Paper 52-2022

PROC SORT (then and) NOW

Derek Morgan, PAREXEL International

ABSTRACT

The SORT procedure has been an integral part of SAS® since its creation. The sort-in-place paradigm made the most of the limited resources at the time, and almost every SAS program had at least one PROC SORT in it. The biggest options at the time were to use something other than the IBM procedure SYNCSORT as the sorting algorithm, or whether you were sorting ASCII data versus EBCDIC data. These days, PROC SORT has fallen out of favor; after all, PROC SQL enables merging without using PROC SORT first, while the performance advantages of HASH sorting cannot be overstated. This leads to the question: Is the SORT procedure still relevant to any other than the SAS novice or the terminally stubborn who refuse to HASH? The answer is a surprisingly clear “yes". PROC SORT has been enhanced to accommodate twenty-first century needs, and this paper discusses those enhancements.

INTRODUCTION

The largest enhancement to the SORT procedure is the addition of collating sequence options. This is first and foremost recognition that SAS is an international software package, and SAS users no longer work exclusively with English-language data. This capability is part of National Language Support (NLS) and doesn’t require any additional modules. You may use standard collations, SAS-provided translation tables, custom translation tables, standard encodings, or rules to produce your sorted dataset. However, you may only use one collation method at a time.

USING STANDARD COLLATIONS, TRANSLATION TABLES AND ENCODINGS

A long time ago, SAS would allow you to sort data using ASCII rules on an EBCDIC system, and vice versa. The following list shows the standard collating sequences now available in SAS 9.4, with the newer ones in italics:

- ASCII
- *DANISH*
- EBCDIC
- *FINNISH*
- *NATIONAL*
- *NORWEGIAN*
- *REVERSE*
- *SWEDISH*

As an example, to use the Finnish collation, you would add it to the PROC SORT statement as follows in:

```
PROC SORT DATA=mydata FINNISH;
BY var1;
RUN;
```

Sample Code 1: Using Built-In SAS Collations

The NATIONAL collation may not be available in your SAS installation; check with your SAS administrator before trying to use it. The standard translation tables provided by SAS add Italian, Polish, and Spanish to the above list as collating sequences. However, these require the use of the SORTSEQ= option. If none of the SAS-provided translation tables work for your situation, you may even create your own translation table. Creation of a custom translation table should be viewed as a last resort. It depends on the

installation of SAS, and there are very specific rules for implementing a custom translation table. Here is an example of using a translation table:

```
PROC SORT DATA=mydata SORTSEQ=ITALIAN;
BY var1;
RUN;
```

Sample Code 2: Using a Translation Table with Your Sort via the SORTSEQ= Option

Encoding values such as “wlatin-1” or “utf-8”, can be used in the SORTSEQ= option, which will perform a binary collation of the character data represented in the specified encoding as follows:

```
PROC SORT DATA=mydata SORTSEQ=latin2; /* Central European ISO Standard */
BY var1;
RUN;
```

Sample Code 3: Using the SORTSEQ= Option to specify an Encoding

A full list of available encodings is in the National Language Support documentation.

Should you need any of this functionality, the SORTSEQ= option will provide it. You may also specify an encoding or translation table as a system option:

```
OPTIONS SORTSEQ=ITALIAN;
```

Sample Code 4: Using SORTSEQ= as a System Option

That will set the default collating sequence throughout your SAS program.

RULES-BASED COLLATION

I believe this is the way to unleash the hidden power inside PROC SORT. It requires the use of the SORTSEQ= option on the PROC SORT statement itself; you cannot specify it as a system option. The keyword to invoke rules-based collation is SORTSEQ=LINGUISTIC.

The LINGUISTIC keyword causes SAS to sort characters according to the linguistic rules associated with the language and locale in effect. However, the LINGUISTIC keyword has multiple options that modify the linguistic collating sequence. Some of these solve problems that have required creative SAS coding for years. You may use more than one LINGUISTIC option, but you cannot use SORTSEQ=LINGUISTIC with a translation table or encoding.

