

Looking Beneath the Surface of Sorting

Andrew T. Kuligowski, FCCI Insurance Group

ABSTRACT

Many things that appear to be simple turn out to be a mask for various complexities. For example, as we all learned early in school, a simple drop of pond water reveals a complete and complex ecosystem when viewed under a microscope. A single snowflake contains a delicate crystalline pattern. Similarly, the decision to use data in a sorted order can conceal an unexpectedly involved series of processing and decisions.

This presentation will examine multiple facets of the process of sorting data, starting with the most basic use of PROC SORT and progressing into options that can be used to extend its flexibility. It will progress to look at some potential uses of sorted data, and contrast them with alternatives that do not require sorted data. For example, we will compare the use of the BY statement vs. the CLASS statement in certain PROCs, as well as investigate alternatives to the MERGE statement to combine multiple datasets together.

SORTING – THE BASICS

sort [sOrt] To arrange into some order, especially numerically, alphabetically or chronologically. From Old French sortir (“allot, sort”), from Latin sortiri (“draw lots, divide, choose”).

It is a safe assertion that all but the most novice SAS® users have used PROC SORT in its most rudimentary form – it is usually covered near the beginning of any introductory course. All that is needed is the name of the dataset to be sorted and the list of variables it is to be sorted by, in order of priority. In fact, one doesn’t even actually need the name of the input dataset, as SAS will default to “last dataset created”. (For the record, this shortcut is frowned upon by most SAS coders. The resulting code lacks the internal documentation that a specific reference can provide; this can cause difficulties and confusion during subsequent maintenance on the routine.) This information is inserted into a PROC SORT, with the source dataset being specified in the DATA= option of the PROC SORT statement, and the variable list added to the mandatory BY statement – after all, from a logical standpoint, one cannot sort anything without knowing the conditions by which that sort should occur!

```
PROC SORT DATA=<DatasetName>;
  BY Variable1 <Variable2 <... VariableN > >;
RUN;
```

It doesn’t take long for that beginning SAS user to introduce some additional complexities into their routine. For example, they may decide to use the OUT= option, so that the sorted output is routed to a new dataset rather than replacing the original unsorted one. The DESCENDING on the BY statement option is another option that is quickly added to the toolkit of the SAS coder. (Of course, saying that it is an option of the BY statement is a bit of a misnomer, since it is actually applied to one or more – perhaps even all – of the variables in the list rather than to the statement as a whole.)

```
PROC SORT DATA=<DatasetName>
  OUT=<NewDatasetName> ;
  BY DESCENDING Variable1 <DESCENDING Variable2
  <... DESCENDING VariableN > >;
RUN;
```

This is also the point where some folks’ interest in sorting under SAS comes to an end. Once they know how to perform a sort on their requested dataset using as many or as few variables as necessary, and that they can do so in either an ascending or descending direction, they believe they have all the knowledge that they will ever need on this particular topic. In some cases they might even be right – that IS everything they will ever need to know. But for many others, stopping there will result in gaps of

knowledge that might cause them to insert unnecessary additional code, or even get unpredictable results!

WHAT IS “NUMERICAL ORDER”?

As a case in point, let us look into the sorting of numeric data. Almost anyone who has gone to school long enough to have covered decimal places and negative numbers should be comfortable in their own ability to sort numeric data – I will not insult anyone’s intelligence by reviewing it here. These individuals – that is, all of us! – would expect a computerized sort utility to produce the same results. However, it is highly doubtful that any of those folks had the concept of missing data covered in their elementary school arithmetic classes. Will the infamous dot “.” be treated as the highest or lowest value when sorted? Or, perhaps it will be analogous to zero, which would fall right between positive and negative numbers?

When addressing questions like these, sometimes the answer can be quickly found online in the appropriate SAS manual, or by doing an internet search for professional papers such as the one you are currently reading. However, it is possible that these techniques may not readily produce an appropriate response – perhaps the answer is hidden among other information, or perhaps it is simply not included in the material in question. When in doubt, the surest way to research and validate such a point is to set up and execute an experiment with an interactive SAS session.

Note: The Appendix of this paper contains a SAS routine that was used to generate most of the test data used in the examples in this text. The reader is encouraged to refer to that routine for complete details of the test data generation. However, each example used in this text will describe the particulars of each variable applicable to the problem and solution at hand.

