A Comparison of Data Step Performance in Versions 6.06 and 5.18
Jim DeFoor, General Dynamics — Fort Worth

INTRODUCTION

Many changes to the operations of the SAS Data Step have been introduced with Version 6.06. For one, the SAS Supervisor, which determines the overall processing of a SAS job, has dropped its control over the output, initialization-to-missing (ITM), and looping actions of the SAS Data Step. These have now been passed to execution phase of the Data Step.

In addition, the I/O processing and data storage methods of the SAS system have been changed. The SAS programmer can now improve the I/O operations of his program by changing the buffer numbers and buffer size established for the read and output actions of each Data Step. The SAS programmer can also reduce storage waste in small SAS data sets through effective control of the page size of those data sets.

Finally, new data subsetting operations have been added. The SAS programmer can now select data in the Data Step using the IN operator or the WHERE statement. The WHERE statement can also be used to subset data being read into a procedure.

These changes and additions were discussed in the 1990 and 1989 SUGI Proceedings. They are also described in various SAS publications. References are given at the end of this paper.

This paper summarizes these changes and additions of Version 6.06 and investigates their impact on the performance of the SAS Data Step. It has three major topics:

Changes to the SAS Data Step
Optimizing SAS I/O Operations and Data Storage
Selectively Subsetting Data

Various tests were established that compared the input, output, and subsetting capabilities of the SAS Data Step under both SAS versions and under different buffer sizes and buffer numbers.

The tests were run in batch on a MVS/370 system using sequential Data Steps that read 140,195 observations with a record length of 30 from SAS data libraries stored on 3350 devices. The tests were evaluated using the CPU and EXCP statistics printed at the bottom of each step in the SAS Log.

Version 5.18 does not show EXCP statistics, so the EXCP counts were used only to evaluate the I/O impact under Version 6.06 of using different buffer sizes, buffer numbers, and physical block sizes.

Due to space limitations, the CPU and EXCP statistics of each test cannot be printed in this paper, but are available from the author upon request.

CHANGES TO THE SAS DATA STEP

In the SAS Supervisor — A Version 6.06 Update, The authors Henderson, Rabb, and Polzin summarized the changes to the Supervisor:

Under Version 5, a variety of work areas, buffers, and flags were created and used to control execution time processing. Under Version 6, much of the work that was previously done at execution time is handled through the generation of machine code specific to the functions being performed. In this way, performance is improved since the logical evaluations are performed at compile time once, and not at execution time for every input observation. The structure of internal work areas has also changed substantively.

The authors explained the changes to the Data Step in this way:

- Control is no longer returned to the SAS Supervisor during the looping process of the SAS Data Step unless a LIST statement is issued or errors occur.
- The initialization to missing (ITM) control has been moved from the SAS Supervisor. It is now a callable function placed in the compiled code of the SAS Data Step. It is located at the top of the SAS-constructed data loop and also where a loop-terminating statement is encountered. Loop-terminators are Subsetting IF, RETURN, or DELETE statements.
- The Program Data Vector (PDV) has been divided into retained and unretained data areas. Version 5.18 checked each variable of the PDV during each execution of the loop of the Data Step to determine which should have their values retained. Version 6.06 uses the ITM control to retain only the variables in the unretained data areas of the PDV.
- The ITM control is not called at the top of the Data loop if the SET statement is placed within a DO UNTIL loop or a RETAIN;** statement is coded.
- Version 5.18 checked whether a default OUTPUT routine needed to be performed at the end of the Data loop, regardless of whether an explicit OUTPUT statement had been coded in the step.
- Version 6.06 places a default OUTPUT statement in the actual compiled code of the SAS Data step when no OUTPUT statement is coded for the step. During the execution phase, Version 6.06 calls the OUTPUT routine whenever an OUTPUT statement is encountered. For that reason, Version 6.06 need not perform any OUTPUT flag checking.

The new execution-time looping process of the normal version 6.06 Data Step can be described as:

Initialize unretained variables to missing
Stop if end of file Read observation into Program Data Vector Execute statements in Data Step Output observations to SAS Data set

In Version 5.18, the normal looping process was more like:

Initialize unretained variables to missing
Check all variables regarding initialization Stop if end of file Read observation into Program Data Vector Execute statements in Data Step Check whether a default OUTPUT should be performed
Output observations to SAS data set

Note that the boxed actions do not occur in Version 6.06.
In Version 5.18, performance improvements could be achieved by placing the \texttt{SET} or \texttt{INPUT} statement, execution statements, and explicit \texttt{OUTPUT} statements inside of a DO UNTIL(EOF) loop.

