

DATA Step Views

Jeffrey A. Polzin, SAS Institute Inc., Cary, NC

ABSTRACT

Traditionally, SAS® software programmers used the DATA step to extract data from external files in order to store the data in SAS data sets for subsequent use by SAS procedures. The DATA step view provides an alternative method of accessing and managing data transparently. DATA step views inherit the power and flexibility of the DATA step, with the ability to manipulate and manage data from a variety of sources including external files and existing SAS data files. This paper describes the input and output DATA step view conceptual models, the application uses of each type, the definition process for each view type, restrictions and requirements, and extensions to the DATA step language to support DATA step views. Performance issues of CPU, I/O, elapsed time, and memory utilization are discussed with comparisons to earlier data extraction methods. More detailed examples and discussion are provided to demonstrate the more complex uses of DATA step views.

INTRODUCTION TO DATA STEP VIEWS

DATA step views extend the power of the DATA step by enabling you to use DATA step programs to generate SAS data views. You can use DATA step views to process directly any file that can be read with an INPUT statement, to generate data dynamically with no external data sources without creating an intermediate SAS data file, or to receive output from a procedure or another DATA step. Once created, DATA step views can be used to migrate data to SAS data sets or to supported database management systems. They can be combined with other data sources using the SQL procedure, and they can be referenced by SAS/ASSIST® users, allowing end-users to perform data management, analysis, and reporting tasks regardless of how the data are stored.

Using DATA step views is a two-part procedure. First you create the view by compiling the SAS source program and storing the compiled code in a SAS DATA step view. Then, you process the view by referencing the view as the input or output data set in a PROC step or another DATA step program.

```
/* Define the view for use */
DATA READ/VIEW=READ;
  INFILE PARTS;
  INPUT PART QUANTITY UNIT COST;
  RUN;
/* Use the view */
PROC PRINT DATA=READ;
  RUN;
```

The previous code provides a simple example of an input DATA step view. The first step creates or defines the input DATA step view. The next step, the PRINT procedure, uses the previously defined input DATA step view to retrieve records out of an external file defined by the fileref PARTS and to read data into a set of variables. These variables are then written to the procedure, PROC PRINT, to be printed. A DATA step or procedure has no information detailing how or where the incoming data are produced; the data could be from a SAS data set, a SAS/ACCESS® data set, an SQL view, or an input DATA step view.

Note: Input DATA step views will be shipped as a production release and output DATA step views will be shipped as an experimental release when the 6.07 release of the SAS System for MVS, CMS, and VMS® is distributed.

DATA Step View Conceptual Model

Conceptually, a DATA step view is the same as other SAS views. A DATA step view is a SAS data set of the type VIEW and like other SAS data views, such as those created with the SQL procedure, the DATA step view does not actually contain data values. Instead, it contains a stored DATA step program that defines data or describes data stored elsewhere. The Stored Program Facility allows the user to compile and store DATA step programs and then execute the stored programs at another time. The Stored Program Facility is used by the DATA step views to store the compiled view and later execute it. For more information on SAS data views refer to "SAS Data Set Model," in Chapter 6, "SAS Files", in *SAS Language: Reference*.

For more information on the Stored Program Facility refer to Appendix 3, "Stored Program Facility," in *SAS Language: Reference*.

The scope of the DATA step view is much broader than that of other SAS data views. PROC SQL views can only retrieve data values from other SAS data files or SAS data views. SAS/ACCESS views only describe data in a single DBMS table or file to the SAS System. But, because DATA step views are generated by the DATA step, they inherit the power of the SAS DATA step language to manipulate and manage input data from a variety of sources including data from external files and data from existing SAS data sets.

DATA step views that function as input SAS data sets are called input DATA step views. DATA step views that function as output SAS data sets are called output DATA step views.

As with all SAS files in Version 6, the SAS System accesses DATA step views through an engine. When you create a SAS DATA step view, the name of the DATA step view engine, SASDSV, is stored with the DATA step view and is internal to the DATA step view engine. The DATA step view engine name cannot be changed by the user. When you use the view as input or output in a SAS procedure or DATA step, the SAS I/O Engine Supervisor automatically selects the SASDSV engine by reading the name stored with the view. For more information on SAS System engines, see Chapter 6 in *SAS Language: Reference*.

