The SAS Supervisor

Started 03-21-2018 | **Modified** 03-21-2018 | **Views** 5,914

Abstract

How SAS processes jobs is the responsibility of the ***SAS Supervisor*** and an understanding of its function is important.

While the details of how it works have changed over time, much of the basics of the ***SAS Supervisor*** have been reasonably consistent over time.

This article contains:

1. links to past SUGI papers on this topic
2. a version of the paper converted to a Communities Article

Links to past SUGI papers on this topic

- SUGI 1983
- SUGI 1987
- SUGI 1988
- SUGI 1990
- SUGI 1991
- SUGI 1992

A Version of the Paper Converted to a Communities Article

THE SAS SUPERVISOR

Don Henderson & Merry Rabb

This paper was originally presented many, many SUGIs ago, specifically, SUGI 83 in New Orleans. It was one of the presentations at the very first Tutorials section. It has been available c

INTRODUCTION

This tutorial discusses the functions of the SAS Supervisor during the execution of a SAS DATA Step program and is a repeat presentation of a paper given in the Tutorial and the Advance

- Compiling SAS Source Code, and
- Executing Resultant Machine Code

The actions of the Supervisor during both the compile and execution phases of a SAS job will be illustrated.

When a SAS DATA Step program is written, the DATA Step6 *module* must be integrated within the structure of the SAS System. This integration is done by the SAS Supervisor. Gaining a

STRUCTURE OF SAS JOBS

There are distinct compile and execute steps for all SAS jobs. This fact is not readily apparent since a single program, the SAS Supervisor, handles the compile and execution (including lin
the first DATA/PROC step is compiled and then executed; this is then followed by the compilation and the execution for the next DATA/PROC step, etc. The SAS Supervisor controls this p

The SAS programmer has tools that allows him or her to take full advantage of the compile/ execute structure for SAS jobs. For example, through the use of the Macro Language, the prog
Step, such as conditional execution of a read operation, or reading data within a loop. The following sections discuss the actions of the Supervisor during compilation and execution of a D

COMPILE TIME PROCESSING

During the compilation of a DATA Step, the Supervisor creates both permanent and transient (in that they *disappear* after the compilation or execution of the current DATA Step) entities. Th
desired output. The following is a partial list of the more important actions taken by the SAS Supervisor during the compilation of a DATA Step:

- Syntax scan;
- Translation from SAS source code to machine language object code;
- Definition of input and output files including variable names, their locations and attributes;
- Creation of the Program Data Vector;
- Specification of variables to be written to the output SAS data set;
- Specification of variables which are to be initialized to missing by the SAS Supervisor between executions of the DATA Step and during read operations; and
- Creation of a variety of *flag variables* which are used by the Supervisor at execution time.

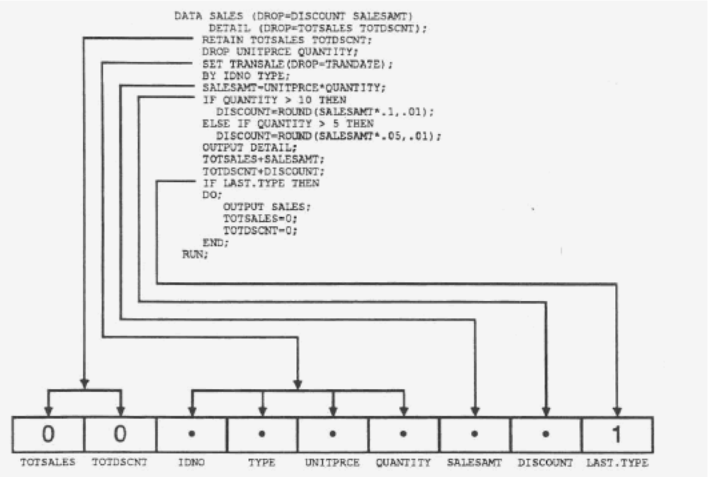
The last four actions in the above list will be discussed in the following subsections.

