



THE
POWER
TO KNOW.

Technical Paper

Linguistic Collation: Everyone Can Get What
They Expect

Sensible Sorting for Global Business Success

Content providers for Linguistic Collation: Everyone Can Get What They Expect were Manfred Kiefer, Globalization Specialist, SAS Heidelberg, Germany and Scott Mebust, Software Developer, SAS Cary, USA.

Table of Contents

Abstract	1
Linguistic Collation – What does it mean?	1
How is it used?	1
Expectations fulfilled	2
Examples of culturally expected sorting	7
Linguistic collation for the global enterprise	13
Conclusion	21
Acknowledgments	21
References	21
Appendix A: Things to be aware of	22
Appendix A1: General Notes	22
Appendix A2: Platform Notes	22
Appendix A3: BY Processing and Formatted Variables	22
Appendix A4: CLASS Processing versus BY Processing	33
Appendix B: Where to Turn for More Information	35

Abstract

In "Creating Order out of Character Chaos: Collation Capabilities of the SAS System" the authors describe the collation capabilities offered by PROC SORT in SAS and explain their respective applicability, advantages, and the processing implications of each approach. This paper concentrates on the true linguistic collation capabilities that SAS offers, and shows how everyone gets their expected results when searching for data in "sorted" order.

Global enterprises have information in many languages and from many different regions. Gathering the right information is crucial to their success. Sorting is ubiquitous in data processing, be it when searching for authors, titles, topics in on-line documentation, or when ordering a customer database by name or address, and so on. However, the expected sort order for the same data can differ a lot across languages and cultures. The SAS System makes sure that everyone can get what he or she expects by supporting sorts for data from one or many languages as well as case-insensitive sorts.

The paper outlines how BY processing now supports linguistic collation in PROC and DATA steps and how a new DATA step function called sortKey allows customers to create keys for sorting.

Linguistic Collation – What does it mean?

Although there are recognized standards for collationⁱ, the way people look at data in "sorted" order differs a lot. German collation is different from French, and a Danish one is again different from both—just to name a few. Even within a language community, there can be subtle differences: a German phone book sort is different from a dictionary sort, traditional Spanish sort order is different from the modern one, and so on. Users of languages based on alphabetic writing systems that make a distinction between upper- and lowercase letters, might want to sort uppercase before lowercase or vice versa or do a case insensitive sort.

Sorting is often called "alphabetization," though collation is not limited to ordering letters of an alphabet. For non-alphabetic writing systems as used in Asian languages, collation can be either phonetic or based on the number of pen strokes or simply on the position of the characters within an encoding (for example, Japanese kanji are usually sorted in the order of their Shift-JIS codes).

To implement linguistic collation, SAS has adopted the International Components for Unicode (ICU). The ICU and its implementation of the Unicode Collation Algorithm (UCA) have become a de facto standard. Nevertheless, people are free to choose: For example, most Japanese customers expect the Shift-JIS order, instead of the UCA.

Therefore, it is best to consider sorting as an 'a la carte' menu that you can make your choices from. People finds an item in a sorted list easily only if it is sorted in the expected order of their particular culture.

How is it used?

Invocation of linguistic collation with PROC SORT is quite simple. The only requirement is the specification of LINGUISTIC as the value to the SORTSEQ procedure option:

```
proc sort data=foo SORTSEQ=LINGUISTIC;
  by x; run;
```

Synonymously, one can specify `SORTSEQ=UCA`. This causes the `SORT` procedure to collate linguistically, in accordance with the current system `LOCALE` setting. The collating sequence used is the default provided by the ICU for the given locale. Options that modify the collating sequence can be specified in parentheses following the `LINGUISTIC` or `UCA` keywords. Generally, it is not necessary to specify option settings because the ICU associates option defaults with the various languages and locales. `PROC SORT` currently allows only a subset of the ICU options to be specified. These options include `STRENGTH`, `CASE_FIRST`, `COLLATION`, and `NUMERIC_COLLATION`. In addition, a `LOCALE` option is available to instruct `SORT` to use a collating sequence that is associated with a locale other than the current locale.

The `STRENGTH` option specifies the collation strength and corresponds to the various levels in the multi-level collating algorithm used by the ICU and UCA. `STRENGTH` can be set to a number between 1 and 5 or to the corresponding values `PRIMARY`, `SECONDARY`, `TERTIARY`, `QUATERNARY`, or `IDENTICAL`. For most languages based on the Latin alphabet, the `PRIMARY` level corresponds to alphabetic differences, the `SECONDARY` level corresponds to diacriticⁱⁱ differences, and the `TERTIARY` level corresponds to differences in character case. The default strength is locale dependent but is `TERTIARY` in most cases. Specification of a particular `STRENGTH` includes all levels up to and including the one specified by the strength value. For example, a `TERTIARY` strength setting includes levels one through three which, for languages with Latin-based alphabets, includes differences in alphabet, differences in accents, and differences in case.

The `CASE_FIRST` option controls whether lowercase characters are collated before uppercase characters or vice versa.

The `COLLATION` option allows selection of different collation types, such as `TRADITIONAL` Spanish collation versus modern Spanish collation, or a `PHONEBOOK` style collation for the German language.

The `NUMERIC_COLLATION` option allows integers, expressed as text in a character string, to be ordered numerically. Normally, numbers that are aligned toward their most significant digits (left aligned for languages that are written from left to right) within a character string will not sort numerically. Instead, the numbers is grouped according to the first digit encountered so that, for example, 1 and 10 appear before 2 and 20. One way to obtain a numeric ordering of character strings is to ensure that the numbers are aligned toward their least significant digits (right aligned for left-to-right languages). Using the `NUMERIC_COLLATION` option allows a numeric ordering regardless of the alignment.

The Unicode Consortium's technical report on the Unicode Collation Algorithm and the ICU documentation provide more details on the various option settings and their meanings.

Linguistic collation not only provides a sequence that is culturally correct and intuitive but also one that is largely independent of the underlying encoding or platform on which the collation is being performed. While the ordering of character strings has some dependence upon the code point values of the characters at the highest level, level 5, the major ordering is established by the lower levels and this ordering is independent of both the encoding of the strings and the system on which the code is executing. The collating sequences obtained with the ICU are, therefore, consistent across systems.

Expectations fulfilled

The implementation of true linguistic sorting has fulfilled long-standing demands of many customers. Let us have a closer look at these in detail.