DATA Step Views Applications

DATA step views can be readily incorporated into current applications. Views are well-suited for referencing infrequently accessed data, such as in the VTOC example described later in this paper. They can be used to access data that require manipulation of the raw data so as to restructure it for consumption by a SAS analytical component. DATA step views provide support for situations in which data independence, encapsulation, and isolation is required, and this can be accomplished through the view definition process or by the use of passwords assigned to a view during view creation. Examples described later in this paper also provide insight into types of applications in which views are well-suited.

Defining DATA Step Views

To create DATA step views, specify the VIEW= option in the DATA statement following all SAS data set names. The VIEW= option tells the SAS System to compile, but not to execute the the SAS

source program and to store the compiled code in the input or output DATA step view named in the option. A message is sent to the SAS log when the DATA step view is saved.

When you create a DATA step view from SAS source code, the DATA step compiler processes the DATA step through the compilation phase and then stores an intermediate code representation of the program in a SAS file. The DATA step program is not executed when defining the view. This SAS file appears in your SAS data library directory and has the member type VIEW.

The intermediate code is processed further when you reference the name of the view in a subsequent invocation of a DATA step or SAS procedure either in the current SAS session or within a SAS session at a later time. The SAS System resolves the intermediate code produced by the compiler and generates machine code for the host environment. The generated code is executed as a DATA step, or a SAS procedure requests observations from the view.

Syntax for Input DATA Step Views

```
DATA output-SAS-data-set-names / VIEW=SAS-data-view-name;
/* SAS DATA step statements */
RUN;
```

You must include a DATA statement that includes the following argument:

output-SAS-data-set-names
specifies a valid SAS name for each output SAS data file and the input DATA step view name to be created by the DATA step. The name can be a one-level name or a two-level name. You can specify more than one data set name in the DATA statement; one and only one of the names must refer to the input SAS DATA step view. For more information about naming SAS data sets, refer to "SAS Data Sets," in Chapter 6 in *SAS Language: Reference*.

You also must include the VIEW= option after a slash and specify the following value:

SAS-data-view-name
specifies which data set you want to store as the DATA step view. The name of the SAS data view that follows the /VIEW= specification must match one of the SAS data set names listed on the DATA statement. If the DATA statement includes data set names other than the view name, those data sets are not created until the view is processed. For more information about naming SAS data sets, refer to "SAS Data Sets," in Chapter 6 in *SAS Language: Reference*.

The following example creates an input DATA step view named A:

```
DATA A / VIEW=A;
/* SAS DATA step statements */
RUN;
```

Additional data set names specified in the DATA statement are not created until the DATA step view is referenced in another DATA step or PROC step.

Syntax for Output DATA Step Views

```
DATA ... / VIEW=SAS-data-view-name;
/* SAS DATA step statements */
SET/MERGE/UPDATE/MODIFY
input-SAS-data-set-names;
/* more SAS DATA step statements */
RUN;
```

You must include a SET, MERGE, UPDATE, or MODIFY statement that includes the following argument:

input-SAS-data-set-names
specifies a valid SAS name for each input SAS data file and the output DATA step view name to be created by the DATA step. The name can be a one-level name or a two-level name. You can specify more than one data set name in a SET, MERGE, UPDATE, or MODIFY statement; one and only one of the names must refer to the output SAS DATA step view. In the case of the MODIFY statement only the transaction data set can reference a view, the master data set is opened for update access. For more information about naming SAS data sets, refer to "SAS Data Sets," in Chapter 6 in *SAS Language: Reference*.

You also must include the VIEW= option after a slash and specify the following value:

SAS-data-view-name
specifies which data set you want to store as the DATA step view. The name of the SAS data view that follows the /VIEW= specification must match one of the SAS data set names listed in a SET, MERGE, UPDATE, or MODIFY statement. If this statement includes data set names other than the view name, those data sets will be used when the output DATA step view is referenced. For more information about naming SAS data sets, refer to "SAS Data Sets," in Chapter 6 in *SAS Language: Reference*.

The following example creates an output DATA step view named A.

```
DATA _NULL_ / VIEW=A;
/* SAS DATA step statements */
SET/MERGE/UPDATE/MODIFY A ...;
/* More SAS DATA step statements */
RUN;
```

Caution: SAS data files and SAS data views in the same library cannot have the same name. The SAS System prevents you from giving the same name to SAS data views and SAS data files in the same library.

Note: If you try to specify a SAS data view name and a SAS data view of that name already exists in the data library, it will be overwritten with the new view definition.

