Dictionary Tables – Metadaten meines Work Flows in SAS

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

Zusammenfassung

Oft soll der Programmablauf flexibel auf unterschiedliche Datengrundlagen reagieren. Fehler oder unerwünschte Meldungen im Log sollen dabei vermieden werden.
Jeder Schritt im Programmablauf hat Voraussetzungen und Folgen. Um einen reibungslosen Programmablauf sicherzustellen kann man die notwendigen Voraussetzungen für den nächsten Schritt abfragen und gegebenenfalls flexibel reagieren.
Eine Möglichkeit, dies zu erreichen bieten die Metadaten-Tabellen, die SAS zur Verfügung stellt. Tabellenoperationen in Abhängigkeit von Bedingungen, wie z.B. der Existenz einer Datentabelle oder Eigenschaften der enthaltenen Zeilen und Spalten, werden möglich. Jeder Schritt im Programmablauf hat auch wieder Folgen, diese werden wiederum in den Metadaten abgebildet.
Dieser Beitrag wendet sich an alle, die die Dictionary Tables kennenlernen und/oder sich anhand von konkreten Beispielen ihren Benefit für die tägliche Programmierung anschauen möchten.

Schlüsselwörter: Dictionary Tables, Metadaten, PROC CATALOG

1 Dictionary Tables und Views

SAS stellt eine große Menge Meta-Information der aktuellen SAS-Sitzung bereit, die der Programmierer nutzen kann. Sie sind abgelegt in den sogenannten Dictionary Tables. Als Views sind sie zusätzlich im Libname SASHELP zugänglich.

Aktuell gibt es 30 Tabellen. Thornton [1] zeigt eine thematisch gruppierte Übersicht von 29 Tabellen (SAS-Version 9.2). In Version 9.3 kam eine Tabelle hinzu (VIEW_SOURCES, [2]). Die Version 9.4 brachte keine Änderungen in den Dictionary Tables. Eine schnelle Übersicht gewinnt man auch mit der Summary of SAS Dictionary Tables and Views [3].

Eine große Gruppe von Tabellen beschäftigt sich mit Informationen über die SAS Sitzung (Optionen, Title, Libnames, Engines, definierte Files, Makro-Variablen). Weitere Themenfelder sind Members (Tables, Catalogs, View), Constraints und Information Maps. Es gibt Tabellen, die detaillierte Information über Data Sets (Tables, Views, Columns, Indexes) und Catalogs (Catalogs, Formats) speichern. Eine übergreifende Tabelle beschreibt alle Dictionary Tables und ihre Spalten (Metadaten der Metadaten).

Viele der Tabellen sind gut dokumentiert und man findet einige praktische Ideen für ihre Anwendung. Andere sind kaum dokumentiert und nicht selbst-erklärend (z.B. VIEW_SOURCES [2], REMEMBER), daher bisher von geringem Nutzen.

Die Dictionary Tables gewähren durch die Art ihrer Ablage ausschließlich Lese-Zugriff. Die Views können mit Schreib-Zugriff geändert werden, was aber keine Auswirkungen auf die tatsächlichen Eigenschaften der SAS-Sitzung hat. Ändert man z.B. eine Option in SASHELP.VOPTION (DICTIONARY.OPTIONS), ist sie dadurch nicht in der Arbeitsumgebung gültig.

Die Dictionary Tables stellen also das tatsächliche Abbild der aktuellen Systemumgebung dar. Das ist eine wichtige Voraussetzung für ihre Nutzung.

2 Abfrage von Dictionary Tables

2.1 Performance

Wenn ein Dictionary Table abgefragt wird, sind viele Prozesse notwendig, um die Informationen zusammenzutragen bzw. zu aktualisieren. Das ist jeweils abhängig vom Umfang der Abfrage wie von der Komplexität der aktuellen SAS-Sitzung. Eine große Anzahl an Datasets, Variablen, Libnames und sonstigen verwendeten Files beansprucht Zeit. Das erstmalige Ansprechen eines Dictionary Tables in einer SAS-Sitzung braucht viel mehr Ressourcen als jedes weitere, wie das folgende Beispiel zeigt:
Ein Datenschritt auf SASHELP.VTABLE (DICTIONARY.TABLES):

```
data TAB1;
  set SASHELP.VTABLE;
  where libname='WORK';
run;
```

Log:

```
real time           10.26 seconds
cpu time            0.39 seconds
```

Läuft der selbe Datenschritt bei gleichen Systembedingungen ein zweites Mal, bekommt man sein Ergebnis schneller, obwohl jetzt ein Data Set mehr in WORK liegt:

```
real time           0.31 seconds
cpu time            0.31 seconds
```

Ein Proc SQL Schritt aus SASHELP.VTABLE (DICTIONARY.TABLES) mit gleichem Ergebnis zum Vergleich:

```
proc sql noprint;
  create table TAB1 as select * from SASHELP.VTABLE
    where libname='WORK';
quit;
```

Log:
```
real time             0.01 seconds
cpu time              0.01 seconds
```
Ein Proc SQL Schritt auf DICTIONARY.TABLES (SASHELP.VTABLE) ist noch sparsamer:
```
proc sql noprint;
  create table TAB1 as select * from DICTIONARY.TABLES
    where libname='WORK';
quit;
real time             0.00 seconds
cpu time              0.00 seconds
```

In den meisten Fällen ist Proc SQL die ökonomischere Lösung [4]. Jedoch bremsen komplexe where-Abfragen die Performance stark aus. Wenn die Where-Abfrage mehrfach auf die Meta-Tabelle zugreift empfiehlt es sich, einmalig ein temporäres Data Set zu erzeugen, auf dem man dann weiter arbeitet.

