

DICTIONARY TABLES - DATA ABOUT YOUR DATA

A component was added in SAS Version 6.07 called dictionary tables. This column will introduce you to these interesting and useful objects in the SAS system.

Dictionary tables are special read-only PROC SQL objects that contain systems catalogs for all SAS datasets and other SAS files.

Some of this information is available from PROC CONTENTS, PROC DATASETS, and other sources, but dictionary tables are more readily available and are usually easier to access.

Using Dictionary Tables

You might use a dictionary table when you want to...

- know if a dataset exists, how many variables it contains, and how many observations it contains
- check the current value of an option or title, change it, but then reset it back to the original value when you are finished
- determine if a macro variable exists
- capture the names of variables, as well as the attributes, from a SAS data file

Dictionary tables are generated at run time to retrieve information about SAS datasets, libraries, SAS catalogs, SAS macros, SAS titles, SAS options, and external files known to SAS. By using PROC SQL and the CREATE TABLE or CREATE VIEW clauses, they can be accessed as any other SAS dataset would be, to provide your programs with the information stored in the dictionary tables. Because they are read-only, you cannot alter dictionary table values directly.

A partial list of the dictionary tables is shown below:

DICTIONARY.MEMBERS	SAS files, catalogs
DICTIONARY.TABLES	SAS datafiles
DICTIONARY.COLUMNS	SAS vars
DICTIONARY.CATALOGS	catalogs and entries
DICTIONARY.EXTFILES	external files

DICTIONARY.VIEWS	SAS views
DICTIONARY.INDEXES	indexing info
DICTIONARY.OPTIONS	SAS options
DICTIONARY.MACROS	SAS macros vars
DICTIONARY.TITLES	SAS Titles

For a more complete listing, reference the SAS documentation or use the PROC SQL DESCRIBE TABLE statement.

Using the DESCRIBE TABLE statement

To list the names of the columns stored in the DICTIONARY.MACROS entry, the following program can be run.

```
proc sql;
  describe table dictionary.macros;
```

The resulting Log:

```
NOTE: SQL table DICTIONARY.MACROS
was created like:

create table DICTIONARY.MACROS
(
  scope char(9) label='Macro Scope',
  name char(32) label='Macro Variable
  Name',
  offset num label='Offset into Macro
  Variable',
  value char(200) label='Macro
  Variable Value'
);
```

To print the data values of the MACROS entry, the PROC SQL CREATE table can make a SAS file which contains an extract of the MACRO entry at that time. The extract file is a normal SAS dataset that can be processed like any SAS dataset.

```
proc sql;
  create table work.macros as
  select * from dictionary.macros;
;
proc print data=work.macros;
  title 'Work.macros';
run;
```

Continued on page 5.....

IN THIS ISSUE

Dictionary Tables - Data About Your Data: Steve First gives an overview of dictionary tables and explains when to use them.....Page 1

Partnering with SAS Institute: Steve First shares some new and exciting information with youPage 2

Using VSAM Files with SAS®: David Beam expounds on the tools available in SAS for working with 'Virtual Storage Access Method' (VSAM) files.....Page 2

Sending PROC Output to SAS® Datasets: Guest Author Kirk Lafler explains a feature of the Output Delivery System (ODS) that you may not be familiar with.....Page 4

SAS® Help Desk I/O: Gerald Frey answers several short questions that were recently submitted to our help deskPage 5

Technical Credit:.....Page 7

Public Class Schedules:.....Page 8



SYSTEMS SEMINAR CONSULTANTS, INC.
 2997 Yarmouth Greenway Drive
 Madison, WI 53711
 Website: www.sys-seminar.com
 Email: train@sys-seminar.com
 Phone: (608) 278-9964
 Fax: (608) 278-0065



PARTNERING WITH SAS INSTITUTE

SAS INSTITUTE WILL OFFER PUBLIC CLASSES AT SSC



While Systems Seminar Consultants (SSC) has been a SAS Quality Partner™ for several years, we and SAS Institute's Training and Education Department are excited to announce a unique partnership.

