SAS Makro %CheckPars – Makroparametercheck Deluxe

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

Zusammenfassung

Dieser Beitrag möchte den Vortrag ‚Richtlinien zur Programmierung von Standard SAS Makros' fortsetzen, der auf der 14. KSFE in Berlin gehalten wurde.
Standard Makros, also Makros, die für den allgemeinen Gebrauch zur Verfügung gestellt werden sollen, müssen u. a. robust und benutzerfreundlich sein. Dazu zählt ein möglichst lückenloser Parametercheck, mit dem eine Fehlbedienung des Makros schon im Vorfeld vermieden werden kann. Mit dem SAS Makro %CheckPars wird der Makroparametercheck zum Kinderspiel. Mit einem einzigen Makroaufruf können die jeweiligen Makroparameter auf folgende Eigenschaften überprüft werden:
Parameter ist nicht leer, Dataset ist nicht leer, Existenz von Libraries, Datasets, Datasetvariablen, Formaten, Verzeichnissen, Dateien, Makrovariablen.
Bevor wir uns dem Makro %CheckPars zuwenden, wird eine Möglichkeit aufgezeigt, wie wir verhindern können, dass im %local Statement Makrovariablen vergessen werden. Eine Ergänzung zur Parameterrückgabe rundet den Beitrag ab.

Schlüsselwörter: Anweisungsüberdeckung, Dictionary Tables, Makroparameter, Makrovariable, Parameterrückgabe, Parameter Check

1 Vorbemerkungen

Nachdem wir uns auf der letzten KSFE [1] damit beschäftig haben, wie wir unsere Makros robust und benutzerfreundlich gestalten können, wollen wir diesmal versuchen, den damit verbundenen Aufwand zu reduzieren. In den nachfolgenden Ausführungen werden wir öfter einmal ein Makro verwenden, welches zu einer Jahreszahl den Austragungsort der KSFE zurückgibt (%KSFECity). Deshalb vorab eine Beschreibung des Makros.

```
%macro KSFECity(pYear, pCity);
%local CityName;
  %let CityName = ;
  %if &pYear. lt 1997 %then %do;
    %put WARNING: KSFE ging erst 1997 los;
  %end;
  %else %do;
    %if &pYear. eq 2011 %then %let CityName = Heidelberg;
    %else %if &pYear. eq 2010 %then %let CityName = Berlin;
  %end;
```

```
  %let &pCity. = &CityName.;
%mend;
```

Beispielaufruf:

```
%let City = ;
%KSFECity(2010, City);
%put 2010 war die KSFE in &City. ;
```

Output ins Logfenster:

```
2010 war die KSFE in Berlin
```

2 %local

Bei größeren Makros ist es nicht einfach, den Überblick über die verwendeten Makrovariablen zu behalten. Schnell passiert es, dass eine Makrovariable nicht gleich im %local Statement eingefügt wird. Eine nachträgliche Überprüfung ist zeitaufwändig und muss theoretisch nach jeder Änderung des Makros wiederholt werden. Andererseits ist es unerlässlich, sämtliche verwendeten Makrovariablen im %local Statement aufzulisten. Andernfalls drohen unerwünschte Fehlersituationen.

Bei der Lösung des Problems bedienen wir uns der Dictionary Table MACROS bzw. SASHELP.VMACRO. Hier werden sämtliche aktuell existierenden Makrovariablen eingetragen. Was wir tun müssen ist also, zu Beginn eines Makros, aber nach dem %local Statement, den Stand von SASHELP.VMACRO zu speichern (mVarsSt) und für einen späteren Merge zu sortieren.

```
data mVarsSt;
  set sashelp.vmacro;
run;
proc sort data=mVarsSt; by scope name; run;
```

Diesen Dataset müssen wir mit dem Stand am Ende des Makros (mVarsEn) vergleichen. Die ggf. gefundenen nicht deklarierten Makrovariablen werden sowohl einmal alle untereinander ins Logfenster geschrieben als auch einmal als fortlaufende Liste.

```
proc sort data=sashelp.vmacro out=mVarsEn; by scope name; run;
data _null_;
length n 8. VarList $2000. ;
retain n 0  VarList;
  merge mVarsEn (in=En) mVarsSt (in=St) end=eof ;
  by scope name;
  if En and not St then do;
    if n eq 0 then put 'Undefined macro variable(s):';
    put name;
    n = n + 1;
    VarList = catx(' ', VarList, name);
  end;
  if eof then do;
```

```
    if n eq 0 then put 'No undefined macro variables found!';
              else put VarList;
  end;
run;
```