Restrictions and Requirements

You can create only one DATA step view in a single DATA step compilation. Once the view has been created, you can use it as input or output to any SAS procedure or DATA step.

A DATA step that generates a DATA step view can include all SAS statements and other components of the SAS language that are available to the DATA step except the following:

- global statements such as the FILENAME, FOOTNOTE, LIBNAME, OPTIONS, and TITLE statements. If you do include global statements in your source program statements, the SAS System stores the DATA step view but not the global statements. In this case, when the view is referenced, actual execution may differ from the intended execution.
- both engine and host-specific data set options.

- host-specific options in the FILE and INFILE statements. Errors are sent to the SAS log for host-specific data set and FILE/INFILE options, but not for global statements. (However, MVS supports the storage and retrieval of host-specific FILE/INFILE options used by the view.)

Remember that when using the MACRO facility in a view definition, the MACRO facility resolution and expansion is performed before compilation time, and that if you are expecting resolution and expansion of MACRO to occur at execution time, then the DATA step program will need to be modified to accomplish that task (for example, use SYMGET, SYMPUT, RESOLVE functions, or the CALL EXECUTE subroutine).

Caution: Save your source code.

The SAS System does not save the source code as part of the view, and it cannot retrieve source code from a stored DATA step view. You need the original source code if you attempt to move the application to another operating environment, if you install a new version of the SAS System, or if you need to regenerate the view for some other reason.

Using DATA Step Views in SAS Processing

You can specify input DATA step views in any DATA or PROC step that requires input data sets, and you can specify output DATA step views in any DATA or PROC step that generates output data sets.

DATA step views can be copied, renamed, or deleted using the DATASETS procedure and the LIBNAME and DIR windows in the SAS Display Manager System. You cannot move DATA step views to another host with an incompatible machine architecture.

To confirm that a particular SAS data set is a DATA step view, run the CONTENTS procedure to list the name of the engine. If the view is a DATA step view, the SASDSV engine and the member type of VIEW will be displayed in the procedure output. In addition, the DATA step view type will be displayed.

```
PROC CONTENT DATA=VIEW;
RUN;
```

When processed, an input view may produce additional output data files. For example, the following statements define a view, B, that will create an output SAS data file, A, when view B is processed.

```
DATA A B / VIEW=B;
/* SAS DATA step statements */
RUN;
```

To illustrate this point, when view B is processed in the following statements, the PRINT procedure lists the observations retrieved from the view B, and the stored DATA step statements create and output new observations to the SAS data set A.

```
PROC PRINT DATA=B;
RUN;
```

DATA Step Language Extensions to Support Views

The OUTPUT statement has been modified to allow applications to return specific codes using the RC= options by which input DATA step views can control the behavior of other DATA steps or procedures reading data through the input DATA step view.

```
DATA IN/VIEW-IN;
RETAIN COUNT 0;
INFILE DEVICE;
INPUT SAMPLE GAPX GAPR;
COUNT = COUNT + 1;
IF COUNT = 10 THEN DO;
OUTPUT IN RC=$SYSRC(_SWPAUAC);
COUNT = 0;
```

```
END;
RUN;

PROC TSHEWHART HISTORY-IN;
XCHART GAP=SAMPLE='*';
RUN;
```

When the input DATA step view has sent 10 samples, it will return a specific code to the consuming procedure TSHEWHART to signal it to perform a set of specific operations. This set of operations might include recalculating new information based on the previous and the latest set of samples, causing a redrawing graphic display information, and so on. The RC= option can be a numeric constant, using the %SYSRC autocall macro for symbolic substitution or the name of a variable. A set of procedures will be modified to use this feature in the future.

The SHEWHART procedure is currently being modified and tested for use with input DATA step views and the OUTPUT RC= option. In the future this procedure and functional support will be released as an experimental procedure TSHEWHART.

The set of return codes and procedures that would use this functionality has not been completely determined at this time. If you would like to assist in identifying this set of procedures and in what circumstances this functionality could be used, please send information to me at SAS Institute detailing your needs and requirements.

PERFORMANCE CONSIDERATIONS OF DATA STEP VIEWS

Using DATA step views can improve programming and application development efficiency with the potential of decreasing or increasing machine resource requirements with respect to the replaced methods of data processing. The impact on machine resources is determined by the access pattern of the consuming task. The consuming task can request the retrieval of data in two ways, a single pass or two or more passes.