Creation of the Program Data Vector

The Program Data Vector (PDV) is a *logical* buffer which includes all variables referenced either explicitly or implicitly in the DATA Step, it is used at execution time as the location where th
of SAS source statements. The following rules are used in defining the variables and their attributes to the PDV:

1. A variable is added to the PDV by its first occurrence (explicit or implicit) in the SAS source statements.
2. DROP and KEEP statements and output data set name parameters are ignored for the purposes of adding variables to the PDV.
3. The SAS automatic variables (e.g., _N_ and _ERROR_ are always added). They are added as they are referenced or created in the DATA Step program.
4. Variables can be implicitly referenced and thus added to the PDV through SET, MERGE or UPDATE statements. Variables referenced in this way are added to the PDV when the *rea*

The use of these rules is illustrated for a sample program in Figure 1.



```
DATA SALES (DROP=DISCOUNT SALESAMT)
     DETAIL (DROP=TOTSALES TOTDSCNT);
    RETAIN TOTSALES TOTDSCNT;
    DROP UNITPRCE QUANTITY;
    SET TRANSALE(DROP=TRANDATE);
    BY IDNO TYPE;
    SALESAMT=UNITPRCE*QUANTITY;
    IF QUANTITY > 10 THEN
      DISCOUNT=ROUND(SALESAMT*.1,.01);
    ELSE IF QUANTITY > 5 THEN
      DISCOUNT=ROUND(SALESAMT*.05,.01);
    OUTPUT DETAIL;
    TOTSALES+SALESAMT;
    TOTDSCNT+DISCOUNT;
    IF LAST.TYPE THEN
    DO;
       OUTPUT SALES;
       TOTSALES=0;
       TOTDSCNT=0;
    END;
  RUN;
```

0	0	•	•	•	•	•	•	1
TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE

Note: _N_, _ERROR_, FIRST.IDNO, and FIRST.TYPE are in the PDV. They are not included in the figure since they are referenced by the program.

FIGURE 1. The Program Data Vector

Specification of Variables for Output

The specification of the list of variables to be copied from the PDV to the output SAS data set is best illustrated by another *logical* buffer called the DROP/KEEP Table (DKT) that has a one
values are supplied at compile time and can not be altered during the execution phase of a DATA Step. The Supervisor uses the following rules in setting DKT values:

1. For each variable in the PDV, set all of its DKT values to "D" if it is a SAS special variable (e-9- _N_, _ERROR_, END=, IN=, POINT=, FIRST, and LAST, variables, ar
2. DROP statement changes to DKT are made before KEEP statement changes.
3. For each variable in a DROP statement with its DKT value equal to "K," change it to "D." If the DROPped variable is not found, set an error condition. The error mess

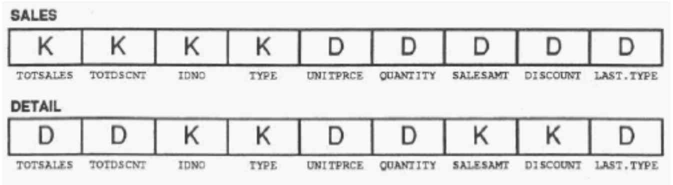
   THE VARIABLE <variable name> IN THE DROP LIST HAS NEVER BEEN REFERENCED.

   This message most often occurs when: a variable is listed in more than one DROP statement or more than once in a single DROP statement; a variable not in the PD
4. If any KEEP statements are present, then create a list of unique variable names from all KEEP statements. The Supervisor compares this list with variables in the PD
   from all KEEP statements, set the DKT value to "D"; for variables in the list compiled from all KEEP statements which do not match variables in the PDV with DKT eq

   THE VARIABLE <variable name> IN THE KEEP LIST HAS NEVER BEEN REFERENCED.

   This message most often occurs when: a variable is listed in a DROP statement and a KEEP statement; a variable which is not in the PDV at all is listed in a KEEP st
5. For each output data set with DROP or KEEP data set name parameters, change its row in the DKT as follows:
   1. Process DROP before KEEP.
   2. For each variable listed in the DROP list, set its DKT value to 'D'.
   3. For each variable not listed in the KEEP list, set its DKT value to 'D'.
   4. Ignore variables in the DROP or KEEP list that are not found in the PDV.

The use of these rules is illustrated in Figure 2.