In the first example, the variable “OrigOrder” is a variable which shows the order that each observation was in upon the creation of the dataset, while “RandomPosNegMiss” contains a random integer between -5 and +5 with some missing values mixed in. (Random500 is simply a random integer between 1 and 500 that was the original variable created in this dataset.) After sorting this dataset by “RandomPosNegMiss”, we can easily see that missing values bubble up to the top in a sort – they should be considered the smallest number possible for this purpose.

Orig				RandomPos			
Obs	Order	Random500	NegMiss	Obs	Order	Random500	NegMiss
1	1	269	1	1	20	372	.
2	3	456	2	2	22	178	-5
3	8	85	4	3	27	350	-4
4	11	152	3	4	19	428	-3
5	14	456	-1	5	78	438	-2
6	17	106	5	6	14	456	-1
7	19	428	-3	7	1	269	1
8	20	372	.	8	3	456	2
9	22	178	-5	9	11	152	3
10	27	350	-4	10	8	85	4
11	78	438	-2	11	17	106	5

➔
SORT

Alert readers will notice that “OrigOrder” does not contain consecutive numbers; there are a number of gaps in the sequence. This presentation will address methods to select and exclude values and variables later in the text. However, the techniques were employed in this early example to prevent the examples from dominating the text in sheer volume!

Having discussed missing values, an experienced SAS coder might inquire, “What about Special Missing Values – how do THEY sort?” A second experiment will demonstrate that all missing values, special or otherwise, bubble to the top of the sequence. Further, we can see that the ordinary “.” dot representation of the missing value will sort prior to the special missing values represented by A through Z; those special missing values sort (as expected) in alphabetical order. The “wild card” is that the underscore “_” can also be used as a special missing value. Unlike A through Z, this character representation of a special missing value actually sorts prior to the dot when sorting missing values. Based on the results of this experiment, it can be concluded that the SAS user should not use the underscore as a special missing

value except in the exceptional case when something is needed that will sort prior to the “ordinary” missing value denoted by the dot.

Orig				RandomPos			
Obs	Order	Random500	NegMiss	Obs	Order	Random500	NegMiss
1	1	269	1	1	85	432	—
2	3	456	2	2	20	372	.
3	8	85	4	3	65	313	A
4	10	29	M	4	10	29	M
5	11	152	3	5	22	178	-5
6	14	456	-1	6	27	350	-4
7	17	106	5	7	19	428	-3
8	19	428	-3	8	78	438	-2
9	20	372	.	9	14	456	-1
10	22	178	-5	10	1	269	1
11	27	350	-4	11	3	456	2
12	65	313	A	12	11	152	3
13	78	438	-2	13	8	85	4
14	85	432	—	14	17	106	5



SORTING ALPHANUMERIC CHARACTERS / VARIATIONS IN OPERATING SYSTEMS?

We all learned the sort order of alphabetic characters early in our lives – and if we ever forget, there is that “ABCDEFGH ...” song to fall back on! As such, we have a clear understanding of what to expect when sorting alphabetic strings. Unfortunately, those lessons never delved into dealing with special characters, or with combinations of alphabetic and numeric characters, or the differences between upper and lower case letters. The librarian at my elementary school advised me that “A” always falls before “B”, and so on – but in the case of a tie, then the Upper Case should be listed before lower case. Further, special characters like the ampersand (“&”) should just be ignored. I suspect this advice might have worked for 3rd graders, but would not suffice as a valid programming specification!

Unfortunately, most sort utilities, including SAS’s, do not work this way. Upper case values sort in sequence, and lower case values sort in sequence – but they do not intertwine. The big question is, of course, which comes first – upper case letters or lower case letters? The answer is, “it depends”.

A veteran user with experience on multiple operating systems might suspect at this point that there could be are differences based on operating systems (and novices may have suspected it based on the title of this section of the presentation!) Indeed, it is necessary to know what operating system is being used; this is one reason why most of the “SAS Companions for the <fill in the blank> Environment” manuals have a separate listing for PROC SORT! Those systems that store data in ASCII will sort the data in ASCII sequence by default, while EBCDIC systems sort in EBCDIC. Alert readers will have caught the “by default” clause; it is possible to override the default behavior by specifying either ASCII or EBCDIC as a keyword on the PROC SORT statement. (Alternatively, the keyword SORTSEQ=ASCII or SORTSEQ=EBCDIC can be used for the same purpose.)

A question may have arisen in some readers’ minds – “How is it possible to sort in EBCDIC on an ASCII machine, and vice versa?” The short answer is, “you aren’t”. By using the keyword, you are telling SAS to use the sort sequence employed by the particular character set.

