INTRODUCTION

This paper describes some enhancements to the DATA step that will be of interest to the more technically oriented SAS users. These enhancements include such aspects as DATA step input and output views, the new MODIFY statement that provides for the ability to update SAS data sets in place, the SET statement with keyed access, performance improvements to the DATA step since the last release, and newly incorporated features in the DATA step language. Also, additional SASware Ballot® requests are discussed.

INPUT AND OUTPUT 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.

Note: An experimental version of the input DATA step views was released in the fall of 1990 for the MVS, CMS, and VMS operating system environments. The Release 6.06 O/S/® and Release 6.07 UNIX® operating system environments will receive the experimental version of the input DATA step views with the final production systems.

You can use DATA step views to directly process any file that can be read with an INPUT statement to dynamically generate data 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/ASIST® 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;
   INPUT PARTS;
   INPUT PART QUANTITY UNIT COST;
RUN;
/* Use the view */
PROC PRINT DATA=READ;
RUN;

The example code above provides a simple example of an input DATA step view. The first step creates or defines the input DATA step view. The next step, PROC PRINT, 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.

The following sections describe the process, provide the syntax, and list requirements and restrictions for using DATA step views.

Description of DATA Step Views

Conceptually, a DATA step view is the same as other SAS data 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, Version 6, First Edition.


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 Release 6.06, 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, is internal to the DATA step view engine, and cannot be changed by the user. When you use the view as input or output in a SAS procedure or DATA step, the SAS System 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.

Processing with DATA Step Views

When you create a DATA step view from SAS source code, the DATA step view engine 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, in the current SAS session or within a SAS session at a later time, of a DATA step or SAS procedure. The SAS System resolves the intermediate code produced by the compiler and generates the executable code for the host environment. The generated code is executed, as a DATA step or a SAS procedure requests observations from the view.
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 program that is 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 program, the SAS System stores the DATA step view but not the global statements: statements. When the view is referenced, actual execution may differ from the intended execution.
- Both global 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, that the MACRO facility resolution and expansion is performed at compilation time only, 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).

Note: The use of DATA step views can improve programming and application development efficiency at the expense of machine resources, particularly if a procedure requires multiple passes through the data. When required, rewinds will be limited to a BY group. Memory requirements for step resources will increase. If a DATA step view is referenced within a DATA step program, the 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 overhead incurred is associated to the DATA step environment that is required to execute the view.

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.

Creating and Using 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.

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 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 must also 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.

This 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/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 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. For more information about naming SAS data sets, refer to "SAS Data Sets" in Chapter 6 in SAS Language: Reference.

You must also 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.
This example creates an output DATA step view named A.

DATA VIEW A;
  /* SAS DATA step statements */
  /* End 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 a SAS data view of that name already exists in the data library, it will be overwritten with the new view definition.

Use of 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 CONTENTS DATA VIEW;
RUN;

When processed, the 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 B VIEW A;
  /* SAS DATA step statements */
RUN;

To illustrate this point, when view B is processed in the following statements, two things happen. 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 View Example

The following example demonstrates how to generate an input and an output DATA step view and shows how both are referenced.

DATA IN/VIEW A;
  INFILE INFILE;
  INPUT PART $ PSYS COCTS;
RUN;

DATA ORDER/VIEW OUT;
  SET OUT;
  IF W = 0 THEN OUTPUT ORDER;
  /* Print a report */
RUN;

PROC SORT DATA IN/OUT;
  BY PART;
RUN;

The input DATA step view reads data from an external file using the INFILE and INPUT statements. The data which are output by this view are passed to the SORT procedure to be sorted by the variable PART. When the INFILE has been completely read, the SORT procedure sorts the data and then writes each of the sorted observations to the output DATA step view. 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. 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.

THE KEY= OUTPUT DATA SET OPTION

The KEY= output data set option allows the user to define indexes when creating a data set instead of building indexes using the DATASETS procedure after creating the data set. The KEY= option allows for the creation of simple and composite key indexes along with specifications to disallow missing values as key values and to guarantee the uniqueness of keys defined within an index.