SALES

K	K	K	K	D	D	D	D	D
TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE

DETAIL

D	D	K	K	D	D	K	K	D
TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE

FIGURE 2. The DROP/KEEP Table (DKT)

Initialization to Missing Values

The specification of the variables that are to be initialized to missing between every execution of the DATA Step program by the SAS Supervisor is also illustrated by a buffer with a one-to-

- Y means initialize to missing between each execution of the DATA Step.
- N means do not initialize to missing.
- R means that the read operation (i.e., SET, MERGE or UPDATE) will perform the initialization to missing values. This value is only used when multiple data sets are bei

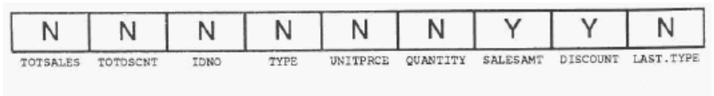
These values, like the values in the DKT, are defined at compile time and can not be changed at execution time. The ITMV values for all variables are initially set to "Y" and are changed tc

1. All SAS special variables.
2. All variables listed in a RETAIN statement (note that "RETAIN;" forces all variables to have ITMV set to "N").
3. All variables which are physically present as the accumulator variable (the variable to the left of the "+" sign) in a sum statement. An exception is when the variable r

All variables which are referenced in SET, MERGE or UPDATE statements will have ITMV values set to "N", or "R" according to the following rules:

1. ITMV values are set to "N" for variables read from a single SAS data set with the SET statement.
2. Where two or more data sets are read with a SET, MERGE or UPDATE statement, ITMV values for the variables from those data sets are set to "R".

These rules are illustrated in Figure 3.

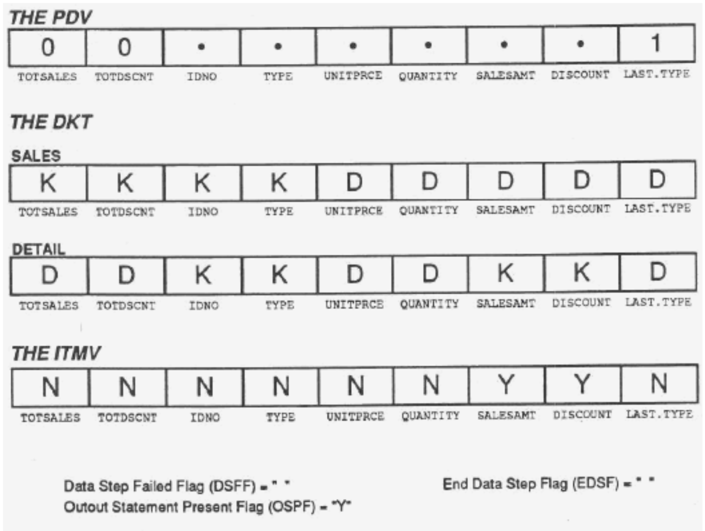


TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE
N	N	N	N	N	N	Y	Y	N

FIGURE 3. The Initialize To Missing Vector (ITMV)

Process Control Flags

In addition to the above buffers or vectors, other flag variables are created during the compile phase of a DATA Step program. The Data Step Failed Flag (DSFF) and the End Data Step Fl
compile time. OSPF is set to "Y" if there is any output statement present in the DATA Step program, otherwise it is set to "N." The DSFF, EDSF and OSPF are all used by the SAS Supervi
the compile phase are illustrated in Figure 4.



THE PDV

TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE
0	0	•	•	•	•	•	•	1

THE DKT

SALES

TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE
K	K	K	K	D	D	D	D	D

DETAIL

TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE
D	D	K	K	D	D	K	K	D

THE ITMV

TOTSALES	TOTDSCNT	IDNO	TYPE	UNITPRCE	QUANTITY	SALESAMT	DISCOUNT	LAST.TYPE
N	N	N	N	N	N	Y	Y	N

Data Step Failed Flag (DSFF) = " "
End Data Step Flag (EDSF) = " "
Outout Statement Present Flag (OSPF) = "Y"

FIGURE 4. Summary of Compile Time Buffers and Flags

Other Compile Time Operations

It should be noted that the above represents only a subset of the SAS Supervisor compile time functions. All of the following statements (*non- executable* or *information* statements) do all

- ARRAY
- ATTRIB
- BY
- DROP
- FORMAT/INFORMAT
- KEEP
- LABEL
- LENGTH
- RENAME
- RETAIN

Because these statements have their effect at compile time, their location within the DATA Step code is irrelevant; they may be placed at the beginning, at the end, or anywhere within the
in mind when writing and debugging SAS programs.