2.2 Abfrage mit Proc SQL: Details

Bei der Abfrage von Dictionary Tables mit Proc SQL treten Verzögerungen bei der Verwendung von Funktionen in der Abfrage auf (z.B. upcase-Funktion). Die Funktionen werden verarbeitet, die Optimierung wird damit jedoch unterlaufen [5]. Hier sollte man abwägen, welche Funktionen wirklich sinnvoll sind. Upcase ist z.B. obsolet, da Namen in den Dictionary Tables immer in Großbuchstaben gespeichert werden. Eine Ausnahme bilden hier nur die Metadaten externer nicht-SAS-Daten, die über Libnames angesprochen werden.

2.3 Abfrage der Tabellen-Struktur

Man muss nicht immer gleich eine ganze, gefüllte Tabelle abfragen. Manchmal reicht die Struktur, um z.B. Spaltennamen und Formate zu erinnern. Hierzu bietet sich ein Schritt nach folgendem Muster an:
```
proc sql;
 describe table
 DICTIONARY.OPTIONS;
quit;
```

Log:
```
NOTE: SQL table DICTIONARY.OPTIONS was created like:

create table DICTIONARY.OPTIONS
  (
   optname char(32) label='Option Name',
   opttype char(8) label='Option type',
   offset num label='Offset into option value',
   setting char(1024) label='Option Setting',
   optdesc char(160) label='Option Description',
   level char(8) label='Option Location',
```

```
   optstart char(8) label='Option Set',
   group char(32) label='Option Group'
  );
```

Alternativ kann eine Abfrage mit proc contents erfolgen:

```
proc contents data=SASHELP.VOPTION /*short*/;
run;
```

3 Beispiele

Die folgenden Beispiele sind Anwendungen aus den Bereichen Data Sets, SAS Sitzung und Kataloge.

3.1 Data Sets: Abfragen von Eigenschaften selektierter Spalten

DICTIONARY.COLUMNS (SASHELP.VCOLUMN) ist gut geeignet, die Eigenschaften bestimmter Spalten in einem oder mehreren Libnames zu vergleichen. Das Beispiel beruht auf zwei Data Sets, die aus SASHELP.CLASS und SASHELP.HEART mit dem folgenden Code erzeugt wurden:

```
data T1;
  set SASHELP.CLASS;
run;

data T2;
  set SASHELP.HEART;
run;
```

Die folgende Abfrage ermittelt alle Variablen, die das Alter abbilden mit ihren Eigenschaften:

```
proc sql;
  create table colinfo_dict as
  select memname, name, type, length, varnum, label, sortedby
    from DICTIONARY.COLUMNS
    where libname='WORK' and (index(label, 'Age') or name='AGE');
quit;
```

Man erhält folgendes Data Set:

VIEWTABLE: WORK.COLINFO_DICT

	MEMNAME	NAME	TYPE	LENGTH	VARNUM	LABEL	SORTEDBY
1	T1	AGE	num	8	3		0
2	T2	AGECHDDIAG	num	8	3	Age CHD Diagnosed	0
3	T2	AGEATSTART	num	8	5	Age at Start	1
4	T2	AGEATDEATH	num	8	12	Age at Death	0

Hiermit hat man eine elegante Möglichkeit, die Konsistenz von Eigenschaften in allen Data Sets zu prüfen (z.B. Länge, Sortierung (Spalte SORTEDBY)).

Die klassische Alternative ist Proc Contents, was jedoch umständlicher zum selben Ergebnis kommt:

```
proc contents data=T1 out=colinfo_T1;
run;
proc contents data=T2 out=colinfo_T2;
run;

data colinfo_cont;
  set colinfo_T1 colinfo_T2;
  where index(label, 'Age') or name='AGE';
  keep memname name type length varnum label sortedby;
run;
```

Manche Informationen aus der Prozedur Proc Contents sind in den Dictionary Tables anderen Tabellen zugeordnet, z.B. befinden sich Creation Date, Modification Date, Engine, Number of Observations in DICTIONARY.TABLES (SASHELP.VTABLE).
Zu beachten ist auf jeden Fall ein Unterschied: die Variable SORTEDBY ist bei Nutzung von Proc Contents anders codiert, wie man beim Vergleich des folgenden Viewtable mit dem aus DICTIONARY.COLUMNS (SASHELP.VCOLUMN) resultierenden Viewtable sieht:

VIEWTABLE: WORK.COLINFO_CONT

	MEMNAME	NAME	TYPE	LENGTH	VARNUM	LABEL	SORTEDBY
1	T1	AGE	1	8	3		.
2	T2	AGEATDEATH	1	8	12	Age at Death	.
3	T2	AGEATSTART	1	8	5	Age at Start	1
4	T2	AGECHDDIAG	1	8	3	Age CHD Diagnosed	.