In earlier versions of the SAS System, international customers had the possibility to use translation tables (trantabs) for defining alternative collating sequences. Though this possibility still exists, it does have serious

restrictions. Translation tables are limited to remapping or reordering up to 256 characters from a single-byte encoding. A TRANTAB is limited in its ability to create a collating sequence that is intuitive or that meets cultural expectations. For example, French compares accented characters from right to left, not from left to right as other languagesⁱⁱⁱ. A sort using a translation table cannot make such a fine distinction, because a translation table can assign only a single “weight” to a character but cannot distinguish between several collation levels. For example: create a simple name^{iv} list with:

```
data list;
  input name $20.;
datalines;
Côté
Boucher
Fournier
Cotée
Legrand
Dubois
Thibeault
Martin
Vaudron
Girard
;
run;
```

Now we can sort it with the Institute-provided translation table FRSOLAT1 like this:

```
proc sort data=list sortseq=frsolat1;
  by name;
run;
proc print data=list; run;
```

to create the following output:

```
1  Boucher
2  Cotée
3  Côté
4  Dubois
5  Fournier
6  Girard
7  Legrand
8  Martin
9  Thibeault
10 Vaudron
```

At first glance, names with accented characters seem to be sorted correctly; however, the French convention requires “Côté” to be sorted before “Cotée”. This can be achieved only with:

```
proc sort data=list sortseq=linguistic (locale=fr_FR);
  by word;
run;
proc print data=list; run;
```

which yields:

```
1  Boucher
2  Côté
3  Cotée
```

4	Dubois
5	Fournier
6	Girard
7	Legrand
8	Martin
9	Thibeault
10	Vaudron

Digraphs that are used in a number of languages cannot be handled with translation tables either. A digraph is a pair of letters used to write a distinct sound. Spanish has “ch” and “ll”, Croatian has “lj”, “nj”, “dž”, and Hungarian is particularly rich in these, with: “cs”, “dz”, “gy”, “ly”, “ny”, “sz”, “ty”, and “zs”. They ought to be treated as a unit or as a single letter.. For example, in a traditional Spanish sort order^v the word “llama” should follow the word “luz,” and “charla” should follow the word “curva”. For obvious reasons, this cannot be achieved with a trantab approach.

The ASCII encoding sorts uppercase before lowercase letters; the EBCDIC encoding does just the opposite. By specifying SORTSEQ=ASCII mainframe users could already do an “ASCII sort” before downloading their data to an “ASCII machine”. This is of particular use for people using English or some other languages (there are very few of them) that use only the upper and lowercase letters A-Z. However, it is not of much use for people using languages (there are many) that use accented characters such as ä, é, or ø, not to speak of those that use a very different script. An ASCII sort will place accented characters arbitrarily after the letters A-Z. Using another encoding value lists characters only according to their position in this particular encoding. So even a SORTSEQ=wlatin1 which order accented characters according to their position in the Windows Latin1 encoding does not produce a correct sort order for a language like Swedish, even though all Swedish characters are represented in Windows Latin1. Using SORTSEQ=LINGUISTIC in conjunction with LOCALE=sv_SE, however, does the trick. That is, it will sort characters adequately, and it provides transparent access after downloading the data from an EBCDIC to an “ASCII” host and or platform. In other words, the sort indicator is maintained, and the collating sequence is recognized.

In SAS 9.2, BY processing has been modified to recognize that a data set has been linguistically sorted so that NOTSORTED is no longer required. This means that a major restriction for international customers has been removed.

Normally, the use of an alternating collating sequence requires either the NOTSORTED option to be used in a BY statement or the system NOBYSORTED option to be specified in order to disable the observation sequence check that is performed when BY processing. Neither option is necessary when BY processing a data set that has been linguistically sorted. In addition to honoring the alternate collating sequence, BY processing is also sensitive to linguistic collation options when determining BY group boundaries. With linguistic collation, specification of a SECONDARY strength groups all character variable values that differ in only character case. Specification of a PRIMARY strength groups values that differ only in case or accents. And, use of the NUMERIC_COLLATION option groups values that are numerically equivalent. This effect on BY processing can be helpful for example: BY processing data that has been entered inconsistently and it is desired that such differences be ignored.

In the example below we are merging two data sets—one containing monthly revenue with another containing a monthly count of customers, to calculate revenue per customer.

```
data clients;
  length mois $ 10;
  infile datalines delimiter=',';
  input mois compte;
  datalines;
  janvier, 370
  février, 400
  mars, 430
  avril, 415
```

```

    mai, 410
    juin, 450
    juillet, 449
    août, 403
    septembre, 339
    novembre, 375
    décembre, 370
;
run;

data revenu;
  length mois $ 10;
  infile datalines delimiter=',';
  input mois ventes;
  datalines;
JANVIER, 376784
FEVRIER, 396911
MARS, 441327
AVRIL, 419272
MAI, 408291
JUIN, 443791
JUILLET, 442111
AOUT, 402771
SEPTEMBRE, 337727
NOVEMBRE, 381929
DECEMBRE, 376771
;
run;

proc sort data=clients sortseq=linguistic(strength=1);
  by mois;
run;

proc sort data=revenu sortseq=linguistic(strength=1);
  by mois;
run;

data resultat;
  merge clients revenu;
  by mois;
  revenuparclient = ventes/compte;
run;

proc print;
run;

```

The following output displays the results:

The SAS System				
Obs	mois	compte	ventes	revenuparclient
1	AOUT	403	402771	999.43
2	AVRIL	415	419272	1010.29
3	DECEMBRE	370	376771	1018.30
4	FEVRIER	400	396911	992.28
5	JANVIER	370	376784	1018.34

6	JUILLET	449	442111	984.66
7	JUIN	450	443791	986.20
8	MAI	410	408291	995.83
9	MARS	430	441327	1026.34
10	NOVEMBRE	375	381929	1018.48
11	SEPTEMBRE	339	337727	996.24

The next example demonstrates (a) how BY processing simply works with linguistically sorted data (without the need for the NOTSORTED option) and (b) how linguistic collation options can be used to ignore inconsistencies in data caused for example, mistakes in data entry for example, differences in capitalization and or accenting).

```

data revenu;
  length mois $ 10;
  infile datalines delimiter=',';
  input mois année ventes;
  datalines;
Janvier, 2007, 376784
Février, 2007, 396911
Mars, 2007, 441327
Avril, 2007, 419272
Mai, 2007, 408291
Juin, 2007, 443791
Juillet, 2007, 442111
Août, 2007, 402771
Septembre, 2007, 337727
Novembre, 2007, 381929
Décembre, 2007, 376771
janvier, 2006, 376297
février, 2006, 397221
mars, 2006, 442176
avril, 2006, 417171
mai, 2006, 409912
juin, 2006, 441376
juillet, 2006, 440191
août, 2006, 399713
septembre, 2006, 335271
novembre, 2006, 371292
décembre, 2006, 377979
JANVIER, 2005, 367487
FEVRIER, 2005, 369119
MARS, 2005, 434127
AVRIL, 2005, 420299
MAI, 2005, 409112
JUIN, 2005, 434917
JUILLET, 2005, 424119
AOUT, 2005, 472117
SEPTEMBRE, 2005, 373272
NOVEMBRE, 2005, 382919
DECEMBRE, 2005, 367171
;
run;

proc sort data=revenu sortseq=linguistic(strength=1);
  by mois;
run;

```

```

proc summary data=revenu ;
  by mois;
  output out=sommaire mean(ventes)= moyenne; run;

proc print data=sommaire;
run;

```

The output makes it obvious that character case and accents are insignificant during BY processing:

Obs	mois	_TYPE_	_FREQ_	moyenne
1	Août	0	3	424867.00
2	Avril	0	3	418914.00
3	Décembre	0	3	373973.67
4	Février	0	3	387750.33
5	Janvier	0	3	373522.67
6	Juillet	0	3	435473.67
7	Juin	0	3	440028.00
8	Mai	0	3	409105.00
9	Mars	0	3	439210.00
10	Novembre	0	3	378713.33
11	Septembre	0	3	348756.67

Linguistic sorting allows differences such as case and accents to be ignored allowing BY groups to contain character BY variable values that are distinct when compared in a binary fashion, character for character. In the example above, because the option STRENGTH=1 was specified, “Août”, “août”, and “AOUT” are considered equivalent values and fall within a single BY group. Such BY groups are denoted by the first variable value encountered within the sorted data. This is most easily observed by examining the BY group boundary values shown in the SAS listing and output by procedures such as PROC PRINT. It is for this reason that the variable MOIS, in the first observation listed in the output of the example above, has the value “Août” and not “août” or “AOUT”.