EXECUTION TIME ACTIVITIES

Once the DATA Step has been successfully compiled and all of the above described buffers and flags have been created, the execution phase of the DATA Step can begin. This is illustrat

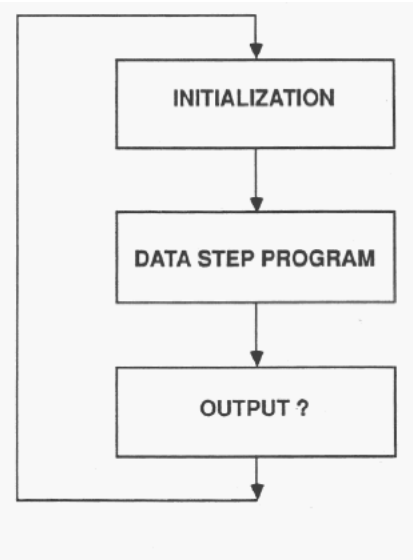


FIGURE 5. Execution Time Flow

The SAS DATA Step can be viewed as a subroutine which is executed repeatedly by the SAS Supervisor, usually until there is no more input data. In a typical SAS job, the Supervisor doe

1. Initialization of variables in the PDV to missing.
2. Execution ("calling") of the DATA Step program.
3. 0utputting or copying values of variables in the PDV to the output SAS data set.
4. Repeating steps 1-3 until the input data source is exhausted.

The details of what happens during the execution of the DATA Step program (step 2 above) is controlled by the user in their SAS code. The details of how the Supervisor performs steps 1,
DATA Step are geared towards one goal: the repeated execution of a DATA Step program. In other words, the DATA Step program can be viewed as the inside of a read-write loop.

Execution Time Program Flow

The SAS Supervisor performs initialization before every execution of our DATA Step program using the PDV and the ITMV as follows:

- For each variable in the PDV with its corresponding ITMV = "Y," the SAS Supervisor will set its position in the PDV to missing ('.' for numeric, *for character)*.

The DATA Step program is then executed (called). The programming statements that comprise the DATA Step are executed, supplying values for the variables in the PDV.

Once the DATA Step program has finished, control is returned to the SAS Supervisor which decides whether to copy the contents of the PDV to the output SAS data set. The OSPF, DSFF

- If OSPF = "N" and DSFF ="N" then execute the OUTPUT routine.

The OUTPUT routine, which is also invoked when an OUTPUT statement is executed from within the DATA Step program, can be described as follows:

- For each variable in the PDV with its corresponding DKT="K", copy its current value from the PDV to the output SAS data set.

The value for the OSPF is set at compile time. Values for DSFF and EDSF are set at execution time. The setting of these flags is discussed in the following paragraphs which also address

On referring to Figure 5, the question arises as to how the SAS Supervisor knows when to stop executing the DATA Step program. The more detailed flow diagram given in Figure 6 is a m

1. During the INITIALIZATION phase, set the values of DSFF and EDSF to "N".
2. Execute the DATA Step program, statement by statement.
    1. When executing the read operation (for this "generic" case, assume a simple SET or INPUT statement) call a Supervisor routine to:
        1. Determine if there is more input data.
        2. If no more data, set DSFF and EDSF to "Y" and skip the rest of the DATA Step program, returning control to the SAS Supervisor.
        3. Otherwise, copy the variables from the input data set to the PDV, set the values of any appropriate special variables and return control to the next exe
3. If OSPF="N" and DSFF="N" then execute the OUTPUT Routine.
4. If EDSF="Y" then end the DATA Step and proceed to the next DATA or PROC step. Otherwise, repeat the above steps.