LETTERS VERSUS SPACES, PUNCTUATION, AND SYMBOLS

The ALTERNATE_HANDLING= option example allows PROC SORT to treat the handling of differences in spaces, punctuation and symbols as less important than differences between letters. Without this option, differences in those characters are of equal weight as letters, which is the default:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(ALTERNATE_HANDLING=SHIFTED);
BY var1;
RUN;
```

Sample Code 5: The ALTERNATE_HANDLING Option

CHARACTER ORDERING

Use the COLLATION= option to specify character ordering. One advantage of specifying collation this way, as opposed to using encoding, is the handling of multiple languages (e.g., STROKE) with a single collation definition:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(COLLATION=collation-value);
BY var1;
RUN;
```

Sample Code 6: Defining Character Ordering with the COLLATION= Option

Table 1 provides the valid values for this option as of SAS 9.4:

BIG5HAN	Specifies Pinyin ordering for Latin and specifies bug5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	Specifies a Hindi variant.
GB21312HAN	Specifies Pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	Specifies a telephone-book style for ordering of characters. Select PHONEBOOK only with the German language.
PINYIN	Specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into Pinyin. This ordering is typically used with simplified Chinese.
POSIX	This option specifies a “C” locale ordering of characters.
STROKE	Specifies a non-alphabetic writing style ordering of characters. Select STROKE with Chinese, Japanese, Korean, or Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	Specifies a traditional style for ordering of characters.

Table 1: Valid Sequences for the COLLATION= Option

THE “ADDRESS PROBLEM”

When the NUMERIC_COLLATION option is set to ON, integer values within a character string will be ordered by their numeric value instead of their character value. The default value is OFF, but it is easy enough to change it:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(NUMERIC_COLLATION=ON);
BY var1;
RUN;
```

Sample Code 7: Ordering by Numbers Within a Character String

As an example, the following unsorted list of addresses is stored in a single variable named ADDRESS, without the numbers being a separate field. How would you sort it without manipulating the ADDRESS field?

1801 Somewhere Ave.	4200 Somewhere Ave.
1652 Somewhere Ave.	7262 Somewhere Ave.
7137 Somewhere Ave.	12425 Somewhere Ave.
10381 Somewhere Ave.	506 Somewhere Ave.
4177 Somewhere Ave.	

Example 1 demonstrates what happens when you use PROC SORT without any options. While it is a defined sorting of these addresses using a standard sorting algorithm, it won't help you if you're a delivery person using this list as a delivery manifest. You'll start at the far end of Somewhere Ave., then go to the 1600 block, and then from the 4200 block to the 500 block before ending up in the middle again:

10381 Somewhere Ave.
12425 Somewhere Ave.
1652 Somewhere Ave.
1801 Somewhere Ave.
4177 Somewhere Ave.
4200 Somewhere Ave.
506 Somewhere Ave.
7137 Somewhere Ave.
7262 Somewhere Ave.

Example 1: Using Default SORT to Order Addresses

For decades, SAS programmers have either designed address datasets to store the street number and the street name separately, or parsed street number from the street name when they receive data where they are not separate, and then sort by the street number and street name. The NUMERIC_COLLATION=ON option removes the need for parsing the address. Example 2 demonstrates:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(NUMERIC_COLLATION=ON);
BY address;
RUN;
```

The above code produces the sorted data shown below without parsing of the ADDRESS variable or any change to the data structure:

506 Somewhere Ave.
1652 Somewhere Ave.
1801 Somewhere Ave.
4177 Somewhere Ave.
4200 Somewhere Ave.
7137 Somewhere Ave.
7262 Somewhere Ave.
10381 Somewhere Ave.
12425 Somewhere Ave.

Example 2: Sorting Addresses using the NUMERIC_COLLATION=ON Option

NUMERIC_COLLATION=ON ALSO SORTS EMBEDDED NUMBERS IN STRINGS

Another example of using the NUMERIC_COLLATION=ON option is with values such as "Day 1 of 365". The possible strategies using PROC SORT without options far have been to sort on the numeric day, keeping that variable along with the output string, or to display the text as "Day 001 of 365". The NUMERIC_COLLATION=ON option avoids this as well, even though the numeric portion is embedded within the character value, as shown in Example 3:

Without NUMERIC_COLLATION=ON	Using NUMERIC_COLLATION=ON
Day 1 of 20	Day 1 of 20
Day 10 of 20	Day 2 of 20
Day 11 of 20	Day 3 of 20
Day 12 of 20	Day 4 of 20
Day 13 of 20	Day 5 of 20
Day 14 of 20	Day 6 of 20
Day 15 of 20	Day 7 of 20
Day 16 of 20	Day 8 of 20
Day 17 of 20	Day 9 of 20
Day 18 of 20	Day 10 of 20
Day 19 of 20	Day 11 of 20
Day 2 of 20	Day 12 of 20
Day 20 of 20	Day 13 of 20
Day 3 of 20	Day 14 of 20
Day 4 of 20	Day 15 of 20
Day 5 of 20	Day 16 of 20
Day 6 of 20	Day 17 of 20
Day 7 of 20	Day 18 of 20
Day 8 of 20	Day 19 of 20
Day 9 of 20	Day 20 of 20

