Updating Data Using the MODIFY Statement and the KEY= Option

Denise J. Moorman and Deanna Warner

Denise J. Moorman is a technical support analyst at SAS Institute. Her area of expertise is base SAS software. Denise has a BS degree in elementary education from Liberty University and a computer programming certificate from North Carolina State University. She has been a SAS software user in Cary, NC, since 1993.

Deanna Warner is a senior technical writer at SAS Institute. She has a Bachelor of Liberal Studies in writing from St. Edward's University. Deanna has been with SAS Institute in Austin, Texas since 1982.

Abstract

The purpose of this article is to explain the nuances of the MODIFY statement and the KEY= option, which is available for both the MODIFY statement and the SET statement. It assumes the reader has used the MODIFY and SET statements and has a general understanding of the KEY= option. The article begins with the MODIFY statement by explaining and comparing the different access methods, one of which uses the KEY= option. In addition, the article contains examples of how to monitor your updates and how to use MODIFY and KEY= to update like-named variables. Then the article goes into more detail regarding the KEY= option for both the MODIFY and SET statements. Examples are provided to explain how different types of data, like duplicate values for a key variable, can affect results when using KEY=.

Contents

Introduction

The MODIFY statement and the KEY= option (for both MODIFY and SET statements) became available in Release 6.07 of the SAS System. Prior to the MODIFY statement and the KEY= option, all DATA step processing of SAS data sets was done by executing the SET, MERGE, or UPDATE statement. MODIFY and KEY= provide the following benefits:

❑ The MODIFY statement uses less disk space than the SET, MERGE, or UPDATE statements.

❑ The KEY= option allows you to take advantage of indexing.

Note the following base SAS terms and definitions, which are used in this article:

data set   a SAS file that consists of descriptor information and data values organized as a table of rows (SAS observations) and columns (SAS variables) that can be processed by the SAS System. A data set can be either a data file or a view:

❑ a *data file* contains both data values and descriptor information. A data file can be indexed.

❑ a *view* contains only the information required to retrieve data values. The data is obtained from another file. There are three types of views:

➢ DATA step view, which is a compiled DATA step program that can read from one or more SAS data sets or external files.

➢ SAS/ACCESS view, which defines data formatted by other software products. When a SAS/ACCESS view is processed, the data it accesses remains in its original format.

➢ PROC SQL view, which is a compiled query that obtains values from one or more data files or views or the SQL Procedure Pass-Through Facility.

index	a file that is created when you define an index for a SAS data file. The index stores values for a specific variable or variables and includes information as to the location of those values within observations in the data file. An index provides quicker and more efficient access to observations, because the index provides the ability to locate an observation by value. An index can be a simple index, which is an index of values for one variable, or a composite index, which is an index of two or more variables. A variable that is indexed is referred to as a *key variable*.

The index consists of entries that represent each distinct value for a key variable. The entries are in ascending value order. Each entry consists of a distinct value and one or more unique record identifiers (referred to as a RID) that identifies each observation containing the value. You can think of the RID as an internal observation number. When an index is used to process a request, the software does a binary search of the index and positions the index to the first entry that contains a qualified value. The value's RID or RIDs are then used to access the observation containing the value. |
| *program data vector (PDV)* | a physical area in memory where the SAS System builds a data set, one observation at a time. When a program executes, the software reads values from the input buffer or creates them by executing SAS language statements. The values are assigned to the appropriate variables in the PDV. From here, the software writes the values to a SAS data set as a single observation. |
| *_IORC_* | an automatic variable created by the software when you use the MODIFY statement or the SET statement with the KEY= option. The value of _IORC_ is a numeric return code that indicates the status of the most recent I/O operation performed on an observation in a SAS data set. The return code indicates whether the retrieval for matching observations was successful. That is

❑ after a successful execution, the variable has the value of 0

❑ if an end-of-file error occurred, the value is -1

❑ all other values indicate a non-match occurrence.

Typically, you use _IORC_ in conjunction with the autocall macro %SYSRC, which allows you to specify a mnemonic name that describes a potential outcome of an I/O operation.

Note: In Version 7, the IORCMSG function returns a formatted error message associated with the current value of _IORC_. |
| *%SYSRC* | an autocall macro that provides a convenient way of testing for a specific I/O error condition created by the most recently executed MODIFY statement or SET statement with the KEY= option. %SYSRC returns a numeric value corresponding to the mnemonic string passed to it. The mnemonic is a literal that corresponds to a numeric value of the _IORC_ automatic variable. SAS Institute supplies a library of autocall macros to each site. The autocall facility enables you to invoke a macro without having previously defined that macro in the same SAS program. To use the autocall facility, specify the MAUTOSOURCE system option. |

Note: The DATA= option is used with the PRINT procedure to specify the MASTER data set. If DATA= is not specified, PROC PRINT displays the last created data set that was opened for output, which may not be the one that was just modified, which was opened for update.

Understanding the MODIFY Statement

The MODIFY statement replaces, deletes, or appends observations in an existing data set without creating a copy of the data set. This is referred to as *updating in place*. Because the MODIFY statement does not create a copy of the data set open for update, the processing uses less disk space than does the MERGE, SET, or UPDATE statements, which create a new data set.

If data set options such as KEEP, RENAME, or DROP are used on the modified data set, the options are in effect only during processing. The descriptor portion of a SAS data set opened in update mode cannot be changed.

The data set referenced in the MODIFY statement must also be referenced in the DATA statement. Then, based on the DATA step logic, you can replace, delete, and append new observations. For example, the following simple DATA step uses the MODIFY statement to update the data set INVTY.STOCK by replacing the date values in all observations for variable RECDATE with the current date:

```
data invty.stock;
   modify invty.stock;
   recdate=today();
run;
```

When the SAS System processes the previous DATA step, the following occurs:

1. The MODIFY statement opens SAS data set INVTY.STOCK for update processing.

2. The first observation is read from the data set and the values are written into the PDV. (Variable RECDATE exists in the data set INVTY.STOCK.)

3. The value of variable RECDATE is replaced with the result of the TODAY function.

4. The values in the PDV replace the values in the data set.

5. The process is repeated for each observation, using a sequential access method, until the end-of-file marker is reached.

To further control processing, you can include the following statements in a DATA step execution in conjunction with the MODIFY statement:

REPLACE statement   writes the current observation to the same physical location; that is, replaces it in the data set. An implicit REPLACE statement at the bottom of the DATA step is the default action.

REMOVE statement   deletes the current observation from the data set. The deletion is either a physical or logical deletion, depending on the SAS I/O engine maintaining the data set.

OUTPUT statement   appends the current observation as a new observation to the end of the data set. The data set specified on the MODIFY statement must also be specified in the DATA statement as the output data set.