When the one-pass access pattern is requested, there is no need for a data set to be created as with traditional methods of processing. The performance of this method is readily apparent with a decrease in I/O operations and elapsed time. When the two- or more-pass access pattern is requested, then the view must build a spill file that contains all generated observations so that subsequent passes will be able to read the same data as in previous passes. An optimization of this access pattern is performed whenever the consuming task just needs to access the data across BY groups, and then the DATA step view engine reuses space within the spill file whenever the BY groups change. With this optimization, the amount of disk space required is not the cumulative size of all data observations generated by the view but the cumulative size of the largest BY group generated.

Either access pattern incurs a certain overhead in relation to CPU and memory requirements. As a general rule CPU time does increase by a factor 10% to 20%. This increase is due to an internal host supervisor requirement and it is currently being investigated and will be addressed in a future release. As for memory utilization, if a DATA step view is referenced within a DATA step program, the additional memory overhead incurred is associated with additional storage required to execute the DATA step view. On the other hand, when a PROC step references a DATA step view, the additional memory overhead incurred is associated to the DATA step environment that is required to execute the view.

It is of critical importance, to ensure optimal performance characteristics, that when using DATA step views the system option STIMER

should be disabled. The CPU overhead consumed by this option can increase CPU usage by a factor of more than 60%.

VIEW EXAMPLES

In this section we examine four detailed examples using views: combined use of INPUT and OUTPUT view processing, VSAM processing, PDS directory processing, and VTOC processing. These examples focus primarily on the OS environment under MVS, but represent the types of applications that could be realized on other operating system environments. DATA step views can be defined and executed on all operating systems that Release 6.07 of the SAS System is supported on. Each example is structured to include the creation of data, if required, the definition of the view, and the execution of the view. Permanent views are created in each example, demonstrated using separate job steps by storing and referencing the views from a SAS data library. Each view example will have to be modified to suit the requirements of your site if you wish to execute the following examples. For brevity the actual output that is generated by these examples is not shown.

INPUT and OUTPUT DATA Step View Example

The following example demonstrates how to generate an input and an output DATA step view and shows how both are referenced. The input DATA step view reads data from an external file using the INFILE and INPUT statements. The data that are output by this view are passed to the SORT procedure to be sorted by the variable PART. When the INFILE statement has been completely read, the SORT procedure sorts the data and then writes each of the sorted observations to the output DATA step view.

```
//INPUTD EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
DATA SAVE.IN/VIEW=SAVE.IN;
INFILE INDATA;
INPUT PART $ N COST;
RUN;
```

The output DATA step view then generates a report based on the sorted data, along with an additional data set named REORDER, which contains observations with the values of PART that need to be reordered.

```
//OUTPUTD EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
DATA REORDER/VIEW=SAVE.OUT;
SET SAVE.OUT;
BY PART;
IF N = 0 THEN OUTPUT REORDER;
/* Print a report */
RUN;
```

With the invocation of the SORT procedure, three different steps are executing simultaneously, the SORT procedure and two independent executions of the DATA step. This example could be expanded further by replacing the REORDER data set with an output DATA step view, which could generate and transmit order information directly.

```
//IOX EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
PROC SORT DATA=SAVE.IN OUT=SAVE.OUT;
BY PART;
RUN;
```

VSAM View Example

VSAM (Virtual Storage Access Method) is a common file structure available under the CMS, OS, and VSE operating systems. VSAM is an IBM data access method that provides three ways to organize records in a disk file: ESDS (Entry-Sequenced Data Set), KSDS (Key-Sequenced Data Set), and RRDS (Relative-Record Data Set).

In addition the records in VSAM files can be accessed three different ways: sequential, direct, or skip sequential. For more information about VSAM processing, refer to the *SAS Guide to VSAM Processing*. This example is broken into three parts: the creation and loading of the sample VSAM disk file, the definition of the view to process the sample data, and an example to execution a view upon the sample data. Begin by first creating the VSAM sample data file using the IDCAMS utility, defining the location of the disk file, file organization, space requirements, and data and index names.

```
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
DELETE (USERNAME.VSAM.ZOO) PURGE CLUSTER
DEFINE CLUSTER -
(NAME(USERNAME.VSAM.ZOO)-
INDEXED -
VOLUME(VOLUME) -
TRACKS(2) -
KEYS(4 11) -
RECORDSIZE(60 120)-
NONSPANNED)-
DATA -
(NAME(USERNAME.VSAM.ZOO.DATA))-
INDEX-
(NAME(USERNAME.VSAM.ZOO.INDEX))
```

