

DATA Step Changes and Enhancements

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

INTRODUCTION

This paper describes some enhancements to the DATA step that will interest more technically oriented SAS users. These enhancements will include such aspects as DATA step input and output views, the output data set option INDEX=, the new MODIFY statement that provides for the ability to update SAS data sets in place, the SET statement with keyed access, newly incorporated features in the DATA step language, and performance improvements to the DATA step since the last release.

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. 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 example above provides a very 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, whether from a SAS data set, a SAS/ACCESS data set, an SQL view, or input DATA step view.

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

DATA Step View Conceptual Model

Conceptually, a DATA step view is the same as other SAS data views. A DATA step view is a SAS data set of 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.

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*.

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 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.

Input DATA Step Views Syntax

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

You must include a DATA statement which must include the following argument:

output-SAS-data-set-names

specifies a valid SAS name for each output SAS data file or 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 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 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 a 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.

Output DATA Step Views Syntax

```
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 must include the following argument:

input-SAS-data-set-names

specifies a valid SAS name for each input SAS data file or 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/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/MODIFY statement. If the SET/MERGE/UPDATE/MODIFY 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 a DATA step view named A.

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

Caution: SAS data files and SAS data views in the same library cannot have the same name.

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

DATA Step Language Extensions to Support Views

The OUTPUT statement has been modified to allow applications to return specific codes through input DATA step views to control the behavior of a DATA step or procedure reading data through the input DATA step view.

```
DATA IN/VIEW=IN;
RETAIN COUNT 0;
INFILE DEVICE;
INPUT SAMPLE;
COUNT = COUNT + 1;
IF COUNT = 10 THEN DO;
  OUTPUT IN RC=XXXXX;
  COUNT = 0;
END;
RUN;

PROC YYYYY DATA=IN;
RUN;
```

Whenever the input DATA step view has sent 10 samples it will return a specific return to the consuming procedure YYYYY to signal it to performing a set of specific operations. This set of operations might include recalculating new information based on previous set and latest set of samples, and causing a redrawing graphic display information. The RC= option can be a numeric constant, using %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 set of return codes and procedures that would use this functionality has not been completely determined at this time. If you have an interest in assisting 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.

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, 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 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;
```

Input and Output DATA Step View Example

Create the view definitions in a different execution of the SAS system by placing the views in a permanent SAS data set library.

```
DATA SAVE.IN/VIEW=SAVE.IN;
  INFILE INDATA;
  INPUT PART $ N COST;
  RUN;

DATA REORDER/VIEW=SAVE.OUT;
  SET SAVE.OUT;
  BY PART;
  IF N = 0 THEN OUTPUT REORDER;
  /* Print a report */
  RUN;
```

Reference the predefined views for processing by the SORT procedure.

```
PROC SORT DATA=SAVE.IN OUT=SAVE.OUT;
  BY PART;
  RUN;
```

The previous example demonstrates how to generate an input and an output DATA step view and how they are both referenced. The input DATA step view reads data from an external file using the INFILE and INPUT statements. The data outputted 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 will sort the data and then write each of the sorted observations to the output DATA step view. The output DATA step view will then generate a report based on the sorted data along with an additional data set named REORDER, which holds observations corresponding to the PART variables 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.

OUTPUT DATA SET OPTION INDEX=

The INDEX= output data set option allows the user to define indexes when creating a data set rather than building indexes using the DATASETS procedure after creating the data set. The INDEX= 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 INDEX= option.

Create a simple index named STOCK.

```
DATA STOCK(INDEX=(STOCK));
```

Create two indexes named AUTHOR and SUBJECT.

```
DATA BOOKS(INDEX=(AUTHOR SUBJECT));
```

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

Create a composite key named CITYST using the variables CITY and STATE.

```
DATA EMPLOYEE(INDEX=(CITYST=(CITY STATE)));
```

Create a simple key index named SSN and a composite key index named CITYST using the variables CITY and STATE.

```
DATA EMPLOYEE(INDEX=(SSN CITYST=(CITY STATE)));
```

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 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. Refer to the DATASETS procedure in Chapter 17 in *SAS Procedures Guide, Version 6, Third Edition* for more information.

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 EMPLOYEE(INDEX=(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 retrieve observations randomly from SAS data set for further manipulation. The MODIFY statement can be used without a BY statement to read a SAS data set sequentially or 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.

MODIFY Without a BY Statement

```
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

```
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 by overwriting existing variable values from the master observation using the same rules the UPDATE statement uses to apply data from the transaction data set to data from the master data set.

MODIFY With a POINT = Statement Option

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

MODIFY With a KEY = Statement Option

```
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 number addressed by the value of the variable NAME, INDATE is 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 an default REPLACE statement generated. This behavior is consistent with the

generation of default OUTPUT statement for output record access data sets.

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

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

For the previous code, 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.

```
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.

```
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;
    ELSE
      REPLACE B;
RUN;
```

I/O Return Code Variable

The automatic variable _IORC_ is used to feedback I/O return codes back to the DATA step application. This type of feedback allows the application more control over these situations. The value returned in the _IORC_ variable is machine independent. The types of feedback that are given currently include :

- MODIFY with POINT option - no matching observation.
- MODIFY with KEY option - no matching observation.
- SET with KEY option - no matching observation.
- MODIFY with BY statement - no matching master observation.
- MODIFY with BY statement - master data set contains duplicates.
- REPLACE statement - executed on non-current observation.
- REMOVE statement - executed on non-current observation.