The processing of the MODIFY statement depends on whether any of those statements are included in the DATA step:

❑ If a REPLACE, REMOVE, or OUTPUT statement is not specified in the DATA step, the software writes the current observation to its original place in the data set…as though a REPLACE statement were the last statement in the DATA step. That is, the MODIFY statement generates a REPLACE statement at the bottom of the DATA step, which is referred to as an *implicit REPLACE*.

❑ If a REPLACE, REMOVE, or OUTPUT statement is included, it overrides the implicit REPLACE.

The REPLACE, REMOVE, and OUTPUT statements are independent of each other. More than one statement can apply to the same observation. Note that if both an OUTPUT statement and a REPLACE or REMOVE statement execute on the same observation, be sure the OUTPUT statement is executed last to keep proper positioning in the index.

The MODIFY statement supports four different access methods:

❑ sequential access

❑ matching access using the BY statement

❑ direct (random) access by observation number using POINT=

❑ direct access by index values using KEY=.

Sequential Access

Sequential access is the simplest form of processing using the MODIFY statement. Sequential access provides less control than the other access methods, but it provides the quickest method for updating all observations in a data set. The syntax for sequential access is:

> **MODIFY** *master-data-set* <(*data-set-option(s)*)> <NOBS=*variable*> <END=*variable*>;

master-data-set	the SAS data set that you want to modify, which can be a SAS data file or a SAS/ACCESS view in Version 6. SQL views and some DBMS engines for the LIBNAME statement are supported beginning with Version 7. (See the chart in **Version 7 and Version 8 Considerations**.) The master data set name must be one of the output data sets named in the DATA statement.
NOBS=	creates and names a temporary variable whose value is usually the total number of observations in the input data set. (For certain SAS views, the SAS System cannot determine the number of observations.)
END=	creates and names a temporary variable that contains an end-of-file indicator. The variable, which is initialized to zero, is set to 1 when the MODIFY statement sequentially processes the last observation of the data set being modified.

In the following code, the SAS System sequentially accesses each observation in the MASTER data set looking for an observation that contains the value u for variable MOD. When an observation is located, the value of variable x is replaced with a 1. The execution of the implicit REPLACE causes the observation to be rewritten with the updated

value. The variable x must exist in the MASTER data set. The MODIFY statement opens the data set in update mode, and the header information for a data set opened for update cannot be modified.

```
data master;
    modify master;
    if mod='u' then x=1;
run;
```

Matching Access Using BY Statement

Matching access matches the value(s) of one or more BY variables in the master data set against the same variables in a second data set, referred to as a *transaction data set.* The syntax for matching access is as follows:

MODIFY *master-data-set* <(*data-set-option*(*s*))> *transaction-data-set* <(*data-set-option*(*s*))>
<NOBS=*variable*><END=*variable*>
<UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;

BY *by-variable*(*s*);

master-data-set	the SAS data set that you want to modify, which can be a SAS data file or a SAS/ACCESS view in Version 6. SQL views and some DBMS engines for the LIBNAME statement are supported beginning with Version 7. (Refer to the chart in **Version 7 and Version 8 Considerations**). The master data sets named must be one of the output data set names in the DATA statement.
transaction-data-set	a SAS data set that provides the values to match and update the master data set.
UPDATEMODE=	specifies how missing values in the transaction data set are handled−that is, whether missing variable values in the transaction data set are to replace existing variable values in the master data set.
	MISSINGCHECK (the default) prevents replacing.
	NOMISSINGCHECK allows replacing.

The BY statement is required. The specified variable(s) must exist in both the master data set and the transaction data set. When using MODIFY with the BY statement, the rules that apply to the UPDATE statement also apply to MODIFY, except that the data sets are not required to be in sorted order.

The MODIFY statement executes as follows:

1. The MODIFY statement opens the specified data set for update processing.

2. MODIFY reads an observation from the transaction data set to a temporary storage location in memory.

3. MODIFY then generates a WHERE statement, specifying the value(s) of the BY variable(s) (for example, PARTNO) from the transaction observation to locate and fetch an observation with a matching BY variable value from the master data set. This observation is placed in the PDV.

4. Value(s) from the transaction observation in temporary storage are used to overlay the values of like-named variables located in the PDV from the master observation.

The following behavior applies to matching access:

❑   When matched on the variable(s) specified in the BY statement, all like-named variables in the master data set are automatically updated with the values from the transaction data set.

❑   If duplicate values for the BY variables exist in the master data set, only the first observation in that BY group is updated; observations with duplicate BY values are ignored. This is because the SAS System locates the observation in the master data set by generating a WHERE statement containing the transaction data set value(s) for the BY variables. The WHERE statement always returns the first occurrence of the BY group for updating in the master data set.

❑   If duplicates exist in a BY group in the transaction data set, the updates are applied one after another to the first matching observation in the master data set. An accumulative statement must be used to accumulate the values from the duplicates. Otherwise, the value from the last duplicate is applied to the matching observation in the master data set.

❑   Because a WHERE statement is generated, neither the master data set nor the transaction data set has to be sorted on the BY variables. However, sorting of both data sets can greatly reduce the I/O to retrieve an observation from the master data set. If the master data set is indexed for the BY variable(s), the generated WHERE statement can use the index.

❑   Missing values from the transaction data set can update the master data set if UPDATEMODE=MISSINGCHECK is the default behavior. If the option UPDATEMODE=NOMISSINGCHECK is not available under your current release, the master data set observation can be updated with a special missing value in the transaction data set observation. (For numeric data, you can differentiate among types of missing values. That is, you can specify up to 27 characters [the underscore (_) and the letters A through Z] to designate different types of missing values.) Another approach is to use the RENAME= data set option to change the names of those variables in the transaction data set so the master data set variable can be set equal to the renamed transaction data set variable.

The following example uses the matching access method to update the master data set INVTY.STOCK using information from the transaction data set WORK.ADDINV as well as to update the date on which stock was received:

```
data invty.stock;
   modify invty.stock addinv;
   by partno;
   recdate=today();
   instock=instock + nwstock;
run;
```

Even though you do not have to sort or index either the transaction data set or the master data set, you can improve performance by

❑   creating an index for the master data set on the variable used as the BY variable
❑   sorting both data sets by the BY variable.

Comparing Matching Access to UPDATE Statement

The MODIFY statement for matching access is almost identical to that of the UPDATE statement. Both require a master and a transaction data set, and both require a BY statement. The MODIFY statement compares to the UPDATE statement as follows:

❑ The MODIFY statement opens the SAS data set specified on the MODIFY and DATA statements in update mode. However, the UPDATE statement opens the specified SAS data set for input, and the SAS data set specified on the DATA statement is opened for output.

