Paper 176-2008

New in SAS® 9.2: It's the Little Things That Count

Diane Olson, SAS Institute Inc., Cary, NC

ABSTRACT

It is easy getting the word out about the new big things that are added to a release, but the little things often go unsung. These additions to Platform SAS can make your coding life easier. This paper talks about some of those additions, including cross-host migration with PROC MIGRATE, a new statement for indexes, two new additions to the DATA step and linguistic sort additions to the utility procedures.

INTRODUCTION

This paper does not contain information about all the new "little things" added to Platform SAS in the SAS 9.2 release, as they are too numerous for one paper. Topics were chosen for universal appeal. All SAS customers should be able to find something in this paper to use in their daily programming.

Cross-host Data Migration

Starting in SAS 9.1, the MIGRATE procedure and the validation tools became available to migrate your SAS libraries forward to any new release. PROC MIGRATE has a number of advantages over the old methods of moving data, with a goal of creating exactly the same data and attributes in the new version of SAS. Previously, PROC MIGRATE was not available to some customers, because it was only for migrating within the same operating system family. In SAS 9.2, migrating your SAS libraries has become simpler. You no longer have to be concerned about whether you are migrating to the same operating system family. PROC MIGRATE can be used to migrate your data from your current operating system to almost any other operating system. Only the z/OS platform, outside of HFS (Hierarchical File System), does not support cross-host migration. There are a few other rare exceptions.[1] The validation tools can also be used cross-host to ensure correct and complete data library migration.

PROC MIGRATE uses Cross-Environment Data Access (CEDA) to read your data sets and views created on a different operating system. PROC MIGRATE ensures that all data attributes, including indexes, integrity constraints, and audit trails are migrated along with your data sets. CEDA is not available on the z/OS platform, so methods that include the COPY procedure and the CPORT/CIMPORT procedures are used for cross-host migration.[2]

CEDA does not work for catalogs, however. In order for migration to occur, a catalog must be read by the operating system on which it was created. Therefore, you must use SAS/SHARE® or SAS/CONNECT® (Remote Library Services (RLS)) to access the catalog from the new operating system. PROC MIGRATE makes use of PROC CPORT/CIMPORT technology when migrating catalogs. If your site does not have RLS, you could migrate the catalogs manually using those procedures. The PROC MIGRATE code for cross-host migration of an entire SAS library with catalogs is simple.

```
Libname lib1 v8 'physical location 1';
Libname lib2 base 'physical location 2';
Libname cats server=server1;

Proc migrate in=lib1 out=lib2 slibref=cats;run;
```

Example 1: Cross-host migration with catalogs

For best practice, we suggest the use of the validation tools. You can find them at http://support.sas.com/rnd/migration/resources/procmigrate/validtools.html.

---

[1] You cannot use PROC MIGRATE to migrate from 64-bit Windows Itanium or AIX platforms.

[2] Even though cross-host migration is not available on z/OS, there are a lot of new "little things" in SAS 9.2 for the z/OS platform, including random access techniques to create and read BSAM files, DSNTYPE=LARGE supported for both external files and direct access bound libraries, and the SASRX REXX exec that provides much greater flexibility in the invocation of SAS under TSO. See more information in the "What's New" section of the z/OS Companion for SAS 9.2.

REBUILD an Index

With the advent of cheaper disk space and more powerful computers to access that space, data sets tend to be much larger than in the past. As a result, indexes can be essential to access points of data. However, building such an index from scratch on a several-gigabyte data set can be time-consuming. If your data set becomes damaged, the index must be re-created from scratch during the repair process. It might take a great deal of time for the index to be rebuilt in order to access your data, but in the past you had no recourse.

New in SAS 9.2 is the value NOINDEX for the system option DLDMGACTION. When a damaged data set is accessed and this option is in force, only the data portion of the data set is repaired. This repaired data set can be opened for input only. The data set is repaired without any indexes or integrity constraints, and the physical index file is deleted. Integrity constraints are involved because they rely on indexes for their maintenance and use. Therefore, all existing integrity constraints are disabled when NOINDEX is specified. After a data set is repaired with NOINDEX in effect, the data set still retains the definitions of the disabled indexes and integrity constraints. If you run the CONTENTS procedure on the data set at this point, you will see that any indexes contain 0 unique values and any integrity constraints are listed as inactive.

```
Alphabetic List of Indexes and Attributes

                          # of
                         Unique
#     Index              Values

1     drug_test            0


                      Alphabetic List of Integrity Constraints

      Integrity                                    Where
#     Constraint          Type        Variables    Clause                Inactive

1     require_felony    Not Null    felony_record                           YES
2     require_fired     Not Null    ever_fired                              YES
3     unique_SSN        Unique      SSN                                     YES
4     val3date_e        Check                    ever_fired in ('n', 'y')   YES
```

Example 2: PROC CONTENTS output for disabled integrity constraints and indexes

The repaired data set can be opened only for input until the REBUILD statement of the DATASETS procedure is used on that data set. The REBUILD statement works on data sets that were created in SAS 7 and later.

