do;
  %let tXWait = %sysfunc(getoption(xwait));   /* create directory if doesn't exist */
  option noxwait;
  x "md %bquote(")&tLibPath.%bquote(")";
             /* " /* to  make the syntax coloring working well again */
  option &tXWait.;
%end;

libname TempLib "&tLibPath."; /* create temporary library */
```

```
  /* ... more SAS code */

  proc datasets lib= TempLib nolist kill memtype=all;
  quit;
  libname TempLib clear;
```

Dabei ist zu beachten, dass am Ende des Makros zwar sämtliche Dateien der Library TempLib gelöscht werden und auch die Library selbst wird deallokiert. Das physikalische Verzeichnis bleibt jedoch bestehen, so dass es beim nächsten Aufruf von %LeftJoin nicht erneut angelegt werden muss. Es wird sowieso mitgelöscht, wenn beim Beenden von SAS® die Library WORK gelöscht wird.
Die Verwendung eines festen Librarynamens diente hier lediglich zur einfacheren Anschauung. Z. B. für rekursive Makros wäre das keine geeignete Methode. Was wir eigentlich benötigen ist ein Makro, welches zusätzlich auch einen freien Namen für die Library findet.
Wir nennen das Makro %CreateLib, das vollständige Listing befindet sich im Anhang.

Beispiel:

SAS Programm	Log Fenster

`%let tNewLib = ;`
`%CreateLib(tNewLib);`
	`NOTE: Libref MLIB0001 was successfully assigned as follows:`
	`Engine: V9`
	`Physical Name: M:\MLIB0001`

`%put The new library is &tNewLib.;`	`The new library is MLIB0001`

Temporäre Formate

Sämtliche im Makro angelegten temporären Formate dürfen bereits vorhandene Formate nicht überschreiben. Auch dies erreichen wir durch anlegen einer temporären Library.

Beispiel:
Wie man eine temporäre Library anlegt haben wir bereits gesehen. Hier wird gezeigt, wie man SAS® dazu bringt, die Formate während der Ausführung des Makros in der temporären Library zu suchen und am Ende des Makros den alten Zustand wiederherstellt.

SAS Programm	Log Fenster

`/* get current value of FMTSEARCH option */`
`%let tFMT1 = %sysfunc(getoption(FMTSEARCH));`
`%put &tFMT1.;`
	`(WORK LIBRARY)`

`/* set FMTSEARCH option to new value */`
`option FMTSEARCH = (TempLib);`
`%let tFMT2 = %sysfunc(getoption(FMTSEARCH));`
`%put &tFMT2.;`
	`(TEMPLIB)`

`proc format library=TEMPLIB;`
`  value $country 'DEU'='GERMANY'`
`'AUT'='AUSTRIA'`
`'FRA'='FRANCE'`
`'ITA'='ITALY'`
`run;`
	`2  proc format library=TEMPLIB;`
	`3  value $country 'DEU'='GERMANY'`
	`4  'AUT'='AUSTRIA'`
	`5  'FRA'='FRANCE'`
	`6  'ITA'='ITALY'`
	`7  run;`
	`NOTE: Format $COUNTRY has been written to TEMPLIB.FORMATS.`

`/* reset FMTSEARCH option to old value */`
`option FMTSEARCH = &tFMT1.;`
`%let tFMT3 = %sysfunc(getoption(FMTSEARCH));`
`%put &tFMT3.;`
	`(WORK LIBRARY)`

In obigem Beispiel werden Formate lediglich in der Library TempLib gesucht. Falls Formate zusätzlich auch in tFMT1 gesucht werden sollen, könnte man statt dessen folgenden SAS® Befehl verwenden:

```
option FMTSEARCH = (TempLib &tFMT1.);
```

System Optionen

Falls es innerhalb eines Makros erforderlich ist, System Optionen zu verändern, sollten diese spätestens vor Beendigung des Makros wieder auf den alten Wert zurückgesetzt werden.

Beispiel:

```
  %let tXWait = %sysfunc(getoption(xwait));   /* create directory if doesn't exist */
  %if %sysfunc(fileexist(&tLibPath.)) ne 1 %then %do;
    option noxwait;
    x "md %bquote(")&tLibPath.%bquote(")";
                /* " /* to  make the syntax coloring working well again */
    option &tXWait.;
  %end;
```

Falls mehrere System Optionen geändert werden müssen, kann man auch die Befehle `optsave` und `optload` verwenden.

Beispiel:

```
proc optsave out=tOptions; run; /* save all options */

/* ... more SAS code */

proc optload data=tOptions; run; /* reset all options */
```

Titles, Footnotes

Falls Titles und Footnotes innerhalb von Makros verändert werden sollen, kann man ähnlich wie bei den Systemoptionen die aktuellen Werte zu Beginn aufheben und spätestens vor Abschluss des Makros wieder restaurieren.

Beispiel:

```
proc sql noprint;               /* save current titles */
  create table tTitles1 as
  select type, number, text
  from sashelp.vtitle
  where upcase(type) = "T";     /* use "F" for footnotes */
quit;

title1 "New Title 1";           /* set titles to new value */
title2 "New Title 2";

/* ... more SAS code */

title;                          /* reset titles to old values */
data _null_;
  set tTitles1;
  call execute('title'||strip(number)||' '||'"'||strip(text)||'";');
run;
```

Meldungen im Logfenster

Zur Dokumentation und Fehlerkontrolle ist es sehr hilfreich, gezielt sinnvolle und verständliche Meldungen (Warnings, Errors) ins Logfenster zu schreiben. Bei Makros, die sehr häufig aufgerufen werden, sollte man allerdings sparsam mit Meldungen ins Logfenster umgehen.
Schön ist es, wenn jede dieser Meldungen zunächst den Namen des Makros angibt. Zu diesem Zweck kann der Makroname zu Beginn eines Makros jeweils in eine Makrovariable gespeichert werden.

Beispiel:

```
%local
  tMacName           /* name of macro */

/* ... more SAS code */

  %let tMacName = &sysmacroname.;

/* ... more SAS code */

  %let tMsgTxt = %str(&tMacName. - Parameter pDSIn1 doesn%'t exist !);
```

Makroaufruf mit aktuellen Parametern ins Log ausgeben

Zu Beginn eines Makros kann der Makroaufruf inklusive der aktuellen Werte der Makroparameter als Meldung ins Logfenster geschrieben werden. Hierfür kann man folgende SAS® Befehle verwenden. Allerdings werden hierbei nur die im Makroaufruf spezifizierten Parameter angezeigt.

Beispiel:

SAS Programm	Log Fenster

`%macro LeftJoin(`
 	
`/* ... more SAS code */`
 	
`) / parmbuff;`
 	
`/* ... more SAS code */`
 	
`  %put &syspbuff.;`	

`%LeftJoin(`
`  pDSIn1    = dm`
`  ,pDSIn2    = country`
`  ,pKeys     = studyid subjid`
`  ,pDSOut    = dm2`
`);`
	`(   pDSIn1    = dm  ,pDSIn2    = country  ,pKeys     = studyid subjid ,pDSOut    = dm2 )`

Einen vollständigen Makroaufruf, bei dem auch die beim Makroaufruf nicht spezifizierten Parameter aufgelistet werden, muss man sich selber erzeugen.

Beispiel:

SAS Programm	Log Fenster

`%macro LeftJoin(`
 	
`/* more ... SAS code */`
 	
`  %put %nrstr(%LeftJoin %();`
`  %put   %str(  pLibIn1   = &pLibIn1.);`
`  %put   %str( ,pDSIn1    = &pDSIn1.);`
`  %put   %str( ,pDSOpt1   = &pDSOpt1.);`
`  %put   %str( ,pLibIn2   = &pLibIn2.);`
`  %put   %str( ,pDSIn2    = &pDSIn2.);`
`  %put   %str( ,pDSOpt2   = &pDSOpt2.);`	

SAS Programm	Log Fenster

`  %put   %str( ,pKeys     = &pKeys.);`
`  %put   %str( ,pLibOut   = &pLibOut.);`
`  %put   %str( ,pDSOut    = &pDSOut.);`
`  %put   %str( ,pDSOptOut = &pDSOptOut.);`
`  %put   %str( ,pDSCode   = &pDSCode.);`
`  %put %nrstr(  %));`
 	
`/* ... more SAS code */`	

`%LeftJoin(`
`  pDSIn1    = dm`
`,pDSIn2    = country`
`,pKeys     = studyid subjid`
`,pDSOut    = dm2`
`);`
	`%LeftJoin (`
	`  pLibIn1   = work`
	`,pDSIn1    = dm`
	`,pDSOpt1   =`
	`,pLibIn2   = work`
	`,pDSIn2    = country`
	`,pDSOpt2   =`
	`,pKeys     = studyid subjid`
	`,pLibOut   = work`
	`,pDSOut    = dm2`
	`,pDSOptOut =`
	`,pDSCode   =`
	`  )`

Beginn und Ende des Makros

Sowohl zu Beginn eines Makros als auch am Ende kann eine Meldung ins Logfenster geschrieben werden. Dadurch wird ein Debugging vereinfacht. Als Beginn des Makros kann auch der oben beschriebene Aufruf des Makros inkl. der Parameter dienen.

Beispiel:

```
  %let tMacName = &sysmacroname.;
  %let tMsgTxt  = ##### Begin of macro &tMacName. #####;
  %put &tMsgTxt.;

  /* ... more SAS code */

  %let tMsgTxt  = ##### End of macro &tMacName. #####;
  %put &tMsgTxt.;
```

Als weiteren Hinweis zu Beginn eines Makros kann man die Versionsnummer des Makros ausgeben.

Dokumentation

Neben einer Dokumentation über Planung, Entwicklung und Test des Makros sollte nicht zuletzt für die Akzeptanz des Makros ein Benutzerhandbuch erstellt werden. Dies gilt auch für vermeintlich einfache Makros. Benutzerhinweise innerhalb des Makroheaders im Sourcecode anstelle eines Handbuchs sind nicht ausreichend. Da man in der Regel nicht nur ein Standard Makro entwickelt, sondern mehrere, empfiehlt es sich, für ein einheitliches Format und Erscheinungsbild des Benutzerhandbuchs zu sorgen. Vorteilhaft ist es auch, sämtliche Handbücher aller Standard Makros in einem Dokument zusammenzufassen. Ein Beispiel für ein Handbuch ist im Anhang zu finden.

Anhang

Listing Makro %LeftJoin

```
/*************************************************************************************************
**  Macro Name:            %LeftJoin
**  Topic:                 Merge
**  Requirements:          ..\Macros\LeftJoin\LeftJoin_URS.doc
**  Source File:           ..\Macros\LeftJoin\LeftJoin.sas
**  Test Program:          ..\Macros\LeftJoin\LeftJoin_UAT.sas
**  Manual:                ..\Macros\LeftJoin\LeftJoin_UM.doc
**  Company/Department:    -
**  Software(Dev/Sys):     SAS Version 9.2 / Windows XP
**-----------------------------------------------------------------------------------------------
**  Short Description:     Perform a left join.
**  Input:                 Two datasets for merge.
**  Output:                Result dataset.
**  Return Value:          -
**  Comments:              -
****************************** H I S T O R Y ************************************************
**  Developer     Date     Remarks
**  ---------  ----------  ---------------------------------------------------------------------
**  HaS        02.08.2009  Start of development
**  HaS        08.08.2009  Implementation finished
*************************************************************************************************/

/*************************************************************************************************
Example call:
%LeftJoin(
  pDSIn1    = dsn1
 ,pDSIn2    = dsn2
 ,pDSOpt2   = where = (country eq 'D')
 ,pKeys     = usubjid dsnseq
 ,pDSOut    = dsn3
);
*************************************************************************************************/