Next load the empty VSAM disk file with sample data for subsequent use.

```
//KSDSL EXEC SAS607
//VSAM DD DISP=SHR,DSN=USERNAME.VSAM.ZOO
/*
** LOAD DATA INTO VSAM KSDS
*/
DATA _NULL_;
INPUT IDNUM $ GENUS $ SPECIES $ SEX $ ZOONAME $ $20.
BIRTHDAY $ COMMON $ $20. MATEDATE $;
NEWANIX + 1;
TODAY=DATE();
FILE VSAM VSAM;
PUT
a1 TODAY DATE7.
a9 NEWANIX 2.0
a12 IDNUM $CHAR4.
a17 GENUS $CHAR13.
a31 SPECIES $CHAR13.
a45 SEX $CHAR1.
a47 ZOONAME $CHAR20.
a68 BIRTHDAY $CHAR5.
a74 COMMON $CHAR20.
a95 MATEDATE $CHAR7.;
```

```
CARDS;
1015 GENUS1 SPECIES1 F BUGS1 NOV83 COMMON RABBIT 0000000
1016 GENUS1 SPECIES1 M BUGS2 DEC83 COMMON RABBIT 0000000
1017 GENUS1 SPECIES1 F BUGS3 DEC83 COMMON RABBIT 0000000
1018 GENUS1 SPECIES1 M BUGS4 DEC83 COMMON RABBIT 0000000
1019 GENUS1 SPECIES1 F BUGS5 DEC83 COMMON RABBIT 0000000
1020 GENUS1 SPECIES1 M BUGS6 DEC83 COMMON RABBIT 0000000
1021 GENUS1 SPECIES1 F BUGS7 DEC83 COMMON RABBIT 0000000
1022 GENUS1 SPECIES1 M BUGS8 DEC83 COMMON RABBIT 0000000
1023 GENUS1 SPECIES1 F BUGS9 DEC83 COMMON RABBIT 0000000
1024 GENUS1 SPECIES1 M BUGS10 DEC83 COMMON RABBIT 0000000
2002 GENUS2 SPECIES2 M ROVER1 AUG80 DOG 04JUL82
2004 GENUS2 SPECIES2 M ROVER3 AUG80 DOG 04JUL82
2006 GENUS2 SPECIES2 M ROVER5 AUG80 DOG 04JUL82
2007 GENUS2 SPECIES2 F ROVER6 AUG80 DOG 04JUL82
2008 GENUS2 SPECIES2 M ROVER7 AUG80 DOG 04JUL82
2009 GENUS2 SPECIES2 F ROVER8 AUG80 DOG 04JUL82
2010 GENUS2 SPECIES2 M ROVER9 AUG80 DOG 04JUL82
2011 GENUS2 SPECIES2 F ROVER10 AUG80 DOG 04JUL82
3011 GENUS3 SPECIES3 M BILLY1 FEB79 GOAT 06DEC81
3012 GENUS3 SPECIES3 M BILLY2 FEB79 GOAT 06DEC81
3013 GENUS3 SPECIES3 M BILLY3 FEB79 GOAT 06DEC81
3014 GENUS3 SPECIES3 M BILLY4 FEB79 GOAT 06DEC81
3015 GENUS3 SPECIES3 M BILLY5 FEB79 GOAT 06DEC81
3016 GENUS3 SPECIES3 M BILLY6 FEB79 GOAT 06DEC81
3017 GENUS3 SPECIES3 M BILLY7 FEB79 GOAT 06DEC81
3018 GENUS3 SPECIES3 M BILLY8 FEB79 GOAT 06DEC81
3019 GENUS3 SPECIES3 M BILLY9 FEB79 GOAT 06DEC81
3020 GENUS3 SPECIES3 M BILLY10 FEB79 GOAT 06DEC81
;;;
```

Define the view over the sample VSAM record structure. Note that the view definition is being performed in a separate JOB step from that of the use of the view.

```
//KSDSD EXEC SAS607
//VSAM DD DISP=SHR,DSN=USERNAME.VSAM.ZOO
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
/*
** CREATE DATASET VIEW OVER VSAM KSDS
*/
DATA SAVE.VSAM / VIEW=SAVE.VSAM;
INFILE VSAM VSAM;
INPUT
  01 TODAY DATE7.
  09 NEWANIM 2.0
  012 IDNUM $CHAR4.
  017 GENUS $CHAR13.
  031 SPECIES $CHAR13.
  047 ZOONAME $CHAR20.
  068 BIRTHDAY $CHAR5.
  074 COMMON $CHAR20.
  095 MATEDATE $CHAR7.;
FORMAT TODAY DATE7.;
RUN;
```