\texttt{DATA WORK;}
\texttt{DO UNTIL(EOF);}
\texttt{SET WORK1 END=EOF;}
\texttt{OUTPUT;}
\texttt{END;}
\texttt{STOP;}
\texttt{RUN;}

The result was a looping process that did no initialization checking, no initialization, and no output checking. It performed only these actions.

- Stop if end of file
- Read observation into Program Data Vector
- Stop if EOF = 1
- Execute statements in Data Step
- Output observations to SAS data set

Version 6.06 eliminated the initialization checking and the default output checking in the normal SAS Data Step. Performance gains are achievable only through the prevention of unnecessary ITM calls. Placing all executable statements inside of a DO UNTIL(EOF) loop prevents ITM calls at the top of the data loop, but will not the use of a 'RETAIN;' statement.

If a 'RETAIN;' statement is used in Version 6.06, the resulting data loop is:

- Stop if end of file
- Read observation into Program Data Vector
- Execute statements in Data Step
- Output observations to SAS data set

The 'RETAIN;' statement approach is better than the DO UNTIL(EOF) approach because it eliminates initialization without additional EOF checking, shown in the box above.

\textbf{Data Step Test Results}

- Formatting input data (INPUT @12 CONTRACT $5; ) in Version 6.06 is noticeably less efficient than Version 5.18. According to Jeff Polzin of SAS Institute, the additional processing occurs because the formatting process is now done with C code. In Version 5.18, it was performed by \texttt{ASSEMBLER} code. C code was used in Version 6.06 because it enhanced the portability of the SAS system.

- Reading data inside of a DO UNTIL(EOF) loop in Version 6.06 is no longer more efficient than reading data inside of a normal SAS Data Step, provided a 'RETAIN;' statement is used. The 'RETAIN;' statement prevents the same top-of-the-loop ITM calls eliminated by the DO UNTIL(EOF), without the EOF checking.

Version 6.07 will have a more efficient input formatting process according to Jeffrey Polzin.

\textbf{OPTIMIZING I/O OPERATIONS AND DATA STORAGE}

Dan Squillace in Tuning the SAS System Version 6 under MVS extensively discussed optimizing data storage and I/O requirements in Version 6.06.

\textbf{Controlling Data Storage Requirements}

In Version 6.06 the physical blocksize of an SAS data library must be:

- between 4096 and 32768 bytes
- a whole-number multiple of 512 bytes

Observations in Version 6.06 are stored in SAS data sets by pages. A page is a contiguous area of space that can span track boundaries. The characteristics of a page must be:

- between 4096 and 16,777,216 bytes
- a whole-number multiple of 512 bytes
- a whole-number multiple of physical blocksize
- no smaller than physical blocksize

Data sets in Version 6.06 start at the beginning of a page. Data sets in Version 5.18 started at the beginning of a track. In Version 6.06, several data sets can reside on the same track. In Version 5.18, only one data set could reside on a track. Version 5.18 wasted storage space at the end of a track when data sets were small.

If the data sets are small in Version 6.06, using a small page size reduces storage waste in the data library by reducing waste at the end of each data set.

SAS Institute recommended a half-track LRECl and BLKSIZE value of 23456 for Version 5.18. That value cannot be used for Version 6.06. It is not a whole-number multiple of 512 bytes.

The nearest whole-number multiple of 512 bytes for a 3380 'half-track' is 23040. The LRECl and BLKSIZE DCB parameters for a SAS data library should be changed to 23040 in Version 6.06 if 'half-tracking' is to be maintained.

The question applicable in Version 6.06, and not in Version 5.18, is whether 'half-track' blocking is the most efficient structure for a SAS library. That depends on page size, since blocksize can never be larger than page size.

If a small page size is used to reduce storage waste, the blocksize has to be small also. SAS recommends a 6144 LRECl and BLKSIZE for data libraries having small data sets. A data library with a 6144 blocksize can have data sets with page sizes of 6144, 12288, 18432, etc.

Page size in Version 6.06 is set using the BUFSIZE (buffer size) option. The page size of a SAS data set is a permanent feature of that data set and is not changed unless that data set is recreated using a different BUFSIZE.