%macro LeftJoin(
  pHelp              /* = ? - print help text to log window and exit macro */
 ,pLibIn1   = work   /* library with input data set 1 */
 ,pDSIn1    =        /* input data set 1 */
 ,pDSOpt1   =        /* options of data set 1 */
 ,pLibIn2   = work   /* library with input data set 2 */
 ,pDSIn2    =        /* input data set 2 */
 ,pDSOpt2   =        /* options of data set 2 */
 ,pKeys     =        /* key variables for by statement */
 ,pLibOut   = work   /* library for output data set */
 ,pDSOut    =        /* output data set */
 ,pDSOptOut =        /* options of output data set */
 ,pDSCode   =        /* additional datastep code */
 ,pDebug    = no     /* = (yes, y) - debug mode on */
                     /* = else     - debug mode off */
);
%local
  tDebug             /* Y/N - debug mode on/off */
  tDSID              /* data set identifier */
  tErr               /* 0 - no errors found */
                     /* 1 - error, stop macro execution */
  tMacName           /* name of macro */
  tMsgTxt            /* text for log message */
  tNewLib            /* name of temporary library */
  tRC                /* result of close statement */
  tVar               /* current variable in list pKeys */
  tVarCnt            /* current element number in list pKeys */
  tVarNum            /* position of the variable in the data set */
;

/*************************************************************************************************
**  print help to log
*************************************************************************************************/
  %if (&pHelp. eq ?) %then %do;
    %put %nrstr(%LeftJoin);
    %put %nrstr(Perform a left join.);
    %put ;
    %put %nrstr(Syntax:);
```

```
    %put %nrstr(%LeftJoin%(<pLibIn1>, pDSIn1, <pDSOpt1>,);
    %put %nrstr(          <pLibIn2>, pDSIn2, <pDSOpt2>, pKeys,);
    %put %nrstr(          <pLibOut>, pDSOut, <pDSOptOut>, <pDSCode>%););
    %put ;
    %put %nrstr(Example call:);
    %put %nrstr(%LeftJoin%();
    %put %nrstr(  pLibIn1   = lib1);
    %put %nrstr( ,pDSIn1    = dsn1);
    %put %nrstr( ,pLibIn2   = lib2);
    %put %nrstr( ,pDSIn2    = dsn2);
    %put %nrstr( ,pKeys     = usubjid dsnseq);
    %put %nrstr( ,pLibOut   = work);
    %put %nrstr( ,pDSOut    = dsnout);
    %put %nrstr(%););

    %GOTO HelpExit;
  %end; /* %if (&pHelp. eq ?) */

/*****************************************************************************************************
**  macro intro
*****************************************************************************************************/
  %let tMacName = &sysmacroname.;
  %let tMsgTxt  = ##### Begin of macro &tMacName. #####;
  %put &tMsgTxt.;

/*****************************************************************************************************
**  set debug mode
*****************************************************************************************************/
  %if (%upcase(&pDebug.) eq YES) or (%upcase(&pDebug.) eq Y) %then %do;
    %let tDebug = Y;
  %end;
  %else %do;
    %let tDebug = N;
  %end;

/*****************************************************************************************************
**  parameter check
*****************************************************************************************************/
  %let tErr = 0;
  /* check parameter exist */
  %if %quote(&pDSIn1.) eq %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Parameter pDSIn1 is empty !);
    %let tErr = 1;
  %end;
  %if %quote(&pDSIn2.) eq %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Parameter pDSIn2 is empty !);
    %let tErr = 1;
  %end;
  %if %quote(&pDSOut.) eq %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Parameter pDSOut is empty !);
    %let tErr = 1;
  %end;
  %if %quote(&pKeys.) eq %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Parameter pKeys is empty !);
    %let tErr = 1;
  %end;
  %if &tErr. ne 0 %then %do; /* if error stop macro execution */
    %let tMsgTxt = %str(&tMacName. - stop macro execution !);
    %GOTO ErrExit;
  %end;