3.2 Data Sets: Sortierungs-Information

In DICTIONARY.TABLES (SASHELP.VTABLE) gibt es eine Variable SORTTYPE, die Sortier-Information über das in dieser Zeile beschriebene Data Set enthält. In diesem Beispiel befüllen wir diese Spalte mit möglichst vielen verschiedenen Testfällen der Sortierung und schauen uns die Abbildung derselben an. Die Data Sets beruhen auf SASHELP.CARS, das nach der Variable MAKE sortiert ist.

A: unsortiert (drop der Sortier-Variable)

```
data cars_a_unsorted;
  set SASHELP.CARS;
  drop make;
run;
```

B: Sortierung implizit beim Erstellen (Ausgangs-Dataset ist sortiert)

```
data cars_b_sort_implicit;
  set SASHELP.CARS;
run;
```

C: Sortierung hartcodiert mitgegeben (hier: falsch!)

```
data cars_c_sort_false (sortedby=type);
  set SASHELP.CARS;
run;
```

D: Sortierung hartcodiert mitgegeben (hier: richtig!)

```
data cars_d_sort_correct (sortedby=make);
  set SASHELP.CARS;
run;
```

E: Sortierung ohne Option

```
proc sort data=SASHELP.CARS out=cars_e_sort_explicit;
  by make;
run;
```

F: Sortierung mit Option nodup

```
proc sort data=SASHELP.CARS out=cars_f_sort_nodup nodup;
  by make;
run;
```

G: Sortierung mit Option nodupkey

```
proc sort data=SASHELP.CARS out=cars_g_sort_nodupkey nodupkey;
  by make;
run;
```

H: Proc SQL implizit

```
proc sql noprint;
  create table cars_h_sql as select *
    from SASHELP.CARS ;
quit;
```

I: Proc SQL mit geänderter Reihenfolge (Sortier-Variable nicht zuerst)

```
proc sql noprint;
  create table cars_i_sql_implicit as select model, type, origin,
  make
    from SASHELP.CARS;
quit;
```

J: Proc SQL unsortiert

```
proc sql noprint;
  create table cars_j_sql_unsorted as select model, type, origin
    from SASHELP.CARS;
quit;
```

K: Proc SQL mit order by

```
proc sql noprint;
  create table cars_k_sql_orderby as select *
    from SASHELP.CARS
    order by make, type;
quit;
```

Sortier-Informationen in SASHELP.VTABLE:

VIEWTABLE: sashelp.vtable

	LIBNAME	MEMNAME	SORTTYPE	SORTCHAR
13	SASHELP	CARS	S	ANSI
1184	WORK	CARS_A_UNSORTED		
1185	WORK	CARS_B_SORT_IMPLICIT		
1186	WORK	CARS_C_SORT_FALSE	W	ANSI
1187	WORK	CARS_D_SORT_CORRECT	W	ANSI
1188	WORK	CARS_E_SORT_EXPLICIT	S	ANSI
1189	WORK	CARS_F_SORT_NODUP	SR	ANSI
1190	WORK	CARS_G_SORT_NODUPKEY	SK	ANSI
1191	WORK	CARS_H_SQL	S	ANSI
1192	WORK	CARS_I_SQL_IMPLICIT	S	ANSI
1193	WORK	CARS_J_SQL_UNSORTED		
1194	WORK	CARS_K_SQL_ORDERBY	S	ANSI

Die Spalte SORTCHAR enthält den Zeichensatz, nach dem sortiert wurde, die Spalte SORTTYPE die eigentliche Sortier-Information. Zur Bedeutung der Buchstaben in dieser Spalte habe ich keine Dokumentation gefunden, man kann sie anhand der vorgenommenen Sortierung erschließen.