An important note. In non-sequential data libraries, the SAS library engine does not attempt to continuously store the pages of a data set. Non-contiguous storage permits better space utilization by allowing the library engine to use space that was deleted from another data set. It also requires the data library to keep pointers to each page of a data set.
According to Keith Buckley of General Dynamics - Fort Worth, a SAS library will appear to run out of storage space when it runs out of pointer space in its directory. If several of the data sets are large, a solution is to increase the page size of those data sets in the library. That will reduce the pointer space needed by the directory. Otherwise, the unused space in the library should be freed and another data library to hold new data sets should be created.

Setting BUFSIZE and BUFNO for SAS Data Sets

BuFSIZE and buffer numbers for SAS data sets are set with the BUFSIZE and BUFNO options.

BUFSIZE is a system option and an output data set option. It cannot be used on the SET statement because it is a permanent part of the input data set.

BUFSIZE establishes the page size in which a data set is stored and the buffer size that will be used to read that data set into a SAS Data Step or Procedure.

BUFNO is a system option and an option for both input and output data sets. It can be used on both SET and DATA statements. It is not a permanent feature of the data set.

BUFNO establishes the number of buffers that will be used to read or write a SAS data set.

The BUFNO and BUFSIZE of the data set being read can differ from the BUFNO and BUFSIZE of the data set being created. Separate buffer areas are used for reading and writing data.

In Version 5.18, all data sets in an SAS library had the same buffer sizes and page sizes. In Version 6.06, the small data sets can have different buffer sizes and page sizes than the large data sets in that library. That flexibility allows the SAS programmer to minimize the memory requirements of an SAS job and its data library.

Below are two examples of the correct syntax for setting BUFNO and BUFSIZE on Data Steps and Procedures such as PROC PRINT:

```
DATA WORK(BUFNO=2,
BUFSIZE=23040);
PROC PRINT DATA=WORK1;
SET WORK(BUFNO=7);
RUN;
RUN;
```

Setting BUFNO and BUFSIZE for External Files

The SAS System and Data options BUFNO and BUFSIZE do not determine the buffer sizes and buffer numbers for reading and writing external files.

SAS uses a buffer size equal to the physical blocksize of the file being read. It uses five (5) buffers if the buffer number is not specified.

The DBC BUFNO parameter in the program's JCL can be used to set the number of buffers for reading an external file. It can also be specified as an option to the INFILE or FILE statement.

Ami Ghosh of SAS Institute indicates that the NCP parameter on the DD statement should also be used if BUFNO is employed. He advises setting it to the same number as BUFNO.

Only small performance gains should be expected. SAS uses IBM's buffering and access methods to retrieve data from external files. Those methods are not nearly as efficient as the SAS system's method for reading and outputting SAS data sets.

More information is available in the SAS Companion for the MVS Environment.

Controlling I/O for SAS Data Sets

The total amount of data passed in a single I/O operation in Version 6.06 is the product of BUFSIZE and BUFNO (buffer number). For example, if the BUFSIZE is 23040 and the BUFNO is 2, the total number of bytes moved would be 23040 * 2, or 46080.

During each loop through a Data Step that reads observations from an SAS data set, Version 6.06 moves the product of BUFNO and BUFSIZE into the total buffer memory. Each move is an I/O operation that is captured in EXCP count and costs CPUs.

Reducing the number of I/O operations used to transfer data into a Data Step saves processing expense and reduces the contention for a channel to move data from one device to another.

Increasing BUFSIZE and/or BUFNO reduces CPU activity by reducing the number of I/O operations needed to move a data set. A larger total buffer memory moves more data with each I/O operation.

Higher buffer memory, however, can slow elapsed times when the system is short of memory due to high demand.

According to Brian Bowman of SAS Institute, after the SAS system moves the input data into the total buffer memory area, it then pages that memory according to the value of BUFSIZE. It also performs blocking and deblocking activities and logically links each page together. For that reason, 6 pages of 7680 bytes should require a few more CPUs for I/O processing than will 2 pages of 23040 bytes, even though the same number of bytes are moved.

The best method, therefore, for reducing I/O expense is to increase BUFSIZE unless it excessively wastes data set storage. If waste is a problem, increase BUFNO. BUFNO does not affect permanent storage.

A large BUFNO and BUFSIZE can be very important for random-access Data Steps which move backwards and forwards over many observations. Such Steps can have a much higher I/O activity than Data Steps that read data sequentially.