❑ The MODIFY statement updates the SAS data set in place. With the UPDATE statement, the original SAS data set is replaced with a copy of the updated version.

❑ The MODIFY statement results in an implicit REPLACE statement being executed at the time the DATA step iterates. UPDATE causes an implicit OUTPUT statement to be executed at the time the DATA step iterates.

The difference between the implicit REPLACE and the implicit OUTPUT statements is illustrated in the following example, which uses two data sets: MASTER and TRANS. Their DATA steps are shown below:

```
data master;                          data trans;
input ssn : $11. nickname $;          input ssn : $11. nickname $;
datalines;                            datalines;
134-56-9094 Megan                     134-56-9094 Meg
160-58-1223 Kathryn                   142-67-9888 Bill
161-60-5881 Joshua                    160-58-1223 Kate
;                                     161-60-5881 .
                                      ;
```

To process the previous data, which is in sorted order, the following UPDATE statement would be successful. UPDATE generates an implicit OUTPUT statement, which appends the current contents of the program data vector to the end of the data set regardless of whether there is a match on the BY variable(s).

```
data master;
    update master trans updatemode=nomissingcheck;
    by ssn;
run;
```

However, if you changed the UPDATE statement to a MODIFY statement as shown below, it would fail. The generated WHERE statement would find no match for SSN 142-67-9888, and the implicit REPLACE statement at the bottom of the DATA step tries to replace an observation that it was unable to retrieve.

```
data master;
    modify master trans updatemode=nomissingcheck;
    by ssn;
run;
```

The SAS log reflects the value 1230013 in the automatic variable _IORC_, because a match for the second observation in the TRANS data set does not exist in the MASTER data set. Because an error occurs, the automatic variable _ERROR_ is set to a value of 1.

9

```
ERROR: The TRANSACTION data set observation does not exist on the MASTER data set.
ERROR: No matching observation was found in MASTER data set.
ssn=142-67-9888 nickname=Bill FIRST.ssn=1 LAST.ssn=1 _ERROR_=1 _IORC_=1230013 _N_=2
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 1 observations read from the dataset WORK.MASTER.
NOTE: The data set WORK.MASTER has been updated.  There were 1 observations
rewritten,      0 observations added and 0 observations deleted.
NOTE: There were 3 observations read from the dataset WORK.TRANS.
```

To prevent the error caused by the implicit REPLACE on a non-match record, use a conditional IF statement so that a REPLACE is performed only when a match is located. The automatic variable _IORC_ holds a value of zero on a successful I/O operation.

```
data master;
   modify master trans updatemode=nomissingcheck;
   by ssn;
   if _iorc_=0 then replace;
run;
```

Note that to prevent displaying the non-match record in the SAS log, reset the automatic variable _ERROR_ equal to zero. (Error checking is discussed later in this article.)

```
data master;
   modify master trans updatemode=nomissingcheck;
   by ssn;
   if _iorc_=0 then replace;
   else _error_=0;
run;
```

Direct (Random) Access by Observation Number Using POINT=

Direct (random) access by observation number requires that you use POINT= to identify the observation to be updated. POINT= specifies a variable from another data source (not the master data set) or one you define in the DATA step whose value is the number of an observation that you want to modify in the master data set. MODIFY uses the values of the POINT= variable to retrieve observations in the data set that you want to modify. The syntax for direct (random) access by observation number is as follows:

MODIFY *master-data-set* <(*data-set-option(s)*)> <NOBS=*variable*> POINT=*variable*;

master-data-set	the SAS data set that you want to modify, which must be a SAS data file in Version 6. Version 7 supports an SQL view that references a SAS data file. The same data set name must be one of the output data set names in the DATA statement.
NOBS=	creates and names a temporary variable whose value is the total number of observations in the input data set.
POINT=	specifies a variable whose value is the number of the observation to read. Note that the concept of an observation number has applied to non-compressed SAS data files only. In Version 7, you can use POINT= with a compressed data file if it is created with the $POINTOBS= data set option set to YES. (For Version 8, the data set option is POINTOBS=.)

10

To illustrate direct access by observation number, assume you have a data set named NEWP. NEWP has variable TOOL_OBS containing the observation number of each tool in the tool company's master data set INVTY.STOCK and variable NEWPRICE containing the new price for each tool.

The following example uses the information in data set NEWP to update data set INVTY.STOCK. NEWP is specified in the SET statement, which reads values to be supplied for the TOOL_OBS and NEWPRICE variables. Variable TOOL_OBS is specified as the value of the POINT= option. Variable NEWPRICE is specified in the assignment statement to replace existing values of PRICE in INVTY.STOCK. As the SET statement executes, values of TOOL_OBS are read from NEWP, placed in the PDV, and then used by the MODIFY statement to retrieve observations directly from INVTY.STOCK. The observation number in TOOL_OBS is used as a key to allow direct retrieval of observations.

```
data invty.stock;
    set newp;
    modify invty.stock point=tool_obs;
    price=newprice;
    recdate=today():
run;
```

Direct Access by Index Values Using KEY=

Direct access by index values requires that you use KEY= to specify an existing index. The software then retrieves observations in the data set through the index based on the index values. The syntax for direct access by index values is as follows:

> **MODIFY** *master-data-set* <(*data-set-option(s)*)> KEY=*index* </UNIQUE>
> <NOBS=*variable*> <END=*variable*>;

master-data-set the SAS data set that you want to modify, which must be a SAS data file in Version 6. Some DBMS engines for the LIBNAME statement are supported beginning with Version 7. (Refer to the chart in **Version 7 and Version 8 Considerations**). The master data set name must be one of the output data set names in the DATA statement. The master data set must have an index defined, which can be either simple or composite.

KEY= required to specify the name of either a simple or composite index that exists for the master data set.

Direct access by index values lets you supply a lookup value from a secondary data source such as another SAS data set. An observation is read using a SET statement to supply a lookup value, which is then used as a key to search the master data set to locate the observation. The search is performed through an index created for that data set. You specify the index name with the KEY= option. Once the observation is located, you can assign variables new values and perform other processing on the observation prior to the implicit REPLACE. If the observation is not located, an appropriate action such as OUTPUT is performed. The SAS data set supplying the lookup value (specified on the SET statement) must have variables defined with the same names as those defined to the index specified on the KEY= option.