Examples of culturally expected sorting

Let us go on a virtual tour around the world and visit several countries and cities. Let us start our journey in the north of Europe, in Sweden. The Swedish alphabet has two interesting features: the characters “å”, “ä”, and “ö” are considered as distinct letters to be sorted after the “z” whereas in most other European languages they are considered as accented characters to come right after the base characters. The letter “W” is used only in family names and in words that have been borrowed from foreign languages; hence, it used to be treated as a mere variant of “V”, and so “V” and “W” have been sorted as one letter. This practice is still commonly encountered in Swedish dictionaries and telephone books, though just recently the Swedish Academy separated the two letters in conformity with international lexicographic practice^{vi}.

Let us see what a linguistically sorted list of Swedish names and cities looks like:

Obs	name	first	city
-----	------	-------	------

1	Bergström	Birgitta	Alingsås
2	Blomqvist	Margareta	Borås
3	Håkansson	Anders	Ängelholm
4	Johansson	Erik	Uppsala
5	Lund	Märta	Åkersberga
6	Strömberg	Gunnar	Ystad
7	Wallin	Lars	Luleå
8	Vikström	Linnéa	Örebro
9	Åström	Eva	Göteborg
10	Öberg	Åsa	Stockholm

Obs	name	first	city
1	Bergström	Birgitta	Alingsås
2	Blomqvist	Margareta	Borås
3	Åström	Eva	Göteborg
4	Wallin	Lars	Luleå
5	Öberg	Åsa	Stockholm
6	Johansson	Erik	Uppsala
7	Strömberg	Gunnar	Ystad
8	Lund	Märta	Åkersberga
9	Håkansson	Anders	Ängelholm
10	Vikström	Linnéa	Örebro

In the first case, it has been sorted by name, in the second by city.

We now proceed to Germany. German distinguishes between a "phone book" sort and a "dictionary" sort. With the former the umlauted vowels "ä, ö, ü" are sorted with "ae, oe, ue" after "ad, od, ud". With the latter (the more frequent), the umlauted vowels are sorted with the simple vowels "a, o, u" while the alternate spellings "ae, oe, ue" in personal names are sorted after "ad, od, ud" respectively. Linguistic sorting allows one to take care of these varieties by specifying the keyword `collation=phonebook`.^{vii} (Dictionary order is the default and cannot be specified as keyword.)

The differences can be clearly seen by comparing the output of a default (dictionary) sort with the one of a phone book sort. With the former, names with "ö" and "ü" sort after names with base characters; with the latter it is the other way round.

Dictionary sort:

Obs	name	first
1	Mader	Ernst
2	Mader	Fritz
3	Mader	Josef
4	Meder	Bruno
5	Meder	Regina
6	Meier	Hans
7	Mlynek	Mike
8	Molitor	Martina
9	Möller	Ellen
10	Möller	Georg
11	Möller	Sabine
12	Mras	Cindy
13	Muller	George
14	Muller	Pam
15	Muller	Susan
16	Müller	Christina

17	Müller	Heinz
18	Müller	Max
19	Müller	Renate
20	Mwamba	Ed
21	Myrzik	Peter
22	Mzyk	Mary

Phonebook sort:

Obs	name	first
1	Mader	Ernst
2	Mader	Fritz
3	Mader	Josef
4	Meder	Bruno
5	Meder	Regina
6	Meier	Hans
7	Mlynek	Mike
8	Möller	Ellen
9	Möller	Georg
10	Möller	Sabine
11	Molitor	Martina
12	Mras	Cindy
13	Müller	Christina
14	Müller	Heinz
15	Müller	Max
16	Müller	Renate
17	Muller	George
18	Muller	Pam
19	Muller	Susan
20	Mwamba	Ed
21	Myrzik	Peter
22	Mzyk	Mary

Similarly, for Spanish you can use both a traditional sort order and a modern sort order. (The latter is the default.) Using the keyword collation=traditional^{viii} allows for treating the digraphs “ch” and “ll” as separate letters.

Let us now travel further to the southeast and develop an understanding of the intricacies of Hungarian culture and language as well as see where the ICU algorithm meets its limits. As mentioned earlier, Hungarian is rich in digraphs, which ought to be sorted as separate letters. Here is our list with Hungarian names:

Balázs Ildikó
 Lynesné Antal Mária
 Szabó Győző
 Zsóri Gergely
 Kovács Szabolcs
 Babsó Zsuzsanna
 Lychnovszky Ferenc
 Tyukász Anna Krisztina
 Tyukász György
 Székely Mihály
 Bundzsák Dezső
 Nyáguly Gergely
 Márkus Zsóka
 Csernus Gábor
 Györffy Szilárd
 Dzsida Jenő

Zsóri Csilla
 Cudar Vilmos
 Nyáguly Csengezy István
 Lyankus István

After sorting, it looks like this:

Obs	name	first
1	Balázs	Ildikó
2	Bazsó	Zsuzsanna
3	Bundzsák	Dezső
4	Cudar	Vilmos
5	Csernus	Gábor
6	Dzsida	Jenő
7	Győrffy	Szilárd
8	Kovács	Szabolcs
9	Lyankus	István
10	Lychnovszky	Ferenc
11	Lynesné	Antal Mária
12	Márkus	Zsóka
13	Nyáguly	Gergely
14	Nyáguly	Csengezy István
15	Szabó	Győző
16	Székely	Mihály
17	Tyukász	Anna Krisztina
18	Tyukász	György
19	Zsóri	Gergely
20	Zsóri	Csilla

"Cudar Vilmos" precedes "Csernus Gábor" as expected. However, "Lychnovszky Ferenc" should precede "Lynesné Antal Mária" and "Lyankus István" because in the first two cases the letters "Ly" are not considered as digraphs because these names are of foreign origin, not Hungarian origin. Admittedly, the example does seem a little far-fetched, and in general, the ICU algorithm works very well. Sorting these names could be managed only by a vocabulary-based solution.

Now, it is time to mention another restriction. Here is a simple list with some glitches:

MacDonald
 Robertson
 Madden
 Brown
 Mackintosh
 McKinley
 Mc Arthur

The sorted output looks like this:

Brown
 MacDonald
 Mackintosh
 Madden
 Mc Arthur
 McKinley
 Robertson

One might have expected the Mac and or Mc spellings to have been sorted together—for example, with McKinley preceding Mackintosh, as if it had been spelled "MacKinley":

```
Brown
Mc Arthur
MacDonald
McKinley
Mackintosh
Madden
Robertson
```

This is the accepted ordering, for some cultures, of these family names when appearing in, for example, a phone book or a dictionary. However, since the advent of computer-sorted lists, this type of alphabetization has fallen out of favor^x.

Finally yet importantly, let us leave the world of alphabetic writing systems and investigate how to sort characters of an Asian language.