When random-access Steps seek to go beyond the beginning or end of the buffer memory, the SAS system must refill the input buffer. Expanding the total buffer memory can minimize the returns to the data set for more observations.
A Constraint to I/O Optimization

As reminded by Bowman, Buffer memory is stored in virtual, not real memory, on MVS/ZA and MVS/ESA systems. The longer a Step requires buffer memory and the more memory it requires, the more likely that memory will be paged by the system onto a disk device. In those cases, the advantage of using more buffer for that job will be lost or reduced.

I/O and Storage Test Results

* Moving data from the input buffer into the PDV is more efficient in Version 6.06 than Version 5.18. This is shown by comparing ‘INPUT;' statement processing in the two versions.
* Moving data from the PDV to the output buffer is equally as efficient in Version 6.06 as Version 5.18.
* Larger buffer memory often does save CPUs in the reading and outputting of SAS observations. But the savings can vary greatly.
* Equal buffer memory, in fewer pages, will save a few more CPUs than the same memory in more pages. But the gain is very small.
* Small page sizes such as 6144 or 7680 do reduce storage waste in small data sets.
* A page size and blocksize of 7680 or a 3380 appears equally as efficient in data storage and I/O operations as a page size and blocksize of 6144.
* Small page sizes require the data library to retain more pointers to the pages. The library will appear to run out of storage space if it runs out of pointer space. Increase the page size of the large data sets in the library, or free space in the library and create another library for new data sets.
* A blocksize of 7680 allows the use of a 23040 page size, which is nearly a 3380 half-track. A blocksize of 6144 cannot be used with a page size of 23040 because a page must be a whole-number multiple of the data library's blocksize.
* Using a 7680 page size for small data sets and 23040 for large data sets in the same data library is a useful method of minimizing storage waste in a data library.

SELECTIVELY SUBSETTING DATA

In Version 5.18, the data could be selectively subset only through the use of subsetting IF statements or IF statements joined with OUTPUT statements.

IF DEPT = 'ACCT' OR IF DEPT = 'ACCT' OR
DEPT = 'PAYROLL' OR DEPT = 'PAYROLL' OR
DEPT = 'LEGAL' OR DEPT = 'LEGAL' OR
DEPT = 'INVOICE'; DEPT = 'INVOICE' THEN
OUTPUT;

Subsetting IF statements (and DELETE and RETURN statements) caused the DATA loop to return to the SAS supervisor, which always checked for a default OUTPUT statement and then evaluated each PDV variable to see whether its value should be retained. Selective OUTPUT statements did not cause the loop to return the SAS supervisor. The SAS supervisor was called only at the bottom of the loop.

Version 5.18 programmers could reduce processing time by placing a selective OUTPUT statement inside of a DO UNTIL(EOF) loop:

DO UNTIL(EOF);
SET WORK1 END=EOF;
IF DEPT = 'ACCT' OR
DEPT = 'PAYROLL' OR
DEPT = 'LEGAL' OR
DEPT = 'INVOICE' THEN
OUTPUT;
END;

Using a DO UNTIL(EOF) loop in version 5.18 to selectively output data prevented both types of returns to the SAS supervisor:

- when the data was selectively output,
- when the bottom of the DATA loop was reached.

Version 5.06 should eliminate this processing opportunity. It has dropped both the default OUTPUT checking and the individual checking of variables to determine a value should be retained. The test results of this paper support that judgement, except when data is selectively output inside of a DO UNTIL(EOF) loop using IF-THEN-ELSE code.

DO UNTIL(EOF)
SET WORK1 END=EOF;
IF IF DEPT = 'ACCT' THEN
OUTPUT;
ELSE IF DEPT = 'PAYROLL' THEN
OUTPUT;
END;

The IN Operator and WHERE Statements

Robina Thornton and John Boling mentioned the IN operator and the WHERE statement in their 1990 proceeding The Painless Path to Release 6.06 of the SAS System. Both can be used to subset observations in the DATA Step. The WHERE statement can also be used to subset data entering Procedures.

The IN operator facilitates comparisons to a list of items that would normally be examined with the = and OR operators. It replaces the coding on the left with that on the right.