The following example uses the KEY= option to specify the index with which to identify observations for retrieval by matching the values of variable PARTNO from data set WORK.ADDINV with the indexed values of variable PARTNO in data set INVTY.STOCK:

```
data invty.stock;
   set addinv;
   modify invty.stock key=partno;
   if _iorc_=0 then do;
     instock=instock+nwstock;
     recdate=today();
     replace;
   end;
   else _error_=0;
run;
```

Comparing Matching Access to Direct Access by Index Values

The matching access method (which uses the BY statement) and the direct access method by using an index (which uses the KEY= option) compare as follows:

MODIFY with BY Statement	MODIFY with KEY= Option
Matching observations in the master data set are retrieved by a generated WHERE statement. The WHERE statement can use an index created on the master data set.	Matching observations in the master data set are retrieved through the index specified with the KEY= option.
If duplicate BY values exist in the master data set, only the first occurrence is updated.	Can update duplicate values of the key variable(s) in the master data set if a DO LOOP is coded to force execution of the KEY= option on the master data set until a non-match condition occurs. See **Using Modify with Duplicate Key Variables in Master Data Set** for more information.
If duplicate BY values exist in the transaction data set, they are applied one after another to the same observation in the master data set unless you write an accumulation statement.Otherwise, the last duplicate is applied.	All consecutive duplicates in the transaction are applied by using the UNIQUE option. If the UNIQUE option is not specified, only the first consecutive duplicate is applied. All non-consecutive duplicates are applied to the observation in the master data set one after the other, so only the last duplicate is applied. See **Using MODIFY with Duplicate Key Variables in Transaction Data Set** for more information.
Performs automatic update of like-named variables in the matching observation of the master data set.	Does not perform automatic update of like-named variables in the matching observation of the master data set. See **Performing Automatic Update of Like-Named Variables Using KEY=** for an explanation.

Monitoring Update Processing

The best way to test for values of _IORC_ is with the %SYSRC autocall macro, which allows you to specify a mnemonic name that describes a potential outcome of an I/O operation. The mnemonic codes provided by %SYSRC are part of the SAS autocall macro library. Each mnemonic code represents a specific condition to determine the success of retrieving matching observations. Below is a list of the most common mnemonics available to %SYSRC:

Access Method	Mnemonic	Description
MODIFY with KEY= option	_DSENOM	Specifies that the master data set does not contain the observation.
MODIFY with BY statement	_DSENMR	Specifies that the transaction data set observation does not exist in the master data set.
MODIFY with BY statement	_DSEMTR	Specifies that multiple transaction data set observations with a given BY value do not exist in the master data set.
MODIFY with KEY= or BY statement	_SOK	Specifies that the observation was located.

The complete list of mnemonics and their current-corresponding numeric values and descriptions for _IORC_ are contained in the SYSRC member of the autocall macro library. You can view the contents of the SYSRC member in the SAS log by submitting the following code:

```
options source2;
%include sasautos(sysrc);
run;
```

The following example uses %SYSRC to test for successful matching of observations. The example uses the mnemonic _SOK, which indicates the I/O operation was successful.

```
data master;
    modify master trans updatemode=nomissingcheck;
    by ssn;
    if _iorc_=%sysrc(_sok) then replace;
    else _error_ =0;
run;
```

Note: Beginning with Version 7, the IORCMSG function returns a formatted error message associated with the current value of _IORC_.

Performing Automatic Update of Like-Named Variables Using KEY=

Unlike using the BY statement, automatic update of like-named variables does not occur with the KEY= option. However, when you have many variables to update, it is convenient to have this ability, which you can achieve by using the SET statement and the POINT= option in conjunction with the MODIFY statement. Observe the order of the following statements:

```
data master;
    set trans;
    modify master key=ssn;
    i+1;
    if _iorc_=%sysrc(_sok) then do;
```

```
        set trans point=i;
        replace;
    end;
    else _error_=0;
run;
```

The SET statement loads the values of the TRANS variables into the PDV. Next, the MODIFY statement is executed. The values of all like-named variables in the PDV are overlaid with the values from the MASTER data set when a fetch is successful. If a REPLACE or OUTPUT statement were executed at this point, MASTER would be updated with the MASTER values in the PDV for the like-named variables.

However, the above code issues a second SET statement, using the POINT= option (with the counter i+1) to read the same observation in the TRANS data set. This effectively updates the PDV with the like-named variables from the TRANS data set. As discussed earlier, the POINT= option specifies a variable from another data source whose value is the number of an observation that you want to modify in the master data set.

If you have only a few variables to update, you can rely on the normal processing of the KEY= option, along with explicit assignment statements. When using KEY=, the TRANS data set must have variable(s) with the same name as those key variable(s) used to create the index on the MASTER data set. The TRANS data set is read with the SET statement and the value(s) for the key variables(s) are loaded into the PDV. The value loaded into the PDV is used against the index values of the MASTER data set specified by the KEY= option to fetch the matching observation.

No automatic update of the values for like-named variables occurs between the TRANS and MASTER data set. So in order to change the value of a given variable, an explicit assignment of the MASTER data set variable is made.

Explicit assignment of like-named variable values presents a problem, however, because the variables you want to assign have the same name in both the MASTER and the TRANS data sets. You can use the RENAME= data set option to rename the variables in one data set before assigning the values.

This use of the RENAME= data set option is illustrated in the following example, in which both data sets have like-named variables other than the key variables.   An explicit assignment must be coded to update the variable NICKNAME in the MASTER data set with the values of the variable NICKNAME in the TRANS data set. To make the explicit assignment, the RENAME= data set option is used to rename the variable NICKNAME to TNICKNAM in the TRANS data set. The NICKNAME variable in the MASTER data set is set equal to the TNICKNAM variable in the TRANS data set.

| ```
data master(index=(ssn));
input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;
``` | ```
data trans;
input ssn : $11. nickname $;
datalines;
161-60-5881 Josh
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;
``` |

```
data master;
    set trans(rename=(nickname=tnicknam));
    modify master key=ssn;
    if _iorc_=%sysrc(_sok) then do;
        nickname=tnicknam;
        replace;
    end;
    else _error_ = 0;
run;
```

14

Using MODIFY in a SAS/SHARE Environment

In a SAS/SHARE environment, the control level for updating using the MODIFY statement is dependent on the access method being used:

❑ A MODIFY statement *without* the POINT= option or KEY= option has a control level of RECORD, which means that other tasks can read or update the data set but no SAS task can open it for output.

❑ A MODIFY statement *with* the POINT= option or KEY= option has a control level of MEMBER, which means that other tasks cannot access the data.

Comparing MODIFY to SET Statement

The differences between the MODIFY and SET statements are listed below.