Sind am Ende des Makros in SASHELP.VMACRO Makrovariablen hinzugekommen, wurden diese im Makro nicht mit %local deklariert. Das bedeutet allerdings auch einschränkend, dass wir nur ein einziges %local Statement zu Beginn eines jeden Makros verwenden dürfen. %local im weiteren Verlauf des Makros könnte bei unserem Vergleich ein falsches Ergebnis liefern.

Es bietet sich an, die beiden Passagen von oben jeweils in Makros zu packen. Dann erhalten wir die Makros %InitCheckMVars und %CheckMVars. Beide Makros können wir nun in jedes unserer Makros fest einbauen. Es wäre jedoch unerwünscht, wenn sie im Alltagsbetrieb ausgeführt würden. Deshalb bekommt jedes unserer Makros einen Parameter pDebug, mit dem wir einstellen können, ob das Makro getestet werden soll (pDebug = yes) oder ob es normal benutzt werden soll.

```
%If &pDebug eq yes %then %do;
  %InitCheckMVars;
%end;
```

Um unsere Makros zu vereinfachen, können wir die if-Abfrage auch in die beiden Makros %InitCheckMVars und %CheckMVars verlagern. Weiterhin müssen die Makrovariablen von %InitCheckMVars bzw. %CheckMVars ausgeblendet werden.

```
%macro InitCheckMVars(pDebug);
  %if &pDebug. eq yes %then %do;
    data mVarsSt;
      set sashelp.vmacro(where = (scope ne 'INITCHECKMVARS'));
    run;
    proc sort data=mVarsSt; by scope name; run;
  %end;
%mend;
```

Der Aufruf würde sich entsprechend vereinfachen:

```
%InitCheckMVars(&pDebug.);
```

%CheckMVars wird entsprechend gestaltet. In unserem Beispielmakro %KSFECity würden die beiden Makros wie folgt eingebaut:

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName;
  %InitCheckMVars(&pDebug.);
```

```
  %let CityName = ;
  %if &pYear. lt 1997 %then %do;
    %put WARNING: KSFE ging erst 1997 los;
  %end;
  %else %do;
    %if &pYear. eq 2011 %then %let CityName = Heidelberg;
    %else %if &pYear. eq 2010 %then %let CityName = Berlin;
  %end;
  %let &pCity. = &CityName.;

  %CheckMVars(&pDebug.);
%mend;
```

Nun rufen wir %KSFECity auf:

```
%let City = ;
%KSFECity(2010, City, yes);
%put 2010 war die KSFE in &City. ;
```

Output im Logfenster:

```
No undefined macro variables found!
2010 war die KSFE in Berlin
```

Hätten wir jedoch das %local Statement in %KSFECity vergessen, würde folgender Output im Logfenster erscheinen:

```
Undefined macro variable(s):
CITYNAME
CITYNAME
2010 war die KSFE in Berlin
```

Nun können wir uns allerdings nicht zurücklehnen. Es wäre ja auch zu schön. Einige Einschränkungen sind noch zu beachten.

2.1 Makrovariablen könnten bereits existieren

Schauen wir uns folgendes Beispiel an. Wir haben wieder die %local Deklaration in %KSFECity vergessen.

```
%let CityName = Frankfurt;
%put Start der Reise ist &CityName.;
%let City = ;
%KSFECity(2011, City, yes);
%put Ziel der Reise ist &City.;
```

Output im Logfenster:

```
No undefined macro variables found!
2010 war die KSFE in Berlin
```

In obigem Beispiel wird vor dem Aufruf von %KSFECity eine Makrovariable verwendet, die genau so heißt, wie die im Makro %KSFECity verwendete, nämlich CityName. Wenn wir nun im Makro %KSFECity wieder die %local Deklaration vergessen haben

sollten, erhalten wir trotzdem keine entsprechende Meldung im Logfenster. Das liegt daran, dass die Makrovariable CityName global bzgl. %KSFECity ist und dadurch auch in %KSFECity bekannt ist.

Wir müssen also dafür sorgen, während unserer Makrotests in den jeweiligen Testprogrammen möglichst keine Makrovariablen zu verwenden. Eine ähnliche Problematik werden wir übrigens in Kapitel 3 wiederfinden.

2.2 Programmzeilen werden nicht durchlaufen

Werden nicht alle Programmzeilen eines Makros durchlaufen, kann es vorkommen, dass eine bestimmte Makrovariable aktuell nicht benutzt wird. Sie würde dann auch nicht im Dictionary Table MACROS bzw. SASHELP.VMACRO eingetragen werden. In diesem Fall würde es nicht auffallen, wenn sie nicht mit %local deklariert wäre. Dem können wir entgegentreten, indem die Testprogramme der jeweiligen Makros mindestens die Bedingung der Anweisungsüberdeckung (C0-Test) erfüllen. Dies ist jedoch sowieso als Mindestanforderung für Makrotests anzusehen.
Im folgenden Makro %KSFECity wird eine Makrovariable Msg verwendet, aber nur dann, wenn pYear kleiner als 1997 ist.

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName;
  %InitCheckMVars(&pDebug.);
  %let CityName = ;
  %if &pYear. lt 1997 %then %do;
    %let Msg = WARNING: KSFE ging erst 1997 los;
    %put &Msg.;
  %end;
  %else %do;
    %if &pYear. eq 2011 %then %let CityName = Heidelberg;
    %else %if &pYear. eq 2010 %then %let CityName = Berlin;
  %end;
  %let &pCity. = &CityName.;
  %CheckMVars(&pDebug.);
%mend;
```

Folgender Aufruf würde die fehlende Deklaration der Makrovariablen Msg nicht entdecken:

```
%let City = ;
%KSFECity(2010, City, yes);
%put 2010 war die KSFE in &City. ;
```

Output ins Logfenster:

```
No undefined macro variables found!
2010 war die KSFE in Berlin
```

Unser Testprogramm müsste auch folgenden Fall abdecken:

```
%let City = ;
%KSFECity(1898, City, yes);
```

Output ins Logfenster:

```
WARNING: KSFE ging erst 1997 los
Undefined macro variable(s):
MSG
MSG
```

2.3 Dynamisch angelegte Makrovariablen

Etwas schwieriger kann es werden, wenn Makrovariablen während der Ausführung des Makros dynamisch angelegt werden. Wir haben unser Makro %KSFECity weiterentwickelt. Die Information, in welcher Stadt die KSFE stattgefunden hat, liegt nun in einem Dataset (KSFECity). In einem data _null_ Step wird für jedes Jahr eine Makrovariable angelegt, in der die jeweilige Stadt enthalten ist, also z.B. Def2010 wird auf Berlin gesetzt.

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName;
  %InitCheckMVars(&pDebug.);
  data _null_;
    set KSFECity;
    call symput('Def'||left(put(year, 8.)), strip(city));
  run;
  %let CityName = ;
  %if &pYear. lt 1997 %then %do;
    %put WARNING: KSFE ging erst 1997 los;
  %end;
  %else %do;
    %let CityName = &&Def&pYear.;
  %end;
  %let &pCity. = &CityName.;
  %CheckMVars(&pDebug.);
%mend;
```

Der Aufruf

```
%let City = ;
%KSFECity(2010, City, yes);
%put 2010 war die KSFE in &City. ;
```

würde folgende Meldung erzeugen:

```
Undefined macro variable(s):
DEF2010
DEF2011
DEF2010 DEF2011
2010 war die KSFE in Berlin
```

Wir müssen also dafür sorgen, dass auch die Makrovariablen Def2011 etc. mit %local deklariert werden. Und das, bevor wir %InitCheckMVars aufrufen. In obigem Beispiel könnten wir dem Problem z.B. mit folgenden Programmzeilen begegnen:

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName VarCnt;
%do VarCnt = 1997 %to 2011;
  %local Def&VarCnt. ;
%end;
  %InitCheckMVars(&pDebug.);
...
```

Das setzt allerdings voraus, dass wir genau wissen, welche Makrovariablen deklariert werden. Ist dies nicht möglich, müssen wir versuchen, den data _null_ Step ein zweites Mal und zwar zu Beginn des Makros auszuführen und dabei die Makrovariablen mit %local deklarieren.