Starting in August, SAS Institute public classes will be taught at Systems Seminar Consultants' Madison, Wisconsin training facility.

In addition, SSC will be providing instructors for public and on-site SAS Institute classes, both locally and nationally. SSC and SAS Institute are also exploring joint development of new course topics. SSC will continue to provide on-site SAS training to our corporate clients, as well as, offer public SAS software classes in St. Paul, Minnesota.

I view this partnership as a unique opportunity for everyone involved: SSC, SAS Institute, and you. SSC instructors can share their real world knowledge and experience with more users than ever, the training curriculum can be expanded to include more topics, and more top quality SAS training classes will be offered in the local Madison area.

We are proud to be part of this new endeavor and are confident that it will better serve your needs in the future!

On another note, we hope you are finding our newsletter useful and interesting. We enjoy putting it together and always welcome your input. In this issue, you will find an article by a guest author, Kirk Lafler. If you would like to share your SAS and writing skills in an article or Quick Tips, please let us know.

Sincerely,

Steve First
President



USING VSAM FILES WITH SAS®

EASY READING OF THIS MVS FILE STRUCTURE

IBM mainframe sites often use a data file structure called "VSAM", or "Virtual Storage Access Method". These files are similar to external flat files but can be accessed in methods other than the "normal" sequential order, such as direct keyed access.

SAS offers exceptional tools for reading, writing, and loading VSAM files in a variety of methods, depending on the data organization in the file. There are several types of VSAM files; we will cover the two most common types:

- **Entry-sequenced (ESDS):** Record sequence determined by the order the records are entered into the data file. These files can be created with an alternate index file, such as something like a person's last name.
- **Key-sequenced (KSDS):** A key contains a unique value (primary key) such as an invoice number or SSN.

Accessing the above types of VSAM files with SAS can be done in three ways:

1. **Sequential** – retrieve records in sequence
 - a. ESDS – by original entry sequence
 - b. KSDS – in order by primary keySAS ends reading the file when the end of file is encountered. The END= option can be used to check for the last record.
2. **Direct Access**
 - a. keyed by key value
 - b. keyed with the alternate indexThe END= option is NOT valid. You must end the DATA step with a STOP statement, or a SET or INPUT statement that reaches end-of-file.
3. **Skip Sequential** – combination of the above two modes: find a starting point with keyed direct access, then retrieve sequentially. Use the STOP or SET statement to end the step.

SAS System Options

There are three global options used with VSAM processing. The first option, VSAMREAD, allows VSAM files to be read by SAS. The second option is VSAMUPDATE, which allows VSAM files to be updated by modifying or erasing existing records or by adding new records. VSAMUPDATE implies VSAMREAD. The third system option, VSAMLOAD, allows you to load records into a new VSAM file.

Normally, the SAS system is shipped with VSAMREAD on, and VSAMUPDATE and VSAMLOAD off. You can check your site's settings by running PROC OPTIONS. You can change the settings by coding OPTIONS VSAMREAD; etc., as long as your installer has not restricted you from changing them.



SYSTEMS SEMINAR CONSULTANTS, INC.

Copyright © 2000 Systems Seminar Consultants, Inc. Madison, WI
All rights reserved. Printed in USA. The Missing Semicolon is a trademark of Systems Seminar Consultants, Inc., SAS is a registered trademark of SAS Institute Inc., and SyncSort is a registered trademark of SyncSort, Inc. in the USA and other countries.

INFILE Options

The INFILE statement has a multitude of options that are used for processing VSAM files. We will discuss a few of them here and then show some sample SAS code using VSAM files.