A simple key index or indexes can be created by specifying the KEY= option.

Begin by creating a simple index named STOCK.

DATA STOCK (KEY=STOCK);

Next, create two indexes named AUTHOR and SUBJECT.

DATA BOOKS (KEY=AUTHOR SUBJECT);

A composite key index or indexes can be created by specifying the KEY= option.

Begin by creating a composite key named CITYST using the variables CITY and STATE.

DATA EMPLOYER (KEY=CITYST (CITY STATE)));

Next, create a simple key index named SSN and a composite key index named CITYST using the variables CITY and STATE.

DATA EMPLOYER (KEY=SSN CITYST (CITY STATE)));

The following options can be specified on each key definition so that restrictions are placed on key values.

NOMISS

Excludes from the index all observations with missing values for all index variables. Observations can still be read from the data set but not through the index.

UNIQUE

Specifies that the combination of values of the index variables must be unique. If you specify UNIQUE for a new data set and multiple observations have the same values for the index variables, the index is not created.


Finally, create a simple key index named SSN restricted by key uniqueness and by non-missing key data. Create a composite key index named CITYST using the variables CITY and STATE restricted by non-missing key data.

DATA EMPLOYER (KEY=SSN/UNIQUE/NOMISS CITYST (CITY STATE)/NOMISS);
MODIFY STATEMENT

The MODIFY statement extends the capabilities of the DATA step to manipulate a SAS data set in place. The MODIFY statement allows for the ability to read and modify records in a data set by sequential, random, or matching access. The MODIFY statement can be used with the POINT= or the KEY= option to randomly retrieve observations from a SAS data set for further manipulation. The MODIFY statement can be used without a BY statement to sequentially read a SAS data set, or it can be used with a BY statement to apply transactions to a master data set in the same manner in which the UPDATE statement applies them.

The MODIFY Statement without a BY Statement

```sas
DATA STOCK;
MODIFY STOCK;
INDATE = TODAY();
RUN;
```

The data set STOCK is read, the variable INDATE is updated with today's date and the updated variable is rewritten to the data set STOCK.

MODIFY with a BY Statement

```sas
DATA STOCK;
MODIFY STOCK SELL;
BY NAME;
INDATE = TODAY();
RUN;
```

The processing of a MODIFY statement with a BY statement is identical to that performed by the UPDATE statement, except that the data sets do not have to be in sorted order. The MODIFY statement reads an observation from the transaction data set SELL and an observation is read from the master data set STOCK that matches the value of the BY variables from the transaction observation. The BY variables are used to create a dynamic WHERE expression that is applied to the master data set during search operations. Indexes on the master data set can improve performance of retrieval operations from the master data set. When a matching observation is found and its variables have been retrieved, the variables on the transaction observation are retrieved overwriting existing variable values from the master observation. The MODIFY statement uses the same rules as the UPDATE statement uses to apply data from the transaction data set to data from the master data set.

The MODIFY Statement with the POINT = Option

```sas
DATA STOCK;
SET SELL;
MODIFY STOCK POINT=P;
INDATE = TODAY();
RUN;
```

After the data set SELL is read, an observation is read from the data set STOCK that corresponds to the observation number addressed by the value of the variable P. INDATE is then updated with today's date, and the updated variable is rewritten to the data set STOCK.

The MODIFY Statement with the KEY = Option

```sas
DATA STOCK;
SET SELL;
MODIFY STOCK KEY=NAME;
INDATE = TODAY();
RUN;
```

After the data set SELL is read, an observation is read from the data set STOCK that corresponds to the observation indexed by the value of the simple key NAME. INDATE is then updated with today's date, and the updated variable is rewritten to the data set STOCK. This type of processing requires the use of indexes.

Additional Support for the MODIFY Statement

OUTPUT Statement

When an OUTPUT statement is executed, the current observation is written to the end of the data set.

REPLACE Statement

When a REPLACE statement is executed, the current observation read from the corresponding MODIFY statement data set is rewritten with any newly-modified data.

REMOVE Statement

When a REMOVE statement is executed, the current observation is deleted. This action may cause the observation to be physically or logically deleted based on what engine is maintaining the data set.