  /* check library exist */
  %if %sysfunc(libref(&pLibIn1.)) ne 0 %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Library &pLibIn1. doesn%'t exist !);
    %let tErr = 1;
  %end;
  %if %sysfunc(libref(&pLibIn2.)) ne 0 %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Library &pLibIn2. doesn%'t exist !);
    %let tErr = 1;
  %end;
  %if %sysfunc(libref(&pLibOut.)) ne 0 %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Library &pLibOut. doesn%'t exist !);
    %let tErr = 1;
  %end;

  /* check datasets exist */
  %if %sysfunc(exist(&pLibIn1..&pDSIn1.)) eq 0 %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Dataset &pLibIn1..&pDSIn1. doesn%'t exist !);
```

```
    %let tErr = 1;
  %end;
  %if %sysfunc(exist(&pLibIn2..&pDSIn2.)) eq 0 %then %do;
    %put %str(ERR)%str(OR: &tMacName. - Dataset &pLibIn2..&pDSIn2. doesn%'t exist !);
    %let tErr = 1;
  %end;
  %if &tErr. ne 0 %then %do; /* if error stop macro execution */
    %let tMsgTxt = %str(&tMacName. - stop macro execution !);
    %GOTO ErrExit;
  %end;

  /* check key variables exist in pDSIn1 */
  %let tVarCnt = 1;
  %let tVar = %qscan(&pKeys., &tVarCnt., ' ');
  %do %while(&tVar. ne); /* loop over all key variables */
    %let tDSID   = %sysfunc(open(&pLibIn1..&pDSIn1., i));
    %let tVarNum = %sysfunc(varnum(&tDSID., &tVar.));
    %let tRC     = %sysfunc(close(&tDSID.));
    %if &tVarNum. le 0 %then %do;
      %let tMsgTxt = %str(Key variable &tVar. doesn%'t exist in Dataset &pLibIn1..&pDSIn1. !);
      %put %str(ERR)%str(OR: &tMacName. - &tMsgTxt.);
      %let tErr = 1;
    %end;
    %let tVarCnt = %eval(&tVarCnt. + 1);
    %let tVar = %qscan(&pKeys., &tVarCnt., ' ');
  %end; /* %do %while(&tVar. ne) */

  /* check key variables exist in pDSIn2 */
  %let tVarCnt = 1;
  %let tVar = %qscan(&pKeys., &tVarCnt., ' ');
  %do %while(&tVar. ne); /* loop over all key variables */
    %let tDSID   = %sysfunc(open(&pLibIn2..&pDSIn2., i));
    %let tVarNum = %sysfunc(varnum(&tDSID., &tVar.));
    %let tRC     = %sysfunc(close(&tDSID.));
    %if &tVarNum. le 0 %then %do;
      %let tMsgTxt = %str(Key variable &tVar. doesn%'t exist in Dataset &pLibIn2..&pDSIn2. !);
      %put %str(ERR)%str(OR: &tMacName. - &tMsgTxt.);
      %let tErr = 1;
    %end;
    %let tVarCnt = %eval(&tVarCnt. + 1);
    %let tVar = %qscan(&pKeys., &tVarCnt., ' ');
  %end; /* %do %while(&tVar. ne) */
  %if &tErr. ne 0 %then %do; /* if error stop macro execution */
    %let tMsgTxt = %str(&tMacName. - stop macro execution !);
    %GOTO ErrExit;
  %end;

  /* add () to option parameter */
  %if (%quote(&pDSOpt1.) ne ) %then %do;
    %let pDSOpt1 = (&pDSOpt1.);
  %end;
  %if (%quote(&pDSOpt2.) ne ) %then %do;
    %let pDSOpt2 = (&pDSOpt2.);
  %end;
  %if %quote(&pDSOptOut.) ne %then %do;
    %let pDSOptOut = (&pDSOptOut.);
  %end;

