Making the most of Dictionary Views in the SASHELP Library.

Author: Simon Wilcock
Organisation: British Biotech

Introduction

The SASHELP library which comes automatically with the Base SAS® software is an often neglected tool which can be used both by advanced programmers looking to create robust code and by a variety of users trying to perform quick ad-hoc queries and data checks.

This paper looks at a number of the dictionary views maintained within the SASHELP library and looks at some of the utility macros written at British Biotech, which put the library to good use in ensuring programs run error free across different data structures.

The paper will also show how one of these views in particular goes some way towards solving the tricky problems involved in the management of macro variables (problems which often appear as requested enhancements in the SASWARE ballot).
The SASHELP views

Dictionary tables are special read-only PROC SQL objects that retrieve information about all of the SAS data libraries, SAS data sets, SAS System options, SAS Macros and external files that are associated with the current SAS session. These permanent PROC SQL views are available in the SASHELP data library. There are 16 of these - at last count - all worth investigating, but this paper will concentrate on the following six:

- SASHELP.VCATALG
- SASHELP.VCOLUMN
- SASHELP.VMACRO
- SASHELP.VSTABLE
- SASHELP.VTABLE
- SASHELP.VOPTION

and look at some of the ways that they have been put to use within the British Biotech programming environment.

British Biotech and Generic Programs

Before proceeding it is worth outlining the kind of SAS programs that are used at British Biotech and the environment in which they run, in order to understand why the views are of so much use.

British Biotech is an Oxford-based Biotechnology company involved in the development and, ultimately, marketing of a number of novel drugs. The SAS System® is used as the main tool for analysing and reporting the results of the Clinical Trials currently under way in order to gain regulatory approval for these drugs.

The SAS programs are run under Windows 95® accessing data stored on a Vax/VMS® server.

In order to avoid duplication of code between the different drug projects each SAS program is written in the SAS Macro language and called via a ‘Template’ (basically a program which sets the given macro variables for a project/study and passes them to the required macro). The macro code, itself, takes care of the differences in database design.

By programming in this way, the SAS group can maintain a library of validated programs in one central area and the Statisticians in the department only need to know a limited amount of SAS in order to produce their tables.

It does, however, mean that the macros themselves can be quite complicated and have to have a certain ‘inside knowledge’ of the data.

By using the SASHELP dictionary views the code has been made more reliable and robust, reducing the need for lengthy user input.
1. SASHELP.VSTABLE

(PROC SQL statement used to create this view:
create view sashelp.vstable as select libname, memname
from dictionary.members
where memtype='DATA'
order by libname, memname)

On the face of it not a very exciting SAS view - just two columns, one with library name
and one with member name.
But consider the situation where SAS has to read in a dataset called X.HAEM and then
concatenate it with a second dataset X.HAEM2 - if it exists.
As the generic macro must work in all cases the code:
DATA ALLHAEM;
SET X.HAEM X.HAEM2;
RUN;
is going to fail in cases where X.HAEM2 does not exist.

(ERROR: File X.HAEM2.DATA does not exist.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.ALLHAEM may be incomplete. When this step
was stopped there were 0 observations and 1 variables.)

This error can be easily avoided by using a utility macro to check for the existence of the
dataset before using it. This is achieved using the VSTABLE view.

%MACRO UFE1(FDIR, FNAME) ;
OPTIONS NOMPRINT;
%GLOBAL EXIST ;
%LET EXIST=0;
DATA _NULL_ ;
SET SASHELP.VSTABLE;
WHERE LIBNAME="&FDIR";
IF MEMNAME="&FNAME" THEN DO;
   CALL SYMPUT('EXIST', 1) ;
   STOP;
END;
RUN ;
%MEND;

This utility macro can then be used within the parent macro simply as follows:

%UFE1(X,HAEM2);

DATA ALLHAEM;
SET X.HAEM
%IF &EXIST=1 %THEN %STR(X.HAEM2); ;
RUN;

A simple but effective use of SASHELP.VSTABLE.