Interactions between OUTPUT, REPLACE, and REMOVE Statements

When using the OUTPUT, REPLACE, and REMOVE statements together in the same DATA step program, there are certain interactions that must be noted. The following restrictions are made so that proper position within the data set and indexes can be maintained.

- An OUTPUT statement must be executed after a REPLACE or REMOVE statement. This restriction is made so that the proper position within the data set and indexes can be maintained.

- When no OUTPUT, REPLACE, or REMOVE statement is present in the DATA step program, any data set opened for update record access will have default REPLACE statement generated. This behavior is consistent with the generation of default OUTPUT statements for output record access data sets.

- When an OUTPUT, REPLACE, or REMOVE statement is present, no default operations will be generated, and it is the responsibility of the programmer to add these statements at appropriate points in the DATA step source code.

```sas
DATA A B;
MODIFY B;
/* Change data from data set B */
RUN;
```

For the code above, a default OUTPUT statement for the data set A and a default REPLACE statement for the data set B will be generated using the rules stated before.

```sas
DATA A B;
MODIFY B;
/* Change data from data set B */
IF <SOME CONDITION> THEN
REMOVE B;
RUN;
```

The presence of the REMOVE statement will not cause the generation of a default OUTPUT statement for the data set A or a default REPLACE statement for the data set B. To have either operation occur will require the addition of the appropriate statements to the DATA step source code.

```sas
DATA A B;
MODIFY B;
/* Change data from data set B */
IF <SOME CONDITION> THEN
REMOVE B;
ELSE
IF <SOME OTHER CONDITION> THEN
OUTPUT A;
```
I/O Return Code Variable
The automatic variable _IORC_ is used to feedback I/O return codes back to the DATA step application, that describe abnormal I/O condition. This type of feedback allows the application more control over these situations. The value returned in the _JORC_ variable is machine-independent. The types of feedback that are currently given include the following:

- MODIFY with POINT option
- MODIFY with KEY option
- MODIFY with BY statement
- SET with KEY option
- MODIFY with BY statement
- REPLACE statement
- OUTPUT statement

The code above is a skeleton of a possible report generation application. An observation is read from the data set DATA, and the RECODE label is linked to recode the CODE variable into a more descriptive value. Traditionally, PROC FORMAT style formats have been used to perform this mapping of compact data into expanded data. Other methods, similar to the KEY option, have been discussed in the past by using HASH algorithms, binary searches, and so on, using the POINT= option to randomly probe the recoding file. Once the CODE variable has been recoded, control is returned to the main body to perform additional operations necessary to generate the application's intended report. As with the MODIFY statement, the automatic variables _ERROR and _JORC_ will be set accordingly, whenever errors or warnings generate an exception during an I/O access operation using the SET statement.

MODIFICATIONS TO CURRENT STATEMENTS
OUTPUT Statement RC = Option
The OUTPUT statement has been modified to allow applications to return specific return codes through input DATA step views to control the behavior of a DATA step or procedure reading data through the input DATA step view.

The above code demonstrates the use of the _JORC_ return code variable in determining error conditions which are retrieving observations from an indexed data set. The data set RECEIVE, which contains information on received shipments of parts, is read. The inventory data set INVEN is read using the variable PART as a key. The RENAME statement and the RENAME Data set option now allows variable list specifications as options.
The MERGE Statement Changes
The MERGE statement will now issue messages detailing information about what variables are overwritten with data from one or more data sets being merged.

The generation of this information will be controlled by a system option.

NEW STATEMENTS

LEAVE Statement
The LEAVE statement controls the flow within a DO loop. The execution of the LEAVE statement provides an early exit from a DO loop and a SELECT statement body. A LEAVE statement causes the innermost enclosing DO loop or SELECT statement body to be exited immediately.

```
DATA NUL.
  ARRAY SAMPLE[100];
  SET SAMPLE;
  /* Determine length of sample set */
  DO 1 = 1 TO 100;
    IF SAMPLE[1] < 0 THEN
      LEAVE;
  END;
  /* ADDITIONAL SAS STATEMENTS */
RUN;
```

When the value of any of the SAMPLEn variables is less than zero, exit the DO loop. If a LEAVE statement is used within the body of a SELECT statement, the body of the SELECT statement is immediately exited.

