Rapid Assessment of Data Set Structures

Phil Vecchione, Cognigen Corporation, Buffalo NY

ABSTRACT

When SAS® data is received from an external source, such as a sponsor, CRO, or another department, it is important to assess the new data to determine if variables have been added or removed, locate specific variables, and identify missing data in key datasets. Changes in dataset structure can have an impact on existing analysis programs, requiring modifications or creation of new programs, which uses up valuable analysis time.

In order to speed up this process and create an efficient way to assess new SAS® datasets, a family of macros has been designed to quickly obtain this information. These macros work for individual datasets as well as for a library of datasets. They rapidly provide output that identifies potential problems, allowing the user to communicate them back to their external source quickly; therefore streamlining the process time from data receipt to final analysis.

INTRODUCTION

Cognigen provides data analysis and consulting services for the pharmaceutical and biotechnology industries during clinical development of new medicines. The population-based statistical and pharmacokinetic/pharmacodynamic modeling performed at Cognigen requires targeted data assembly of multiple datasets within and across clinical trials. As a data analysis firm, we receive data from a number of different sponsors for various projects. Each of these sponsors has their own data management systems and database structures. Many sponsors also contract with laboratories or CRO's that generate data and transmit it directly to Cognigen, during the course of the study. Because of this, we encounter data in a variety of formats and structures and need to quickly navigate the data so that we can assemble the data into a usable form and perform our analysis.

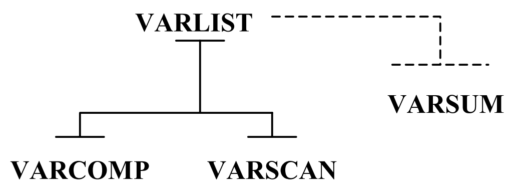
Typically we receive clinical data as SAS® transport files containing 20 or more datasets, comprising the various database tables used by the sponsor. In the beginning of a project we will receive a transmission of data and need to determine which variables are common between datasets, locate specific variables by keywords in their variable name or label, and determine the completeness of data for critical tables. During the course of a given project we may receive updates of data several times. The sponsor may have added or removed variables from some of the datasets, as they get closer to their internal primary analysis. This change in variables can directly impact the programs that have been written to analyze the data.

In order to efficiently analyze data, we require the ability to rapidly summarize the structure of the incoming data, determine if variables have been added or removed, locate specific variables by keyword, and determine missing data in key datasets. To assist us in these efforts, a collection of macros, which we refer to as the VAR family, has been designed to quickly obtain this information. These macros, when added to a program written to unpack a transport file or to load data from another file format, provide concise output, which enables the programmer to identify potential problems with the data. The data issues can then be communicated to the sponsor quickly and efficiently for resolution. Rapid prospective identification and communication of data issues decreases the time it takes to resolve the issues and affords more time to analyze the data.

THE VAR FAMILY OF MACROS

Designed to quickly identify the common data problems we have encountered in various projects, each member of the VAR family of macros has a specific use. The VAR family consists of the following four macros: **varlist**, which creates a list of all the variable names for a given library of data and shows which variables are common across different datasets; **varcomp**, which compares the variable names between two libraries of data and reports which variables have been added or deleted; **varscan**, which searches a list of variable names and variable labels for a given keyword; and **varsum**, which displays all the variables for a given table and lists the number of missing and non-missing values for each variable.

The VAR family is considered a family of macros because each macro is related to the others. Varcomp and varscan both call varlist as part of their program. This makes varlist the parent macro, and the other two the children macros. Varsum is the cousin of varlist because it does not call varlist directly but contains code that was originally from varlist, which was modified to run on a single dataset. The diagram below shows the relationship among the macros in the VAR family.



Varlist utilizes the OUT option of PROC CONTENTS to create a dataset that contains all the metadata about the dataset. At the time that the VAR family was created, I was not aware of how to access the Data Dictionary tables. The vcolumn table contains the same data that varlist generates and could be obtained with less code. The advantage of using PROC CONTENTS is that you can also generate other information about each dataset, such as number of observations, that vcolumn does not contain but is located in some of the other dictionary tables. Using the Data Dictionary tables to assemble all the information from the various tables would require more code then using PROC CONTENTS, where the data is already assembled.

Varcomp and varscan are considered the child macros of varlist because they use the dataset created by varlist to perform specific functions. Because of their relation to varlist, other VAR macros can be made using varlist as the parent, and expanding on varlist's output.

