Richtlinien zur Programmierung von Standard SAS Makros

Hanspeter Schnitzer
Merz Pharmaceuticals GmbH
Eckenheimer Landstraße 100
D-60318 Frankfurt am Main
Hanspeter.Schnitzer@Merz.de

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. 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.

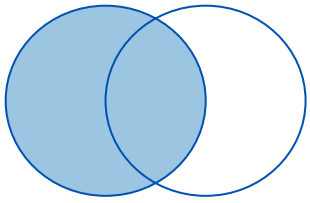
Schlüsselwörter: Merge, Left Join, Keyvariablen, Makroparameter, Keyword Parameter, Positionsparameter, Parameter Default Value, Datasetoption, Parameterrückgabe, Parameter Check, temporäre Library, Systemoption

1 Beispielmakros

In den nachfolgenden Ausführungen wird auf die Makros %LeftJoin und %CreateLib Bezug genommen, die als Beispiele entwickelt wurden. Ein Listing der beiden Makros befindet sich im Anhang.

1.1 %LeftJoin

Das Makro %LeftJoin führt einen Merge bzw. einen 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;
```

1.2 %CreateLib

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

Beispielaufruf:

```
%let tNewLib = ;
%CreateLib(tNewLib);
```

2 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.

2.1 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 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

2.2 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);
```

2.3 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.

2.4 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
...
%if (&pHelp. eq ?) %then %do;
  %put %nrstr(%LeftJoin);
  %put   %nrstr(Perform   a   left
join.);
 ...
```

Beispielaufruf:

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

2.5 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')
...
);
```

Beispiel (kombinierte Parameter):

```
%LeftJoin(
  pDSIn1    = lib1.dm(where = (sex eq 'male'))
...
);
```

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.

2.6 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:

```
%LeftJoin(
  pDSIn1    = dm
 ,pDSIn2    = country
 ,pKeys     = studyid subjid
 ,pDSOut    = dm2
 ,pDSCode   = if missing(country) then country = '*'
);
```

Resultierender Datastep:

```
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.

2.7 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(
 ...
,pDebug    = no                    /* = (yes, y) - debug mode on */
                                   /* = else     - debug mode off */
);
 ...
%if (%upcase(&pDebug.) eq YES) or
    (%upcase(&pDebug.) eq Y) %then %do;
  %let tDebug = Y;
%end;
%else %do;
  %let tDebug = N;
%end;
 ...
%if &tDebug. eq N %then %do;                    /* debug mode off */
  proc datasets lib=&tNewLib. nolist kill memtype=all;
  quit;
  libname &tNewLib. clear;
%end;
```

2.8 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.

```
%macro CreateLib(                  %let tNewLib = ;
  pNewLib                          %CreateLib(tNewLib);
...
  %let &pNewLib. = &tLibName.;
%mend;
```

2.9 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.

3 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.
Im Nachfolgenden sind Beispiele für einige Testfälle aufgeführt.

3.1 Inhalt des Makroparameters ist leer

```
%if %quote(&pParam.) eq %then %do;
  %put %str(Parameter pParam is empty !);
   ...
```

3.2 Library existiert nicht

```
%if %sysfunc(libref(&pParam.)) ne 0 %then %do;
  %put %str(Library &pParam. doesn%'t exist !);
  ...
```

3.3 Dataset existiert nicht

```
%if %sysfunc(exist(&pParam.)) eq 0 %then %do;
  %put %str(Dataset &pParam. doesn%'t exist !);
  ...
```

3.4 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(Dataset variable &pParam. doesn%'t exist !);
   ...
```

3.5 Typ der Datasetvariablen ist ungültig

Es wird überprüft, ob die Datasetvariable vom Typ numerisch ist:

```
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;
  %put %str(Dataset variable &pParam. isn%'t from type numeric !);
  ...
```

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

3.6 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(Format &pParam. not found !);
  ...
```

3.7 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(Dataset &pParam. is empty !);
  ...
```

3.8 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(Directory &pParam. doesn%'t exist !);
  ...
%end;
%else %do;
  %let tResult = %sysfunc(dclose(&tRes.));
%end;
%let tResult = %sysfunc(filename(tFileRef));
```

3.9 Datei existiert nicht

```
%if %sysfunc(fileexist(&pParam.)) ne 1 %then %do;
  %put %str(File &pParam. doesn%'t exist !);
  ...
```

3.10 Makrovariable existiert nicht

```
%if %symexist(&pParam.) eq 0 %then %do;
  %put %str(Macro variable &pParam. doesn%'t exist !);
  ...
```

4 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.