BACKWARD	Tells SAS to read sequentially, backwards
FEEDBACK= <i>variable</i>	Defines numeric variable that SAS sets to the VSAM error code
GENKEY	Used to specify generic-key processing
KEY= <i>variable(s)</i>	Indicates keyed direct access for KSDS, or use of alternate index for ESDS
KEYLEN= <i>variable</i>	Numeric variable used with GENKEY to specify length of keys to compare
KEYPOS= <i>variable</i>	Numeric value set to the position of the key field
VSAM	Indicates the file referenced on the INFILE statement is a VSAM file (optional in MVS batch jobs)

Reading a KSDS Sequentially, in KEY Order

To read a KSDS sequentially the good news is you don't have to do anything out of the ordinary. You can read the file like you would any other sequential file.

The example below from an auto manufacturer has a VSAM file called VIN master which contains vehicle identification numbers as well as other fields about the autos that have been built. To sequentially read all of the records, just code a data step to read the vin master normally.

```
//MYJOB JOB accting etc
//      EXEC SAS
//VINMAST DD DSN=PROD.VIN.MASTER,DISP=SHR

data vins;
infile vinmast;
input @1  modelyr $char2.
      @3  vin      $char18.
      @31  blddate yymmdd6.
      @106 vdlrno $char4.
      +102 vtrfno $char4.;
bldmonth = intnx('month',blddate,0);
format blddate bldmonth yymmdd10.;
run;
```

Reading a VSAM File Backwards

Suppose we would like the above data, but sorted into descending MODELyr and VIN order. Obviously we could sort the file, but VSAM can also return the records in reverse order using the BACKWARD infile option. In some cases because optimized system level routines do the sorting, the overall resources consumed may be less, or it may just make your programming more convenient.

```
data backvins;
infile vinmast backward;
input @1  modelyr $char2.
      @3  vin      $char18.
      @31  blddate yymmdd6.
      @106 vdlrno $char4.
      +102 vtrfno $char4.;
bldmonth = intnx('month',blddate,0);
format blddate bldmonth yymmdd10.;
run;
```

QUICK TIP

To condense the size of your proc print output, use the width=min option.



Keyed Access

One of the most powerful features of VSAM is the ability to read records directly by some key value. This is sometimes called random access, though it's not really a random process at all. Keyed access is usually performed when a match is needed to some other file and we need to read only a small percentage of the VSAM records. If you need to read more than 10-20% of the VSAM records, it may actually be more efficient to sequentially sweep the file like the example above and then merge the files sequentially. Keyed access really can have enormous time and resource savings when the volume of records read is small.

Reading a VSAM file with a key is somewhat complex as there are several issues:

1. You will probably need to read two files in the data step. The easiest type of this second file to read is a SAS dataset which you can SET. Other files of course could be used as long as the data step stops eventually (usually when the file is out of data). It is also important to note that when accessing VSAM with a KEY there will never be an automatic end-of-file indication on the VSAM file that will stop the datastep. VSAM will tell us of end-of-file, but we must stop the step.
2. There will be additional options needed for the VSAM read as we now need to pass key information to VSAM and will need to check status codes after the read.
3. We will need to check whether the read was successful. A desired record may not exist or there could be a more catastrophic error such as the VSAM file not being available or it is damaged.
4. SAS may consider the above an error and may produce error listings that we may want to suppress.

The example below reads a SAS dataset containing MODELyr and VIN, constructs a key, then reads the VSAM file with the corresponding key value. If a matching record is found, an observation is written to QGOOD file, otherwise the observation is written to the QERROR file.

```
data qgood(keep=vin bldmonth modelyr)
  qerror(keep=error vin);
length error $ 72 keyvar $21;
attrib bldmonth length=4 format=monyy5.;
/* klvar - 21 byte generic key for infile */
/* kposvar - starting position of key in vsam */
/*****
/* read sas dataset qvin containing vin, modelyr */
/* then randomly access vsam vin master file */
*****/
```

Continued on page 7.....

SENDING PROC OUTPUT TO SAS® DATASETS USING OUTPUT DELIVERY SYSTEM (ODS)

Much of the emphasis on the Output Delivery System (ODS) is for producing HTML output and that is well documented, and perhaps a topic for another issue. A lesser known feature of ODS however, is that it can also capture output from SAS procs and store it in a SAS dataset.