/*****************************************************************************************************
**  print macro parameter setting to log window
*****************************************************************************************************/
  %put %nrstr(%LeftJoin %();
  %put   %str(  pLibIn1   = &pLibIn1.);
  %put   %str( ,pDSIn1    = &pDSIn1.);
  %put   %str( ,pDSOpt1   = &pDSOpt1.);
  %put   %str( ,pLibIn2   = &pLibIn2.);
  %put   %str( ,pDSIn2    = &pDSIn2.);
  %put   %str( ,pDSOpt2   = &pDSOpt2.);
  %put   %str( ,pKeys     = &pKeys.);
  %put   %str( ,pLibOut   = &pLibOut.);
  %put   %str( ,pDSOut    = &pDSOut.);
  %put   %str( ,pDSOptOut = &pDSOptOut.);
  %put   %str( ,pDSCode   = &pDSCode.);
  %put %nrstr(  %));

/*****************************************************************************************************
**  create temporary library
*****************************************************************************************************/
```

```
  %CreateLib(tNewLib);

/*****************************************************************************************************
**  sort both datasets
*****************************************************************************************************/
  proc sort data = &pLibIn1..&pDSIn1. &pDSOpt1. out = &tNewLib..tDSIn1;
    by &pKeys.;
  run;
  proc sort data = &pLibIn2..&pDSIn2. &pDSOpt2. out = &tNewLib..tDSIn2;
    by &pKeys.;
  run;

/*****************************************************************************************************
**  merge both datasets
*****************************************************************************************************/
  data &pLibOut..&pDSOut. &pDSOptOut.;
    merge &tNewLib..tDSIn1(in = a) &tNewLib..tDSIn2;
    by &pKeys.;
    if a;
    &pDSCode.;
  run;

/*****************************************************************************************************
**  delete temporary datasets
*****************************************************************************************************/
  %if &tDebug. eq N %then %do; /* debug mode off */
    proc datasets lib=&tNewLib. nolist kill memtype=all;
    quit;
    libname &tNewLib. clear;
  %end;

  %GOTO Exit;

%ErrExit:
  %put %str(ERR)%str(OR: &tMsgTxt.);

%Exit:
  %let tMsgTxt  = ##### End of macro &tMacName. #####;
  %put &tMsgTxt.;

%HelpExit:
%mend;
/*****************************************************************************************************
**  End of Macro
*****************************************************************************************************/
```

Listing Makro %CreateLib

```
/*************************************************************************************************
**  Macro Name:            %CreateLib
**  Topic:                 SAS FILE I/O
**  Requirements:          ..\Macros\CreateLib\CreateLib_URS.doc
**  Source File:           ..\Macros\CreateLib\CreateLib.sas
**  Test Program:          ..\Macros\CreateLib\CreateLib_UAT.sas
**  Manual:                ..\Macros\CreateLib\CreateLib_UM.doc
**  Company/Department:    -
**  Software(Dev/Sys):     SAS Version 9.1.3 / Windows XP
**-----------------------------------------------------------------------------------------------
**  Short Description:     Create a library in the directory of library WORK.
**  Input:                 -
**  Output:                Name of new library.
**  Return Value:          -
**  Comments:              -
***************************** H I S T O R Y ***************************************************
**  Developer     Date     Remarks
**  ---------  ----------  ---------------------------------------------------------------------
**  HaS        02.08.2009  Start of development
**  HaS        07.08.2009  Implementation finished
*************************************************************************************************/

/*************************************************************************************************
Example call:
%let tNewLib = ;
%CreateLib(tNewLib);
%put &tNewLib.;
*************************************************************************************************/