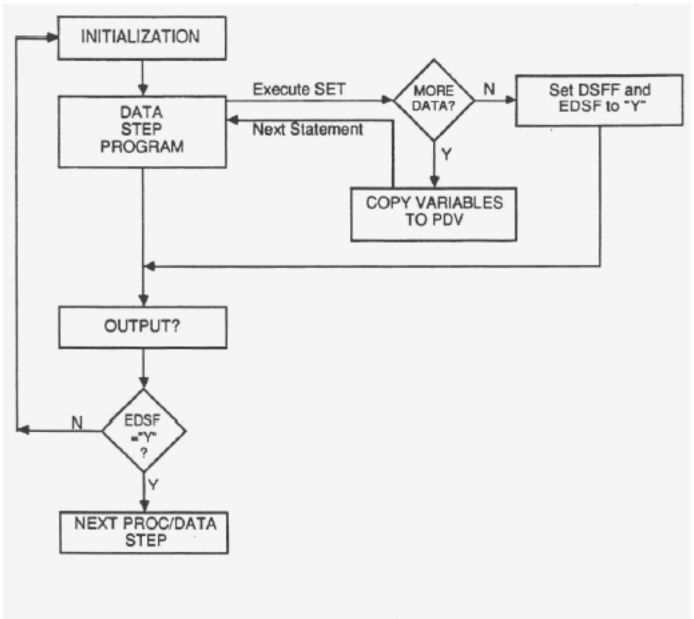


FIGURE 6. Detailed Execution Time Flow

In writing a SAS DATA Step program, it is crucial to keep in mind the statements that return control to the Supervisor and how they impact the values of the DSFF and EDSF flags. Executi

Statement	DSFF	EDSF
ABORT	Y	Y
DELETE	Y	N
IF false <expression>	Y	N
RETURN	N	N
STOP	Y	Y
Failed read operation(i.e. INPUT, SET, MERGE or UPDATE)	Y	Y

On reviewing the above table and the rules for the SAS Supervisor OUTPUT, it is clear that when OSPF="Y", DELETE, a false subsetting IF <expression> and RETURN are equivalent, si
DATA Step execution.

Read Operation Details

The SAS read operations SET and MERGE perform two general actions when executed:

- Call a SAS Supervisor routine to initialize selected variables in the PDV to missing.
- Copy variable values from one or more SAS data sets to the PDV.

These actions are performed according to a set of rules, depending on which type of read operation is being performed and whether or not a BY statement is present. The rules for each ty

When a SET statement references more than one SAS data set, and no BY statement is present, the data sets listed on the SET statement are concatenated. The SET statement perform

1. Determine which data set is being read and set IN= and END= variable values.
2. If the SET statement will read from a different data set compared to its last execution, then initialize all variables in the PDV with ITMV values of "R" to missing.
3. Copy the values of variables from the current data set to the PDV.

When a SET statement referencing more than one SAS data set has a BY statement associated with it, the data sets listed on the SET statement are interleaved. The SET statement perfo

1. Determine which data set is being read by looking ahead to the values of the variables in the BY statement for the next observation in each data set. Set values for I
2. If the observation to be read is the first observation for a new BY group, then do the following:
    1. Set the appropriate FIRST, variables to 1.
    2. Set all variables in the PDV with ITMV values of "R" to missing.
3. If the SET statement will read from a different data set compared to its last execution, regardless of whether the BY group changes, then initialize all variables in the
4. Copy variable values to the PDV from the current data set.

5. Look ahead to the values of the variables in the BY statement for the next observation in each data set. If there are no more observations for this BY group then set

When a MERGE statement with no BY statement is present, the observations in the data sets listed on the MERGE statement are merged one-to-one. The MERGE statement performs th

1. Copy variables values to the PDV from next observation in the first data set listed on the MERGE statement, then the second data set, and so on until all data sets h
2. If end-of-file has been reached for a data set and no observation is read, initialize variables unique to that data set to missing.
3. Set IN= variables depending on which data sets are read.

When a MERGE statement with a BY statement is executed, the observations in the data sets listed are merged according to the values of the variables on the BY statement.