2. SASHELP.VCATALG

(PROC SQL statement used to create this view:
create view sashelp.vscatalg as select libname, memname
from dictionary.members
where memtype='CATALOG'
order by libname, memname)

Similar to VSTABLE this view is very useful when trying to manage SAS catalog entries. Because of the way the GOUTMODE=APPEND option works in SAS/GRAPH®, it is often useful to check for the existence of a catalog entry before actually doing anything with it.

For example, if the code needs to add a graph called PKANAL to the catalog X.DRUG1, then GOUTMODE=APPEND must be used. However, if a graph called PKANAL already exists in this catalog then the new graph will be called PKANAL1, which can be very disconcerting and even cause the wrong graph to be printed later.

So, using VCATALG, consider the following macro:

```
%MACRO UCE1(LIB, CAT, ENTNAME, ENTTYPE=GRSEG) ;
  OPTIONS NOXWAIT NOMPRINT;
  %GLOBAL EXIST ;
  %LET EXIST=0;
  DATA _NULL_;
    SET SASHELP.VCATALG;
    WHERE LIBNAME="&LIB"
    AND MEMNAME="&CAT"
    AND MEMTYPE="CATALOG";

    IF OBJNAME="&ENTNAME"
    AND OBJTYPE="&ENTTYPE" THEN DO;
      CALL SYMPUT('EXIST', 1) ;
      STOP;
    END;
  RUN ;
  OPTION MPRINT;
  %MEND ;
```

This macro can then be used, in stream, to check for the existence of an entry name and then, if necessary, take steps to delete or rename it.
3. SASHELP.VTABLE

(PROC SQL statement used to create this view:
create view sashelp.vtable as select *
from dictionary.tables)

This view is awash with useful information about the datasets and data views assigned to
the system - there is: number of observations, number of variables, creation dates, data
file labels, buffer sizes and much more.
Most of this information can, of course, be accessed easily (and often faster) by using
PROC CONTENTS. The real strength of this view is that it works across librefs and
datasets.
This can prove invaluable within the pharmaceutical industry, as a requirement of any
submission is to perform cross protocol analysis. Thus, data has to be pulled in from
many different sources and the VTABLE view offers an excellent way of managing this.

So, for example, when performing data for regulatory submissions, it is a requirement of
the internal QA department that all tables have a footnote containing the creation date of
the SAS datasets. If the table involves data from multiple studies then a range of creation
dates must be given (i.e. from the study with the earliest data to the most recent). This is
easily achieved using VTABLE in the following macro:

%MACRO UDC1(LIBS, DSETS);
%GLOBAL MINDATE MAXDATE;
*** Find min and max values of CRDATE ***;
PROC SQL NOPRINT;
SELECT MAX(DATEPART( CRDATE)) FORMAT=DATE.,
MIN(DATEPART( CRDATE)) FORMAT=DATE.
 INTO :MAXDATE, :MINDATE
FROM SASHELP.VTABLE
WHERE LIBNAME IN (&LIBS) AND MEMNAME IN(&DSETS);
QUIT;
%MEND;

Eg:
LIBNAME X1 ‘G:\big_project\first_study\data’;
LIBNAME X2 ‘G:\big_project\second_study\data’;
LIBNAME X3 ‘G:\big_project\third_study\data’;
%UDC1(“X1, X2, X3”, ”PKDATA”);

These macro variables can then be used in a footnote statement:
FOOTNOTE1 “Pharmacokinetic data created from &MAXDATE - &MINDATE”

Another potential use of this view, is to use it to pinpoint studies with missing data. Using the
NOBS variable it is easy to see which tables/datasets are empty for which projects.
4. SASHELP.VCOLUMN

(PROC SQL statement used to create this view:
create view sashelp.vcolumn as select *
from dictionary.columns)

Like the VTABLE view, the VCOLUMN view really comes into its own when performing cross study analysis. As already stated, as part of a regulatory submission data must be pooled from a multiple of sources. Problems inevitably arise when concatenating datasets from these different sources - sometimes variables have a different type between studies (which is easy to spot) or worse still, have different lengths.