The sections that follow describe each macro, outline their uses, and detail some sample output from each one. In addition, several small case studies are presented where the VAR family was used to identify data anomalies.

VARLIST

MACRO CALL:
%varlist (whatlib=*libref*, wherelib=*libref*, print=(0 | Null))

FUNCTION: To create a list of all the variable names within a given library and to list all cases where a variable exists in more than one dataset.

USE: To summarize newly received libraries of data and to show how the datasets are related. We use this macro to get our first look at new data. From this list we can usually determine what the major merge variables are, as they will be located in all tables. Because we work with a number of different sponsors, each of whom have their own database systems, a variable as simple as a Subject Number could have many possible variable names which sometimes are not obvious to an external user. Also, when performing merges of tables, it is helpful to know if two tables have variables with the same name so that the data is not overwritten in the merge.

MECHANISM: Varlist runs on a specified library of data that is defined in the whatlib parameter. It creates a list of all the datasets in the specified library and then begins to assemble a dataset that contains the memname (dataset), variable name, type, length, label, format, and informat.

Once the dataset is assembled, a list of the dataset is printed using PROC PRINT. Then a second list is created using a DATA NULL step that prints a summary for each variable contained in multiple datasets. This list contains the variable name, label, and a list of all the datasets in which the variable is contained.

Varlist has two other parameters that can be used by other programs and are used by the other members of VAR family. The first is the wherelib parameter. This parameter is used to determine the library where the metadata dataset will be created. Typically WORK is the specified directory, so no permanent dataset is created. If a permanent dataset is desired, then another libref could be given, and varlist would output the dataset to that library.

The second parameter is the print parameter. This parameter is optional. When it has the value of 0, varlist does not create the two lists described above. When the parameter is not specified or given any value other then 0, the lists are printed. This parameter is used by the other VAR family macros so that varlist creates the necessary datasets but does not create a text output, when called.

EXAMPLE OUTPUT: The example below shows a typical output from the varlist macro. The first part is an excerpt of the variable list. The second part is an example of a variable located in two different datasets.

```
      Complete list of Variables from:
                    SAMPLIB

MEMNAME  NAME   TYPE LENGTH  LABEL        FORMAT

BLODSAMP DAY        1    8   STUDY DAY
DRUGADMN DAY        1    8   STUDY DAY
EXAM     DAY        1    8   DAY
MEAL     DAY        1    8   STUDY DAY
QUESTION DAY        1    8   STUDY DAY
VITALS   DAY        1    8   DAY
FORMATS  DEFAULT    1    5   DEFAULT
EXAM     DESCR1     2    40  DESC1
EXAM     DESCR2     2    40  DESC2
DEMOG    DOB        1    8   DATE OF BIRTH  DATE


Variable Name:   BIRTHD
Label:  Date of Birth
-------------------------------
Data Sets:  DEMOG
            SUMMARY
```

VARCOMP
MACRO CALL: %varcomp (newlib=*libref*, oldlib=*libref*)

FUNCTION: To compare the variable lists from two libraries of data and to determine which variables have been added, which have been removed, and which are the same.

USE: This macro is used when we receive an updated library of data. Typically we receive data at the beginning of a project, several times during the course of the study, and then the final locked dataset. This macro looks at the library of data we currently have and compares the newly arrived data to determine if any changes have occurred. When changes are noted, we are able to assess if those variable changes are going to impact our data assembly and data analysis programs.

MECHANISM: Varcomp starts by creating datasets of metadata from the two libraries. This is done by calling varlist with the print=0 parameter, twice. Once the datasets have been made, they are then merged by memname and variable name and put into three datasets using the IN option: newvar, lostvar, and merglst.

The newly created list begins with a summary of the number of variables in each of the three datasets followed by a list of newvar and lostvar.

EXAMPLE OUTPUT: The example below shows a typical output from the varcomp macro where there are differences in the variables between two libraries.

```
              VarComp Summary Report
   -------------------------------------------

  Number of Variables that are the same:  1128

  Number of Variables that are new:       9

  Number of Variables that are lost:      2
   -------------------------------------------

       Variables Not In Previous Data Set
                 (New Variables)

  OBS    MEMNAME     NAME        LABEL

   1    ADE         DRUG
   2    ADE         PAGEDONE    PAGE WAS DONE
   3    DOSTUDY     DRUG
   4    EOS         DRUG
   5    MED_HX      DRUG
   6    OTHMED      DRUG
   7    SMEDACC     DRUG
   8    VITAL       DRUG
   9    VITAL       PAGENO      PAGE NUMBER

                 N = 9

 Variables In Previous Data Set but Not In New
                    Data Set
               (Missing Variables)

 OBS    MEMNAME     NAME          LABEL

  1     ADE       PTINIT    PATIENT INITIALS
  2     OTHMED    PTINIT    PATIENT INITIALS

                 N = 2
```

VARSCAN
MACRO CALL:
%varscan (whatlib=*libref*, keyword=*text*, field=[N,L,B])

FUNCTION: To search a list of variable names and labels in order to locate variables that match the keyword.