The table can be part of a bigger database that contains the names of Chinese customers:

Name	Name (English)	First name	First name (English)
李	Li	伟	Wei
王	Wang	建国	Jianguo
张	Zhang	东	Dong
陈	Chen	英	Ying
馬	Ma	雪	Xue

There are various possibilities for sorting the data—say, by name. You can use the COLLATION= option to do so. For Simplified Chinese you can specify GB2312HAN, PINYIN, or STROKE^x. GB2312HAN means that characters are ordered according to their position in the GB2312 standard (or EUC-CN encoding); PINYIN means using a phonetic spelling system in Latin characters for ordering, and STROKE means using the stroke order (the number of strokes it takes to draw a character) for defining the collating sequence. The results look very different. For example, the following code:

```
proc sort data=list out=sorted sortseq=linguistic (locale=zh_CN
  collation=GB2312HAN);
  by name;
run;
proc print data=sorted; run;
```

yields:

Name	Name (English)	First name	First name (English)
陈	Chen	英	Ying
李	Li	伟	Wei
王	Wang	建国	Jianguo
张	Zhang	东	Dong
馬	Ma	雪	Xue

Sorted with collation=PINYIN the result looks like this:

Name	Name (English)	First name	First name (English)
陈	Chen	英	Ying
李	Li	伟	Wei
馬	Ma	雪	Xue
王	Wang	建国	Jianguo
张	Zhang	东	Dong

and with collation=STROKE like this:

Name	Name (English)	First name	First name (English)
王	Wang	建国	Jianguo
李	Li	伟	Wei
馬	Ma	雪	Xue
张	Zhang	东	Dong
陈	Chen	英	Ying

Linguistic collation for the global enterprise

Global enterprises have data from all over the world, in all kinds of scripts and encodings. A good way to store such data centrally is to use a form of Unicode. You can then have data in native script and transliterated to Latin characters or just in English, for convenience. From there it is easy to create subsets and views for one locale or for multiple locales. If multilingual data are kept together and there is a need to order them, but the context does not define a particular locale, the Unicode Collation Algorithm (UCA) provides a convenient way to put them in sequence.

Let us imagine that we have a big database with names and addresses of customers both in native script and with plain Latin characters. Below is an excerpt:

```

name=Воронин name_e=Voronin first=Борис first_e=Boris city=Санкт-Петербург
city_e=Saint Petersburg country=Россия country_e=Russia
name=Śmigowska name_e=Smigowska first=Świetłana first_e=Swietlana
city=Warszawa city_e=Warsaw country=Polska country_e=Poland
name=Παπαρίζου name_e=Paparizou first=Ζωή first_e=Zoe city=Θεσσαλονίκη
city_e=Thessaloniki country=Ελλάδα country_e=Greece
name=Crespo name_e=Crespo first=Gustavo first_e=Gustavo city=Lleida
city_e=Lleida country=España country_e=Spain
name=Christensen name_e=Christensen first=Astrid first_e=Astrid city=Århus
city_e=Aarhus country=Danmark country_e=Denmark
name=Vikström name_e=Vikstrom first=Linnéa first_e=Linnea city=Örebro
city_e=Orebro country=Sverige country_e=Sweden
name=Müller name_e=Muller first=Alois first_e=Alois city=München
city_e=Munich country=Deutschland country_e=Germany
name=Côte name_e=Cote first=Frédéric first_e=Frederic city=Châteauroux
city_e=Chateauroux country=France country_e=France
name=רַחֵל name_e=Peretz first=רַחֵל first_e=Rachel city=תל אביב תל אביב
city_e=Tel Aviv country=יִשְׂרָאֵל country_e=Israel
name=Yılmaz name_e=Yilmaz first=Ekrem first_e=Ekrem city=İstanbul
city_e=Istanbul country=Türkiye country_e=Turkey
name=佐藤 name_e=Sato first=明子 first_e=Akiko city=東京 city_e=Tokyo
country=日本 country_e=Japan
name=馬 name_e=Ma first=雪 first_e=Xue city=南京 city_e=Nanjing country=中国
country_e=China
name=Lynesné name_e=Lynesne first=Antal Mária first_e=Antal Maria
city=Csabapuszta city_e=Csabapuszta country=Magyarország country_e=Hungary

```

If you do not decide to sort everything by the English name or English city but by name or city what does the sorted output look like? The answer: In a multilingual environment, the relative ordering of scripts has been defined by UCA as (using the above example) Latin – Greek – Cyrillic – Hebrew – CJK (Chinese – Korean – Japanese).

Will a city like Aarhus be sorted together with other cities that start with an “Å”? The answer is yes. But will they sort at the end of the alphabet that is. after a “z”^{xi}? No, unless you specify a LOCALE= option that requires such a convention (for example: Danish or Swedish). Will the city of “Lleida” be sorted after “Lugo”? No, unless you use the keyword collation=traditional together with a Spanish locale. Will Herr Müller’s record be sorted before Frau Muller’s? No, unless you use the keyword collation=phonebook together with a German locale.

Therefore, the general answer is: How to sort your data in a multilingual environment depends on your situation. The UCA algorithm provides enough flexibility to do so.

The addition of linguistic collation abilities to the SAS System is an evolutionary change and, of course, has been done in a way as to be backward compatible and not break existing code. These abilities have not, however, been extended into all parts of the system that manipulate and compare character string values. For example, basic string comparison operations such as EQ, LE, LT, GE, GT, NE, and IN, which can be used within a WHERE statement or SQL WHERE clause, have not been modified to be implicitly locale sensitive and to be performed in a linguistic fashion. Likewise, data set indexes created using character variables are not linguistically organized. Such can be made in the future but until that time, many of these abilities can be achieved manually through the explicit creation of linguistic sorting keys. The SORTKEY function has been provided for this purpose.

This function offers a convenient way of sorting multilingual data according to different language conventions. In our example above, we could create custom-made sort keys for sorting our data according to Swedish, French, German, Spanish convention, and so on.

The syntax is quite intuitive:

```
sortKey( string, <locale, strength, case, numeric, type>)
```

Only the first parameter, the data set variable or string constant from which to form the key, is required. The rest of the parameters are optional and, if constants, are specified within quotation marks. The *locale* name must be in the form of a POSIX name (for example: "sv_SE"). *Strength* uses the collation levels as explained above and is specified using a letter ("P", "S" for Secondary, and so on,). *Case* specifies, for Level 3, which character variant appears first and can be either "U" for upper or "L" for lower. *Numeric*, indicating numeric collation, is used to order numbers by the numeric value rather than by the characters that make up the number. And, collation *type* can be any of the types supported by the COLLATION option for example; P for PHONEBOOK or T for TRADITIONAL.

Hence, for creating four sort keys we would run the following code:

```
data list ;
  set list ;
  key_fr = sortkey(name, "fr_FR" ) ;
  key_de = sortkey(name, "de_DE" , , , 'P' ) ;
  key_sv = sortkey(name, "sv_SE" ) ;
  key_es = sortkey(name, "es_ES" , , , 'T' ) ;
run ;
```

The sort key for German ("key_de") would use the German phone book collating sequence; the Spanish one ("key_es") would use the traditional Spanish collating sequence.

You could then create copies of your data that are sorted according to the language convention of your choice for example:

```
PROC SQL ;
  create table fre_sort as
  SELECT * FROM list
  ORDER BY key_fr;
QUIT ;
```

Please note that a sort key should be considered as temporary and not as suitable for permanent storage. This is because the sortKey function uses ICU functionality, and ICU does not guarantee compatibility with future versions. Further, sort keys created using one set of options should not be compared with keys created with another set of options. Such a comparison is not valid and might return unexpected results.