→ nicht sortiert oder keine Information über Sortierung
S → sortiert ohne Zusatzoption
W → sort Information aus Data Set Option SORTEDBY („unvalidiert“)
SR → sortiert mit Option nodup
SK → sortiert mit Option nodupkey

Zwei Fälle sind hier nicht so dargestellt, wie man sie intuitiv erwarten würde. Beide sind jedoch in der SAS-Hilfe so dokumentiert [6], [7].
In Fall B (implizite Sortierung des Data Sets) wird das Data Set als nicht sortiert geführt, weil SAS weder mitgeteilt bekam, dass es sortiert ist, noch ein Sortierschritt ablief. SAS „weiß“ nicht, dass es sortiert ist. In Fall C wurde SAS über die SORTEDBY Data Set Option eine falsche Sortiervariable mitgeteilt. Mit Standard-Einstellungen wird dies so übernommen und nicht geprüft. Die Verwendung der Systemoption SORTVALIDATE könnte in diesem Fall helfen, siehe SAS Dokumentation.
Zusammenfassend lässt sich festhalten, dass Proc SQL hier zuverlässiger die intuitiv erwartete Information transportiert als der Data Step. Welche Variablen zur Sortierung verwendet wurden, findet man in SASHELP.VCOLUMN (DICTIONARY.COLUMNS). Proc Contents ist eine klassische Alternative zu diesem Ansatz. Zu beachten ist, dass die Codierung von SORTEDBY bei beiden Herangehensweisen unterschiedlich ist (siehe Beispiel 1).

3.3 Data Sets: Spaltennamen und Spaltenanzahl für Tabellierung auslesen

Angenommen es liegt ein Data Set mit einer unbekannten Zahl von Ausprägungen einer Analysevariablen vor. Diese Ausprägungen sollen in den Spalten einer Tabelle dargestellt werden. Das Programm soll dynamisch auf die Anzahl der Ausprägungen reagieren, z.B. durch eine Teilung der Tabelle in zwei einzelne, falls mehr als 5 Ausprägungen vorliegen. Konkret könnte es um die Erstlinientherapie in einem onkologischen Registerprojekt gehen, in dem es keine Vorgaben für Treatment gibt.
Folgender SAS-Code generiert ein Beispiel:

```
data colnumber;
  set SASHELP.HEART;
  TRT01A='Treatment 1';
  if _n_<=200 then output;
  TRT01A='Treatment 2';
  if 200 <_n_<=250 then output;
  TRT01A='Treatment 3';
  if 250<_n_<=600 then output;
  TRT01A='Treatment 4';
  if 600<_n_<=1000 then output;
  TRT01A='Treatment 5';
  if 1000<_n_ then output;
run;
```

Anschließend werden die Daten analysiert, hier beispielhaft mit folgendem Programm:

```
proc freq data=colnumber noprint;
  tables TRT01A / nocum out=colnum_freq;
run;

data colnumber_;
  set colnum_freq;
  outvar=put(count, 4.0)||' ('||put(percent, 4.1)||')';
run;

proc transpose data=colnumber_ out=colnum_trans prefix=_OUT_;
  id TRT01A;
  var outvar;
run;
```

Man erhält ein Data Set mit der Anzahl von Spalten, die die unterschiedlichen Treatments abbilden:

VIEWTABLE: WORK.COLNUM_TRANS

	NAME	_OUT_TREATMENT_1	_OUT_TREATMENT_2	_OUT_TREATMENT_3	_OUT_TREATMENT_4	_OUT_TREATMENT_5
1	OUTVAR	200 (3.8)	50 (1.0)	350 (6.7)	400 (7.7)	4209 (80.8)

Jetzt können sowohl die Spalten-Namen als auch die Anzahl der Spalten sehr einfach aus DICTIONARY.COLUMNS (SASHELP.VCOLUMN) ausgelesen und in zwei Makro-Variablen gespeichert werden:

```
proc sql noprint;
  select distinct name into: COLS separated by ' '
    from DICTIONARY.COLUMNS
    where memname='COLNUM_TRANS' and substr(name, 1, 5)='_OUT_'
  ;
  select count(name) into: NCOLS trimmed
    from DICTIONARY.COLUMNS
    where memname='COLNUM_TRANS' and substr(name, 1, 5)='_OUT_'
  ;
quit;
```

Makro-Variable COLS enthält Leerzeichengetrennt die Namen:

```
%put &=COLS;
```

Log-Ausgabe:

```
COLS= _OUT_TREATMENT_1 _OUT_TREATMENT_2 _OUT_TREATMENT_3
_OUT_TREATMENT_4 _OUT_TREATMENT_5
```

Makro-Variable NCOLS enthält die Anzahl der Spalten:

```
%put &=NCOLS;
```

Log-Ausgabe:

```
NCOLS=5
```

Diese Makro-Variablen liefern nun Informationen für die Tabellierung. Proc Report kann ein- oder zweimal aufgerufen werden in Abhängigkeit des Wertes von NCOLS. COLS kann dem Column-Statement mitgegeben werden, NCOLS als Ziel-Wert einer Schleife eingesetzt werden,... Die denkbaren Anwendungen sind vielfältig.

