SESUG Paper 146-2020

Efficiently Apply a Change File to a Large File via DATA Step MODIFY and _IORC_

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California
James Walker, MUFG Union Bank, N.A., Santa Barbara, California

ABSTRACT

The MODIFY statement of the SAS® DATA step is less commonly used than SET or MERGE; it supports changing a SAS file in-place. There are multiple versions of the MODIFY statement; herein we focus on MODIFY with a KEY= option and an associated SET statement. We begin with an overview of MODIFY and then discuss the automatic variable _IORC_ which functions as an “input/output return code” from I/O operations. The %SYSRC macro function is provided to test return codes using SAS-supplied mnemonics; this avoids hard-coded values and provides a degree of portability. We provide test data and sample MODIFY code that is annotated with explanations. We conclude by discussing whether _IORC_ has a wider range of applications than just MODIFY.

Keywords: MODIFY, KEY=, SET, macro functions, %SYSRC, _IORC_, IORCMSG, error handling.
SAS products: Base SAS.
User level: intermediate+.

BACKGROUND: MODIFY AND _IORC_

The MODIFY statement of the SAS® DATA step supports changing a SAS file in-place. MODIFY can also be used with relational database files, via a LIBNAME that points to an RDBMS file. To use MODIFY on an RDBMS file, one must have the relevant access permissions and the LIBNAME must be setup correctly. Mack (2008) is recommended as a reference to readers interested in RDBMS applications of MODIFY; the emphasis here is on using MODIFY on native SAS files.

There are 4 main versions of MODIFY:

1. MODIFY alone – no BY, no SET. This provides a single pass through a dataset and is similar to a simple SQL UPDATE operation.
2. MODIFY with BY – here the datasets are merged via BY.
3. MODIFY with the KEY= Option and an associated SET Statement – this is an index-based lookup/record match where a change file (the SET statement) is applied to a master file (the MODIFY statement).
4. MODIFY with the POINT= Statement – this is similar to the KEY= approach but instead of an index, the lookup process is driven by row numbers. [The KEY= approach is better/preferred over the POINT= approach.]

Mack (2008) describes a 5th, undocumented version: MODIFY with the KEY = Option and an INFILE statement instead of a SET statement. In this case the INFILE (and INPUT statement) provides the change records. As it is undocumented, this version is not recommended for production or critical applications.

The focus here is on version #3: MODIFY with KEY= and a SET operation. In this approach, an update file is applied to a master file, via an indexed match-merge. Now an update file might have changes for only a small number of rows, and records in the 2 files might not match – among other possible issues that may arise. To control/manage the merger of update rows with the master file, the SAS system provides 3 associated statements for MODIFY applications: OUTPUT (adds new record to master file), REPLACE (replaces the current record in the master file), and REMOVE (delete = mark as logically deleted the current record in the master file).

Given that the MODIFY with KEY= is a match-merge, the SAS system provides tools that inform the user of the status of I/O operations during the match merge. These tools allow the user to write code to manage the merge/update process. The tools provided include:

- Automatic variable _IORC_ and associated IORCMSG() function
- Autocall macro: %SYSRC()
- Mnemonics to designate/identify _IORC_ values; for portability and to avoid hardcoded values

The automatic variable _IORC_ is set after each I/O operation in a DATA step; it can be considered an acronym for “input/output (operation) return code”. Note the “after each operation” part – that is significant here as the subject DATA steps will have first a SET operation, followed by a MODIFY operation, and _IORC_ is reset after each distinct operation. The IORCMSG() function provides an English-language explanation of the _IORC_ status.

Now _IORC_ values are return codes and these can vary by operating system (portability) and over time in the same O/S (updates/changes). Instead of checking hardcoded values, use the %SYSRC macro with the relevant mnemonics to check completion codes; this allows the user to control the merge/update processing. The sample code herein uses these mnemonic values:

- _sok - I/O operation was successful,
- _dsenom – unmatched; no observation in master dataset for the change dataset observation.

Many additional mnemonics are available; see Mack (2008).

Readers may be asking why one would use MODIFY? In short: updating a large file with a small change file via MODIFY is usually more efficient than rewriting an entire, large file. Also, MODIFY can be simpler to use than SQL UPDATE and provides more controls/error handling features.

Next, we create artificial test data – both master and change files – and use that data to illustrate the code for a simple use of MODIFY (with KEY= and an associated SET statement) to update a SAS file in-place.

TEST DATA

In the code below we generate a master file and a change file, for testing/demonstration usage. Seeds are used for the random number generators to make the runs reproducible.

```
data master_file;
      call streaminit(94704);

      do i=1 to 15;
            my_key = i;
            v1 = rand('uniform');
            v2 = rand('uniform');
            output;
      end;

      drop i;
```

```
run;

proc print data=master_file;
	title "Master file";
run;

data change_file;
	call streaminit(32952);

	do i=1 to 5;
		my_key = 3*i;
		v1 = rand('uniform');
		v2 = rand('uniform');
		output;
	end;

	drop i;
run;

proc print data=change_file;
	title "Change file";
run;
```

The code above produces these 2 files:

Master file

Obs	my_key	v1	v2
1	1	0.95894	0.77739
2	2	0.11951	0.33168
3	3	0.90233	0.44476
4	4	0.54178	0.47867
5	5	0.5999	0.27089
6	6	0.78692	0.63691
7	7	0.70473	0.03992
8	8	0.18763	0.09963
9	9	0.33444	0.78542
10	10	0.06949	0.0848
11	11	0.69373	0.17284
12	12	0.27136	0.27158
13	13	0.34698	0.42868
14	14	0.82883	0.80414
15	15	0.74502	0.46302