Also note that within SCL, PROC SQL, and WHERE statements, missing For example: parameters must be specified by an empty string.

```
key_es = sortkey(name, "es_ES" , ' ' , ' ' , ' ' , 'T' ) ;
```

Another way to create a sorted copy and immediately drop the key could be achieved with the code below. It also bypasses the default length of 200 characters for an undeclared variable being assigned the results of a function returning a character value, which could cause truncation.

```
PROC SQL ;
  create table fre_sort as
  select *, sortkey(name,"fr_FR") as skey length=700 format=$hex20.
  label="French Key"
  from list
  ORDER BY skey ;
  alter table query drop skey ;
QUIT ;
```

The length of the key returned from the SORTKEY function is not constant but can vary and depends upon not only the SORTKEY function options but also on the input string. If the receiving character variable is insufficient in size then the SORTKEY function returns an error indicating that the key has been truncated. While quite large, an estimate of twelve times the length of the longest string from which a key is created and can be used as the initial length of the receiving character variable. To increase efficiency and decrease storage requirements, this estimate can be reduced to a size just above that for which a truncation error I occurs.

There is even more you can do with this function. You can use it for comparisons, creating subsets of your data, and indexing a data set.

Let us have another look at the database that contains names and addresses of customers. Here is an excerpt with some European addresses:

Name	Name (English)	First name	First name (English)	City	City (English)	Country	Country (English)
Gómez	Gomez	Juan	Juan	Barcelona	Barcelona	España	Spain
Martínez-Monés	Martinez-Mones	Leonardo	Leonardo	La Coruña	La Coruna	España	Spain
López Fernandez	Lopez Fernandez	Ángela	Angela	León	Leon	España	Spain
Sánchez	Sanchez	Miguel José	Miguel Jose	Algeciras	Algeciras	España	Spain
Llinares Sellés	Llinares Selles	Cristina	Cristina	Lugo	Lugo	España	Spain
Nuñez Navarro	Nunez Navarro	Ignacio	Ignacio	Madrid	Madrid	España	Spain
Chuliá Mengual	Chulia Mengual	David	David	Oviedo	Oviedo	España	Spain

Name	Name (English)	First name	First name (English)	City	City (English)	Country	Country (English)
Crespo	Crespo	Gustavo	Gustavo	Lleida	Lleida	España	Spain
Hernández Cerrillo	Hernandez Cerrillo	María del Pilar	Maria del Pilar	Zaragoza	Zaragoza	España	Spain
...
Côté	Cote	Frédéric	Frederic	Châteauroux	Chateauroux	France	France
Boucher	Boucher	Corinne	Corinne	Paris	Paris	France	France
Fournier	Fournier	Étienne	Etienne	Mâcon	Macon	France	France
Cotée	Cotee	Madeleine	Madeleine	Nîmes	Nimes	France	France
Legrand	Legrand	Claire	Claire	Orléans	Orleans	France	France
Dubois	Dubois	Benoît	Benoit	Yerres	Yerres	France	France
Thibeault	Thibeault	Georges	Georges	Évry	Evry	France	France
Martin	Martin	Désirée	Desiree	Fréjus	Frejus	France	France
Vaudron	Vaudron	Sébastien	Sebastien	Marseille	Marseille	France	France
Girard	Girard	Régine	Regine	Lyon	Lyon	France	France

Now, we would like to create a subset with all cities sorted after "Lleida". To do so we can use something like the code below:

```

data cities ;
  set list ;
  key = sortkey(city) ;
  put key= $hex40. ;
  where (sortkey(city) > sortkey("Lleida")) ;
run ;

/* The sorted list will start with "Lugo" */
PROC SQL ;
  create table sorted as
  SELECT * FROM cities
  ORDER BY key;
QUIT ;

```

As a result, the data set WORK.CITIES has 24 observations and 9 variables.

The SAS System

Obs	name	name_e	first	first_e	city	city_e	country	country_e	key
1	Llinares Sellés	Llinares Selles	Cristina	Cristina	Lugo	Lugo	España	Spain	>P4D0
2	Wallin	Wallin	Lars	Lars	Luleå	Lulea	Sverige	Sweden	>P>0(
3	Girard	Girard	Régine	Regine	Lyon	Lyon	France	France	>XDB0
4	Fournier	Fournier	Étienne	Etienne	Mâcon	Macon	France	France	@(.DB
5	Nuñez Navarro	Nunez Navarro	Ignacio	Ignacio	Madrid	Madrid	España	Spain	@(.J8.
6	Vaudron	Vaudron	Sébastien	Sebastien	Marseille	Marseille	France	France	@(JL0
7	Hoffmann	Hoffmann	Gustav	Gustav	Moers	Moers	Deutschland	Germany	@D0Jl
8	Mzyk	Mzyk	Regine	Regine	Mönchengladbach	Moenchengladbach	Deutschland	Germany	@DB.6 (6@...1
9	Müller	Muller	Alois	Alois	München	Munich	Deutschland	Germany	@PB.6 01
10	Cotée	Cotee	Madeleine	Madeleine	Nimes	Nimes	France	France	B8@0l
11	Sørensen	Sorensen	Niels	Niels	Odense	Odense	Danmark	Denmark	D.0BL

Remember that in the traditional Spanish collating sequence “ll” is treated as a separate character; so our output looks different with the code below:

```
data cities2 ;
  set list ;
  key = sortkey(city,"es_ES",,,, 'T') ;
  put key= key= $hex40. ;
  where (sortkey(city,"es_ES",',',',',', 'T')
    > sortkey("Lleida", "es_ES",',',',',', 'T'));
run ;

/* The sorted list will start with "Mâcon" */
PROC SQL ;
  create table sorted2 as
  SELECT * FROM cities2
  ORDER BY key;
QUIT ;
```

As a result, the data set WORK.CITIES2 has 21 observations and 9 variables.

Obs	name	name_e	first	first_e	city	city_e	country	country_e	key
1	Fournier	Fournier	Étienne	Etienne	Mâcon	Macon	France	France	@(.DB
2	Nuñez Navarro	Nunez Navarro	Ignacio	Ignacio	Madrid	Madrid	España	Spain	@(.J8.0
3	Vaudron	Vaudron	Sébastien	Sebastien	Marseille	Marseille	France	France	@(.JL0
4	Hoffmann	Hoffmann	Gustav	Gustav	Moers	Moers	Deutschland	Germany	@D0JL
5	Mzyk	Mzyk	Regine	Regine	Mönchengladbach	Moenchengladbach	Deutschland	Germany	@DB-1
6	Müller	Muller	Alois	Alois	München	Munich	Deutschland	Germany	@PB-1
7	Cotée	Cotee	Madeleine	Madeleine	Nîmes	Nimes	France	France	B8@0L
8	Sørensen	Sorensen	Niels	Niels	Odense	Odense	Danmark	Denmark	D.0BL
9	Hedegaard	Hedegaard	Marie	Marie	Ølstykke	Olstykke	Danmark	Denmark	D>LN>
10	Vikström	Vikstrom	Linnéa	Linnea	Örebro	Orebro	Sverige	Sweden	DJ0*JL
11	Legrand	Legrand	Claire	Claire	Orléans	Orleans	France	France	DJ>0

The following example demonstrates two more ways the SORTKEY function can be used: (a) to enhance the capability of a WHERE clause to select observations and (b) to optimize the performance of the WHERE clause processing by using an index.

```

data lista;
  set list;
  length llave $ 30;
  format llave $hex60.;
  llave=sortkey(country,','P');
run;

proc datasets;
  modify lista;
  index create llave;
quit;

options msglevel=i;
proc sql;
  select name, first, city from lista
  where llave = sortkey('España','P'); quit;

```

The MSGLEVEL=I option shows that the index is used for optimization.