3.4 Data Sets: Leere SAS Data Sets erkennen

Die logische Programmfolge in SAS ist abhängig von der Datengrundlage. Data Sets werden erzeugt und weiter verwendet. Nicht immer weiß man vorher, ob die Voraussetzungen für eine Analyse gegeben sind. Arbeitet man z.B. mit ein und demselben Analyseprogramm für verschiedene Subgruppen, kann es sein, dass in einer Subgruppe alle Ausprägungen in ausreichender Anzahl für die Analyse vorhanden sind, in einer anderen nicht.
Wird ein Data Set nicht erzeugt, ist dies sehr einfach mit der EXIST()-Funktion festzustellen. Ist ein Data Set leer, kann man diese Information aus DICTIONARY.TABLES (SASHELP.VTABLE) entnehmen. Ein leeres Data Set hat eine definierte Spaltenstruktur, aber 0 Zeilen.
Angenommen es liegt ein leeres Data Set namens EMPTYDS in der WORK Library, erzeugt der folgende Code eine Makro-Variable NROWS mit dem Inhalt 0:

```
proc sql noprint;
  select nobs into: NROWS trimmed
    from DICTIONARY.TABLES
    where libname='WORK' and memname='EMPTYDS';
quit;
```

Diese Information kann als Bedingung vor der Ausgabe der Ergebnisse verwendet werden und stellt eine elegante Alternative zu Proc Contents dar.

3.5 SAS Sitzung: Makro-Variablen auslesen, ansprechen, löschen

Zu guter Programmierung gehören Übersicht und gezieltes Löschen nicht mehr benötigter Elemente. Im Bereich der Makro-Variablen hilft dabei die Meta-Information aus DICTIONARY.MACROS (SASHELP.VMACRO).
Für dieses Beispiel werden einige globale Makro-Variablen definiert:

```
%global _usermv1 _usermv2 _usermv3 _usermv4;
%let _gPath=U:\&sysuserid.\;
%let _gDBDat=20160630;
```

Eine Abfrage für alle Makro-Variablen lautet:

```
proc sql;
  create table macinfo as
  select * from DICTIONARY.MACROS;
quit;
```

Man erhält folgendes Data Set (Ausschnitt):

VIEWTABLE: WORK.MACINFO

	SCOPE	NAME	VALUE
1	GLOBAL	SQLEXITCODE	0
2	GLOBAL	SQLOBS	0
3	GLOBAL	SQLOOPS	0
4	GLOBAL	SQLRC	0
5	GLOBAL	SQLXOBS	0
6	GLOBAL	SYS_SQL_IP_ALL	-1
7	GLOBAL	SYS_SQL_IP_STMT	
8	GLOBAL	_GDBDAT	20160630
9	GLOBAL	_GPATH	U:\scheiner-sparna\
10	GLOBAL	_USERMV1	
11	GLOBAL	_USERMV2	
12	GLOBAL	_USERMV3	
13	GLOBAL	_USERMV4	
14	AUTOMATIC	AFDSID	0
15	AUTOMATIC	AFDSNAME	
16	AUTOMATIC	AFLIB	

Die nutzerdefinierten Makro-Variablen haben bei SCOPE den Eintrag GLOBAL wie andere System-Makrovariablen auch. Leider gibt es keinen Identifier zur Unterscheidung beider Typen. Man kann sich mit einer Namenskonvention helfen. In diesem Beispiel bedeutet dies, den Namen mit _G oder _USER beginnen zu lassen.
Dann kann man sie Makro-Variablen gezielt gruppenweise auslesen:

```
proc sql noprint;
  select name into: usermv separated by ' '
    from SASHELP.VMACRO
    where index(name, '_USER') or index(name, '_G');
quit;
```

Die Namen werden leerzeichengetrennt in eine Makro-Variable USERMV geschrieben. Das kann man sich mit `%put` im Log anzeigen lassen. Log-Ausgabe:

```
USERMV=_GDBDAT _GPATH _USERMV1 _USERMV2 _USERMV3 _USERMV4
```

Den Textinhalt dieser Makrovariable kann man nun zum sicheren Löschen aller dieser Namens-Konvention entsprechenden Makrovariablen verwenden. Auch die Makro-Variable USERMV wird anschließend gelöscht:

```
%symdel &usermv. usermv;
```

3.6 SAS Sitzung: Selektierte Optionen merken und zurücksetzen

Manchmal verwendet man Programmbausteine, die System-Optionen verändern, was sehr sinnvoll sein kann. Es gibt verschiedene Möglichkeiten, Optionen in der ursprünglichen Form zu speichern und später wieder so zu setzen. Eine einfache Möglichkeit bietet DICTIONARY.OPTIONS (SASHELP.VOPTION).
Angenommen werden die folgenden Optionen (alle vom Typ char) in DICTIONARY.OPTIONS:

OPTNAME	SETTING	OPTTYPE
ODSSTYLE	AUTO	char
ORIENTATION	PORTRAIT	char
PAPERSIZE	LETTER	char

Ein kleines Makro speichert diese Settings:

```
%macro optkeep(opt);
  %* globale Makro-Variable fuer aktuelle Settings;
  %global old&opt.;

  %* aktuelle Option speichern;
  proc sql noprint;
      select setting into: old&opt. trimmed
      from DICTIONARY.OPTIONS
      where optname = "&opt.";
  quit;
%mend optkeep;
```