The REBUILD statement in PROC DATASETS enables you to be flexible in the repair of the index. You can rebuild or delete the indexes and integrity constraints at a time that is convenient for you, instead of being required to do so when the data set is repaired. The REBUILD statement has syntax that is similar to the REPAIR statement, also in PROC DATASETS:

REBUILD SAS-file <ALTER=password GENNUM=n MEMTYPE=mtype NOINDEX>;

When the REBUILD statement is executed, all of the indexes and integrity constraints that were originally on the data set at the time it was damaged are rebuilt. During the rebuild, the data set is locked at the member level, meaning that it is not available to any other SAS process. Once the rebuild is complete, the data set is fully accessible and no longer restricted to INPUT mode, and indexes and integrity constraints are available. This is the default behavior of the REBUILD statement.

```
proc datasets lib=mylib;
rebuild mydata; run;
quit;
```

Use of the REBUILD statement's NOINDEX option will delete all indexes and integrity constraints and will allow full access of the data set once again. Here is a simple example of using REBUILD with the NOINDEX option:

```
proc datasets lib=mylib;
rebuild mydata noindex;run;
quit;
```

Note that because referential integrity constraints maintain a relationship of data between two or more data sets, data sets containing referential integrity constraints cannot be rebuilt with the NOINDEX option. To use the REBUILD statement with the NOINDEX option, referential integrity constraints must be deleted.

```
proc datasets lib=mylib;
   modify mydata;
      ic delete foreign1;run;
      ic delete foreign2;run;
   modify mydata2;
      ic delete primary1;run;
   rebuild mydata noindex;run;
quit;
```

Example 3: Deleting referential integrity constraints before NOINDEX repair

After the REBUILD statement is used on the data set, the referential integrity constraints can be re-created manually. If deleting the referential integrity constraints is not an option, the indexes must be repaired along with the data set.

IBUFNO System Option

SAS programmers who add indexes to their data usually do so for only one purpose—to speed access to the particular observations that are subsetted by the index. SAS automatically allocates a minimal number of buffers to be used with the index to navigate the index file. The IBUFNO system option enables you to determine the number of extra buffers to be used by the index, instead of relying on the default. This increase in the number of buffers can increase the performance of an index. Of course, that increase in performance comes with a price—increased memory usage.

In order to determine the best number of index buffers, trial and error is necessary to identify the least number of buffers that produce satisfactory performance results. The system option accepts values from MIN (which is 0) to MAX (which is 10,000).

The IBUFNO system option was available as an internal option for many years and made known to customers by Technical Support as needed. It has now been added to our documentation and is supported along with IBUFSIZE to enable you to customize the index buffers to your particular needs. The IBUFSIZE system option determines the size of one index buffer, and IBUFNO determines the number of extra buffers to be used. The total memory that is used for index access is the sum of the default number of buffers and the value of IBUFNO times the IBUFSIZE system option value.

DATA Step Data Set Lists

Before SAS 9.2, creating a data set from similarly named data sets required that all the names be painstakingly listed on the SET statement of the DATA step. You can now use either a dash list or a colon list to indicate which data sets to read. To use the dash list syntax to create a numbered range list, specify a data set name that ends in one or more digits, followed by a dash. Indicate the ending data set by the same name, ending in different digits. For example, you can use A1–A100, A1B1–A1B5, or A100–A1. The colon list enables you to select a group of members whose names begin with the same letter or letters by entering the common letters followed by a colon (:).

Compare the differences in these three examples:

```
data results;
  set a1 a2 a3 a4 a5 a6 a7 a8 a9 a10
      a11 a12 a13 a14 a15 a16 a17 a18 a19 a20
      a21 a22 a23 a24 a25 a26 a27 a28 a29 a30
      a31 a32 a33 a34 a35 a36 a37 a38 a39 a40
      a41 a42 a43 a44 a45 a46 a47 a48 a49 a50
      a51 a52 a53 a54 a55 a56 a57 a58 a59 a60
      a61 a62 a63 a64 a65 a66 a67 a68 a69 a70
      a71 a72 a73 a74 a75 a76 a77 a78 a79 a80
      a81 a82 a83 a84 a85 a86 a87 a88 a89 a90
      a91 a92 a93 a94 a95 a96 a97 a98 a99 a100;
run;
```

Example 4: The hard way

```
data results;
  set a1-a100;
run;
```

Example 5: Data set numeric range list

```
data results;
  set a:;
run;
```

Example 6: Data set colon list

The dash list is for data set names that end in numerics, but the colon list is for names that begin with any letter or series of letters. Be careful when you use the colon list syntax. If there are other data sets in your library that begin with the prefix specified, they would be included in the SET statement as well.

DATA Step SET Statement with INDSNAME Option

SAS programmers might need to know what data set contributed a particular observation in a data set drawn from many sources. Knowing the contributing data set name is especially helpful when metadata is coded in data set names. Before SAS 9.2, this was difficult to code because it required an IN= option for each input data set. The new INDSNAME= option on the SET statement provides the easily-coded functionality. When INDSNAME=*variable-name* is specified on the SET statement, *variable-name* is populated with the name of the data set that contributed the last observation read. This data set name is a fully qualified libname.member. Unless a LENGTH statement is used, *variable-name* has a length of 41 characters, based on a possible eight-character libname, a 32-character member name, and the connecting '.' between them. The value that is placed in *variable-name* is uppercase.