MODIFY Statement	SET Statement
Updates the original data set opened in update mode without creating a copy.	Creates a temporary data set with the updates applied. Upon successful completion of the DATA step, the original data set is replaced with the temporary data set if the DATA statement and the SET statement refer to the same data set name. If the data set name is different, the temporary data set is renamed to that existing on the DATA statement.
Generates an implicit REPLACE statement at the bottom of the DATA step.	Generates an implicit OUTPUT statement at the bottom of the DATA step.
Variables cannot be added or dropped from the modified data set. The header information for a data set opened for update cannot be modified.	The output data set can contain new variables and variables existing in the lookup and primary data sets.
The REPLACE statement updates the current observation of the original data set.	The REPLACE statement is not valid since a copy of the original observation is updated and OUTPUT.
The original data set is updated with missing values from the transaction data set using the UPDATEMODE= option.	The UPDATEMODE= option is not valid since a copy of the original observation is updated with missing values and OUTPUT.

Understanding the KEY= Option

Both the MODIFY and SET statements support the KEY= option, which allows applications to specify an index in order to retrieve particular observations in a data set based on the indexed values. Using KEY= with the MODIFY statement is explained earlier in this article for the direct access by index values method. The next topic explains using KEY= with the SET statement. Then subsequent topics cover issues regarding KEY= for both MODIFY and SET statements such as when duplicate values exist for the key variable.

Using KEY= with SET Statement

The topic does not explain the SET statement in general but does explain using the KEY= option in the SET statement. For more details on the SET statement, see *SAS Language Reference: Dictionary*.

The syntax for the SET statement is as follows:

> **SET** *data-set*(*s*) <(*data-set option*(*s*))> <**NOBS**=*variable*> <**END**=*variable*>
> <**POINT**=*variable* | **KEY**=*index*> </**UNIQUE**>;

KEY= provides non-sequential access to observations in a data set through an index created for one or more variables. The index can be either simple or composite. You cannot, however, use KEY= with the POINT= option.

As with the MODIFY statement, using the _IORC_ automatic variable in conjunction with the %SYSRC autocall macro provides more error-handling information. When you use the SET statement with KEY=, _IORC_ is created and set to a return code that indicates the status of the most recent I/O operation performed on an observation in the data set. If the KEY= value is not found in the master data set, _IORC_ returns a numeric value that corresponds to the %SYSRC autocall macro's mnemonic _DSENOM and the automatic variable _ERROR_ is set to 1.

Note: When issuing multiple SET statements using the KEY= option, you must perform separate error checking of the automatic variable _IORC_on each data set. That is, in order to produce an accurate output data set, test the _IORC_ variable following each SET statement using the KEY= option.

Using the SET statement with KEY= supports the concept of lookup values. The lookup value can be provided through another SAS data set, an external file, a view, or a FRAME entry. For Version 6, the lookup data set must be a native SAS data set. Version 7 supports SAS/ACCESS views and the DBMS engine for the LIBNAME statement.

```
data combine;
    set lookup;
    set primary key=partno;
    select(_iorc_);
        when (%sysrc(_sok)) do;
            output;
        end;
        when (%sysrc(_dsenom)) do;
            _error_ = 0;
        end;
        otherwise;
    end;
run;
```

The lookup value from the lookup data source is used as a key to locate observations in the primary or MASTER data set. This primary data set must be indexed (either simple or composite) and specified with the KEY= option. The lookup values must be provided through variable(s) named the same as the key variable(s) in the primary data set. For example, if the lookup data source is a SAS data set, it must contain variables(s) with the same name as those defined to the index of the primary data set. Once the observation is successfully fetched from the primary data set, an action such as OUTPUT can be performed on the observation. (See **Combining Data Sets with Non-Matching Observations** for information on behavior when the fetch is not successful.)

Note: if a DROP or KEEP is not specified, using the KEY= option combines all variables on the lookup data set and the primary data set. That is, the output data set contains not only the variables from the lookup data set, but also all the variables in the primary data set. This does not happen with KEY= and the MODIFY statement.

Handling Duplicate Values for Key Variable

When specifying the KEY= option to specify that an index should be used, you may encounter situations where multiple observations contain the same values for the key variable in either the data set being accessed with the KEY= option, or the data set supplying the index values, or both.

Using MODIFY with Duplicate Key Values in Transaction Data Set

Having duplicate values for the key variable in the transaction data set can affect the results when they are applied to the master data set. In addition, whether the duplicate values are consecutive or non-consecutive also affect the results.

For example, consider the following two programs, which are identical except for the order of the duplicate values for the key variable SSN in the TRANS data set. Notice that in Program 1, the TRANS data set has duplicate key values of 160-58-1223, but they are not consecutive; Program 2, however, also has the duplicate values, but they are consecutive.

Program 1	Program 2
| ```
data master(index=(ssn));
input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;

data trans;
input ssn : $11. tnicknam $;
datalines;
 161-60-5881 Josh
160-58-1223 Kathy
134-56-9094 Meg
142-67-9888 Bill
160-58-1223 Kate
;

data master;
set trans;
modify master key=ssn;
if _iorc_=%sysrc(_sok) then do;
   nickname=tnicknam;
   replace;
end;
else_error_=0;
run;
``` | ```
data master(index=(ssn));
input ssn : $11. nickname $;
datalines;
161-60-5881 Joshua
160-58-1223 Kathryn
134-56-9094 Megan
;

data trans;
input ssn : $11. tnicknam $;
datalines;
161-60-5881 Josh
160-58-1223 Kathy
160-58-1223 Kate
134-56-9094 Meg
142-67-9888 Bill
;

data master;
set trans;
modify master key=ssn;
if _iorc_=%sysrc(_sok) then do;
   nickname=tnicknam;
   replace;
end;
else_error_=0;
run;
``` |

The following PRINT procedure shows the results of the above programs.

Note: The DATA= option is used with PROC PRINT to specify the MASTER data set. If DATA= is not specified, PROC PRINT displays the last created data set opened for output, which may not be the one just modified, which was opened for update.

```
proc print data=master;
run;
```

❑ From Program 1, the value for NICKNAME in the resulting MASTER data set is `Kate`, which is the value of the **last** observation in the TRANS data set with the corresponding social security number.

```
OBS    SSN              NICKNAME

1      161-60-5881      Josh
2      160-58-1223      Kate
3      134-56-9094      Meg
```

❑ From Program 2, the consecutive value of TNICKNAM is `Kate` for SSN in the TRANS data set. That value does not update the MASTER data set. The consecutive record of `Kate` actually causes a non-match to occur, and only the **first** observation in TRANS with the corresponding value is applied.

```
OBS    SSN              NICKNAME

1      161-60-5881      Josh
2      160-58-1223      Kathy
3      134-56-9094      Meg
```

Whether the duplicate key values in TRANS are consecutive or non-consecutive affects the results in the MASTER data set, because when searching the index, the software begins at the top of the index only when the value of the key variable changes in the TRANS data set between executions of the statement with the KEY= option.