Für jede Option wird ein Aufruf des Makros ausgeführt, der eine Makro-Variable mit dem Eintrag der Option generiert:

```
%optkeep(ORIENTATION)
%optkeep(PAPERSIZE)
%optkeep(ODSSTYLE)
```

Die Makro-Variablen können mit `%put` im Log angezeigt werden. Log-Ausgabe:

```
OLDORIENTATION=PORTRAIT
OLDPAPERSIZE=LETTER
OLDODSSTYLE=AUTO
```

Nachdem die Programmsequenz mit den geänderten Optionen abgelaufen ist, können die alten Einstellungen wiederhergestellt werden.

```
options   ORIENTATION=&oldORIENTATION.
          PAPERSIZE=&oldPAPERSIZE.
          ODSSTYLE=&oldODSSTYLE.;
```

Diese Art, Optionen zu merken und zurückzusetzen, lohnt sich nur für wenige, gezielt ausgewählte Optionen. Das nächste Beispiel befasst sich mit Gruppen von Optionen.

3.7 SAS Sitzung: Gruppen von Optionen merken und zurücksetzen

In DICTIONARY.OPTIONS (SASHELP.VOPTION) sind die Optionen in thematische Gruppen eingeteilt. Exemplarisch seien hier drei Optionen unterschiedlichen Typs der Gruppe MACRO dargestellt. Hier ein Ausschnitt aus DICTIONARY.OPTIONS:

VIEWTABLE: WORK.OPTINFO2

	OPTNAME	OPTTYPE	SETTING	OPTDESC
15	MINDELIMITER	char		Specifies the character delimiter for the macro IN operator.
17	MLOGIC	Boolean	NOMLOGIC	Traces macro execution and writes the results to the SAS log.
25	MVARSIZE	num	65534	Specifies the maximum size for a macro variable that is stored in memory.

Die ursprünglichen Optionen werden mit einem Proc SQL in ein Data Set geschrieben:

```
data PREVIOUS;
  set SASHELP.VOPTION;
  where group='MACRO' and optstart ne 'startup'
                              and optname ne 'SASAUTOS';
  length opt_keyw $1060;
  opt_keyw=getoption(optname, "keyword");
  keep optname opttype setting opt_keyw;
run;
```

Mit `group='MACRO'` wird die Abfrage eingeschränkt auf die Gruppe MACRO. Andere Gruppen wären z.B. ODSPRINT, PERFORMANCE, ERRORHANDLING, GRAPHICS, LOGCONTROL. `optstart ne 'startup'` schließt Optionen aus, die nach dem SAS Start nicht mehr veränderbar sind.
Die Funktion `getoption()` erzeugt die Syntax für ein Options Statement, das passend ist für den jeweiligen Typ der Option.

Typ char z.B. `options ORIENTATION=landscape;`
Typ boolean z.B. `options MPRINT;`
Typ num z.B. `options MVARSIZE=65530;`

Dieses Proc SQL schreibt Notes ins Log: “A conflicting combination of parameters was specified for the GETOPTION function.” Diese Meldung wird in SAS Foren diskutiert und als hinnehmbar bewertet [8]. Es werden alle Optionen richtig ausgelesen. Die drei Beispiel-Optionen werden verändert mit:

```
options MVARSIZE=6000 MLOGIC MINDELIMITER='#';
```

Der folgende Schritt erzeugt ein Data Set, in dem die neuen und die alten Optionen nebeneinander erscheinen:

```
proc sql noprint;
  create table optreset as select c.optname, c.opttype,
                          c.setting as setting_current,
                          p.setting as setting_previous,
                          p.opt_keyw
    from DICTIONARY.OPTIONS as c
    left join previous as p
    on c.optname=p.optname
    where p.optname ne ''
  ;
quit;
```

Für die drei Beispiel-Optionen ergibt sich daraus:

VIEWTABLE: WORK.OPTRESET

	OPTNAME	OPTTYPE	SETTING_CURRENT	SETTING_PREVIOUS	OPT_KEYW
14	MINDELIMITER	char	#		MINDELIMITER="
16	MLOGIC	Boolean	MLOGIC	NOMLOGIC	NOMLOGIC
24	MVARSIZE	num	6000	65534	MVARSIZE=65534

Mit einem Data Step und Call Execute können die Optionen auf den ursprünglichen Stand zurückgesetzt werden. Da die getoption-Funktion mit der Option keyword verwendet wurde, ist die Syntax für die unterschiedlichen Typen von Optionen in der Variable OPT_KEYW richtig vorbereitet.

```
data _null_;
  set optreset;
  if setting_current ne setting_previous then do;
    call execute("option "||compress(opt_keyw)||";");
  end;
run;
```

Im Log erscheint:

```
NOTE: CALL EXECUTE generated line.
1   + option MINDELIMITER='';
2   + option NOMLOGIC;
3   + option MVARSIZE=65534;
```

Diese Methode ist eine flexible Alternative zu Proc OptSave und Proc OptLoad. Ein Vorteil liegt darin, dass nicht alle Optionen angesprochen werden müssen, sondern vorab selektiert werden kann.