CONTINUE Statement
The execution of the CONTINUE statement causes the DO loop to suspend execution of the current loop iteration, and causes the next loop iteration to begin. The CONTINUE statement only applies to DO loops, not to SELECT statement bodies. A CONTINUE statement inside a SELECT statement body will cause an error message to be generated.

```
DATA NUL.
  ARRAY SAMPLE[100];
  SET SAMPLE;
  DO 1 = 1 TO 100;
    IF SAMPLE[1] < 0 THEN
      CONTINUE;
  END;
  /* ADDITIONAL SAS STATEMENTS */
RUN;
```

When the value of the SAMPLE set is less than zero, do not include the sample value in any further calculations of the DO loop. Begin execution of the DO loop at the next iteration value.

DATA STEP FUNCTIONS AND SUBROUTINES
A set of new functions and subroutines has been added which deal with interfacing with the macro facility, string manipulation, numerical operations, memory access, and bitwise logical operations.

```
TRIMN(V)  removes trailing blanks from the character argument V and returns the trimmed string. This function can return a zero length character value and is useful in concatenation operations.
INDEXW(S,P)  works like the INDEX function, but the indexed substring pattern must begin on a word boundary.
RECIPE(N)  returns the reciprocal of N, (1/N).
ADDR(V)  returns the memory address of the storage corresponding to V.
PEEK(V,L)  returns a numeric value corresponding to the memory address V using a data length of L. L defaults to a length of a double precision floating-point value.
PEEK(V,L)  returns a character value corresponding to the memory address V using a data length of L. L defaults to a length of 8.
CALL POKE(A,V,L)  places into memory, pointed to by address A, the value of the variable V using the length of L. L defaults to a length of a double precision floating-point value if V is a numeric expression, and to a length of 8 if V is a character expression.
BOR(A,B)  returns the bitwise logical OR of A and B.
BAND(A,B)  returns the bitwise logical AND of A and B.
BNOT(A)  returns the bitwise logical NOT of A.
BSHIFT(A,B)  returns the bitwise logical right shift of A for B bits.
BLSHIFT(A,B)  returns the bitwise logical left shift of A for B bits.
BXOR(A,B)  returns the bitwise logical EXCLUSIVE OR of A and B.
EXECUTE(A)  resolves the macro expression contained in the variable A, and issues the resolved macro value for execution upon the termination of the DATA step.
RESOLVE(A)  returns the value of the resolved macro expression contained in the variable A.
```

PERFORMANCE ENHANCEMENTS

Specifics of areas in the DATA step environment have been identified and modified to improve the performance of the DATA step and of the SAS System as a whole. The areas of performance improvements include:

- Bit string operations
- LAG/DIFF functions
- Numeric to character and character to numeric conversion processing
- Perform more inline code generation
- Format and Informat performance improvements
- Function and subroutine performance improvements
- FILE/INFILE performance improvements
- Internal logic and algorithm performance improvements
- Data set processing performance enhancements.
SASWARE BALLOT

Each year the SASware Ballot is distributed for the purpose of finding out what SAS users are content with, as well as what they are looking for. Here are some ideas that are currently under investigation.

• The RENAME statement and the RENAME data set option now allow variable list specifications as options. This item will be implemented in Release 6.08 of the SAS System.

• Provide a true syntax checker for SAS programs. Under investigation.

• Provide an option to convert all numeric missing values to 0 or to a specified value. Under investigation.

• Store array definitions with a data set. Under investigation.

• Issue a warning to indicate each variable that is being overwritten if one or more variables are common to data sets being merged. This item will be implemented in Release 6.08.

• Enhance invalid data message so it lists which file contained invalid data when using INFILE statements. Under investigation.

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, 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, SAS/ASSIST, and SASware Ballot are registered trademarks of SAS Institute, Inc., Cary, NC, USA.

OS/2 is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of AT&T.

VMS is a trademark of Digital Equipment Corporation.