Change file

Obs	my_key	v1	v2
1	3	0.22749	0.73412
2	6	0.53324	0.86428
3	9	0.58554	0.14757
4	12	0.42799	0.42836

SAMPLE CODE: MODIFY WITH KEY= AND SET

Now we are ready to illustrate and describe SAS code to use MODIFY with SET to apply a transaction or change file to a larger master file. The first thing we need is an index on the KEY= variable, in the master file. The change file does not need to be indexed. Indexes can be created via PROC SQL or PROC DATASETS; here we use the latter:

```
proc datasets lib=work nolist;
      modify master_file;
      index create my_key;
      run;
quit;
```

A PROC DATASETS subcommand - a MODIFY run-group - is used above to create the index. This MODIFY should not be confused with the DATA step MODIFY statement. The code above is a single variable index, i.e., a simple index. Composite indexes (containing multiple variables) can be created and used; in that case the index must be assigned a name, and the index name is used as the KEY= parameter.

Now we are ready to apply the update/change file to the master file via MODIFY with KEY=. Annotations to explain the code are provided after the code.

```
data master_file;   [1]
      set change_file  (rename= (v1 = new_v1   [2]
                                 v2 = new_v2) );
      do until (_iorc_=%sysrc(_dsenom));      [3]
            modify master_file key=my_key;    [4]
            select (_iorc_);                  [5]
                  when (%sysrc(_sok)) do;     [6]
                        v1 = new_v1;         [7]
                        v2 = new_v2;
                        replace master_file;  [8]
                  end;
                  when (%sysrc(_dsenom)) do;   [9]
                        _error_=0;
                  end;
                  otherwise;                   [10]
            end;
      end;  [11]
run;
```

Annotations:

[1] Here we change, in-place, SAS dataset master_file.
[2] The SET pulls in a change record. RENAME is required for relevant variables as the following MODIFY operation will overwrite variables with the same name in the program data vector.
[3] The _IORC_ here refers to the SET statement, not MODIFY, as that statement has not executed yet. This line can be interpreted as "do until a mismatched record is detected" (OR end-of-file is reached). Note: the DO UNTIL here could be replaced by logic that checks for %SYSRC(_SOK).
[4] MODIFY pulls in records from master_file, using the index key my_key, to match the value of my_key pulled in via the preceding SET operation from the change_file.
[5] This _IORC_ is for the MODIFY operation, and starts a SELECT-WHEN block.
[6] This translates to: WHEN there is a successful match, i.e., records exist in both files and are matched.
[7] If a successful match, over-write old values of variables in master_file with new values from change_file.
[8] Re-write the master_file record, i.e., update it in-place.
[9] This translates to: WHEN there is no successful match, i.e., in this context , records exist in change_file but not in master_file, set _ERROR_ to 0 to avoid error messages. With this logic, unmatched records are discarded/ignored.
[10] OTHERWISE statement is required for correct SELECT-WHEN syntax.
[11] The renamed variables (in the change file) are not added to the master file as MODIFY does not change the program data vector.

Check the results:

The code above successfully applied the changes in change_file to master_file; we check it via:

```
proc print data=master_file;
      title "Master file after MODIFY";
run;
```

The results below show that the 5 records in change_file have been applied in master_file; compare to the change_file and previous master_file listings above:

Master file after MODIFY

Obs	my_key	v1	v2
1	1	0.95894	0.77739
2	2	0.11951	0.33168
3	3	0.22749	0.73412
4	4	0.54178	0.47867
5	5	0.5999	0.27089
6	6	0.53324	0.86428
7	7	0.70473	0.03992
8	8	0.18763	0.09963
9	9	0.58554	0.14757
10	10	0.06949	0.0848
11	11	0.69373	0.17284
12	12	0.42799	0.42836
13	13	0.34698	0.42868
14	14	0.82883	0.80414
15	15	0.99791	0.6714

The example above is simple; _IORC_ and %SYSRC can also be used to export rows that fail to meet integrity constraints or are unmatched, to a separate “error file” for remediation. Of course, this requires more complex logic and checking additional mnemonic values.

IS _IORC_ USEFUL IN OTHER APPLICATIONS?

Mack (2008) reports that _IORC_ is set in all user-directed DATA step I/O operations. This writer’s direct experience using the associated IORCMSG() function indicates that this is correct. That suggests that _IORC_ is potentially useful in I/O error-handling logic. However, the documented usage of _IORC_ is (only) in association with MODIFY, and other usages might be considered undocumented and unsupported. Unfortunately that makes its usage in other contexts, questionable for production.

EPILOGUE

Given the efficiency of updating a large file in-place, SAS programmers should be aware of the MODIFY statement, and use it when appropriate/optimal.

Note: As of the date of publication, the authors are employees of MUFG Union Bank, N.A. The content of this paper does not reflect the views or opinions or work product of MUFG Union Bank.

APPENDIX 1: BSD 2-CLAUSE COPYRIGHT LICENSE (OPEN SOURCE)

```
* All program code in this paper is released under a Berkeley Systems
  Distribution BSD-2-Clause license, an open-source license that permits
  free reuse and republication under conditions;
```

REFERENCES

Mack, C (2008). MODIFY The Most Under-Appreciated of the Data Step File Handling Statements *Pacific Northwest SAS Users Group (PNWSUG) Conference Proceedings.* URL: https://www.lexjansen.com/pnwsug/2008/CurtisMack-Modify.pdf

CONTACT INFORMATION

A list of the author’s SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): https://goo.gl/uCUHoa

Note: Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
Email: tebillings@gmail.com