If the duplicate key values in the TRANS data set are *not consecutive*, the search starts from the top of the index in both searches for social security number `160-58-1223`. Therefore, the one and only matched value in the MASTER data set is located and updated both times, so the value in the **last** duplicate observation is applied.

For *consecutive* duplicate key variable values in the TRANS data set, when the first value of `160-58-1223` is supplied by the TRANS data set, the value for the key variable changes -- the index search of the MASTER data set begins at the top and the observation is found. The NICKNAME variable in MASTER is updated to `Kathy`. When the consecutive value of `160-58-1223` is supplied by the TRANS data set, the value does not change. Therefore, the search does not begin at the top of the MASTER index; rather, it begins from the current position in the index structure. The observation is not found, and an update is not performed for the subsequent duplicate observations. Only the value in the **first** duplicate is applied.

To assure the same result whether duplicates in TRANS are consecutive or not, you can force the software to begin the search at the top of the index by specifying the UNIQUE option in the MODIFY statement. The UNIQUE option specifies to search from the top of the index, regardless of whether the key variable value changes from one iteration to the next.

The following code uses the UNIQUE option to produce the same results for each program. For readability purposes, the IF statement is coded as a SELECT statement instead.

```
data master(index=(ssn));            data trans;
input ssn : $11. Nickname $;         input ssn : $11. tnicknam $;
datalines;                           datalines;
161-60-5881 Joshua                   161-60-5881 Josh
160-58-1223 Kathryn                  160-58-1223 Kathy
134-56-9094 Megan                    160-58-1223 Kate
;                                     134-56-9094 Meg
                                      142-67-9888 Bill
                                      ;
```

```
data master;
set trans;
modify master key=ssn/unique;
select (_iorc_);
   when (%sysrc(_sok)) do;
      nickname=tnicknam;
      replace master;
   end;
   when (%sysrc(_dsenom)) do;
      _error_=0;
   end;
   otherwise;
end;

proc print data=master;
run;
```

```
OBS   SSN             NICKNAME

1     161-60-5881     Josh
2     160-58-1223     Kate
3     134-56-9094     Meg
```

Using MODIFY with Duplicate Key Values in Master Data Set

Having duplicate values for the key variable in the master data set can affect the results if you want to update all duplicates in the master with a corresponding unique value in the transaction data set.

For example, the TRANS data set observation with the SSN value of 161-60-5881 only updates the first corresponding observation of the SSN variable in the MASTER data set to the value of Josh. Notice that in the PROC PRINT results, the bolded record is updated.

```
data master(index=(ssn));            data trans;
input ssn : $11. nickname $;         input ssn : $11. nickname $;
datalines;                           datalines;
161-60-5881 Joshua                   161-60-5881 Josh
161-60-5881 Joshua                   160-58-1223 Kathy
160-58-1223 Kathryn                  134-56-9094 Meg
134-56-9094 Megan                    142-67-9888 Bill
;                                     ;
```

19

```
data master;
    set trans(rename=(nickname=tnicknam));
    modify master key=ssn;
    select (_iorc_);
        when (%sysrc(_sok)) do;
            nickname=tnicknam;
            replace master;
        end;
        when (%sysrc(_dsenom)) do;
            _error_=0;
         end;
        otherwise;
     end;

    proc print data=master;
    run;
```

```
OBS    SSN            NICKNAME

1      161-60-5881    Josh
2      161-60-5881    Joshua
3      160-58-1223    Kathy
4      134-56-9094    Meg
```

To update variable NICKNAME for multiple observations in MASTER with the unique, corresponding observation in TRANS, force a continuous search of the MASTER index file using a DO LOOP until a non-match condition is encountered. Notice that the bolded observations from the PRINT procedure indicate all corresponding records of the SSN 161-60-5881 are now updated in MASTER.

```
data master(index=(ssn));             data trans;
input ssn : $11. nickname $;          input ssn : $11. tnicknam $;
datalines;                            datalines;
161-60-5881 Joshua                    161-60-5881 Josh
161-60-5881 Joshua                    160-58-1223 Kathy
160-58-1223 Kathryn                   134-56-9094 Meg
134-56-9094 Megan                     142-67-9888 Bill
;                                     ;
```

```
data master;
set trans;
do until (_iorc_=%sysrc(_dsenom));
    modify master key=ssn;
    select (_iorc_);
        when (%sysrc(_sok)) do;
            nickname=tnicknam;
            replace master;
        end;

        when (%sysrc(_dsenom)) do;
            _error_=0;
        end;
        otherwise;
    end;
end;

proc print data=master;
run;
```

```
OBS     SSN             NICKNAME

1       161-60-5881     Josh
2       161-60-5881     Josh
3       160-58-1223     Kathy
4       134-56-9094     Meg
```

Using MODIFY with Duplicate Key Values in Master Data Set and Transaction Data Set

If duplicate values of the key variable exist in both the master data set and the transaction data set, updating each corresponding master observation with each corresponding transaction observation requires additional BY processing.

When consecutive duplicate values of SSN are supplied by the transaction data set, you have seen how the UNIQUE option forces a search to start from the top of the index. For this situation, however, if you use the UNIQUE option, the same observation in the MASTER data set is located and updated. Therefore, the duplicate observations in MASTER are not updated.

You must force the search to start at the top of the index by changing the key variable without using the UNIQUE option, and it must happen between each observation of the same SSN. Because you need to change the key variable between consecutive values of SSN in the TRANS data set, use BY processing to determine if more observations of the same SSN in TRANS are supplied. Therefore, the TRANS data set must be sorted for BY processing. To continue the search in the index for all corresponding matches, use a DO UNTIL statement to process until a non-match situation is encountered in MASTER.

The following example shows how to temporarily change the key variable value to a value not located in the MASTER data set. For a full discussion of this example, refer Example 4.6 in *Combining and Modifying SAS Data Sets: Examples.*

```
data master(index=(ssn));          data trans;
input ssn : $11. nickname $;       input ssn : $11. tnicknam $;
datalines;                         datalines;
161-60-5881 Joshua                 161-60-5881 Josh
161-60-5881 Joshua                 160-58-1223 Kathy
160-58-1223 Kathryn                160-58-1223 Kate
160-58-1223 Kathryn                134-56-9094 Meg
134-56-9094 Megan                  142-67-9888 Bill
;                                  ;
```

```
proc sort data=trans;
    by ssn;

data master;
set trans;
by ssn;
dummy=0;
do until (_iorc_=%sysrc(_dsenom));
if dummy then ssn='999-99-9999';
modify master key=ssn;
    select (_iorc_);
        when (%sysrc(_sok)) do;
            nickname=tnicknam;
            replace master;
        end;
        when (%sysrc(_dsenom)) do;
           _error=0;
           if not last.ssn and not dummy then do;
              dummy=1;
              _iorc_=0;
              end;
        end;
        otherwise;
    end;
  end;

proc print data=master;
run;
```

OBS	SSN	NICKNAME
1	161-60-5881	Josh
2	161-60-5881	Josh
3	**160-58-1223**	**Kate**
4	**160-58-1223**	**Kate**
5	134-56-9094	Meg

Using SET with Duplicate Key Values in Lookup Data Set

Like KEY= with MODIFY, execution of the SET statement with the KEY= option forces a search to begin at the top of the index only when the value of the KEY= variable changes in the lookup data set between fetch operations. If duplicate values for the key variable exist consecutively in the lookup data set, the value does not change between executions. Therefore, to force the search to begin at the top, use the UNIQUE option. With non-consecutive duplicate values, the search begins at the top of the index structure.