Process the sample data using the view definition as input to a SAS procedure, as in the case with PROC PRINT.

```
//KSDSX EXEC SAS607
//VSAM DD DISP=SHR,DSN=USERNAME.VSAM.ZOO
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
/*
** PRINT DATA MATERIALIZED BY DATASET VIEW FROM VSAM KSDS
*/
PROC PRINT DATA=SAVE.VSAM UNIFORM PRETTY;
RUN;
```

PDS View Example

The PDS (Partitioned Data Set) example demonstrates the use of views to read the directory structure of an MVS PDS. A PDS directory structure describes the members contained within the PDS along with the information to access the data for each member. This construction is very similar to that of a directory structure, which is found on more recent operating system environments, with the exception that PDS's can not be nested. Create the view that will traverse through the directory member entries for any referenced PDS. The structural details within the PDS is left as an exercise to the reader.

```
//PDSD EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
/*
** CREATE DATASET VIEW OVER PDS DIRECTORY
*/
DATA SAVE.PDS / VIEW=SAVE.PDS;
LENGTH DSNAME $44 LAST_MOD $14 JFCB $176;
RETAIN CC 3 LEN 0 EOY 1 DSNAME;
INFILE PDS JFCB=JFCB RECFM=F LRECL=256 BLKSIZE=256 EOY=EOY END=END;
IF CC > LEN THEN DO;
  INPUT 01 LEN PIB2. 00;
  CC = 3;
END;
IF EOY OR _L_=1 THEN
  DSNAME = SUBSTR (JFCB, 1, 44);
EOY = 0;
INPUT 0CC NAME $CHARS.
      TT PIB2.
      R PIB1.
      C $CHAR1. 00;
CC + 12;
IF NAME = '#####'X THEN DO;
  IF END THEN STOP;
  DO UNTIL (END OR EOY);
    INPUT;
    INPUT 00;
  END;
  CC = 256;
  IF END THEN INPUT;
  DELETE;
```

```
END;
IF C = '...01111'B THEN DO;
  INPUT 0CC VER PIB1.
      LEVEL PIB1.
      +2 C_DATE PD4.
      M_DATE PD4.
      M_HH PK1.
      M_MM PK1.
      SIZE PIB2.
      INIT PIB2.
      MOD PIB2.
      ID $CHAR7. 00;
  CC + 30;
  CREATED = DATEJUL (C_DATE);
  M_DATE = DATEJUL (M_DATE);
  M_TIME = HMS (M_HH, M_MM, 0);
  VER_MOD = VER + LEVEL / 100;
  LAST_MOD = PUT (M_DATE, YMMDD8.) || ' ' ||
    PUT (M_TIME, HHMM5.);
END;
IF CC > LEN THEN INPUT;
FORMAT VER_MOD 5.2 CREATED YMMDD8. SIZE INIT MOD 4. ;
KEEP NAME VER_MOD CREATED LAST_MOD SIZE INIT MOD ID DSNAME;
RUN;
```

After the view has been defined, reference the view, and be sure to point to a PDS that you wish to process with your favorite SAS analytical tool.

```
//PDSD EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
//PDS DD DISP=SHR,DSN=USERNAME.PDS
/*
** PRINT DATA MATERIALIZED BY DATASET VIEW FROM PDS DIRECTORY
*/
PROC PRINT DATA=SAVE.PDS UNIFORM PRETTY;
ID NAME;
RUN;
```