A non-current observation is a condition in which the DATA step determines that a MODIFY statement has not been executed or that a MODIFY statement has been executed but the observation matching conditions failed.

Example

The following code demonstrates the use of the `_IORC_` return code variable in determining error conditions retrieving observations from a indexed data set. The data set `RECEIVE` is read, and it contains information on received shipments of parts. The inventory data set `INVEN` is read using the variable `PART` as a key, the index for `PART` must have been previously defined for the data set `INVEN`. If `_IORC_` is set to `_SWEOF`, END OF FILE, a new observation is added to the `INVEN` data set otherwise the number of current parts are updated and the observation is replaced in the `INVEN` data set. The autocall macro `%SYSRC` is used to expand defined I/O return code names to specific values, so that checks for specific I/O conditions can be made during the execution of an application.

```
DATA INVEN;
  SET RECEIVE;
  MODIFY INVEN KEY=PART;
  IF _IORC_ = %SYSRC(_SWEOF) THEN
    OUTPUT;
  ELSE DO;
    N = N + NR;
    REPLACE;
  END;
RUN;
```

SET STATEMENT WITH KEY= OPTION

The `SET` statement with the `KEY=` option provides the capability to retrieve data from a SAS data set through direct use of indexes. The `KEY=` value must be the name of a primary key or the name of a composite key, a key composed of more than one variable. When the `SET` statement is executed, a data observation is retrieved using the value of the key variable value(s). If no matching observation can be retrieved, then the `_ERROR_` variable is set to 1 and the `_IORC_` automatic variable is set to the corresponding error resulting from the failed I/O access.

```
DATA _NULL_;
  SET DATA;
  LINK RECODE;
  /* Generate a report from read data */
  RETURN;
RECODE:
  SET CODES KEY=CODE;
  RETURN;
RUN;
```

The previous code 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 probe the recoding file randomly. 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 `_IORC_` will be set accordingly, whenever errors or warnings generate an exception during an I/O access operation using the `SET` statement.

DATA STEP LANGUAGE EXTENSIONS

The `DATA` step language has been updated to include additional enhancements other than the constructs that have been previously discussed. These enhancements include flow of control statements for use within `DO` loops and new `DATA` step functions and subroutines.

LEAVE Statement

The `LEAVE` statement controls the flow of control 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 _NULL_;
  ARRAY SAMPLE{100};
  SET SAMPLES;
  /* Determine length of sample set */
  DO I = 1 TO 100;
    IF SAMPLE[I] < 0 THEN
      LEAVE;
  END;
  /* ADDITIONAL SAS STATEMENTS */
RUN;
```

Whenever the value of any of the `SAMPLE` variables is less than 0 then 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 _NULL_;
  ARRAY SAMPLE{100};
  SET SAMPLES;
  DO I=1 TO NSAMPLES;
    IF SAMPLE[I] < 0 THEN
      CONTINUE;
  /* ADDITIONAL SAS STATEMENTS */
  END;
RUN;
```

Whenever the value of `SAMPLE` is less than 0 then do not include sample value in any further calculations of the `DO` loop, and begin execution of the `DO` loop at the next iteration value.

DATA Step Functions and Subroutines

A set of new functions and subroutines have been added dealing with macro interfacing, string manipulation, numerical operations, memory access, and bitwise logical operations.

- `COMPBL(V)` - removes multiple blanks between words in the character string `V`.
- `DEQUOTE(V)` - removes single or double quotes from the character string `V`.
- `QUOTE(V)` - quotes the character string `V`. If embedded quotes are found within the character string `V`, then those will be double quoted.
- `INPUTC(V,INFORMAT<,W<,D>>)` - allows for the specification of a character informat at runtime.
- `INPUTN(V,INFORMAT<,W<,D>>)` - allows for the specification of a numeric informat at runtime.
- `PUTC(V,INFORMAT<,W<,D>>)` - allows for the specification of a character format at runtime.

- PUTN(V,INFORMAT<,W<,D>>) - allows for the specification of a numeric format at runtime.
- LOWCASE(V) - lowercases the character string V.
- TRIMN(V) - removes trailing blanks from the character argument V, and returns the trimmed string. This function can return a 0-length character value and is useful in concatenation operations.
- INDEXW(S,P) - works just like the INDEX function, but the indexed substring pattern must begin on a word boundary.
- RECIP(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.
- PEEKC(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(V,A<,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 the size 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.
- SOUNDIX(V) - encodes a character string according to the SOUNDIX algorithm.
- 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.
- BRSHIFT(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.
- CALL EXECUTE(A) - resolves the macro expression contained in the variable A and issues the resolved macro value for execution at the termination of the DATA step.

- RESOLVE(A) - returns the value of the resolved macro expression contained in the variable A.

PERFORMANCE ENHANCEMENTS

Specific 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 improvement include :

- bit string operations
- LAG/DIF 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.

ACKNOWLEDGMENTS

The author greatly appreciates the contributions by Annette Green for providing the discussion of DATA step views, the assistance from Susan O'Connor, Bruce Tindall, and Pat Herbert 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, and SAS/ASSIST are registered trademarks of SAS Institute, Inc., Cary, NC, USA. *indicates USA registration.

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