The sort sequence is not the same for these two character sets. The most apparent difference is that under the English language implementation of EBCDIC, lower case letters sort ahead of upper case letters, which are sorted ahead of numbers. In ASCII’s English language implementation, that precedence is reversed; numbers dominate upper case letters which in turn precede lower case letters. Special characters sort in different orders under each collating sequence, as well. The precise order is documented under “SORTING ORDERS FOR CHARACTER VALUES” in the SAS Procedures Guide.

Let us try an example to illustrate this point. We will create a temporary dataset containing some numbers, some alphabetic characters – both upper and lower case – and several special characters

including the space or blank (placed in pre-sort sequence between the right square bracket and the left curly bracket: **a f A F _ [] { } @ # \$, . < > ;**

PROC SORT DATA=AsciiEbcDic			
ASCII;			
BY OneByte;			
RUN;			
PROC PRINT UNIFORM;			
RUN;			
Obs	OneByte	Obs	OneByte
1		11	>
2	#	12	@
3	\$	13	A
4	,	14	F
5	.	15	[
6	0	16	]
7	1	17	_
8	9	18	a
9	;	19	f
10	<	20	{
		21	}

VS.

PROC SORT DATA=AsciiEbcDic			
EBCDIC;			
BY OneByte;			
RUN;			
PROC PRINT UNIFORM;			
RUN;			
Obs	OneByte	Obs	OneByte
1		11	a
2	.	12	f
3	<	13	[
4	\$	14	]
5	;	15	{
6	,	16	A
7	_	17	F
8	>	18	}
9	#	19	0
10	@	20	1
		21	9

To emphasize the point - SAS may return different results from PROC SORT depending on whether it is employed under Z/OS (mainframe) vs. Windows or Unix (server / workstation) because of this operating system-dependent option. This violates one of the standard assumptions of SAS; that is, the same input passed through the same routine will return the same output regardless of operating system! If one is insistent that their sort results always stay the same regardless of operating system, then a standard sort sequence should be selected and hard coded. However, most users *want* their sort to be in sync with their operating system of choice, and only care about this anomaly from a curiosity standpoint.

Alert readers may have caught a specific reference to “English” when referring to EBCDIC and ASCII earlier in this presentation. The SAS developers are fully aware that there are talented coders out there who might be using one of the Scandinavian languages. To facilitate their use of PROC SORT, the keywords “DANISH”, “FINNISH”, “NORWEGIAN”, and “SWEDISH” can be used to point to the appropriate collating sequence. These 4 keywords can also be used with SORTSEQ=, as can “SPANISH” and “ITALIAN”. Under SAS 9.2, “POLISH” has also been added to the mix as both a separate keyword and in conjunction with SORTSEQ.

Despite all of these new options, there is a chance that someone may be using a language, or more precisely a character set, for which a standard sort sequence does not yet exist in SAS. (Perhaps our Scandinavian friends also need to deal with Icelandic, for example.) The good folks who support base SAS realized this, and provide an additional procedure called PROC TRANTAB. The TRANTAB Procedure can be used to create, edit and/or display a customized translation table. Discussion of the syntax and usage of this procedure is outside of the scope of this presentation; interested readers are directed to the online SAS documentation for further information.

Of course, one would not expect to be able to specify multiple sort sequences for the same sort; why would anyone need to request both Danish AND Finnish simultaneously? It is probably obvious, but for the record, only one collating sequence is allowed per unique SORT.

Most if not all of the European and European-based languages contain the concept of upper and lower case letters, so let us return to the original discussion of upper case vs. lower case. Anyone who has looked at the sort order for characters in either EBCDIC or ASCII will soon realize that PROC SORT will not handle this situation in the same way for the two collating sequences. The two character representations do not agree as to whether the upper case sequence dominates over the lower case, or vice versa. However, at least both handle the entire series of 26 letters by case before addressing the

opposite case of 26 letters! It may be necessary to code a workaround if a traditional English (or other) language sort is desired.