Using SET with Duplicate Key Values in Primary Data Set

Duplicate values for the key variable in the primary data set have the same effect with SET as previously described with MODIFY. The software locates and returns the first occurrence of the key variable in the index, unless the statement with the KEY= option is forced to execute again using the same key value. As with MODIFY, you can force the continuation of this search of the primary index structure with the DO UNTIL statement.   Notice that the bolded observations from the PRINT procedure indicate that all corresponding observations for PARTNO A066 exist in the output data set.

```
   data lookup;                          data primary(index=(partno));
   input partno $ quantity;              input partno $ desc $;
   datalines;                            datalines;
   A066 8                                A021 motor
   A220 2                                A033 bolt
   A812 10                               A043 nut
   ;                                     A055 radiator
                                         A066 hose
                                         A066 hose2
                                         A066 hose3
                                         A078 knob
                                         A220 switch
                                         A223 bridge
                                         ;
```

```
data combine;
   set lookup;
   do until(_iorc_=%sysrc(_dsenom));
   set primary key=partno;
   select(_iorc_);
      when (%sysrc(_sok)) do;
         output;
      end;
      when (%sysrc(_dsenom)) do;
         _error_=0;
      end;
      otherwise;
   end;
end;

proc print data=combine;
run;
```

OBS	PARTNO	QUANTITY	DESC
1	**A066**	**8**	**hose**
2	**A066**	**8**	**hose2**
3	**A066**	**8**	**hose3**
4	A220	2	switch

Using SET with Duplicate Key Values in Primary Data Set and Lookup Data Set

To combine data sets when duplicate values for a key variable exist in both the primary data set and the lookup data set, use the same concept discussed for the MODIFY statement. See **Using MODIFY with Duplicate Key Values in Master Data Set and Transaction Data Set**.

Note: Rather than using the SET statement, consider using the SQL procedure, which gives the same results with simplified code as shown in the following PROC SQL example. Refer to Example 4.6 in *Combining and Modifying SAS Data Sets: Examples* for more information on both techniques.

```
     data lookup;                              data primary(index=(partno));
     input partno $ district;                  input partno $ desc $;
     datalines;                                datalines;
     A066 1                                    A021 motor
     A066 2                                    A033 bolt
     A220 2                                    A043 nut
     A812 10                                   A055 radiator
     ;                                         A066 hose
                                               A066 hose2
                                               A066 hose3
                                               A078 knob
                                               A220 switch
                                               223 bridge
                                               ;
```

```
     proc sql;
     create table shiplst as
     select a.*, b.desc
     from lookup as a, primary as b
     where a.partno=b.partno;
     quit;

     proc print data=shiplst;
     run;
```

```
     OBS      PARTNO        DISTRICT      DESC

     1        A066          1             hose
     2        A066          2             hose
     3        A066          1             hose2
     4        A066          2             hose2
     5        A066          1             hose3
     6        A066          2             hose3
     7        A220          2             switch
```

Combining Data Sets with Non-Matching Observations

Each time the SET statement is executed, one observation is read from the current data set opened for input into the
PDV. SET reads all variables and all observations from the input data sets unless you specify to do otherwise. You
can use multiple SET statements to perform one-to-one reading (also called one-to-one matching) of the specified
data sets. The new data set contains all the variables from all the input data sets. The number of observations in the
new data set is the number of observations in the smallest original data set. If the data sets contain common
variables, the values that are read in from the last data set replace those read in from previous ones.

The rules for combining SAS data sets when using multiple SET statements dictate that existing variables are not
reinitialized to missing for each iteration. The non-missing value on the output data set is the retained value in the
PDV from the most recent match that occurred.

In the following example, the observation in the lookup data set with the value A812 for PARTNO is not located in
the primary data set. The DESC variable loaded in the PDV currently holds the value switch from the last successful
match for PARTNO value A220. Because a match does not occur, this DESC variable value is not overwritten in the
PDV with a new value for PARTNO A812.

```
data lookup;                          data primary(index=(partno));
input partno $ quantity;              input partno $ desc $;
datalines;                            datalines;
A066 8                                A021 motor
A220 2                                A033 bolt
A812 10                               A043 nut
;                                     A055 radiator
                                      A066 hose
                                      A078 knob
                                      A220 switch
                                      A223 bridge
                                      ;
```

```
data combine;
set lookup;
set primary key=partno;
select(_iorc_);
   when (%sysrc(_sok)) do;
      output;
   end;
   when (%sysrc(_dsenom)) do;
      _error_=0;
      *desc = ' ';
      output;
   end;
   otherwise;
end;

proc print data=combine;
run;
```

```
OBS     PARTNO     QUANTITY     DESC

1       A066          8         hose
2       A220          2         switch
3       A812         10         switch
```

To output a missing value for DESC in the output data set, you would have to execute an explicit assignment statement, setting DESC equal to missing when an observation is not located in the primary data set. This overwrites the retained value in the PDV with the desired value of missing. In the example code above, uncomment the DESC= code for the desired output.

Adding Variables to the Output Data Set

Using the SET statement to open a data set for output enables you to add a new variable through an assignment statement to an output data set. When this occurs, the new variable is reset to missing for each iteration of the DATA step. The following example shows that the STATUS variable is automatically set to missing without assigning a missing value, when a non-match occurs in an observation from the primary data set.

```
     data lookup;                          data primary(index=(partno));
     input partno $ quantity;              input partno $ desc $;
     datalines;                            datalines;
     A066 8                                A021 motor
     A220 2                                A033 bolt
     A812 10                               A043 nut
     ;                                     A055 radiator
                                           A066 hose
                                           A078 knob
                                           A220 switch
                                           A223 bridge
                                           ;
```

```
     data combine;
     set lookup;
     set primary key=partno;
     select(_iorc_);
        when (%sysrc(_sok)) do;
           status="available";
           output;
        end;
        when (%sysrc(_dsenom)) do;
           desc=' ';
           _error_=0;
           output;
        end;
        otherwise;
     end;

     proc print data=combine;
     run;
```

```
   OBS     PARTNO     QUANTITY     DESC     STATUS


   1       A066          8         hose     available
   2       A220          2         switch   available
   3       A812         10
```

Using SET with KEY= in SAS/SHARE Environment

In a SAS/SHARE environment, the control level for updating using SET with KEY= is MEMBER. This means that other tasks can read the data set, but no SAS task can open it for update or output.