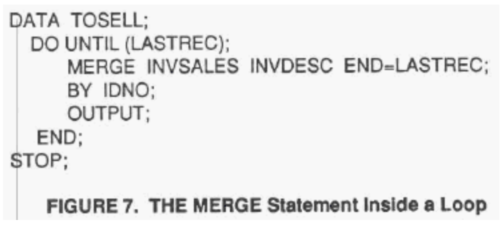
The MERGE statement performs the following actions when executed:

1. Determine which data sets are being read by looking ahead to the values of the variables in the BY statement for the next observation in each data set.
2. If the observation(s) to be read represent a new BY group, then do the following:
    1. Set the appropriate FIRST, variables to 1.
    2. Set all of the IN= variables to 0.
    3. Set all variables with ITMV values of "R" to missing.
3. For each data set listed on the MERGE statement having another observation for this BY group, do the following:
    1. Set the appropriate IN= variable to 1.
    2. Copy variable values from the data set to the PDV.
4. Look ahead to the next observation in each data set to determine if any more observations are present for this BY group. If not, set the appropriate LAST, variable va

Understanding these rules can be helpful when writing or debugging a SAS program with complex DATA Step code. The following sections discuss how the SAS programmer can take cor

Reading SAS Data Sets Within A Loop

SAS programmers can take control from the SAS Supervisor within a DATA Step by placing the read operation statement inside a DO loop. When the SAS Supervisor does the looping, th efficiency of the DATA Step. There might be other reasons for reading data within a loop, such as when a data set must be searched to find a specific observation.



```
DATA TOSELL;
  DO UNTIL (LASTREC);
      MERGE INVSALES INVDESC END=LASTREC;
      BY IDNO;
      OUTPUT;
   END;
STOP;
```

FIGURE 7. THE MERGE Statement Inside a Loop

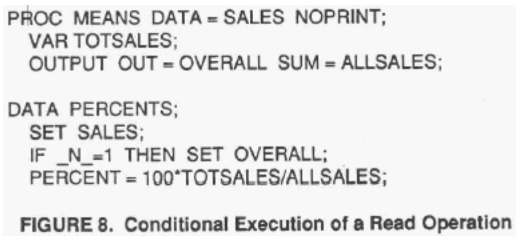
The program shown in Figure 7 merges two SAS data sets in one execution of the DATA Step. Because control is not returned to the Supervisor each time, an OUTPUT statement must be Step, thus the value of _N_ in the PDV will be equal to 1 for every observation processed.

Looping also has an effect on how variables in the PDV are initialized to missing. Variables with ITMV values of "Y" are only initialized to missing when program control is returned to the S

It is important to remember which statements automatically return control to the Supervisor when executed. These statements should not be placed within the DO loop where the data is be

Conditional Execution Of A Read Operation

Any SAS read operation can be executed conditionally. For example, suppose the programmer has a SAS data set with a single observation that contains a constant. This constant value i



```
PROC MEANS DATA = SALES NOPRINT;
  VAR TOTSALES;
  OUTPUT OUT = OVERALL SUM = ALLSALES;

DATA PERCENTS;
  SET SALES;
  IF _N_=1 THEN SET OVERALL;
  PERCENT = 100*TOTSALES/ALLSALES;
```

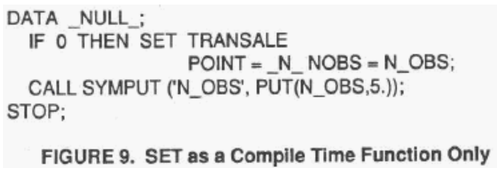
FIGURE 8. Conditional Execution of a Read Operation

Since variables read from a SET statement referencing a single SAS data set have ITMV values set to "N", the constant will not be initialized to missing on subsequent executions of the D

Referencing A Data Set At Compile Time

Any SAS data set may be referenced at compile time only. For example, suppose the programmer wants to add variables to the PDV, even though those variables will not be read in this D will ever be read since 0 is false. This can be a useful technique for creating a "shell" of a SAS data set, where values are to be added later using FSEDIT or UPDATE.

Using the SET statement at compile time only can also be used in conjunction with the NOBS and POINT options.



```
DATA _NULL_;
  IF 0 THEN SET TRANSALE
                POINT = _N_ NOBS = N_OBS;
  CALL SYMPUT ('N_OBS', PUT(N_OBS,5.));
STOP;
```

FIGURE 9. SET as a Compile Time Function Only

The program in Figure 9 sets a macro variable whose value is the number of observations in data set INVDESC. The SET statement is never executed because the "IF 0" condition is neve

CONCLUSION

These examples illustrate that by gaining a more complete understanding of what the SAS Supervisor is doing at both compile and execution time, SAS programmers can make more infor

Feel free to contact @DonH if you have questions.