The SAS System

name	first	city
Gómez	Juan	Barcelona
Martínez-Monés	Leonardo	La Coruña
López Fernandez	Ángela	León
Sánchez	Miguel	Algeciras
Llinares Sellés	Cristina	Lugo
Núñez Navarro	Ignacio	Madrid
Chuliá Mengual	David	Oviedo
Crespo	Gustavo	Lleida
Hernández Cerrillo	María del Pilar	Zaragoza

By default, linguistic sorting considers character case only after considering letters and accents to establish a basic alphabetic ordering. Let us check the following unordered list of commodities:

```

Housekeeping Supplies
Kitchen Supplies
Medical Supplies, Sports Medicine
Plastic Bags & Liners
Medical Supplies, Occupational Therapy
Medical Supplies, Rehabilitation
BOOKS
Plumbing Supplies
Bulbs & Lighting
Industrial Supplies
Burn Garments
Paper Disposables
WATER COOLERS
PHARMACEUTICALS
SUPPLIES, RADIOACTIVE
Medical Supplies, Orthotic Lab
Hardware
Beverages
Trophies & Plaques
Utensils, Kitchen
Periodicals, Publications
Water, Distilled Spring, & Bottled
Tableware
Pool Supplies
Business Cards
MEDICAL SUPPLIES, ORTHOPAEDIC DEVICES
MEDICAL SUPPLIES, PHYSICAL THERAPY

```

A "normal" (ASCII) sort results in:

```

1   BOOKS
2   Beverages
3   Bulbs & Lighting
4   Burn Garments
5   Business Cards
6   Hardware
7   Housekeeping Supplies
8   Industrial Supplies
9   Kitchen Supplies
10  MEDICAL SUPPLIES, ORTHOPAEDIC DEVICES

```

```

11 MEDICAL SUPPLIES, PHYSICAL THERAPY
12 Medical Supplies, Occupational Therapy
13 Medical Supplies, Orthotic Lab
14 Medical Supplies, Rehabilitation
15 Medical Supplies, Sports Medicine
16 PHARMACEUTICALS
17 Paper Disposables
18 Periodicals, Publications
19 Plastic Bags & Liners
20 Plumbing Supplies
21 Pool Supplies
22 SUPPLIES, RADIOACTIVE
23 Tableware
24 Trophies & Plaques
25 Utensils, Kitchen
26 WATER COOLERS
27 Water, Distilled Spring, & Bottled

```

In this case, the group of medical supplies is not sorted in the expected order for example:, Medical Supplies, Occupational Therapy should sort before MEDICAL SUPPLIES, ORTHOPAEDIC DEVICES. A PROC SORT run with SORTSEQ=LINGUISTIC will do this:

```

1 Beverages
2 BOOKS
3 Bulbs & Lighting
4 Burn Garments
5 Business Cards
6 Hardware
7 Housekeeping Supplies
8 Industrial Supplies
9 Kitchen Supplies
10 Medical Supplies, Occupational Therapy
11 MEDICAL SUPPLIES, ORTHOPAEDIC DEVICES
12 Medical Supplies, Orthotic Lab
13 MEDICAL SUPPLIES, PHYSICAL THERAPY
14 Medical Supplies, Rehabilitation
15 Medical Supplies, Sports Medicine
16 Paper Disposables
17 Periodicals, Publications
18 PHARMACEUTICALS
19 Plastic Bags & Liners
20 Plumbing Supplies
21 Pool Supplies
22 SUPPLIES, RADIOACTIVE
23 Tableware
24 Trophies & Plaques
25 Utensils, Kitchen
26 WATER COOLERS
27 Water, Distilled Spring, & Bottled

```

But does this also work when we add some data from other locales? Yes, it does, as we can see from the excerpt here:

```

9 Kitchen Supplies
10 Medical Supplies, Occupational Therapy
11 MEDICAL SUPPLIES, ORTHOPAEDIC DEVICES
12 Medical Supplies, Orthotic Lab

```

13	MEDICAL SUPPLIES, PHYSICAL THERAPY
14	Medical Supplies, Rehabilitation
15	Medical Supplies, Sports Medicine
16	Medizinisches Zubehör, Orthesen
17	MEDIZINISCHES ZUBEHÖR, ORTHOPÄDISCHE HILFSMITTEL
18	Medizinisches Zubehör, physikalische Therapie
19	Medizinisches Zubehör, Sportmedizin
20	Paper Disposables

Conclusion

As shown above, there are numerous ways to sort data according to different conventions. To be successful in the global economy, it is essential that you make accommodations for all possible situations. The linguistic collation capabilities ensure that you cannot only sort your data according to local language conventions and also appropriately order global character data and make use of extended character operations in a way that is linguistically appropriate.

No system is perfect and systems that are evolutionary are likely to have developed interesting behaviors and restrictions. SAS is no exception to this so some behaviors and restrictions that you can encounter or of which you should be aware are discussed in the appendix. Regardless of these caveats, we at SAS believe you find the addition of linguistic collation to be useful and that it allows you to get the results from SAS that you expect.

Acknowledgments

The authors want to express their thanks to the reviewers whose valuable feedback on the drafts has resulted in several enhancements. A special thank you goes to John Kohl whose suggestions have greatly improved the readability and consistency of this paper.

References

- Mebust, Scott and Michael Bridgers. 2006. "Creating Order out of Character Chaos: Collation Capabilities of the SAS System."
- SAS Institute Inc. 2008. SAS® 9.2 National Language Support (NLS): Reference Guide. Cary, NC: SAS Institute Inc.
- Sorting Your Linguistic Data Inside an Oracle Database. 2005. An Oracle Technical White Paper.
- Wissink, Cathy and Michael Kaplan. 2002. "Sorting it all out: An introduction to collation." Twenty-first International Unicode Conference Presentations
- Xiao Hui Zhu et al. 2002. "E-business globalization solution design guide: getting started" IBM Redbooks ISBN:0738426563

Appendix A: Things to be aware of

Appendix A1: General Notes

In 9.2, graphical user interface (GUI) dialog boxes have not been modified to support linguistic collation so, to sort linguistically, you need to use PROC SORT. Further SORTSEQ=LINGUISTIC is recognized only when set as a SORT procedure option; SORTSEQ=LINGUISTIC are not recognized when set as a system option.

The following locales are not supported by PROC SORT: Afrikaans_SouthAfrica (af_ZA), Cornish_UnitedKingdom (kw_GB), ManxGaelic_UnitedKingdom (gv_GB), and Bosnian_BosniaHerzegovina (bs_BA). In these cases, there is a fall back to the default *root* collation rules. That is, collation might not be tailored specifically to these locales but it is still using the UCA (that is for multilingual collation), so the results should be generally reasonable. This is acceptable because none of these locales uses extended characters. In the case of Bosnian_BosniaHerzegovina (bs_BA) we recommend using sh_BA instead.

Appendix A2: Platform Notes

LINGUISTIC sorting is available on all platforms except for 64-bit Windows on Itanium and VMS on Itanium.