Consider two datasets:

X1.EVENT. (length of variable causality = $8)
SUBJECT=3  Causality='RELATED';

X2.EVENT. (length of variable causality = $16)
SUBJECT=1  Causality = 'PROBABLY RELATED';
SUBJECT=2  Causality='PROBABLY UNRELATED';

When they are concatenated (DATA EVENT; SET X1.EVENT X2.EVENT;) the variable Causality is set to length $8 and both Subject 1 & 2 have Causality='PROBABLY'; This is not good! Therefore, use the following macro to check for consistency of variables:

%MACRO UVC1(LIBS, DSET);
%GLOBAL VARERR;
%LET VARERR=0;
PROC SQL;
   CREATE TABLE VCHECK AS
   SELECT LIBNAME, MEMNAME, NAME, TYPE, LENGTH, FORMAT
   FROM SASHELP.VCOLUMN
   WHERE LIBNAME IN (&LIBS) AND MEMNAME="&DSET";
PROC SORT NODUPKEY;
   BY NAME TYPE LENGTH FORMAT;
DATA VERROR;
   SET VCHECK;
   BY NAME TYPE LENGTH FORMAT;
   IF FIRST.NAME AND NOT LAST.NAME THEN DO;
      OUTPUT;
      CALL SYMPUT('VARERR',1);
   END;
   IF NOT FIRST.NAME THEN OUTPUT;
RUN;
%IF &VARERR EQ 1 %THEN %DO;
   TITLE1 "THE FOLLOWING VARIABLES ARE INCONSISTENT ACROSS STUDIES";
   PROC PRINT LABEL NOOBS;
RUN;
%END;
%MEND;
5. SASHELP.VMACRO

(PROC SQL statement used to create this view: create view sashelp.vmacro as select * from dictionary.macros)

This view is, by far and away, the most useful view in the SASHELP library. It provides a complete view of all the macros defined to the system, the values associated with them and the scope (AUTOMATIC, GLOBAL, LOCAL). This view has been used to great effect within the British Biotech environment to solve two problems that have always caused great headaches.

The first of these problems occur when a macro can be called by two different sources. For example, the ‘Serious Adverse Event’ reporting macro (MSAE1) calls part of the ‘Adverse Event’ reporting macro (MAE1), in order to avoid duplication of code. However, the latter macro tends to be more complicated and, as such, has more macro variables assigned. Thus, whenever a macro variable is added to MAE1, it must also be added to MSAE1.

This problem is compounded when the calls to a given macro can come from a multitude of sources. If the relevant macro variables haven’t been set up, then the inevitable error message follows:

WARNING: Apparent symbolic reference TRTGROUP not resolved.
ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: &mvar =
ERROR: The macro will stop executing. (or similar !)

Errors of this type can be avoided by using the VMACRO view to ensure a macro variable exists before attempting to use it. An example utility macro for doing this is as follows:

%MACRO UMA1(MVAR) ;
  OPTIONS NOXWAIT NOMPRINT;
  %GLOBAL EXIST;
  %LET EXIST=0;
  DATA _NULL_;
  SET SASHELP.VMACRO;
  IF NAME EQ "&MVAR" THEN DO;
    CALL SYMPUT("EXIST",1);
    STOP;
  END;
RUN;

OPTIONS MPRINT;
%MEND;

This macro can then be used, in a reporting macro, to ensure that macro variables that are particular to only one type of call do not cause errors when called from an alternative source. The macro above is similar to the one used at British Biotech, although it needs slight ‘tweaking’ in order to cope with the second problem that VMACRO has helped to solve.

The second problem that VMACRO has helped to solve is the that of ‘clearing out’ macro variables between executions of a macro. A request often seen in the SASWARE ballot is to ‘provide the ability to delete macro variables’ - because if, for example, two programs
are run consecutively in the same Windows 95® session, then it is possible that a macro variable set in the first program is inadvertently picked up in the second. Potentially this could cause invalid analysis. (The same of course is true of work datasets and formats but these are easily handled with PROC DATASETS).