We'll see how data from any output-producing procedure can be sent to a dataset using ODS. The beauty of using ODS to accomplish this is that the syntax remains consistent from procedure to procedure. There's no need to remember the unique syntax requirements for each procedure as it was prior to ODS. So here's a tip to help make your programming experience more rewarding.

The ability to send output, without the use of any fancy programming techniques, from any output-producing procedure to a SAS dataset is a feature many SAS users have wanted for along time. Fortunately the SAS Institute listened and delivered this capability in ODS. ODS provides users with a consistent way to output the results of any output-producing procedure to a SAS dataset. Whether you use detail-level or summary-level procedures, output from either procedure can be sent to a SAS dataset the same way. This not only saves time - it makes performing these types of tasks a simple matter.

Before you can successfully send a procedure's output to a SAS dataset, you'll need to know two things:

- 1) how to identify an output object's name
- 2) the naming conventions for naming a SAS dataset

Identifying the names of output objects created in a procedure is relatively easy. For example, to identify the names of all the output objects created in the UNIVARIATE procedure for a dataset containing "movie" data is accomplished by using the following ODS statement code:

```
ODS TRACE ON ;  
PROC UNIVARIATE DATA = MOVIES ;  
RUN ;  
ODS TRACE OFF ;
```

QUICK TIP

To change the colors in your SAS environment while working with display manager, submit the following statement:

```
DM 'COLOR area color';
```

Example: DM 'COLOR BACKGROUND YELLOW';

The areas you can color are: background, banner, border, command, foreground, and message.



The results of the trace are automatically displayed in the Log window as follows:

```
Output Added:  
-----  
Name:           Moments  
Label:          Moments  
Template:       base.univariate.Moments  
Path:           Univariate.age.Moments  
-----  
Output Added:  
-----  
Name:           BasicMeasures  
Label:          Basic Measures of Location and Variability  
Template:       base.univariate.Measures  
Path:           Univariate.age.BasicMeasures  
-----  
Output Added:  
-----  
Name:           TestsForLocation  
Label:          Tests For Location  
Template:       base.univariate.Location  
Path:           Univariate.age.TestsForLocation  
-----  
Output Added:  
-----  
Name:           Quantiles  
Label:          Quantiles  
Template:       base.univariate.Quantiles  
Path:           Univariate.age.Quantiles  
-----  
Output Added:  
-----  
Name:           ExtremeObs  
Label:          Extreme Observations  
Template:       base.univariate.ExtObs  
Path:           Univariate.age.ExtremeObs  
-----
```

As can be seen from viewing the SAS Log, there are five output objects created by UNIVARIATE (Moments, BasicMeasures, TestsForLocation, Quantiles, and ExtremeObs). Once this is known, we are ready to send output to a SAS dataset.

The syntax used to send output to a SAS dataset using ODS follows:

```
ODS OUTPUT output-object = user-defined-SAS-dataset-name ;
```

where the **output-object** is the name assigned to the piece of information created by the specific procedure. The output-object name is identified by using the ODS TRACE ON statement (see previous example). The **user-defined-SAS-dataset-name** can be any valid user-supplied SAS name such as MOMENTS_RESULTS.

Continuing with the UNIVARIATE example, we can send the results of just the MOMENTS output object to a SAS dataset as follows:

```
ODS LISTING CLOSE ;  
ODS OUPUT MOMENTS = MOMENTS_RESULTS ;  
PROC UNIVARIATE DATA = MOVIES ;  
RUN ;  
ODS OUTPUT CLOSE ;  
ODS LISTING ;
```

Continued on page 5.....

SAS® HELP DESK I/O

SOLUTIONS FROM OUR HELP DESK SERVICE

Q: "I want to format a value, but not to print. I want the formatted value in my SAS dataset. How can I do that?"

A: To keep the formatted value in a new SAS dataset, use the following code:

```
VAR1=put (VAR,REGFMT.);
```

This will move the value (via the put function) to the new variable, instead of just printing the var with the format. In this example, VAR contains the region number and we want VAR1 to contain the region name. REGFMT is the name of the format we created containing all region names.