IF DEPT = 'ACCT' OR IF DEPT IN ('ACCT',
DEPT = 'PAYROLL' OR 'PAYROLL',
DEPT = 'LEGAL' OR 'LEGAL',
DEPT = 'INVOICE'; 'INVOICE');

Furthermore, and more importantly, according to the SAS Programming Tips: A Guide to Efficient SAS Processing, it terminates comparisons once a match is found. An IF statement composed of = and OR operators continues its evaluation even when a match has been found. Thus, an IF statement using an IN operator should work like this IF-THEN-ELSE code that uses GOTO statements:

IF DEPT = 'ACCT' THEN
GOTO BYPASS;
ELSE IF DEPT = 'PAYROLL' THEN
GOTO BYPASS;
ELSE IF DEPT = 'LEGAL' THEN
GOTO BYPASS;
ELSE IF DEPT = 'INVOICE' THEN
GOTO BYPASS;
DELETE;
BYPASS;

51
The above code is a proven technique of increasing the performance efficiency of Data Steps which subset observations. It selects data with fewer comparisons than Data Steps which contain IF statements that use = and OR operators. If an IN operator performs similarly to IF-THEN-ELSE code, it should be more efficient than IF statements that use = and OR operators and equally as efficient as IF-THEN-ELSE statements. The test results of this paper show otherwise.

WHERE statements subset data very differently than do other subsetting structures in the SAS system. Subsetting IF statements, DELETE statements, RETURN statements, and code constructions that bypass OUTPUT statements execute as part of the SAS Data Step. They filter the data being transferred from the PDV to an output data set.

WHERE statements work within the I/O Engine Supervisor of the SAS system. They filter the data flowing from the input buffer to the PDV. They can filter the data as it is read or they can use an INDEX (key) to retrieve only the matching observations from the data set.

Observations eliminated by WHERE statements never enter the PDV. They are never evaluated by any of the statements of a Data Step.

WHERE statements can be placed within a Procedure or a Data Step.

DATA WORK;
PROC PRINT DATA = WORK;
SET WORK;
WHERE DEPT = 'ACCT' OR DEPT = 'PAYROLL' OR DEPT = 'LEGAL' OR DEPT = 'INVOICE';

Or they can be treated as an option to a data set.

DATA WORK;
SET WORK WHERE DEPT = 'ACCT' OR DEPT = 'PAYROLL' OR DEPT = 'LEGAL' OR DEPT = 'INVOICE';

When WHERE statements are used as a data set option, they subset observations only from that data set. Subsetting statements which work inside of the Data Step do not have that capability. They operate against all observations that enter the PDV.

Since WHERE statements do not become part of the Data Step, they cannot be performed conditionally, nor can they perform any action other than subsetting data. These WHERE constructions are illegal:

DATA WORK;
SET WORK;
IF DAY = 30 THEN WHERE DEPT = 'ACCT' OR DEPT = 'PAYROLL';
END;

According to the SAS Language Reference, WHERE statements should be more efficient than subsetting IF statements since they never bring unwanted observations into the Data Step. Also, since they have been used within Procedures, they should be more efficient than using a preliminary Data Step to subset the Data before passing that data to the Procedure.

Subsetting Test Results

* Subsetting IF statements in Version 6.06 are equally as efficient as in Version 5.18.
* Well-structured IF-THEN-ELSE statements improve processing efficiency in Version 6.06 as much as they did in Version 5.18.
* Well-structured IF-THEN-ELSE code which conditionally outputs data from within a DO UNTIL (EOF) loop is the most efficient method for subsetting data.
* The IN operator is less efficient than evaluating data than an IF statement that uses = and OR operators.
* The IN operator is much less efficient at evaluating data than is well-constructed IF-THEN-ELSE code.
* Non-indexed WHERE statements are less efficient than simple subsetting IF statements when selecting a small number of observations.
* Non-indexed WHERE statements in a PROC PRINT are less efficient for selecting and reporting a small number of observations than a subsetting Data Step and a subsequent PROC PRINT.

According to Technical Support, SAS Institute, the WHERE statement is less efficient than a subsetting IF statement when the length of the record being moved from the input buffer to the PDV is small. As the length increases, the WHERE statement becomes more efficient. Thirty (30) was the length of the record moved in the tests of this paper.

Also, tests ran by Technical Support indicated that the IN operator is less efficient than IF statements that use = and OR operators.
REFERENCES


SAS is a registered trademark of SAS Institute, Cary, NC, USA.

Special appreciation goes to Dan Squillace, Jeff Polzin, Brian Bowman, John Boling, and Ami Ghosh of SAS Institute for patiently explaining the topics of their papers and for answering my many questions.