4.1 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.

4.2 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 lassen, 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.

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;
%if %sysfunc(fileexist(&tLibPath.)) ne 1 %then %do;
  %let tXWait = %sysfunc(getoption(xwait));
  option noxwait;
  x "md %bquote(")&tLibPath.%bquote(")";
  option &tXWait.;
%end;
libname TempLib "&tLibPath.";          /* create temporary library */
...
```

```
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 des Makros 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. 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.

4.3 Temporäre Formate

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

Hierzu verwenden wir die Systemoption FMTSEARCH. Zunächst müssen wir den aktuellen Wert der Option auslesen und zwischenspeichern:

```
%let tFMT1 = %sysfunc(getoption(FMTSEARCH));
```

Nun können wir die Option ändern und die temporäre Library TempLib einsetzen:

```
option FMTSEARCH = (TempLib);
```

Falls nicht nur die Formate aus der Library TempLib verwendet werden sollen, sondern zusätzlich auch die Formate, die in den vor Aufruf des Makros definierten Libraries liegen, modifizieren wir den Befehl wie folgt:

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

Formate, die wir temporär anlegen wollen, speichern wir in die Library TempLib, z. B.:

```
proc format library= TempLib;
  value $country 'DEU'='GERMANY'
                 'AUT'='AUSTRIA'
                 'FRA'='FRANCE'
                 'ITA'='ITALY'
run;
```

Am Ende des Makros müssen wir die Systemoption wieder in den ursprünglichen Zustand versetzen:

```
option FMTSEARCH = &tFMT1.;
```

4.4 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. Hierzu speichern wir zu Beginn des Makros die aktuelle Einstellung der Option in eine Makrovariable:

```
%let tXWait = %sysfunc(getoption(xwait));
```

Im weiteren Verlauf können wir die Option auf den gewünschten Wert setzen:

```
option noxwait;
```

Am Ende des Makros wird die Option auf den ursprünglichen Wert zurückgesetzt:

```
option &tXWait.;
```

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

```
proc optsave out=tOptions; run;                /* save all options */
  ...
proc optload data=tOptions; run;              /* reset all options */
```

4.5 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.

Zunächst speichern wir den aktuellen Wert der Titles:

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

Am Ende des Makros setzen wir die Titles wieder zurück:

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

Anhang A: Listing Makro %LeftJoin

Das komplette SAS-Makro %LeftJoin finden Sie im deutschsprachigen SAS-Wiki unter http://de.saswiki.org/wiki/KSFE2010.

Anhang B: 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
*************************************************************/
%macro CreateLib(
  pNewLib             /* output parameter - name of new library */
                      /* = ? - print help text to log window */
);
%local
  tResult             /* number of macro library entries */
  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 */
  tXWait              /* current value of system option XWAIT */
;
%let tMaxLibNum = 10000;
```

```
/*************************************************************
**  macro intro
*************************************************************/
%let tMacName = &sysmacroname.;
%put %str(##### Begin of macro &tMacName. #####);

/*************************************************************
**  parameter check
*************************************************************/
/* check parameter exist */
%if %quote(&pNewLib.) eq %then %do;
  %put %str(ERR)%str(OR: &tMacName. - Parameter pNewLib is empty !);
  %GOTO ErrExit;
%end;

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

/*************************************************************
**  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.));
%end; /* %do %until ((%eval(&tResult.) ne 0) or ... */
%if &tLibNum. gt &tMaxLibNum. %then %do;
  %put %str(ERR)%str(OR: &tMacName. - A library couldn%'t be
                                      allocated !);
  %GOTO ErrExit;
%end;

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

```
%let tXWait = %sysfunc(getoption(xwait));
%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.; /* return value with library name */

%ErrExit:
  %put %str(##### End of macro &tMacName. #####);
%HelpExit:
%mend;
/*************************************************************
**  End of Macro
*************************************************************/
```

Literatur

[1] H. Stürzl: SAS Makro UNISTATS – Ein universelles Werkzeug, 13. KSFE 2009, Halle/Saale

[2] G. Redner, L. Zhang, C. Herremans: Techniques for Developing Quality SAS Macros, SESUG 2008, Paper SBC-121

[3] F. Ivis: Guidelines on Writing SAS Macros for Public Use, SUGI 29, Paper 047-29

[4] H. Schnitzer: Richtlinien zur Programmierung von Standard SAS Makros, www.sasmacro.de