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName VarStr;
data _null_;
length VarStr $2000.;
retain VarStr;
  set KSFECity end=eof;
  VarStr = catx(' ', VarStr, 'Def'||left(put(year, 8.)));
  if eof then call symput('VarStr', strip(VarStr));
run;
%local &VarStr.;
  %InitCheckMVars(&pDebug.);
...
```

2.4 PROC SQL

Ein letztes Problem hängt mit PROC SQL zusammen. PROC SQL legt beim ersten Aufruf Makrovariablen an, die teilweise lokale teilweise aber auch globale Gültigkeit haben. Würde unser Makro %KSFECity PROC SQL verwenden, bekämen wir z.B. folgende Meldung ins Logfenster:

```
Undefined macro variable(s):
SYS_SQL_IP_ALL
SYS_SQL_IP_STMT
SQLEXITCODE
SQLOBS
SQLOOPS
SQLRC
SQLXOBS
SYS_SQL_IP_ALL  SYS_SQL_IP_STMT  SQLEXITCODE  SQLOBS  SQLOOPS  SQLRC
SQLXOBS
```

Diesem Problem können wir am einfachsten aus dem Weg gehen, indem wir zu Beginn unseres Testprogramms einfach einmal PROC SQL aufrufen. Dadurch werden sämtliche von PROC SQL erzeugten Makrovariablen angelegt bzw. deklariert und sind dann global gültig.

3 Parameterrückgabe

Bleiben wir bei unserem Beispiel %KSFECity und wenden uns folgenden Programmzeilen zu:

```
%let CityName = xxx;
%KSFECity(2010, CityName, yes);
%put 2010 war die KSFE in &CityName. ;
```

Output im Logfenster:

```
No undefined macro variables found!
2010 war die KSFE in xxx
```

Was ist passiert? Nun, das Problem liegt darin, dass unser rufendes Programm jetzt die Makrovariable CityName verwendet und nicht mehr City. Auch %KSFECity verwendet die Makrovariable CityName. Schauen wir uns die entscheidende Stelle ganz am Ende von %KSFECity einmal an.

```
  %let &pCity. = &CityName.;
```

Hier wird die gefundene Stadt unserem Rückgabeparameter zugewiesen. Wenn wir die Makrovariablen in dieser Zeile einmal auflösen, bleibt

```
  %let CityName = Berlin;
```

Wie soll das Makro nun wissen, dass mit CityName nicht die lokale Makrovariable CityName gemeint ist, sondern die vom rufenden Programm angelegte Makrovariable CityName? Tatsächlich kennt %KSFECity lediglich die eigens als lokal deklarierte Makrovariable CityName und schreibt den Wert Berlin dorthin, was folgender Instruktion entspricht:

```
  %let CityName = &CityName.;
```

Die vom rufenden Programm angelegte Makrovariable CityName bleibt also unberührt. Diesem Problem können wir begegnen, indem wir überprüfen, ob die vom Anwender im Rückgabeparameter pCity spezifizierte Makrovariable nicht zufällig auch vom Makro %KSFECity verwendet wird. Ist dies der Fall, verlassen wir %KSFECity mit einer Fehlermeldung.

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName;
  %InitCheckMVars(&pDebug.);
  %if %upcase(&pCity.) eq CITYNAME %then %do;
    %put ERROR: pCity=&pCity. is not a valid macro variable name !;
  %end;
```

```
  %else %do;
    %let CityName = ;
    %if &pYear. lt 1997 %then %do;
      %put WARNING: KSFE ging erst 1997 los;
    %end;
    %else %do;
      %if &pYear. eq 2011 %then %let CityName = Heidelberg;
      %else %if &pYear. eq 2010 %then %let CityName = Berlin;
    %end;
    %let &pCity. = &CityName.;
  %end;
  %CheckMVars(&pDebug.);
%mend;
```