3.8 Kataloge: Temporäre Formate auslesen und löschen

Formate werden in SAS-Katalogen abgelegt. In den Metadaten gibt es sowohl eine Tabelle zu Katalogen allgemein, als auch eine speziell zu Formaten. Für dieses Beispiel wird die Tabelle DICTIONARY.FORMATS (SASHELP.VFORMAT) verwendet.

Wie bereits für Makro-Variablen dargestellt, sollten auch temporär angelegte Formate aus Gründen der Fehlervermeidung und Übersicht wieder gelöscht werden, wenn sie nicht mehr gebraucht werden.
Zur Demonstration werden zuerst einige Formate in der WORK Library erzeugt.

```
proc format lib = WORK cntlout=NEWFMT;
  value $ BP
    'Normal' = 'OK'
    'Optimal'= 'OK'
    'High'   = 'Not OK'
    'Low'    = 'Not OK'
  ;
  value $ _BPB
    'Normal' = 'Norm'
    'Optimal'= 'Opt.'
    'High'   = 'Bad'
    'Low'    = 'Bad'
  ;
  value $ _ABC
    'Normal' = '1'
    'Optimal'= '1'
    'High'   = '2'
    'Low'    = '0'
  ;
  value _AGE
    20 - 49   = 'Low'
    50 - 59   = 'Medium'
    60 - high = 'High'
  ;
run;
```

Die herkömmliche Art, Formate zu löschen, ist global mit Hilfe von Proc Catalog:

```
proc catalog c=work.formats force kill;
quit;
```

Will man differenzierter vorgehen und bestimmte Formate erhalten, kann man sie über DICTIONARY.FORMATS abfragen. Eine Namenskonvention für Formate, die mit einem Standard-Clean-up gelöscht werden sollen, bietet sich an. In diesem Beispiel ist der führende Unterstrich das Merkmal, das das Format für die Standard-Löschung qualifiziert.
Mit dem folgenden Code werden alle Formate, deren Name mit diesem Zeichen beginnt, leerzeichengetrennt in zwei Makro-Variable geschrieben. FMTC enthält alle Character-Formate, FMTN alle numerischen Formate. Mit folgendem `%put` werden sie ins Log ausgegeben:

```
%put Temporary character formats: & FMTC.;
%put Temporary numeric formats: & FMTN.;
```

Log-Ausgabe:

```
Temporary character formats: _ABC _BPB
Temporary numeric formats: _AGE
```

Jetzt kann das Proc Catalog speziell für diese Formate aufgerufen werden:

```
proc catalog c=work.formats;
  delete &FMTC. / entrytype=formatc;
  delete &FMTN. / entrytype=format;
quit;
```

Log-Ausgabe:

```
NOTE: Deleting entry _ABC.FORMATC in catalog WORK.FORMATS.
NOTE: Deleting entry _BPB.FORMATC in catalog WORK.FORMATS.
NOTE: Deleting entry _AGE.FORMAT in catalog WORK.FORMATS.
```

3.9 Kataloge: Format-Dokumentation

Die Tabelle DICTIONARY.CATALOGS (SASHELP.VCATALG) bietet die Möglichkeit, in einer Arbeitsumgebung verfügbare Formate abzurufen und als Teil der Dokumentation einer Projekt-Umgebung zu verwenden. Dieses Beispiel beruht auf denselben Formaten wie Beispiel 3.8. SASHELP.VCATALG zeigt sie mit folgenden Spalten:

LIBNAME	MEMNAME	MEMTYPE	OBJNAME	OBJTYPE	OBJDESC	CREATED
WORK	FORMATS	CATALOG	_AGE	FORMAT		05JAN17:12:51:32
WORK	FORMATS	CATALOG	BP	FORMATC		05JAN17:12:51:32
WORK	FORMATS	CATALOG	CHARF	FORMATC		05JAN17:12:51:32
WORK	FORMATS	CATALOG	_ABC	FORMATC		05JAN17:12:51:32

Der Vorteil dieser Tabelle gegenüber SASHELP.VFORMAT ist die Spalte OBJDESC, in die eine Beschreibung eingefügt werden kann. Hier wird diese relativ aufwändig hartcodiert eingefügt:

```
proc catalog c=formats;
  modify BP.formatc
    (description="Blood Pressure: Normal=OK, Optimal=OK, High=Not
OK, Low=Not OK");
  modify CHARF.formatc
    (description="Type Format: FORMAT=num FORMATC=char");
quit;
```

Die Beschreibung kann jedoch auch aus dem Dataset cntlout=NEWFMT ausgelesen und mit einem Makro aufbereitet werden. Ein übersichtliches Listing z.B. mit Proc Report könnte dann Bestandteil einer Projekt-Dokumentation sein.