At the time of writing, there still does not appear to be a way of deleting macro variables - except of course by exiting the SAS session, but this is very time consuming when there are multiple jobs to be run. However, because the VMACRO view lists all defined macro variables, there is always the option of resetting them all. At British Biotech, because a null variable still represents information for the macro, before a program is run all macro variables are set to the string UNDEFINED. All macros within the system recognise this as an undefined macro variable and therefore not to be used until it is given a ‘real’ value. The utility macro that does this is as follows:

```sas
%MACRO UZZ1;
%*** Pick up all the currently defined macro variables ***;
%*** from the dictionary table and place in a dataset ***;
PROC SQL NOPRINT;
CREATE TABLE VARLIST AS SELECT NAME FROM SASHELP.VMACRO WHERE SCOPE NOT IN ('AUTOMATIC');
QUIT;
%*** Create one macro variable for each distinct variable in the dataset. ***;
%*** These (local) macro variables are VAR1- VARn ***;
%LOCAL  I   TOTVARS;
PROC SQL NOPRINT;
SELECT COUNT(NAME) INTO: TOTVARS FROM VARLIST;
%LET TOTVARS=%EVAL(&TOTVARS);
%DO I = 1 %TO &TOTVARS; %LOCAL VAR&I; %END;
SELECT DISTINCT NAME INTO: VAR1 - : VAR&TOTVARS FROM VARLIST; QUIT;
%*** Now loop around the macro variable names setting them to UNDEFINED ***;
%DO I = 1 %TO &TOTVARS;
  %LET &&VAR&I=UNDEFINED;
%END;
%* And finally ... clean up after yourself **;
PROC DATASETS LIBRARY=WORK NOLIST;
DELETE VARLIST;
QUIT;
%MEND;
```

6. SASHELP.VOPTION

(PROC SQL statement used to create this view: create view sashelp.voption as select * from dictionary.options)
The VOPTION view is a useful component for managing the running of identical code within different SAS sessions. It contains information on each System Option and the current settings.

This means that if, for any reason, a System Option needs to be changed during the run of a program, it is easy to create a macro variable representing the setting of the option before the change. Then, when the program has run, this macro variable can be used to reset the System Option back to its original value.

Another very powerful application of this view is to use it to ensure that macros are being called from the correct libraries.

At British Biotech, when an analysis or table is run for a draft version of a study report the search path for the macros is ‘E:\TEST’ ‘E:\LIBRARY’ ‘E:\MACLIB’.
But when a final version is produced, all relevant software has to have been fully documented, validated and promoted into the library. And so the search path is changed to ‘E:\LIBRARY’ ‘E:\MACLIB’

The software can use the VOPTION view to check the search path in operation and raise any relevant warning/error messages if an incorrect path is being used:

```
DATA _NULL_;
SET SASHELP.VOPTION(WHERE=(OPTNAME='SASAUTOS'));
IF INDEX(SETTING,'TEST') > 0 THEN TEST=1;
ELSE TEST=0;
CALL SYMPUT('TEST',TEST);
RUN;
%IF &TEST EQ 1 %THEN
%STR(FOOTNOTE1 "THIS TABLE IS PRODUCED USING TEST SOFTWARE");
```

A simple, but effective, method of ensuring that results derived from unvalidated programs do not find themselves appearing in final reports.
Conclusion

By making good use of the meta data available in the SASHELP dictionary views, simple utility macros can be easily produced to ensure that all SAS programs run reliably and across a range of data structures. The experience at British Biotech has been that, by using these simple utility macros, less errors have occurred and the ‘generic library’ of software has been accepted by the user community as robust, and therefore it is used extensively.

This paper has only focused on a few of the views available but there are plenty more where they came from! The author strongly recommends that any programmers who may have overlooked the existence of the SASHELP library should take some time out to investigate its potential.

Notices

The SAS System, Base SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc., Cary, NC, USA. Other brand or product names are registered trademarks of their respective companies.