Für %KSFECity haben wir das Problem damit gelöst. Stellen wir uns aber ein Makro vor, welches viele lokale Makrovariablen verwendet, so wird es doch recht umständlich. Wir müssten für jede lokale Makrovariable eine Abfrage einbauen. Hierfür hätte ich folgenden Vorschlag, nicht unbedingt schön, aber auf jeden Fall einfacher. Wir lassen jede lokale Makrovariable mit einem speziellen Text beginnen, z.B. dem Makronamen. Dann kommen wir mit einer Abfrage aus. Die vom rufenden Programm spezifizierte Makrovariable darf nun nicht mit diesem Text beginnen.

```
%macro KSFECity(pYear, pCity, pDebug);
%local KSFECityCity;
  %InitCheckMVars(&pDebug.);
  %if %length(&pCity.) ge 8 %then %do;
    %if %upcase(%substr(&pCity., 1, 8)) eq KSFECITY %then %do;
      %put ERROR: pCity=&pCity. is not a valid macro variable name!;
      %GOTO ErrExit;
    %end;
  %end;
  %else %do;
    %let KSFECityCity = ;
    %if &pYear. lt 1997 %then %do;
      %put WARNING: KSFE ging erst 1997 los;
    %end;
    %else %do;
      %if &pYear. eq 2011 %then %let KSFECityCity = Heidelberg;
      %else %if &pYear. eq 2010 %then %let KSFECityCity = Berlin;
    %end;
    %let &pCity. = &KSFECityCity.;
  %end;
%ErrExit:
  %CheckMVars(&pDebug.);
%mend;
```

4 %CheckPars

Wenden wir uns nun dem Parametercheck zu. Folgende Checks sowie die zugehörigen Abfragetechniken haben wir in [2] behandelt:

- Inhalt des Makroparameters ist leer
- Library existiert nicht
- Dataset existiert nicht
- Datasetvariable existiert nicht
- Typ der Datasetvariablen ist ungültig
- Format existiert nicht
- Dataset ist leer
- Verzeichnis existiert nicht
- Datei existiert nicht
- Makrovariable existiert nicht

Hat unser Makro, in das wir den Parametercheck einbauen wollen, viele Parameter mit vielen notwendigen Checks, kann der Aufwand für den Check sehr groß werden. Zusätzlich wird das Makro unübersichtlich und unnötig lang. Also bietet es sich an, ein Makro zu schreiben, welches uns diese Arbeit abnimmt. Zumal sich die oben aufgelisteten Abfragen sinnvoll in einem einzigen Makro realisieren lassen.
Ein Vorschlag für ein solches Makro ist %CheckPars, welches im Folgenden beschrieben wird.

%CheckPars führt einen Parametercheck durch und gibt eine Fehlermeldung im Logfenster aus, falls ein Parameter eine Checkbedingung nicht erfüllt. Zusätzlich wird ein Rückgabeparameter mit dem Ergebnis des Checks gesetzt.

4.1 Parameter

Beim Aufruf von %CheckPars werden die Parameter, die überprüft werden sollen, direkt übergeben, also nicht referenziert.
Es können beliebig viele auch verschiedene Checks mit einem einzigen Aufruf von %CheckPars ausgeführt werden.
Falls nicht anders spezifiziert, werden die zu überprüfenden Parameter innerhalb eines Checks mit Leerzeichen getrennt.
Sollte ein Check einen Dataset benötigen, kann dieser Dataset durch einen Parameter mit dem Datasetnamen spezifiziert werden, falls sich der Dataset in der Library WORK befindet (z.B. pDS). Andernfalls werden zwei Parameter benötigt, einer mit dem Librarynamen und einer mit dem Datasetnamen. Beide werden durch einen Punkt getrennt (z.B. pLib.pDS).

Auf unser Makro %KSFECity angewendet würde %CheckPars wie folgt verwendet:

```
%macro KSFECity(pYear, pCity, pDebug);
%local CityName tErr;
```

```
  %InitCheckMVars(&pDebug.);
  %let tErr = 0;
  %CheckPars(
    pNotEmpty  = %str(pYear pCity)
   ,pMVarExist = %str(pCity)
   ,pError     = tErr
  );
...
```

Sollte entweder mindestens einer der Parameter pYear und pCity leer sein oder die dem Parameter pCity zugewiesene Makrovariable nicht existieren, wird eine Fehlermeldung ins Logfenster geschrieben. Zusätzlich wird die Makrovariable tErr auf 1 gesetzt.