In the following example, assume the existence of the data sets whose names begin with 'gas' and 'coal'. Use the INDSNAME= option and the SCAN function to parse the pieces of the data set name into separate variables.

```
data results;
  set gas_price_option
      gas_rbid_option
      gas_price_forward
      gas_rbid_forward
      coal_price_option
      coal_rbid_option
      coal_price_forward
      coal_rbid_forward
      indsname = cur_dataset;

  fuel = scan(cur_dataset, 2, '._');
  value = scan(cur_dataset, 3, '._');
  type = scan(cur_dataset, 4, '._');
run;

proc print data=results; run;
```

Obs	x	fuel	value	type
1	1	GAS	PRICE	OPTION
2	2	GAS	RBID	OPTION
3	3	GAS	PRICE	FORWARD
4	4	GAS	RBID	FORWARD
5	5	COAL	PRICE	OPTION
6	7	COAL	RBID	OPTION
7	6	COAL	PRICE	FORWARD
8	8	COAL	RBID	FORWARD

Example 7: PRINT Procedure of INDSNAME results

Of course, if you use the data sets list in the previous example, this becomes simpler. The colon list makes it easy to select all the data sets that begin with ‘gas’ or ‘coal’. Using INDSNAME= in conjunction with a colon list is helpful when you might not know the names of all the data sets that are read.

```
data results;
  set gas: coal: indsname = cur_dataset;

  fuel = scan(cur_dataset, 2, '._');
  value = scan(cur_dataset, 3, '._');
  type = scan(cur_dataset, 4, '._');
run;
```

Example 8: Data set colon list with INDSNAME

Linguistic Collation and the Utility Procedures

One of the big projects accomplished in SAS 9.2 is the availability of linguistic collation. Linguistic collation gives every language and culture in the world the ability to get their expected sort order when sorting a data set. There are already a couple of excellent white papers on this subject, including “Creating Order out of Character Chaos: Collation Capabilities of the SAS System” by Scott Mebust and Michael Bridgers. This paper points out the “little things” that go along with a large project like linguistic collation.

When PROC CONTENTS is run on a sorted data set, the listing output contains a section on sort information. The sort information output was modified to reflect the new metadata when the data set is linguistically sorted. Locale, strength, collation order, case first, alternate handling, and numeric collation are all pieces of information you might see in the sort information in SAS 9.2, depending on what was specified in options to the SORT procedure. See the SAS 9.2 PROC SORT documentation for more information about these options.

```
data lib1.sorted;
  length x $ 20;
  input x;
  datalines;
  aardvark
  azimuth
  zebra
  Aaron
  Aztec
  Zeus
  ;
run;
proc sort in=lib1.sorted out=lib1.sorted2 sortseq=linguistic (locale=ko_KR
strength=identical ) ; by x;
run;

proc contents data=lib1.sorted2;run;
```

Example 9: Creating a linguistically sorted data set and using PROC CONTENTS

```
     Sort Information

Sortedby              x
Validated             YES
Character Set         ANSI

Collating Sequence    LINGUISTIC
Locale                ko_KR
Strength              5
```

Example 10: PROC CONTENTS of linguistically sorted data set

With new features such as linguistic collation, SAS provides you with behavior you expect from legacy procedures. For example, by default PROC COPY and PROC MIGRATE will always try to set an attribute from the input data set on the output data set. The same is true for linguistic collation. If the output library engine supports linguistic collation, sort settings will be kept on the new data set. There are some cases where this is not possible; consider PROC COPY to a SAS 6 library. Linguistic sorting is not understood in SAS 6, so it cannot be set. Similarly, VMI Itanium and 64-bit Windows Itanium do not support linguistic collation, so using PROC MIGRATE on that platform will not retain the linguistic collation. When linguistic sort information cannot be set on the output data set by PROC COPY or PROC MIGRATE, a warning is written to the log. If you are using the validation tools with either procedure, they will note the difference between your source and target data libraries.

CONCLUSION

The developers at SAS Institute Inc. care about your software needs. Every single one of these additions was conceived by a customer like you with a need or idea, who contacted Technical Support, e-mailed a suggestion to someone at SAS, or talked with SAS employees at SAS Global Forum. We're interested in making you a happy, productive customer, right down to the little things that count.

ACKNOWLEDGMENTS

Thanks to all the customers who generated the ideas for the additions to SAS 9.2, along with the SAS developers who made them happen. Special thanks for reviewing this paper or providing content go to Miguel Bamberger, David Wiehle, Jason Secosky, Claire Bonney, Lewis King, Gary Franklin and Lisa Brown. Any mistakes are my own.

CONTACT INFORMATION

Your suggestions, comments, and questions are always welcome. Please contact the author:

Diane Olson
SAS Institute Inc.
SAS Campus Drive
Cary, NC, 27513
E-mail: Diane.Olson@sas.com