%macro CreateLib(
  pNewLib            /* output parameter - name of new library */
                     /* = ? - print help text to log window and exit macro */
 ,pDebug    = no     /* = (yes, y) - debug mode on */
                     /* = else     - debug mode off */
);
%local
  tDebug             /* Y/N - debug mode on/off */
  tResult            /* number of macro library entries in sashelp.vlibnam */
  tLibName           /* name of temporary macro library */
  tLibNum            /* number of temporary macro library */
  tLibPath           /* path of temporary library */
  tMacName           /* name of macro */
  tMaxLibNum         /* max. number of macro libraries */
  tMsgTxt            /* text for log message */
  tWorkPath          /* path of library WORK */
  tXWait             /* current value of system option XWAIT */
;
  %let tMaxLibNum = 10000;

/*************************************************************************************************
**  print help to log
*************************************************************************************************/
  %if (&pNewLib. eq ?) %then %do;
    %put %nrstr(%CreateLib);
    %put %nrstr(Create a library in the directory of library WORK.);
    %put ;
    %put %nrstr(Syntax:);
    %put %nrstr(%CreateLib%(pNewLib;);
    %put ;
    %put %nrstr(Example call:);
    %put %nrstr(%let tNewLib = ;);
    %put %nrstr(%CreateLib%(tNewLib%););
    %put %nrstr(%put &tNewLib.;);

    %GOTO HelpExit;
  %end; /* %if (&pNewLib. eq ?) */

/*************************************************************************************************
**  macro intro
*************************************************************************************************/
  %let tMacName = &sysmacroname.;
  %let tMsgTxt  = ##### Begin of macro &tMacName. #####;
  %put &tMsgTxt.;

/*************************************************************************************************
```

```
**  set debug mode
*************************************************************************************************/
  %if (%upcase(&pDebug.) eq YES) or (%upcase(&pDebug.) eq Y) %then %do;
    %let tDebug = Y;
  %end;
  %else %do;
    %let tDebug = N;
  %end;

/************************************************************************************************
**  parameter check
*************************************************************************************************/
  /* check parameter exist */
  %if %quote(&pNewLib.) eq %then %do;
    %let tMsgTxt = %str(&tMacName. - Parameter pNewLib is empty !);
    %GOTO ErrExit;
  %end;

  /* check macro variable exist */
  %if %symexist(&pNewLib.) eq 0 %then %do;
    %let tMsgTxt = %str(&tMacName. - Macro variable &pNewLib. doesn%'t exist !);
    %GOTO ErrExit;
  %end;

/************************************************************************************************
**  print macro parameter setting to log window
*************************************************************************************************/
  %put %nrstr(%CreateLib %();
  %put   %str(  pNewLib   = &pNewLib.);
  %put %nrstr(  %));

/************************************************************************************************
**  build name for temporary library
*************************************************************************************************/
  %let tLibName = ;
  %let tLibNum = 0;
  %do %until ((%eval(&tResult.) ne 0) or (&tLibNum. ge &tMaxLibNum.)) ;
    %let tLibNum  = %eval(&tLibNum. + 1);
    %let tLibName = %cmpres(%sysfunc(upcase(MLib%sysfunc(putn(&tLibNum., z4.)))));
    %let tResult  = %sysfunc(libref(&tLibName.));
    %if &tDebug. eq Y %then %do; /* debug mode on */
      %put tLibNum  - &tLibNum.;
      %put tLibName - &tLibName.;
      %put tResult  - &tResult.;
    %end;
  %end; /* %do %until ((%eval(&tResult.) ne 0) or (&tLibNum. ge &tMaxLibNum.)) */
  %if &tLibNum. gt &tMaxLibNum. %then %do;
    %let tMsgTxt = %str(&tMacName. - A library couldn%'t be allocated !);
    %GOTO ErrExit;
  %end;

/************************************************************************************************
**  create directory for temporary library
*************************************************************************************************/
  %let tLibPath = %sysfunc(pathname(WORK))\&tLibName.;  /* path for temporary library */

  %if &tDebug. eq Y %then %do; /* debug mode on */
    %let tMsgTxt = %str(&tMacName. - Directory for temporary library: &tLibPath. !);
    %put &tMsgTxt.;
  %end;