VTOC View Example

The last example demonstrates the use of a view to process a VTOC (Volume Table of Contents). On each volume within the disk organization at your site, there exists a VTOC describing the contents of the volume. This VTOC is similar to a PDS directory, but it contains information specific to organization, structure, space requirements and utilization, for all data files contained on a volume. The view definition reads each VTOC entry, checks for proper entry type, and then extracts and decodes information contained in each VTOC entry. Appropriate SAS variables are created detailing the information contained within the VTOC for later use with other SAS analytical tools.

```
//VTOCVIEW EXEC SAS607
//SAVE DD DISP=OLD,DSN=USERNAME.V607.SASLIB
/*
** CREATE DATASET VIEW OVER VTOC
*/
DATA SAVE.VTOC / VIEW=SAVE.VTOC;
KEEP VOLUME DSNAME DSORG RECFM LRECL BLKSIZE
  CREATED EXPIRES LASTMOD LASTREF;
LENGTH VOLUME $6 CCHHR $5;
FORMAT CCHHR $HEX10.
  CREATED EXPIRES LASTMOD LASTREF DATE7.;
INFILE VTOC VTOC CVAF VOLUME=VOLUME CCHHR=CCHHR COLUMN=COL;
INPUT 045 DSCBTYPE $CHAR1. 0;
IF DSCBTYPE='1' THEN DELETE;
INPUT
  001 DSNAME $CHAR4.
  046 SER $CHAR6.
  046 VM PIB1.
  DAYM PIB2.
  COUNT PIB2.
  IND PIB1.
  054 YC PIB1.
  DAYC PIB2.
  YE PIB1.
  DAYE PIB2.
  EXTENTS PIB1.
```

```

076 YR      PIB1.
      DAYR   PIB2.
083 DSORG1  PIB1.
      DSORG2 PIB1.
      RECFM0 PIB1.
087 BLKSIZE PIB2.
      LRECL  PIB2.
      KEYLNG PIB1.;
IF YC>0 THEN CREATED=DATEJUL(1000*YC+DAYC);
IF YE>0 THEN DO;
  YE=1000*YE+DAYE;
  IF YE=99366 THEN YE=99365;
  EXPIRES=DATEJUL(YE);
END;
IF YR>0 THEN LASTREF=DATEJUL(1000*YR+DAYR);
IF VERIFY(SER,'-$$9ABCDEFGHIJKLMN0PQRSTUVWXYZ0123456789') AND
  YR>0 AND 0<DAYR<367 AND (IND=00X | IND=40X)
  THEN LASTMOD=DATEJUL(1000*YR+DAYR); ELSE COUNT=.;
DSORG='';
IF MOD(INT(DSORG1/2),2) THEN DSORG='PO';
IF DSORG1>=32      THEN DSORG='DA';
IF DSORG1>=64      THEN DSORG='PS';
IF DSORG1>=128     THEN DSORG='IS';
IF DSORG2>= 8      THEN DSORG='VS';
RECFM='';
Z=INT(RECFM0/64);
IF Z=1 THEN RECFM='V';
IF Z=2 THEN RECFM='F';
IF Z=3 THEN RECFM='U';
IF MOD(INT(RECFM0/16),2) THEN SUBSTR(RECFM,2,1)='B';
IF MOD(INT(RECFM0/4),2) THEN SUBSTR(RECFM,3,1)='A';
IF MOD(INT(RECFM0/8),2) THEN SUBSTR(RECFM,4,1)='S';
RECFM=COMPRESS(RECFM);
RUN;

```

After the view has been defined, reference the view, and be sure to point to a VTOC volume that you wish to process with your favorite SAS analytical tool.

```

//VTOCVIEW EXEC SAS607
//SAVE      DD DISP=OLD,DSN=USERNAME.V607.SASLIB
//VTOC      DD DISP=SHR,UNIT=SYSDA,VOL=SER=VOLUME

```

```

/*
** PRINT DATA MATERIALIZED BY DATASTEP VIEW FROM VTOC
*/
PROC PRINT DATA=SAVE.VTOC UNIFORM PRETTY;
  ID DSNAME;
  VAR DSORG RECFM LRECL BLKSIZE CREATED LASTREF;
RUN;

```

ACKNOWLEDGMENTS

The author greatly appreciates the contributions by Annette Green for providing the discussion of DATA step views, the assistance from Susan O'Connor and Bruce Tindall for reviewing this paper, Robert Cross for the VSAM PDS and VTOC DATA step view examples, and the Publications Division for their work in producing the final paper.

The author can be contacted at:

Jeff Polzin
 SAS Institute Inc.
 SAS Campus Drive
 Cary, NC 27513
 Phone (919) 677-8000
 FAX (919) 677-8123

SAS, SAS/ACCESS, and SAS/ASSIST are registered trademarks of SAS Institute, Inc., Cary, NC, USA. *indicates USA registration.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

Other brand and product names are registered trademarks or trademarks of the respective companies.