Version 7 and Version 8 Considerations

Using the DBKEY= SAS/ACCESS Data Set Option with KEY=

SAS/ACCESS software provides several data set options to improve performance when accessing data in a DBMS. The DBKEY= data set option enables you to specify variable names to use as an index. The software uses the specified variable name(s) as an index by constructing and passing a WHERE expression to the DBMS.

DBKEY= can be used to improve performance just as indexing a SAS data file can improve performance. For example, to improve the performance, the DBKEY= option can be used in a DATA step with the KEY= option in a SET statement. Note that you must specify the keyword DBKEY as the value of the KEY= option.

The following DATA step creates a new data file by joining data file KEYVALUES with the DBMS table MYTABLE. The software uses the variable DEPTNO with the DBKEY= data set option to cause a WHERE expression to be passed to the DBMS. Performance benefits may occur if the DBMS optimizer selects to optimize the WHERE expression with an existing index on the DBMS table.

```
data sasuser.new;
    set sasuser.keyvalues;
    set dblib.mytable (dbkey=deptno) key=dbkey;
run;
```

Open Mode of Views for both MODIFY and SET

Under Version 6, only SAS/ACCESS views are open for update under sequential and matching access methods. All the views are open for input, which allows read-only access to the data set. Version 7 and beyond allow this same functionality but also add update access for SQL views with the following restrictions.

The SQL view must

❑ Reference a single data file or view

❑ Not reference a DBMS via the pass-through facility

❑ Not contain derived variable names to update.

The enhancement of DBMS engines on the LIBNAME statement allows dynamic connection to the DBMS server through a libref. The DBMS engines also allow data sets to be opened for update for sequential, matching, and direct access. The following libref of DB on the LIBNAME statement uses the DB2 engine to dynamically connect to a DB2 server.

```
Libname DB DB2 ssid=DB2;
```

The chart below shows the open mode for SAS/ACCESS views, SQL views, PASS-THROUGH views, DATA views, and DBMS engines on the LIBNAME statement for each access method. The MODIFY statement can be used only for an access method that specifies an open mode of *update*. The SET statement can be used for an access method that specifies an open mode of either *update* or *input*.

	DBMS Engine	SQL view		Pass-through		Access View		DATA Step View	
	V7	V6	V7	V6	V7	V6	V7	V6	V7
Sequential Access	update	input	update *	input	input	update	update	input	input
Matching Access (BY)	update	input	update *	input	input	update	update	input	input
Random Access (POINT=)	n/a	input #	update #	n/a	n/a	n/a	n/a	input	input
Direct Access (KEY=)	update	n/a	n/a	n/a	n/a	n/a	input	n/a	n/a

* single data set reference

must reference a native SAS data set

Other Enhancements

Integrity constraints are new beginning in Version 7. Integrity constraints are assigned by the user to define a set of rules for data file modifications that guarantee the validity of the data as specified by those rules. They restrict the data values that can be updated or inserted into a table. Integrity constraints are enforced by the SAS System for each update, delete, or add. The MODIFY statement honors the defined integrity constraints.

Generation data sets, also new beginning in Version 7, are historical versions of SAS data files, SAS views, and SAS/ACCESS views. The SAS System can now keep several generations of a data set. The MODIFY statement also supports this feature.

The IORCMSG function and the POINTOBS= data set option are also new. The IORCMSG function returns a formatted error message associated with the current value of _IORC_ . The POINTOBS= option ($POINTOBS= in Version 7) allows observations to be accessed on compressed data sets.

Conclusion

The MODIFY statement allows you to replace, delete, and append observations in the original data set without creating a copy. Therefore, MODIFY uses less disk space than the SET, MERGE, and UPDATE statements. The different access methods for MODIFY are

- ❑ sequential access

- ❑ matching access using the BY statement

- ❑ direct (random) access by observation number using the POINT= option

- ❑ direct access by index values using the KEY= option.

You cannot use MODIFY with DATA step views, and MODIFY cannot modify the descriptor portion of a SAS data set, such as adding a variable.

When processing MODIFY for matching access using the BY statement, the SAS System updates only the **first** matching observation in the master if there are duplicates. All like-named variables in the master data set are updated automatically from the transaction data set.

You can use the KEY= option for both the MODIFY statement and the SET statement to specify either a simple index or a composite index. The SAS System retrieves observations based on the index values. The rules for using KEY= are the same for both the MODIFY statement and the SET statement, except for the way the variables are reinitialized in the PDV between iterations, and the fact that MODIFY updates in place.

When using the KEY= option for either the MODIFY statement or the SET statement, you need to be aware of any duplicate values for the key variable in any data set involved. If duplicates exist in the data, be sure the source code properly handles the situation.

- ❑ When duplicate values for the key variable exist in the updating data set and unique values of the key variable exist in the modified data set, use the UNIQUE option to process all duplicates in the updating data set.

❑ When duplicate values for the key variable exist in the modified data set and unique values for the key variable exist in the updating data set, use a DO LOOP to continue a search in the index structure or, if possible, eliminate the duplicates prior to processing.

When using the KEY= option and a match occurs on the key variable, like-named variables in the modified data set are NOT updated automatically from the updating data set. However, you can achieve the same result by reissuing the SET statement and using POINT= option for the updating data set.

Updating data sets with MODIFY and the KEY= option allows only variables in the modified data set to appear on the output data set. Updating data sets with SET and the KEY= option combines variables of both data sets so that all variables are in the output data set.

Multiple SET statements dictate that existing variables are not reinitialized to missing for each iteration; therefore, be especially careful to appropriately update variables when a non-match observation appears in the primary data set.

References

SAS Institute Inc. (1999), *SAS Language Reference: Dictionary*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS Procedures Guide*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1995), *Combining and Modifying SAS Data Sets: Examples, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1999), *SAS/SHARE User's Guide,* Cary, NC: SAS Institute Inc.

Jacobs, Charles (1992), "DATA Step Programming Using the MODIFY Statement," O*bservations, The Technical Journal for SAS Software Users, Fourth Quarter 1992, Volume 2, Number 1.*