Formats in WORK library

Name	Type	Format definition
_AGE	num	
BP	char	Blood Pressure: Normal=OK, Optimal=OK, High=Not OK, Low=Not OK
CHARF	char	Type Format: FORMAT=num FORMATC=char
_ABC	char	
_BPB	char	

Abbildung 1: Beispiel für Format-Dokumentation

3.10 Kataloge: Makro-Dokumentation

Ähnlich wie die Formate, kann man auch die in einer SAS-Umgebung zur Verfügung stehenden Makros aus DICTIONARY.CATALOGS (SASHELP.VCATALG) herauslesen:

```
proc sql noprint;
  create table project_mac as
      select OBJNAME, CREATED as RUN_DATE, OBJDESC
    from DICTIONARY.CATALOGS
    where OBJTYPE='MACRO' and libname='WORK';
quit;
```

Hier sind ein paar Eigenschaften zu beachten:

- Es erscheinen alle Makros, die ausgeführt wurden.
- Zusätzlich reicht das Kompilieren eines Makros aus, um in der Liste zu erscheinen, auch wenn das Makro nicht ausgeführt wurde.
- SAS unterscheidet nur SASHELP und WORK Makros, es wird nicht nach Speicherort der Makros unterschieden.

Den letzten Punkt kann man mit zwei Optionen die Dokumentation des Pfades hinzufügen, jedoch nur in einigen Fällen:

```
options MAUTOLOCINDES
        SASAUTOS     = ("L:\Statistik\SASLibrary\prod"
                        "L:\Statistik\SASLibrary\dev"
                        SASAUTOS);
```

Die Option SASAUTOS enthält Pfade, die SAS in dieser Reihenfolge nach einem aufgerufenen Makro durchsucht (Auto-Search). Aktiviert man zusätzlich MAUTOLOCINDES, werden die Pfade der hier gefundenen Makros den Metadaten hinzugefügt. Dies funktioniert leider nur für diese Pfade und nur für Makros, die nicht in Sammel-Files gespeichert wurden. Der Makro-Name muss mit dem File-Namen identisch sein. Makros, die mit %include kompiliert oder im Programm-Code lokal definiert werden, bekommen keinen Pfad.

In unserem Beispiel erscheint der Pfad bei einem Makro in der Spalte OBJDESC:

VIEWTABLE: WORK.PROJECT_MAC

	OBJNAME	RUN_DATE	OBJDESC
1	AUDITTRAIL	09FEB17:13:19:41	
2	UGE2	09FEB17:13:19:41	
3	DOPKEY_V1	09FEB17:13:21:24	
4	SGE2	09FEB17:13:19:41	
5	GM_CREATE	09FEB17:12:10:26	L:\Statistik\SASLibrary\prod\gm_create

Denkbar ist es, wie bei den Formaten in Beispiel 3.9 die Spalte für alle anderweitig abgelegten Makros über Proc Catalog zu befüllen.

4 Schlusswort

Die vielfältigen Möglichkeiten zur Verwendung der SAS Metadaten können hier nur ansatzweise ausgearbeitet werden. Beispiele zu DICTIONARY.TABLES (SASHELP.VTABLE) sind schon vielfach beschrieben worden. Weitere Anwendungen werden dem aufmerksamen Programmierer beim Lösen seiner Aufgaben sicher einfallen. Der große Vorteil der Dictionary Tables ist, dass sie das tatsächliche Abbild der SAS-Sitzung reflektieren, sich sofort an neue Gegebenheiten anpassen. Wer hat nicht schon einmal bei einem umfangreichen Clean-up-Prozess mit Löschen vieler Data Sets, Makro-Variablen etc. einige Sekunden gewartet und sich gefragt, was SAS da im Hintergrund arbeitet. Bei der Beschäftigung mit den Metadaten ist mir das klar geworden.

Literatur

[1] Thornton, P: SAS Dictionary: Step by Step. SUGI paper 264-2011. http://support.sas.com/resources/papers/proceedings11/264-2011.pdf

[2] DICTIONARY.VIEW_SOURCES https://de.scribd.com/doc/135543119/Pratique-de-SAS-Windows-9-3-Volume-2

[3] Summary of SAS Dictionary Tables and Views http://www.codecraftersinc.com/pdf/DictionaryTablesRefCard.pdf

[4] SAS SQL Procedure User's Guide (2011): http://support.sas.com/documentation/cdl/en/sqlproc/63043/PDF/default/sqlproc.pdf

[5] Accessing SAS System Information by Using DICTIONARY Tables http://support.sas.com/documentation/cdl/en/sqlproc/62086/HTML/default/viewer.htm#a001385596.htm

[6] Sort information in DICTIONARY.TABLES (SAS Support): http://stackoverflow.com/questions/29008938/check-if-sas-datasets-are-sorted

[7] Sort information in DICTIONARY.TABLES: http://support.sas.com/ documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000766829.htm

[8] getoption() Note in Log Window https://communities.sas.com/t5/Base-SAS-Programming/Getoption-argument-to-sysfunc-9-3-changes/td-p/34617