Example 3: The NUMERIC_COLLATION=ON Option with Embedded Numeric Values

LOCATION-BASED SORTING

The LOCALE= option that is a part of National Language Support can also be used as an option for rule-based sorting. All the LOCALE values can be found in the National Language Support documentation. This setting will override the LOCALE setting in effect, allowing PROC SORT to use a different LOCALE (and usually language) from the remainder of the program as shown in Sample Code 8:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(LOCALE=locale-value);
BY var1;
RUN;
```

Sample Code 8: Using a Different LOCALE for Sorting from the one in Effect for the SAS Session

WHAT DIFFERENCES DO I WANT MY SORT TO IGNORE?

The final set of LINGUISTIC options has to do with the sensitivity of the sorting algorithm. Sample Code 9 provides the syntax with the STRENGTH= option:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(STRENGTH=strength-value);
BY var1;
RUN;
```

Sample Code 9: Using SORTSEQ=LINGUISTIC

The values for the STRENGTH= option are shown in Table 2:

VALUE	ALIAS	Explanation
PRIMARY	1	PRIMARY specifies differences between characters, but not case or accents. For example, “a” < “b”, but “a” = “A”, and “Á” = “A”). It is the strongest difference.
SECONDARY	2	Accents in characters are considered secondary differences (for example, "as" < "às" < "at"). A secondary difference is ignored when there is a primary difference anywhere in the strings. Depending on the language, other differences between letters will also be considered secondary differences.
TERTIARY	3	**This is the default sort strength for US English**. Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò"). A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. A non-English example would be the difference between large and small Kana
QUATERNARY	4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "a-b" < "ab" < "aB"). The quaternary level should be used if ignoring punctuation is required or when processing Japanese text. This difference is ignored when there is a primary, secondary, or tertiary difference.
IDENTICAL	5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4. This level should be used sparingly, because code-point value differences between two strings rarely occur. In practical terms, only Hebrew cantillation marks are distinguished at this level.

Table 2: Options for Detecting Differences Between Records

The CASE_FIRST= option allows you to specify if uppercase letters sort before lower-case letters and vice versa. This option only works in conjunction with the STRENGTH= option, and only when STRENGTH is TERTIARY, QUATERNARY, or IDENTICAL, as shown in Sample Code 10.:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(STRENGTH=TERTIARY
                                         CASE_FIRST=UPPER);
BY var1;
RUN;
```

Sample Code 10: Using both the STRENGTH= and CASE_FIRST= Options

If the STRENGTH= value is PRIMARY or SECONDARY, the CASE_FIRST option has no effect because upper- and lower-case differences are not detected at these levels.

Instead of using the UPPERCASE() function to create an identical-case version of the variable you want to sort on, you can use STRENGTH=PRIMARY or SECONDARY to perform case-insensitive sorting, and avoid modifying your data before sorting. The default strength for English-language sorting differentiates between upper and lower case, so a simple alphanumeric sort won’t produce a case-insensitive result. For decades, SAS programmers have been using Sample Code 11 or something like it:

```
DATA mydata2;
SET mydata;
uname = UPCASE(name);
RUN;

PROC SORT DATA=mydata2;
BY uname;
RUN;
```

Sample Code 11: The Old Way to Perform Case-Insensitive Sorting

Now you can reduce the sensitivity of the SORT to ignore case as shown in Sample Code 12, avoiding the DATA step with the function call and the extra variable:

```
PROC SORT DATA=mydata SORTSEQ=LINGUISTIC(STRENGTH=PRIMARY);
BY name;
RUN;
```

Sample Code 12: Using the STRENGTH= Option to Perform a Case-Insensitive Sort

Example 4 is a side-by-side comparison of the original unsorted data, and the different ways to sort that data. The third column shows the result of a plain PROC SORT, while column four shows the sorted data using the variable created by the UPPERCASE() function in a prior DATA step. Note the sort variable is different (UNAME vs. NAME), and the value being sorted remains in upper-case. You can drop the upper-case variable from the output dataset using the DROP= option on the **output** dataset to avoid adding it to your dataset. The final column displays the result of Sample Code 12. Unlike the upper-case sort, the values are not changed from their original case, and there is no extra variable to remove from the output dataset.

	Original Unsorted Order	Sorting on NAME	Sorting on an Upper-case alias created in a DATA Step	Sorting with a Case-insensitive Sort
OBS	**name**	**name**	***uname***	**name**
1	Macarthur	MACK	MACALLEN	***Macallen***
2	Mc Grady	MacCarron	MACARTHUR	***Macarthur***
3	MACK	MacManus	MACCARAY	***Maccaray***
4	MacManus	Macallen	MACCARRON	***MacCarron***
5	Maccarron	Macarthur	MACCARRON	***Maccarron***
6	Macallen	Maccaray	MACK	***MACK***
7	McAllen	Maccarron	MACMANUS	***MacManus***
8	Maccaray	Mc Grady	MC GRADY	***Mc Grady***
9	MacCarron	McAllen	MCALLEN	***McAllen***

Example 4: Effect of Different Sorting Methods for Case-Insensitive Sorting

With very few accented characters (mostly appropriated from foreign words incorporated into the language), TERTIARY works well as the default sorting strength for English. However, the ability to ignore or account for accents may be very helpful in other languages.

ENHANCED HANDLING OF RECORDS WITH DUPLICATE KEYS

Everyone who uses PROC SORT should be familiar with the NODUPKEY option, which removes records with identical keys. Even this has been enhanced, with the ability to choose which of the observations is kept, and you can send the duplicates that aren't kept to a dataset. The next set of examples will use a dataset containing customer ID numbers and the level of ticket purchased for a given event, shown in

Sample Data 1. The customers who have purchased tickets to more than one event are bolded in the table. The original sort order is by customer ID and descending ticket level.

Customer ID	Ticket Level
10270	7
12230	3
19323	**5**
19323	**4**
22779	3
28819	6
29252	2
30457	**7**
30457	**3**
30457	**2**
30457	**2**
31918	**4**
31918	**1**

Sample Data 1: Customer Ticket Data

How can you find out which records have been eliminated? Sample Code 13 sends the duplicates to a dataset, shown in Example 5:

```
PROC SORT DATA=sortsamp OUT=sort1 NODUPKEY DUPOUT=dups;
BY cid;
RUN;
```

Sample Code 13: Sending Duplicate Records to a Dataset

Dataset SORT1			Dataset DUPS	
Customer ID	Ticket Level		Customer ID	Ticket Level
10270	7		19323	4
12230	3		30457	3
19323	5		30457	2
22779	3		30457	2
28819	6		31918	1
29252	2			
30457	7			
31918	4			

Example 5: Using the DUPOUT= Option to Remove Records with Duplicate Key Values

This is handy when you have multiple keys and a lot of records and you have to track down why you have duplicate-keyed records; perhaps you don't have enough keys for uniqueness or you have a different problem with the data. Again, this is all done within PROC SORT, so you don't have to write DATA step or SQL code to keep your removed duplicates.

Another PROC SORT option lets you pull out all the duplicate keyed records. Instead of running a DATA step like Sample Code 14 as SAS programmers have done for decades, use Sample Code 15:

```
DATA inspectdups;
SET sortsamp;
BY cid;
IF NOT (FIRST.cid AND last.cid) THEN
   OUTPUT;
RUN;
```

Sample Code 14: The Old Way of Setting Records with Duplicate Keys Aside

```
PROC SORT DATA=sortsamp NOUNIQUEKEY OUT=alldups UNIQUEOUT=uniques;
BY cid;
RUN;
```

Sample Code 15: Setting Records with Duplicate Keys Aside using the NOUNIQUEKEY Option, and the UNIQUEOUT= and OUT= Options

That produces the dataset on the right in Example 6. As you can see, it is identical to the result you would get from Sample Code 14 (on the left,) but without the overhead of the extra data step.

Dataset INSPECTDUPS Created by DATA step			Dataset ALLDUPS Created by PROC SORT and NOUNIQUEKEY option	
Customer ID	**Ticket Level**		**Customer ID**	**Ticket Level**
19323	5		19323	5
19323	4		19323	4
30457	7		30457	7
30457	3		30457	3
30457	2		30457	2
30457	2		30457	2
31918	4		31918	4
31918	1		31918	1

Example 6: Removing Records with Duplicate Keys

Another benefit of using PROC SORT and the NOUNIQUEKEY option is that by adding the UNIQUEOUT= option the same SORT procedure will send all your uniquely-keyed records to their own dataset, without adding conditional processing to your DATA step code. Example 7 is produced by the same PROC SORT in Sample Code 15:

Dataset UNIQUES	
Customer ID	**Ticket Level**
10270	7
12230	3
22779	3
28819	6
29252	2

Example 7: Removing Records with Unique Keys from a Data Set

DETERMINING DATASET KEYS

Another use of the NOUNIQUEKEY option would be to determine dataset keys. If you have selected your key variables correctly, using the NOUNIQUEKEY and OUT= options will immediately let you know if your keys uniquely identify records using Sample Code 16:

```
PROC SORT DATA=dataset_to_test OUT=keytest NOUNIQUEKEY;
BY key-variable-1 key-variable-2... key-variable-n;
RUN;
```

Sample Code 16: Using the NOUNIQUEKEY Option to Determine Dataset Keys

If the output dataset (KEYTEST) specified by the OUT= option has any records in it, then your key variables do not uniquely identify records, and you will have to add at least one additional variable. Unfortunately, this does not help you identify if you've included extraneous variables in your dataset key. However, you can remove the last variable from the BY statement and run the sort again. If KEYTEST contains any records after removing it, then you know that variable is required for your key. If no records are in KEYTEST, then you know the variable you removed is not required as a key variable for the dataset.

WHAT ABOUT THE NODUPLICATES OPTION?

Surprisingly, the NODUPLICATES option is no longer documented as a part of PROC SORT, although it is documented in the SORT() function in SAS Component Language. However, the option still functions. The NODUPLICATES option has always come with the warning that it doesn't test each record against all other records for a duplicate; it only tests against adjacent records. Therefore, it doesn't always work in an unsorted dataset. PROC SQL provides a better way to remove duplicate records as shown in Sample Code 17:

```
PROC SQL:
CREATE TABLE all_duplicates_removed AS
SELECT DISTINCT *
FROM data_with_dup_records
ORDER BY keyvar1… keyvarN
;
QUIT:
```

Sample Code 17: Using PROC SQL to Reliably Remove Duplicate Records

ADDITIONAL SORT OPTIONS

There are other PROC SORT options available, such as DATECOPY, which retains the date and time of the original, unsorted dataset in the sorted version. This can be useful with version control.

The REVERSE option collates in reverse, according to the character set in use. This is also the same as SORTSEQ=REVERSE. If you use the REVERSE option with SORTSEQ=REVERSE, they will cancel each other out, and the dataset will be sorted in regular order.

The PRESORTED option is an efficiency aid with SAS datasets. If you think the data are already sorted, using this option will skip the sort process if the dataset is already sorted. This ONLY applies to SAS datasets. Do NOT use this option when using other DBMS or SAS/ACCESS®, as the records may not be accessed in the assumed order.

CONCLUSION

PROC SORT has grown in its capabilities since the early days of SAS. Although you no longer need to sort in order to merge, and there are much more efficient ways to sort data, version 9 of SAS has returned PROC SORT to relevance. Many of the enhancements are linked with National Language Support, and acknowledge that SAS is truly an international software package working with data in multiple languages.

Several options for PROC SORT replace DATA step or SQL code. You can perform case-insensitive sorting and/or select uniquely-keyed records while placing the eliminated records into a data set for later inspection. Conversely, you can select all the duplicate-keyed records, eliminating the uniquely-keyed ones.

Other options maintain the date of the unsorted data set, reverse the collating sequence (not quite the same as sorting in descending order), and allow SAS to check the sort status of a data set before sorting it, which can prove to be quite an efficiency boost.

While HASH sorting is undeniably more efficient, the language enhancements in PROC SORT and the options that replace common SAS code may even those odds. PROC SORT is once again a useful tool in the SAS programmer's toolkit. For more in-depth detail, including how PROC SORT works with National Language Support (NLS), refer to the technical paper from SAS Institute in the first reference below.

ACKNOWLEDGMENTS

Thanks to SAS Technical Support in general for their assistance over the years.

REFERENCES

Kiefer, Manfred and Mebust, Scott. "Linguistic Collation: Everyone Can Get What They Expect: Sensible Sorting for Global Business Success". http://support.sas.com/resources/papers/linguistic_collation.pdf. Accessed 01 February 2018

Mebust, Scott and Bridgers, Michael. Creating Order out of Character Chaos: Collation Capabilities of the SAS System. *Proceedings of the SAS® Global Forum 2007 Conference,* SAS Institute Inc. Cary, NC

RECOMMENDED READING

- *National Language Support Documentation*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Derek Morgan
mrdatesandtimes@gmail.com