Q: "I need to pull data from DB2 via PROC SQL pass-thru and specify a date range with dates in the form 'YYYY-MM-DD'. Is there a way that I can pass a parm to SAS but still insert the quotes and date values in the WHERE statement correctly?"

A: The SYSPARM option can be used to pass up to 200 characters of any text you would like. To SAS, the value just looks like a text string. You can break it into pieces and, using SAS functions, convert the string into individual dates and wrap single quotes as necessary. Macro variables can then be created that can be used in the PROC SQL step.

```
//JOB1 JOB ABC
// EXEC SAS,OPTIONS='SYSPARM="1970-01-01 1970-01-02"'
OPTIONS SYMBOLGEN;
DATA NULL ;
LENGTH DATE1 DATE2 $12;          /*NULL STEP */
DATE1=SCAN("&SYSPARM",1,' ');      /*LEN OF DATES PLUS QUOTES */
DATE2=SCAN("&SYSPARM",2,' ');      /*1ST WORD OF SYSPARM */
DATE1="'" !! TRIM(DATE1) !!"'";    /*SECOND WORD OF SYSPARM */
DATE2="'" !! TRIM(DATE2) !!"'";    /*WRAP DATE1 WITH QUOTES */
CALL SYMPUT('MDATE1',DATE1);     /*WRAP DATE2 WITH QUOTES */
CALL SYMPUT('MDATE2',DATE2);     /* CREATE MAC VAR */
RUN;
%PUT ****SYSPARM=&SYSPARM        /* CREATE MAC VAR */
      MDATE1=&MDATE1
      MDATE2=&MDATE2;
PROC SQL;
CONNECT to etc.
CREATE TABLE XYZ etc.
WHERE DATE BETWEEN &MDATE1 AND &MDATE2; /*USE PARM IN WHERE */
QUIT;
```

Q: "I know I can use the &SYSDATE macro variable to put current date like 16JUN00 into a title, but is there a way to format it as 06/16/2000?"

A: You can create your own macro variable using %SYSFUNC which allows you to call most datastep functions and apply a format. Calling the DATE() function will return current date, and applying MMDDYY10. format should give the desired results.

```
%let mydate=%sysfunc(date(),MMDDYY10.);
TITLE "My report as of &mydate";
```

the title will print as: My report as of 06/16/2000

QUICK TIP

To take extra blanks out of a variable (character), use the COMPBL function.

Example: BIGNAME="MARY JANE SMITH"
SMALLER=COMPBL(BIGNAME);
Resulting in: "MARY JANE SMITH"

DICTIONARY TABLES

CONTINUED FROM PAGE 1

Partial Output:

Obs	scope	name	offset	value
1	GLOBAL	SQLOBS	0	0
2	GLOBAL	SQLOOPS	0	0
3	AUTOMATIC	SYSDAY	0	Thursday
4	AUTOMATIC	SYSTIME	0	09:14
.

Although using the PROC SQL code is probably the fastest way to access dictionary table data, SAS software comes standard with several views in the SASHELP library that you can access without PROC SQL.

Some of the views are listed below:

SASHELP.VMEMBER	SAS files and catalogs
SASHELP.VTABLE	SAS datafiles
SASHELP.VCOLUMN	SAS vars in all datasets
SASHELP.VCATALG	SAS catalogs and entries
SASHELP.VEXTFL	external files allocated
SASHELP.VVIEW	SAS views
SASHELP.VINDEX	SAS indexing information
SASHELP.VOPTION	SAS options
SASHELP.VSLIB	Sorted libname list
SASHELP.VSTABLE	Sorted table list
SASHELP.VMACRO	SAS macros
SASHELP.VTITLE	SAS Titles

Continued on page 6.....