Our test dataset contains a variable called ThreeLetterMixed. As the name might imply, this contains a 3 character string randomly generated using a combination (potentially) of upper and lower case letters. When we sort the dataset by this variable, we find that PROC SORT respects case – upper or lower – more than it does the actual order of the alphabet. (Aside: Note the use of the KEEP option on the DATA= option of PROC SORT and the WHERE= option on the DATA= option of PROC PRINT – we will briefly discuss these options later. The intent at this point is to select a small subset of the data in order to demonstrate the point at hand; the details will be discussed later.)

```
PROC SORT DATA=TEMP (KEEP=OrigOrder Random500 ThreeLetterMixed)
    OUT=TEMPMIXED;
    BY ThreeLetterMixed ;
RUN;

PROC PRINT
    DATA=TEMPMIXED (WHERE= (UPCASE (SUBSTR (ThreeLetterMixed,1,1))='M'))
    UNIFORM;
RUN;
```

Obs	Orig Order	Random500	Three Letter Mixed
46	187	348	MEH
47	20	99	MJv
48	44	344	MPY
49	143	276	MWz
50	4	283	MZx
51	159	253	Mda
52	140	117	MfE
187	62	315	mCA
188	42	358	mVX
189	191	416	moU

Note that the upper case “M” bubbles ahead of the lower case “m” – this indicates an ASCII sort.

One quick solution to the problem would be to add an additional variable to the dataset – one that contains the same contents as ThreeLetterMixed, only converted to Upper Case. (As a caution, we cannot simply convert ThreeLetterMixed, as it is highly likely that someone might want to see the original value of the variable at some point!) This variable can be truly transient in nature by using the DROP= option to remove it – or if preferred, by using KEEP= on every variable except for it.

```
DATA TEMP2;
    SET TEMP (KEEP=OrigOrder Random500
                ThreeLetterMixed ThreeLetterCaps);
    ThreeLetterCaps = UPCASE ( ThreeLetterMixed );
RUN;

PROC SORT DATA=TEMP2 ;
    BY ThreeLetterCaps ;
RUN;

PROC PRINT
    DATA=TEMP2 (WHERE= (UPCASE (SUBSTR (ThreeLetterMixed,1,1))='M'))
    UNIFORM;
RUN;
```

Obs	Orig Order	Mixed Order	Random500	Three Letter Mixed	Three Letter Caps
109	62	187	315	mCA	MCA
110	159	51	253	Mda	MDA
111	187	46	348	MEq	MEQ
112	140	52	117	MfE	MFE
113	20	47	99	MJv	MJV
114	191	189	416	moU	MOU
115	44	48	344	MPY	MPY
116	42	188	358	mVX	MVX
117	143	49	276	MWz	MWZ
118	4	50	283	MZx	MZX

The 3 letter string is now alphabetized without regard to whether we are dealing with upper or lower case.

This quick addition to our routine allows us to satisfy the requirement of a true alphanumeric sort. However, “quick” is a relative term. From a coding standpoint, it may only take seconds to add the necessary components to the routine in question. However, there are many factors to consider over and above the programmer’s time. For example, how much additional testing does this change require according to the company or agency standards, and does that effort require a separate independent testing resource? How much extra CPU (and clock) time will the additional step take, and how much additional disk space will the sort require? The “simple” change may, in fact, be the tip of the iceberg!

Individuals who have upgraded to SAS 9.2 have a new option at their disposal which truly does make the process trivial. A new option available for SORTSEQ= – sortseq=*linguistic* – handles mixed case sorts without the need for jury-rigging ones existing routine and data.

```
PROC SORT DATA=TEMP2 SORTSEQ=linguistic;
  BY ThreeLetterMixed ;
RUN;

PROC PRINT
  DATA=TEMP2 (WHERE= (UPCASE (SUBSTR (ThreeLetterMixed,1,1))='M'))
  UNIFORM;
RUN;
```

Obs	Orig Order	Mixed Order	Random500	Three Letter Mixed	Three Letter Caps
109	62	187	315	mCA	MCA
110	159	51	253	Mda	MDA
111	187	46	348	MEq	MEQ
112	140	52	117	MfE	MFE
113	20	47	99	MJv	MJV
114	191	189	416	moU	MOU
115	44	48	344	MPY	MPY
116	42	188	358	mVX	MVX
117	143	49	276	MWz	MWZ
118	4	50	283	MZx	MZX

This table looks exactly like the one produced earlier when we added the extra field with all capital letters and sorted by it.

The SAS 9.2 documentation states that the linguistic option on SORTSEQ “are largely compatible with” the Unicode Collation Algorithms (UCA) – which can also be conversely interpreted to mean that they are not 100% compatible with UCA, if this is of concern. There are a number of options available for “linguistic” which are documented in the manual.