On the mainframe (with z/OS), LINGUISTIC sorting requires more memory. Specifically, you might need to set your REGION to 50M or higher. This must be done in JCL, if running in batch, or in the VERIFY screen if running interactively. This is simply to get the ICU libraries to load properly and does not have anything to do with how much memory is used for sorting (although that increases with LINGUISTIC sorting as well).

Appendix A3: BY Processing and Formatted Variables

Sorting a data set linguistically by a character variable that has an associated format does not result in linguistic ordering of the formatted variable values. BY processing such a data set can produce unexpected results.

General BY processing groups observations based on the formatted value of a BY variable, when a format has been applied to the variable, but PROC SORT orders data only by the raw, internal value. If the formatted values of a variable do not collate in the same relative order as the raw values, then the result can be multiple BY groups for any single format value. Further, DATA step BY processing behaves differently, by default, than general BY processing. Normally, for the DATA step, BY groups are determined using internal variable values. This DATA step behavior can be changed to match that of general BY processing using the GROUPFORMAT option on the BY statement.

The following examples illustrate these issues and present a method should one want to linguistically order and BY process formatted values.

```

/*****
  BY processing data sorted BY a variable that has an associated format
  can produce unexpected results. However, linguistic collation and
  formats can be combined with a bit of additional work.
  *****/
options locale=en_US;

/*****
  Create a format which assigns each country in the LIST data set to a
  specific world region.

```

```

*****/
proc format;
  value $REGION (default=20)
    "China"      = "Asia"
    "Denmark"    = "Scandinavia"
    "France"     = "Europe"
    "Germany"    = "Europe"
    "Greece"     = "Europe"
    "Hungary"    = "Europe"
    "Israel"     = "Middle-East"
    "Japan"      = "Asia"
    "Poland"     = "Europe"
    "Russia"     = "Asia"
    "Spain"      = "Europe"
    "Sweden"     = "Scandinavia"
    "Turkey"    = "Asia"
  ;
run;

/*****
Create a subset of the LIST data set to demonstrate issues involving
the mixture of linguistic collation and formatted character variables.
*****/
data cities;
  set list;
  where city_e >= "Chateauroux" and city_e < "Leon";
  keep city_e country_e;
run;

TITLE "CITIES SUBSET";
proc print data=cities;
run;

/*****
Create some input data for the first three examples.  For this
demonstration, we create a new variable REGION_E, which contains the
same raw value as variable COUNTRY_E but is formatted with $REGION.
Note that the SORT procedure orders the data set by the unformatted
value of variable REGION_E!
*****/
data cities_1;
  set cities;
  region_e=country_e;
  format region_e $REGION.;
run;

proc sort data=cities_1 SORTSEQ=LINGUISTIC;
  by region_e;
run;

/*****
EXAMPLE 1

In general BY processing, BY groups will be defined by the formatted
value of a BY variable if that variable has an associated format.

```

Note that (1) the data is sorted linguistically by the name of the country (the unformatted value of REGION_E) and (2) BY processing recognizes that the data is sorted BY REGION_E. However, the presence of the \$REGION format causes the BY groups to be defined by the FORMATTED value!

This example, perhaps unexpectedly, results in multiple BY groups for a single formatted BY variable value because the formatted values are not properly grouped. Formatted BY variable values are not grouped because the data set is sorted by the unformatted BY variable value.

```
*****/
TITLE "EXAMPLE 1";
proc print data=cities_1;
by region_e;
run;
```

```
/******
EXAMPLE 2
```

In DATA step BY processing, the default is to define BY groups based on the unformatted value of a BY variable.

Note that (1) the data is sorted linguistically by the name of the country (the unformatted value of REGION_E) and (2) the DATA step recognizes that the data is sorted BY REGION_E. However, by default, the presence of the format is ignored and BY groups are defined by the UNFORMATTED value!

```
*****/
data example_2;
set cities_1;
by region_e;
if first.region_e
then NewGroup="YES";
else NewGroup="NO";
run;

TITLE "EXAMPLE 2";
proc print data=example_2;
run;
```

```
/******
EXAMPLE 3
```

The default behavior for DATA step BY processing can be overridden with the GROUPFORMAT option to achieve the same results as general BY processing.

Note that (1) the data is sorted linguistically by the name of the country (the unformatted value of REGION_E) and (2) the DATA step recognizes that the data is sorted BY REGION_E. However, the GROUPFORMAT option causes BY groups to be defined by the FORMATTED value.

This example produces the same results as EXAMPLE 1 and does so for the same reason. See EXAMPLE 1 for a description.

```

*****/
data example_3;
  set cities_1;
  by region_e GROUPFORMAT;
  if first.region_e
  then NewGroup="YES";
  else NewGroup="NO";
run;

TITLE "EXAMPLE 3";
proc print data=example_3;
run;

```

```

/*****
EXAMPLE 4

```

To combine linguistic collation and formatted character BY variables, create a separate variable to contain the formatted value and sort linguistically BY that variable. This technique works with both general BY processing and DATA step BY processing. It can also be done with a VIEW should one want to avoid the I/O costs associated with creating and reading a new data set.

Here, we put the formatted value of COUNTRY_E into the variable REGION_E and then sort linguistically BY the (unformatted) value of REGION_E.

```

*****/
data cities_2;
  set cities;
  length region_e $ 20;
  region_e=put(country_e,$REGION.);
run;

proc sort data=cities_2 SORTSEQ=LINGUISTIC;
  by region_e;
run;

TITLE "EXAMPLE 4A";
proc print data=cities_2;
  by region_e;
run;

data example_4;
  set cities_2;
  by region_e;
  if first.region_e
  then NewGroup="YES";
  else NewGroup="NO";
run;

TITLE "EXAMPLE 4B";
proc print data=example_4;
run;

```

CITIES SUBSET

Obs	city_e	country_e
1	Cracow	Poland
2	Czestochowa	Poland
3	Corinth	Greece
4	Ioannina	Greece
5	La Coruna	Spain
6	Copenhagen	Denmark
7	Göteborg	Sweden
8	Frankfurt	Germany
9	Duesseldorf	Germany
10	Goerlitz	Germany
11	Hamburg	Germany
12	Chateauroux	France
13	Evry	France
14	Frejus	France
15	Haifa	Israel
16	Istanbul	Turkey
17	Izmir	Turkey
18	Elazig	Turkey
19	Corlu	Turkey
20	Kawasaki	Japan
21	Guangzhou	China
22	Gyor	Hungary
23	Csabapuszta	Hungary
24	Csempeszkopacs	Hungary
25	Csempeszkopacs	Hungary

EXAMPLE 1

----- region_e=Asia -----

Obs	city_e	country_e
1	Guangzhou	China

----- region_e=Scandinavia -----

Obs	city_e	country_e
2	Copenhagen	Denmark

----- region_e=Europe -----

Obs	city_e	country_e
3	Chateauroux	France
4	Evry	France
5	Frejus	France
6	Frankfurt	Germany
7	Duesseldorf	Germany
8	Goerlitz	Germany
9	Hamburg	Germany
10	Corinth	Greece
11	Ioannina	Greece
12	Gyor	Hungary
13	Csabapuszta	Hungary
14	Csempeszkopacs	Hungary
15	Csempeszkopacs	Hungary