USE: This macro is used when we receive a new library of data and want to locate commonly used variables such as sex, race, concentration, dosing, etc. Rather then reviewing the output of varlist or a stack of PROC CONTENTS, this macro will locate all possible instances and display them in a list.

MECHANISM: Varscan starts by creating a dataset of metadata by calling varlist with a print=0 parameter. Then the dataset is pared down by using a where statement which uses both the sounds-like and like options to find potential matches to the keyword that is specified. The keyword has to be a single word.

The field parameter allows the user to specify which fields varscan should search. A value of N searches only the variable name field. A value of L searches only the variable label field. A value of B searches both the variable name and label fields. The default setting is B, and if the field parameter is not included, the search is performed on both the variable name and label fields.

EXAMPLE OUTPUT: The example below shows a search for both variable names and variable labels that contain the keyword "dose."

```
       B: Variable Name Search for Labels and
          Variables that sound like: dose
               From Library: lib018

Obs    MEMNAME     NAME    LABEL

1      CONMED      DOSE    DOSE
2      DRUGADMN    DOSE    COMPOUND DOSAGE

                       N = 2

       B: Variable Name Search for Labels and
            Variables that contain: dose
               From Library: lib018

Obs    MEMNAME     NAME     LABEL

1      CONMED      DOSE     DOSE
2      DRUGADMN    DOSE     COMPOUND DOSAGE
3      CONMED      UNITS    DOSE UNITS

                       N = 3
```

VARSUM
MACRO CALL: %varsum (whatlib=*libref*, dset=*dataset*)

FUNCTION: To summarize a given table and determine for each variable the number of missing and non-missing values.

USE: In a library of data, there may be some datasets that are of critical importance, such as patient demographics. In these cases it is important to know how complete the data is for every variable. If important merge variables are missing, then the data will not merge properly. If dosing times and dates are missing, then it will be more difficult to assemble a dataset for analysis. This macro creates a summary table and provides a list of missing and non-missing values, allowing us to rapidly identify holes in the data.

Varsum is the workhorse of the VAR family. Using varsum, we are able to see the completeness of the data for any dataset via a straightforward summary table. This output is very useful to project team members who need to see a concise summary of the data in order to make analysis decisions based on the completeness.

MECHANISM: The cousin of varlist, varsum similarly creates a dataset of variable names from the dataset specified in the dset parameter. Using that metadata list as the list of variables in the table, varsum begins to count the blank and non-blank values for each variable. It does this by taking each variable one at a time and, for each record, determining if the value is missing or not missing and summing these counts. The metadata for the variable and the number of missing and non-missing values are then transferred to a new dataset as a single record.

When all the variables have been scanned, the final dataset of metadata and counts is then outputted as a list.

EXAMPLE OUTPUT:
```
   Variable List for dataset:  lib1.conc

Num Var     Label           Type      N  MISS
1  PROT    Protocol        Character  884    0
2  CENTER  Center #        Character  884    0
3  SUBJECT Subject #       Character  884    0
4  SAMPLE  SampleType      Character  884    0
5  DRDTRAW Draw Date       Character  884    0
6  DRAWDT  Draw Date (SAS)Numeric     884    0
7  VISIT   Visit Desc      Character  884    0
8  ANALYT  Analyte         Character  884    0
9  LAB     Analytical Lab  Character  884    0
10 EXTID   External ID #   Character  884    0
11 CONC    Concen.         Character  873   11
12 UNITS   Units           Character  884    0
13 COMENT  Lab Comments    Character  17   867
```

CASE STUDIES
The next section shows three cases where the VAR family of macros was used to gain knowledge of newly arriving data. In each case, the macros were called at the end of the program that was used to process the data. From the macro output, we were able to locate potential issues and have them resolved the same day the data was loaded.

CASE STUDY #1- NEWLY ARRIVED DATA; NO PAPERWORK
In this first case, a client sent us a transport file of SAS® datasets without sending any paperwork regarding database mappings, key fields, etc. The data needed to be unpacked, and data assembly needed to occur as soon as possible.