One of the options available for “linguistic” does deserve special mention, because it helps to solve another long-standing problem with sorting. Many programmers, at one time or another, have been forced with processing address data. This data can be problematic; street addresses are traditionally stored as character data, but actually contain a combination of numbers and letters. This can present

problems during sorting; numbers stored in character format are sorted based on a character collation rather than as numeric values. Most of us have seen a dataset sorted with a sequence such as “1”, “10”, “2” This is easily explainable from a processing standpoint –the sequence “1” and “blank” comes before the sequence “1” and “0”. However, it is not so easily explainable to non-computer types who cannot understand any scenario where “10” would come before “2”.

A quick solution to the problem would be to store the street number and the street name in separate fields, then sort by number then name. This is not a complete solution; one glaring weakness is that it does not handle the common situation of street names that contain numbers. SAS 9.2 users can expand upon the options just discussed with “sortseq=linguistic(Numeric_Collation=ON)” to easily address (no pun intended) this type of data.

Traditional SAS (and other) sort techniques handle the data in a strict character collating sequence, which may not conform to a numerical order expected by the end user. To illustrate this point, let us take a dataset containing assorted addresses – specialized addresses using a lot of numeric characters, and sort it using a traditional character sequence. We will see that the “sorted” data does not respect the concept of numeric order since all sorting is done on the character representations of those numbers.

<pre>PROC SORT DATA=ADDRESSES OUT=ADDRESSES_CHAR ; BY ADDRESS ; RUN; PROC PRINT DATA=ADDRESSES_CHAR UNIFORM; RUN;</pre>	 <i>SORT</i>	<table><tr><th>Obs</th><th>ADDRESS</th></tr><tr><td>Obs</td><td>ADDRESS</td></tr><tr><td>1</td><td>14 12th Street</td></tr><tr><td>2</td><td>14 121st Street</td></tr><tr><td>3</td><td>141 12th Street Apt 1</td></tr><tr><td>4</td><td>141 12th Street Apt 11</td></tr><tr><td>5</td><td>141 12th Street Apt 2</td></tr><tr><td>6</td><td>141 121st Street</td></tr><tr><td>7</td><td>1411 12th Street</td></tr></table>	Obs	ADDRESS	Obs	ADDRESS	1	14 12th Street	2	14 121st Street	3	141 12th Street Apt 1	4	141 12th Street Apt 11	5	141 12th Street Apt 2	6	141 121st Street	7	1411 12th Street
Obs	ADDRESS																			
Obs	ADDRESS																			
1	14 12th Street																			
2	14 121st Street																			
3	141 12th Street Apt 1																			
4	141 12th Street Apt 11																			
5	141 12th Street Apt 2																			
6	141 121st Street																			
7	1411 12th Street																			

The new NUMERIC_COLLATION option allows for a more traditional (from a human standpoint) sort. Repeating the sort on the same dataset used above, we will find that the street addresses are sorted in such a way that both numeric street addresses and alphabetic characters are respected. Further, it is not limited to only the first number in the string; the street number, the street address, and even the apartment number are treated as numeric values during the sort, while the rest of the field is still handled as character.

<pre>PROC SORT DATA=ADDRESSES OUT=ADDRESSES_NUM SORTSEQ=linguistic (NUMERIC_COLLATION=ON) ; BY ADDRESS ; RUN; PROC PRINT DATA=ADDRESSES_NUM UNIFORM; RUN;</pre>	 <i>SORT</i>	<table><tr><th>Obs</th><th>ADDRESS</th></tr><tr><td>Obs</td><td>ADDRESS</td></tr><tr><td>1</td><td>14 12th Street</td></tr><tr><td>2</td><td>14 121st Street</td></tr><tr><td>3</td><td>141 12th Street Apt 1</td></tr><tr><td>4</td><td>141 12th Street Apt 2</td></tr><tr><td>5</td><td>141 12th Street Apt 11</td></tr><tr><td>6</td><td>141 121st Street</td></tr><tr><td>7</td><td>1411 12th Street</td></tr></table>	Obs	ADDRESS	Obs	ADDRESS	1	14 12th Street	2	14 121st Street	3	141 12th Street Apt 1	4	141 12th Street Apt 2	5	141 12th Street Apt 11	6	141 121st Street	7	1411 12th Street
Obs	ADDRESS																			
Obs	ADDRESS																			
1	14 12th Street																			
2	14 121st Street																			
3	141 12th Street Apt 1																			
4	141 12th Street Apt 2																			
5	141 12th Street Apt 11																			
6	141 121st Street																			
7	1411 12th Street																			

ADDITIONAL TOPICS

Time constraints prevent the author from completing this paper prior to the publication deadline for the conference Proceedings. Topics yet to be addressed include:

- *Space concerns? Cutting back records with KEEP= / DROP= / WHERE=, and actual disk space discussion.*
- *Selective record retention and removal - NoDup / NoDupKey. Plus “tiebreakers”.*
- *Dealing with pre-sorted data (and sorted data that doesn’t act sorted).*
- *System options dealing with sort (including discussion of multi-threading and DBMS.)*
- *Comparing “BY” vs. “CLASS” statements.*
- *Discussion of Indexes.*
- *Brief mention of Hashing.*

There is a PowerPoint presentation associated with this presentation; it has been completed and will be discussed at the lecture associated with this paper.