SENDING PROC OUTPUT TO SAS® DATASETS

CONTINUED FROM PAGE 4

In the preceding example, we first close the Listing destination (this prevents output from being sent to both the Listing and Output destinations). We then specify the desired output object (e.g., MOMENTS) in the ODS OUTPUT statement and assign a user-defined-SAS-dataset-name (e.g., MOMENTS_RESULTS).

After the UNIVARIATE procedure sends the MOMENTS output to the user-defined dataset, we close the Output destination and reopen the Listing destination. It's that simple. If you would like more information about this article, please contact Kirk Lafler at KirkLafler@cs.com.

DICTIONARY TABLES

CONTINUED FROM PAGE 5

To see what is stored in each of the views, we could use a separate PROC CONTENTS and/or PROC PRINT, or we could use the SASHELP.VVIEW member to get the member names of all views that SAS knows of.

If we are interested in columns (variables) that are stored in the SASHELP views, the SASHELP.VCOLUMN view contains a row for each variable stored in all views known to SAS at that time.

Examples Using SASHELP Views

To show all of the columns in all of the SASHELP views, use the code that follows:

```
proc print data=sashelp.vcolumn;
  where libname='SASHELP'
  and substr(memname,1,1)='V';
run;
```

Partial Output:

Obs	libname	memname	memtype	name	type...
156	SASHELP	VCATALOG	VIEW	libname	char
157	SASHELP	VCATALOG	VIEW	memname	char
158	SASHELP	VCATALOG	VIEW	memtype	char
159	SASHELP	VCATALOG	VIEW	objname	char
160	SASHELP	VCATALOG	VIEW	objtype	char
161	SASHELP	VCATALOG	VIEW	objdesc	char
162	SASHELP	VCATALOG	VIEW	created	num
.

To use the SASHELP.VTABLE view, again just reference it like you would any other SAS dataset.

If you want to list all datafiles in the SASDATA library with corresponding create dates and number of observations, while selecting only members created after 1 January, 2000, use the following code:

```
proc print data=sashelp.vtable;
  where crdate ge '01JAN2000:0:0'dt
  and libname='SASDATA';
  var memname crdate nobs;
run;
```

Output:

OBS	MEMNAME	CRDATE	NOBS
1	ADDRDATA	13JAN00:10:19:58	2
2	BOTH	13JAN00:10:20:07	1730
3	COMPFILE	13JAN00:10:19:52	1
4	FILE1	13JAN00:10:19:59	1730

Other Applications

Let's look at an example where you are writing a macro that will accept a SAS library and member as the input dataset. If the dataset doesn't exist or has 0 rows or columns, you want to fail the process.

```
%MACRO SSCTEST (MLIB=,MMEM=);
%LET MSASDS=&MLIB..&MMEM;
PROC SQL;
  CREATE TABLE WORK.VCOLUMN AS
  SELECT NAME, TYPE, FORMAT, LENGTH, LABEL
  FROM DICTIONARY.COLUMNS
  WHERE LIBNAME=UPCASE("&MLIB")
  & MEMNAME=UPCASE("&MMEM");
DATA _NULL_;
  IF NOBS=0 THEN
  DO;
    FILE LOG;
    PUT "*** MSASDS=&MSASDS DOESNT EXIST OR
  HAS 0 VARS**";
    PUT "*** MLIB=&MLIB MMEM=&MMEM **";
    STOP;
  END;
  SET WORK.VCOLUMN END=EOF NOBS=NOBS;
/* rest of macro */
RUN;
%MEND SSCTEST;
```

If the macro is called using a non-existent dataset or one with no variables, WORK.VCOLUMN will have no observations. The data step checks for the number of observations and will issue the message and stop running if there are no observations. If there are rows and columns in the passed dataset, the program above now has access to the other column attributes such as name, type, length, format, label, and more.

Capturing a SAS Option Value

Suppose you would like to capture the current linesize setting from SAS, then do a report with a different linesize, and when finished reset the linesize to the original value.