Tabelle 1: Parameter von %CheckPars

Parameter	Beschreibung
pNotEmpty	Es wird überprüft, dass der Makroparameter nicht leer ist. Alle weiteren nachfolgend beschriebenen Checks werden nur ausgeführt, wenn der Parameter nicht leer ist. Beispiel: `pNotEmpty = %str(pYear pCity)`
pLibExist	Es wird überprüft, ob die Library existiert. Beispiel: `pLibExist = %str(pLib1 pLib2)`
pDSExist	Es wird überprüft, ob der Dataset existiert. Falls der zu überprüfende Dataset nicht in WORK gesucht werden soll, sondern in einer anderen Library, wird diese Library im zu überprüfenden Makro als separater Makroparameter konzipiert und kann beim Check in der Form pLibrary.pDataset übergeben werden. Beispiel: `pDSExist = %str(pLib1.pDS1 pDS2)`
pDSNotEmpty	Es wird überprüft, dass der Dataset nicht leer ist. Auch hier kann wie bei pDSExist die Library mit übergeben werden. Beispiel: `pDSNotEmpty = %str(pLib1.pDS1 pDS2)`

pDSVarExist	Es wird überprüft, ob die Datasetvariablen im angegebenen Dataset vorhanden sind. Jede Checkdefinition besteht aus dem Dataset (ggf. inkl. der Library), der Liste mit den Datasetvariablen, die vorhanden sein sollen und ggf. der Typ der Datasetvariablen (C für Character Variablen, N für numerische Variablen). Folgende Trennzeichen werden verwendet:
	Zwischen den Checks: '|'
	Zwischen Dataset und Datasetvariablen: '-'
	Zwischen Datasetvariablen und Datentyp: '-'
	Zwischen den Datasetvariablen: ' '
	Beispiel: `pDSVarExist = %str(`
	`pLib1.pDS1 - pVar1 pVar2 pVar3 |`
	`pDS2 - pVar4 pVar5 pVar6 - C )`

pFmtExist	Es wird überprüft, ob das Format existiert. Beispiel: `pFmtExist = %str(pSex pYesNo)`

Parameter	Beschreibung
pDirExist	Es wird überprüft, ob das Verzeichnis existiert. Beispiel: `pDirExist = %str(pDirIn pDirOut)`
pFileExist	Es wird überprüft, ob die Datei existiert. Beispiel: `pFileExist = %str(pFile1 pFile2)`
pMVarExist	Es wird überprüft, ob die Makrovariable existiert. Beispiel: `pMVarExist = %str(pMVar1 pMVar2)`
pError	Rückgabeparameter. Hier wird die Makrovariable spezifiziert, in der das Ergebnis des Parameterchecks zurückgegeben werden soll. Mögliche Werte in der Makrovariablen: 0 - Parametercheck war erfolgreich. 1 - Parametercheck war nicht erfolgreich. Mindestens ein Check hat einen Fehler gemeldet.

4.2 Makroquelltext

Der Makroquelltext von %CheckPars steht als Open Source frei zur Verfügung und kann unter

http://www.sasmacro.de

angefordert werden.

Literatur

[1] H. Schnitzer: Proceedings der 14. KSFE Berlin. Shaker-Verlag, 2010.

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

Anhang A: Listing Makro %InitCheckMVars

```
%macro InitCheckMVars(pDebug);
  %if &pDebug. eq yes %then %do;
    data mVarsSt;
      set sashelp.vmacro(where = (scope ne 'INITCHECKMVARS'));
    run;
    proc sort data=mVarsSt; by scope name; run;
  %end;
%mend;
```

Anhang B: Listing Makro %CheckMVars

```
%macro CheckMVars(pDebug);
  %if &pDebug. eq yes %then %do;
    proc sort data = sashelp.vmacro(where = (scope ne 'CHECKMVARS'))
              out  = mVarsEn;
              by scope name;
```

```
    run;
    data _null_;
    length n 8. VarList $2000. ;
    retain n 0  VarList;
      merge mVarsEn (in=En) mVarsSt (in=St) end=eof ;
      by scope name;
      if En and not St then do;
        if n eq 0 then put 'Undefined macro variable(s):';
        put name;
        n = n + 1;
        VarList = catx(' ', VarList, name);
      end;
      if eof then do;
        if n eq 0 then put 'No undefined macro variables found!';
                  else put VarList;
      end;
    run;
  %end;
%mend;
```