  %let tXWait = %sysfunc(getoption(xwait));   /* create directory if doesn't exist */
  %if %sysfunc(fileexist(&tLibPath.)) ne 1 %then %do;
    option noxwait;
    x "md %bquote(")&tLibPath.%bquote(")";
                /* " /* to  make the syntax coloring working well again */
    option &tXWait.;
  %end;

/************************************************************************************************
**  create temporary library
*************************************************************************************************/
  libname &tLibName. "&tLibPath.";
  %let &pNewLib. = &tLibName.; /* save library name to macro parameter */

  %GOTO Exit;

%ErrExit:
```

```
  %put %str(ERR)%str(OR:  &tMsgTxt.);

%Exit:
  %let tMsgTxt  = ##### End of macro &tMacName. #####;
  %put &tMsgTxt.;

%HelpExit:
%mend;
/*************************************************************************************************
**  End of Macro
*************************************************************************************************/
```

Benutzerhandbuch Makro %LeftJoin

%LeftJoin
Führt einen Left Join aus.

Typ: Macro
Topic: Merge

Syntax

%LeftJoin*(<pLibIn1>, pDSIn1, <pDSOpt1>, <pLibIn2>, pDSIn2, <pDSOpt2>, pKeys, <pLibOut>, pDSOut, <pDSOptOut>, <pDSCode>)*

Parameter

pLibIn1
Name der Library des ersten Input Datasets.
Falls **pLibIn1** nicht angegeben wird, wird der erste Input Dataset in der Library WORK gesucht.

pDSIn1
Name des ersten Input Dataset.

pDSOpt1
Dataset Option zum ersten Input Dataset.

pLibIn2
Name der Library des zweiten Input Datasets.
Falls **pLibIn2** nicht angegeben wird, wird der zweite Input Dataset in der Library WORK gesucht.

pDSIn2
Name des zweiten Input Dataset.

pDSOpt2
Dataset Option zum zweiten Input Dataset.

pKeys
Liste mit Key-Variablen für das By-Statement beim Merge.

pLibOut
Name der Library für den Output Dataset.
Falls **pLibOut** nicht angegeben wird, wird der Ergebnis-Dataset in der Library WORK abgelegt.

pDSOut
Name des Output Datasets.

pDSOptOut
Dataset Option zum Output Dataset.

pDSCode
Zusätzliche Datastep Instruktionen.

Beschreibung

Ein Left Join wird ausgeführt, nachdem beide Input Datasets sortiert wurden:

```
data &pLibOut..&pDSOut. &pDSOptOut.;
  merge &pLibIn1..&pDSIn1(in = a) &pLibIn2..&pDSIn2;
  by &pKeys.;
  if a;
  &pDSCode;
run;
```

Beispiele

Beispiel 1:

Leftjoin zweier Datasets.

dm:

STUDYID	SUBJID	SEX
studyA	2001	male
studyA	2002	female
studyA	2003	female
studyA	2004	male
studyA	2005	female
studyA	2006	male
studyB	2001	male
studyB	2002	female

country:

STUDYID	SUBJID	COUNTRY
studyA	2001	GERMANY
studyA	2004	ITALY
studyA	2002	AUSTRIA
studyA	2003	FRANCE

SAS Statements

```
%LeftJoin(
  pDSIn1    = dm
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = dm2
);
```

Ergebnisse

dm2:

STUDYID	SUBJID	SEX	COUNTRY
studyA	2001	male	GERMANY
studyA	2002	female	AUSTRIA
studyA	2003	female	FRANCE
studyA	2004	male	ITALY
studyA	2005	female	
studyA	2006	male	
studyB	2001	male	
studyB	2002	female	

Referenzen

H. Stürzl: SAS Makro UNISTATS – Ein universelles Werkzeug, 13. KSFE 2009, Halle/Saale
G. Redner, L. Zhang, C. Herremans: Techniques for Developing Quality SAS® Macros, SESUG 2008, Paper SBC-121
F. Ivis: Guidelines on Writing SAS® Macros for Public Use, SUGI 29, Paper 047-29