PROC SQL can query DICTIONARY.OPTIONS and can store the current linesize value into a macro variable called mlinesiz. The program can then change the linesize value, do the report, then by using the macro variable mlinesiz, set the option back to its original value.

```
proc sql noprint; /* capture setting */
  select setting into :mlinesiz
  from dictionary.options
  where optname='LINESIZE';
quit;

options linesize=132; /* set linesize to 132 */
proc print data=mydata;
run;

options linesize=&mlinesiz; /* set it back */
```

This technique can be used to capture any SAS option setting. Similar logic could be used to store SAS titles and footnotes if necessary.

In summary, there are lots of ways to capture data about your data. Dictionary tables are often an appropriate and easy way to incorporate that information into your applications.

QUICK TIP

To avoid division by zero errors in your log, code division statements as:

```
IF A NOT IN (0,.) THEN C=B/A;
ELSE C=0;
```



USING VSAM FILES WITH SAS®

CONTINUED FROM PAGE 3

```
retain klvar 21 kposvar 1;
infile vinmast /* fileref of vsam file */
  Vsam
  Keypos=kposvar
  Keylen=klvar
  Key=keyvar
  Feedback=feedback ;
set qvin; /*sas ds with vins */
keyvar=modelyr||vin||'1'; /* construct key */
input @ ; /* issue vsam read */
if iorc = 0 and
feedback = 0 then do; /* found it ? */
  Input @31 bldate yymmdd6.
    @106 vdlrno $char4.
    +102 vtrfno $char4. @;
  bldmonth = intnx('month',blddate,0);
  format blddate yymmdd10.;
  output qgood;
end;

/* not found,send msg to error log (not sas log) */
if feedback = 16 then do;
  error = 'vin not found in vin master file';
  output qerror;
  _error_=0; /* reset these variables*/
  feedback=0; /* after an error!!! */
end; run;
```

GENKEY Option

GENKEY specifies generic key processing. SAS treats the value given by the KEY= variable as a partial key (the leading portion) of the desired record. The input statement will read only the first record matching the partial key (unless you do skip sequential processing). Changing the value of the KEY= variable indicates a new generic key request.

The example below reads only the first record for each MODELRYR and VIN. Since the earlier program also had a more detailed key specifying just a '1' after the VIN, the results are identical in this example.

```
data qgood(keep=vin bldmonth modelyr)
  qerror(keep=error vin);
length error $ 72 keyvar $21;
attrib bldmonth length=4 format=monyy5.;
/* klvar - 20 byte generic key for infile */
/* kposvar - starting position of key in vsam */
retain klvar 20 kposvar 1;
infile vinmast /* fileref of vsam file */
  Vsam
  Genkey
  Keypos=kposvar
  Keylen=klvar
  Key=keyvar
  Feedback=feedback ;
set qvin; /*sas ds with vins */
keyvar=modelyr||vin; /* construct key */
input @ ; /* issue vsam read */
if iorc = 0 and
feedback = 0 then do; /* found it ? */
  Input @31 bldate yymmdd6.
    @106 vdlrno $char4.
    +102 vtrfno $char4. @;
  bldmonth = intnx('month',blddate,0);
  format blddate bldmonth yymmdd10.;
  output qgood;
end;
/* not found,send msg to error log (not sas log) */
if feedback = 16 then do;
  error = 'vin not found in vin master file';
  output qerror;
  _error_=0; /* reset these variables*/
  feedback=0; /* after an error!!! */
end; run;
```

QUICK TIP

———— To change a datetime variable to a date only, use the *datepart* function.

Example: MYDATE=DATEPART(MYDATE) ;



Skip Sequential Access

Skip sequential access is a hybrid of keyed and sequential access to VSAM. An initial record of a series is located using keyed direct access, and sequential access is then used to return following records usually containing the same generic key. In some cases this type of access can be much more efficient than either straight sequential or direct accessing alone. When you use the SKIP INFILE options, leaving the value of the KEY= variable unchanged turns off direct access and indicates that subsequent records are read sequentially. Also, when doing SKIP sequential processing, you must explicitly end the data step with a SET or a STOP statement, since a VSAM end-of-file is a feedback code of 4, which is not automatically checked in the datastep.