CONCLUSION

The basics of sorting in SAS can be taught and understood in a manner of minutes. However, like an onion, there are layers upon layers of details that can be studied and employed if the reader is interested in doing so. It is not simply sorting that should be examined, but sorted data in general; it is possible to use data as though it were sorted, even though it is not.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

The author can be contacted via e-mail as follows:
KuligowskiConference@gmail.com

REFERENCES / FOR FURTHER INFORMATION

Clifford, Billy. (2005) “Frequently Asked Questions About SAS® Indexes”. *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

First, Steven. (2006). *Class Notes: Introduction to SAS®*. Madison, WI: Systems Seminar Consultants, Inc.

First, Steven. (2006). *Class Notes: SAS® Efficiencies*. Madison, WI: Systems Seminar Consultants, Inc.

Langston, Rick (2007). *What’s Coming in Version 9.2 of Base SAS Software*. Lecture presented as Opening Session at Fifteenth Annual Conference of the SouthEast SAS Users Group, November 2007.

Raithel, Michael. (2005) “*The Basics of Using SAS® Indexes*”. *Proceedings of the Thirtieth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.

Raithel, Michael. (2006) *The Complete Guide to SAS® Indexes*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2006). *SAS® Online Documentation, Version 9*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2008). *What’s New in Base SAS® 9.2*. Cary, NC: SAS Institute, Inc.

Wiktionary contributor(s), uncredited. (2009) *Sort*. <http://en.wiktionary.org/wiki/sort>

ACKNOWLEDGMENTS

I would like to thank Lisa Eckler for suggesting the topic and issuing the necessary invitations and acceptances, and Tony McClay for providing a second opinion or two along the way.

APPENDIX – Sample Data Generation Routine

The routine below was used to generate most of the test data used for the examples in this paper.

```
%LET LoopLen = 250;

DATA TEMP(KEEP=OrigOrder RandomBool Random500 RandomPosNegMiss
          ThreeLetterUpOnly ThreeLetterMixed);
  LENGTH Alphabet $ 52. ;
  RETAIN Alphabet 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
  MISSING A M Z ;
  DO OrigOrder = 1 TO &LoopLen;
    RandomBool = INT( RANUNI(0)+ .5 );
    Random500 = INT( RANUNI(0)*500 ) + 1;
    IF MOD( OrigOrder, 5 ) = 0 THEN DO;
      RanMiss = RANUNI(0) ;
      IF RanMiss < .25 THEN RandomPosNegMiss = . ;
      ELSE IF RanMiss < .50 THEN RandomPosNegMiss = .A ;
      ELSE IF RanMiss < .75 THEN RandomPosNegMiss = .M ;
      ELSE RandomPosNegMiss = .Z ;
    END;
    ELSE DO;
      RandomPosNegMiss = INT( RANUNI(0)*5 ) + 1;
      IF RANUNI(0) > .75 THEN
        RandomPosNegMiss = RandomPosNegMiss* -1 ;
    END;
    ThreeLetterUpOnly = SUBSTR( Alphabet, INT(RANUNI(0)*26)+1, 1 ) ||
                      SUBSTR( Alphabet, INT(RANUNI(0)*26)+1, 1 ) ||
                      SUBSTR( Alphabet, INT(RANUNI(0)*26)+1, 1 ) ;
    ThreeLetterMixed = SUBSTR( Alphabet, INT(RANUNI(0)*52)+1, 1 ) ||
                      SUBSTR( Alphabet, INT(RANUNI(0)*52)+1, 1 ) ||
                      SUBSTR( Alphabet, INT(RANUNI(0)*52)+1, 1 ) ;

    OUTPUT;
  END;
RUN;

PROC SORT DATA=TEMP;
  BY RandomBool RandomPosNegMiss ;
  /** replace with other variables as needed ***/
RUN;

PROC PRINT DATA=TEMP UNIFORM;
RUN;
```