Varlist and varscan were both used. Varlist generated a list of all the variable names in the library of data as well as lists of which variables were located in multiple datasets. From the second part of the list we were able to determine what the primary key variable names were because certain variables appeared in every dataset. This helped us understand how the various data tables were related to each other.

We employed varscan several times using the keywords "conc," "dose," "age," "sex," and "race." The terms "conc" and "dose" are common for datasets that contain drug concentration levels. The like search of varscan will match "conc" to the word concentration or to the common abbreviation of "conc." Terms such as "age," "sex," and "race" are typical for demographics tables.

From the varscan output we were able to locate the demographics table by keyword matches to “age,” “sex,” and “race.” We were also able to determine that concentration data was not sent to us by the lack of the keyword “conc.” The keyword “dose” showed up in several other tables, as expected.

All of the varlist and varscan data was generated at the same time that the data was unpacked, allowing us to make the determinations above within 15 minutes of unpacking the data.

From the information that was obtained, the programmer assembling the data knew which variables were potential merge variables and that the concentration data was not present.

Had the programmer not had this up-front information, she would have had to manually search various PROC CONTENTS listings or visually survey tables using SAS View® to look for merge variables. During the manual process she would have determined that the concentration data was not present, but only after an extensive review of the datasets. The whole process would have taken hours, adding more time to data assembly and taking time away from data analysis.

CASE STUDY #2- UPDATED DATASET

In this case we had an established relationship with our sponsor and had previously received data from them in the form of transport files containing their CRF database. On this day, a new transport file arrived and needed to be unpacked.

The library was unpacked and varcomp was run on it. The list from varcomp showed that 130 new variables had been added. With some checking, it was discovered that the addition of most of these variables was due to tables that had been added to the library. Varcomp also identified several derived variables that the sponsor, as they performed their internal analysis, added to the datasets.

Using the information from varcomp, the project team was aware of the additional variables and datasets minutes after the data was unpacked. The team was then able to evaluate the new analysis variables and datasets and determine if they could be used in the analysis.

The sponsor’s cover letter did not indicate that new tables and analysis variables had been added since their last data transmission. Unless the programmer looked to count the tables, they would not have immediately known which tables were added, and their contents. Without varcomp, finding the new variables would have been an even harder task, and could only be determined by looking over PROC CONTENTS for each table received.

CASE STUDY #3- CONCENTRATION FILE RECEIVED

In this case, we were working with a sponsor who had been transmitting their CRF data to us and had contracted a CRO to process PK samples and transmit the data directly to us. The CRF data would come as a SAS® transport file, but the CRO sent the PK concentrations as an Excel® file.

The study had reached the end of data collection and the database had been cleaned and locked. The CRO transmitted their locked data for the PK concentrations. Once all the data was received, we would begin our analysis.

The Excel® file of concentrations was processed and transformed into a SAS® dataset and a varsum call was added to the end of the program. From the output of varsum, it was shown that 11 records had missing concentration values. Because this data was locked and our analysis of it was about to proceed, we contacted the sponsor the same day the data arrived and confirmed with them that the missing values could not be recovered because they were lost samples and were unable to be processed.

This confirmation was able to take place within an hour of processing the data. Without using varscan to summarize the completeness of the data, it may have been days before someone working with the data realized the missing values and only then queried the sponsor, at a marked loss of analysis time.

CONCLUSION

The VAR family of macros has become a standard tool for analyzing incoming data at Cognigen. Using the VAR macros, we are able to identify potential problems with newly processed data and to communicate our questions to our sponsors in the same day. This has taken the burden off our project programmers, who assemble and analyze the data, and allows basic data checking to be performed upon receipt of the data instead of during data assembly.

With the structure of the VAR family, it is easy to create new members of the family by creating calls to the other family members. As we encounter new issues, we plan on creating additional members of the family to address these issues.

ACKNOWLEDGMENTS

I would like to acknowledge the Data Management Department of Cognigen Corporation. As a relatively new SAS® programmer, I have worked with every member of the Data Management team to answer questions and solve problems I encountered while creating the VAR family. Thank you to the all the SAS® Programmers in the Data Management Department for all your time and help.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Phil Vecchione
Cognigen Corp.
395 Youngs Road.
Buffalo, NY 14221-5831
Work Phone: 716-633-3463 (1-800-248-4244)
Fax: 716-633-7404
Email: phil.vecchione@cognigencorp.com
Web: www.cognigencorp.com