Suppose we want to read multiple records for a particular MODELRYR and VIN. We are interested in relatively few records and our VSAM file is huge. The following program should perform the above, though it has not been tested.

Continued on page 8.....

TECHNICAL CREDIT AND RECOGNITION



Steve First

President

Author of this issue's:

Dictionary Tables-Data About Your Data, Page 1



David Beam

Vice President

Author of this issue's:

Using VSAM Files with SAS®, Page 2

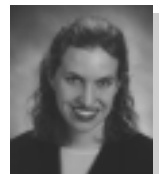


Gerald Frey

Trainer/Consultant

Author of this issue's:

SAS® Help Desk I/O, Page 5



Katie Minten

Trainer/Consultant

Author of this issue's:

Quick Tips, Pages 3-7

Sending PROC Output to SAS® Datasets.....Guest Author, Kirk Lafler
PublisherCindy Kersten
EditorsDavid Beam, Gerald Frey, & Cindy Kersten



USING VSAM FILES WITH SAS®

CONTINUED FROM PAGE 7

```
data qgood(keep=vin bldmonth modelyr);
length keyvar $20;
attrib bldmonth length=4 format=monyy5.;
/* klvar - 20 byte generic key for infile */
/* kposvar - starting position of key in vsam */
retain klvar 20 kposvar 1;
infile vinmast /* fileref of vsam file */
       Vsam
       Genkey
       SKIP
       Keypos=kposvar
       Keylen=klvar
       Key=keyvar
       Feedback=feedback ;
set qvin; /*sas ds with vins */
keyvar=modelyr||vin; /* construct key */
input @; /* read 1stin group*/
if feedback = 4 then stop; /* eof */
if feedback = 16 then
  Put '** keyvar=not found';
if feedback=0; /* found it */
keysave=keyvar; /* save key value */

do while(keysave=keyvar);
  input @1 keysave $CHAR20.
        @31 blddate yymmdd6.
        @106 vdlrno $char4.
        +102 vtrfno $char4. @;
  If feedback=4 then stop;
  bldmonth = intnx('month',blddate,0);
  format blddate bldmonth yymmdd10.;
  output qgood;
end;
run;
```

Writing to VSAM Files

Even though most SAS VSAM applications are read-only, SAS can also add new records as well as update records in place in VSAM files. While this makes many a disk manager nervous, if it is done correctly, writing to VSAM with SAS should not pose any more risk than would any other programming language, and in fact SAS is quite often the best utility to use for such an update application. Obviously, the programmer needs to be extra vigilant when writing and overlaying data.



SYSTEMS SEMINAR CONSULTANTS, INC.

2997 Yarmouth Greenway Drive
Madison, WI 53711

SSC PUBLIC CLASS SCHEDULE ST PAUL, MN

Introduction to SAS®	SAS® Efficiencies
August 28-30 \$725	August 31 \$350
October 16-18 \$725	November 16 \$350
December 4-6 \$725	
The SAS® SQL Procedure	SAS® Report Writing
October 20 \$350	September 11-12 \$525
Advanced SAS®	SAS® Macros
September 13-15 \$725	November 14-15 \$525
December 11-13 \$725	SyncSort®
	November 17 \$350
To register call (608) 278-9964 or visit www.sys-seminar.com	

SAS INSTITUTE PUBLIC CLASS SCHEDULE MADISON, WI

New Features in Version 8 of the SAS® System
August 16-17 \$650
SAS® Programming I: Essentials
September 12-14 \$975
Statistics I: Introduction to ANOVA, Regression, and Logistic Regression
October 10-12 \$975
SAS® Programming II: Manipulating Data with the DATA Step
October 17-19 \$975
SAS® Macro Language
November 15-16 \$650
To register call (800) 333-7660 or visit www.sas.com/training

PRSR STD
U.S. POSTAGE
PAID
MADISON, WI
PERMIT #2783

Call or visit our website for your
free subscription to
The Missing Semicolon™ .