----- region_e=Middle-East -----

Obs	city_e	country_e
16	Haifa	Israel

----- region_e=Asia -----

Obs	city_e	country_e
17	Kawasaki	Japan

----- region_e=Europe -----

Obs	city_e	country_e
-----	--------	-----------

18	Cracow	Poland
19	Czestochowa	Poland
20	La Coruna	Spain

----- region_e=Scandinavia -----

Obs	city_e	country_e
21	Gothenburg	Sweden

EXAMPLE 1

----- region_e=Asia -----

Obs	city_e	country_e
22	Istanbul	Turkey
23	Izmir	Turkey
24	Elazig	Turkey
25	Corlu	Turkey

EXAMPLE 2

Obs	city_e	country_ e	region_e	New Group
1	Guangzhou	China	Asia	YES
2	Copenhagen	Denmark	Scandinavia	YES
3	Chateauroux	France	Europe	YES
4	Evry	France	Europe	NO
5	Frejus	France	Europe	NO
6	Frankfurt	Germany	Europe	YES
7	Duesseldorf	Germany	Europe	NO
8	Goerlitz	Germany	Europe	NO
9	Hamburg	Germany	Europe	NO
10	Corinth	Greece	Europe	YES
11	Ioannina	Greece	Europe	NO
12	Gyor	Hungary	Europe	YES
13	Csabapuszta	Hungary	Europe	NO
14	Csempeszkopacs	Hungary	Europe	NO
15	Csempeszkopacs	Hungary	Europe	NO
16	Haifa	Israel	Middle-East	YES
17	Kawasaki	Japan	Asia	YES
18	Cracow	Poland	Europe	YES
19	Czestochowa	Poland	Europe	NO
20	La Coruna	Spain	Europe	YES
21	Gothenburg	Sweden	Scandinavia	YES
22	Istanbul	Turkey	Asia	YES
23	Izmir	Turkey	Asia	NO
24	Elazig	Turkey	Asia	NO
25	Corlu	Turkey	Asia	NO

EXAMPLE 3

Obs	city_e	country_ e	region_e	New Group
1	Guangzhou	China	Asia	YES
2	Copenhagen	Denmark	Scandinavia	YES
3	Chateauroux	France	Europe	YES
4	Evry	France	Europe	NO
5	Frejus	France	Europe	NO
6	Frankfurt	Germany	Europe	NO
7	Duesseldorf	Germany	Europe	NO
8	Goerlitz	Germany	Europe	NO
9	Hamburg	Germany	Europe	NO
10	Corinth	Greece	Europe	NO
11	Ioannina	Greece	Europe	NO
12	Gyor	Hungary	Europe	NO
13	Csabapuszta	Hungary	Europe	NO
14	Csempeszkopacs	Hungary	Europe	NO
15	Csempeszkopacs	Hungary	Europe	NO
16	Haifa	Israel	Middle-East	YES
17	Kawasaki	Japan	Asia	YES
18	Cracow	Poland	Europe	YES
19	Czestochowa	Poland	Europe	NO
20	La Coruna	Spain	Europe	NO
21	Göteborg	Sweden	Scandinavia	YES
22	Istanbul	Turkey	Asia	YES
23	Izmir	Turkey	Asia	NO
24	Elazig	Turkey	Asia	NO
25	Corlu	Turkey	Asia	NO

EXAMPLE 4A

----- region_e=Asia -----

Obs	city_e	country_e
1	Istanbul	Turkey
2	Izmir	Turkey
3	Elazig	Turkey
4	Corlu	Turkey
5	Kawasaki	Japan
6	Guangzhou	China

----- region_e=Europe -----

Obs	city_e	country_e
7	Cracow	Poland
8	Czestochowa	Poland
9	Corinth	Greece
10	Ioannina	Greece
11	La Coruna	Spain
12	Frankfurt	Germany
13	Duesseldorf	Germany
14	Goerlitz	Germany
15	Hamburg	Germany
16	Chateauroux	France
17	Evry	France
18	Frejus	France
19	Gyor	Hungary
20	Csabapuszta	Hungary
21	Csempeszkopacs	Hungary
22	Csempeszkopacs	Hungary

----- region_e=Middle-East -----

Obs	city_e	country_e
23	Haifa	Israel

----- region_e=Scandinavia -----

Obs	city_e	country_e
24	Copenhagen	Denmark
25	Gothenburg	Sweden

EXAMPLE 4B

Obs	city_e	country_e	region_e	New Group
1	Istanbul	Turkey	Asia	YES
2	Izmir	Turkey	Asia	NO
3	Elazig	Turkey	Asia	NO
4	Corlu	Turkey	Asia	NO
5	Kawasaki	Japan	Asia	NO
6	Guangzhou	China	Asia	NO
7	Cracow	Poland	Europe	YES
8	Czestochowa	Poland	Europe	NO
9	Corinth	Greece	Europe	NO
10	Ioannina	Greece	Europe	NO
11	La Coruna	Spain	Europe	NO
12	Frankfurt	Germany	Europe	NO
13	Duesseldorf	Germany	Europe	NO
14	Goerlitz	Germany	Europe	NO
15	Hamburg	Germany	Europe	NO
16	Chateauroux	France	Europe	NO
17	Evry	France	Europe	NO
18	Frejus	France	Europe	NO
19	Gyor	Hungary	Europe	NO
20	Csabapuszta	Hungary	Europe	NO
21	Csempeszkopacs	Hungary	Europe	NO
22	Csempeszkopacs	Hungary	Europe	NO
23	Haifa	Israel	Middle-East	YES
24	Copenhagen	Denmark	Scandinavia	YES
25	Gothenburg	Sweden	Scandinavia	NO

Appendix A4: CLASS Processing versus BY Processing

CLASS processing does not order or group data linguistically nor is it sensitive to an existing linguistic collation sequence of a data set. CLASS processing can produce results that are different from those obtained using BY processing because BY processing is now sensitive to collating sequences.

For example, with the SUMMARY procedure, class processing is normally performed by grouping formatted values of a class variable (or raw values, if the GROUPINTERNAL option is specified). If a data set is sorted, the ORDER=DATA option can be used to preserve the order in which class levels are output for the NWAY type. However, if the data is sorted linguistically, classification boundaries are still determined by a binary difference in the formatted (or unformatted) class variable values. For example, if a case-insensitive linguistic collating sequence was used (that is STRENGTH=2), changes in character case still denotes a new level in the NWAY type.

The following example shows the difference in output between BY processing and CLASS processing using PROC MEANS when, in the first case, the input data set is linguistically sorted in a case-insensitive manner and, in the second case, when the ORDER=DATA option is used with the same input data set. We can address this particular problem, as shown in the third PROC MEANS invocation, by using the \$UPCASE format but this type of solution is not universally applicable.

```
data survey;
```

```

length gender $ 10;
input gender age;
cards;
Male 27
Female 31
MALE 25
FEMALE 23
;
run;

proc sort data=survey SORTSEQ=LINGUISTIC(STRENGTH=2);
  by gender;
run;

TITLE "BY GENDER";
proc means data=survey n mean;
  var age;
  BY GENDER;
run;

TITLE "CLASS GENDER";
proc means data=survey n mean ORDER=DATA;
  var age;
  CLASS GENDER;
run;

TITLE "CLASS GENDER with $UPCASE format";
proc means data=survey n mean ORDER=DATA;
  var age;
  CLASS GENDER;
  format gender $upcase.;
run;

```

BY GENDER

1

----- gender=Female -----

The MEANS Procedure

Analysis Variable : age

N	Mean
2	27.0000000

----- gender=Male -----

Analysis Variable : age

N	Mean
2	26.0000000

CLASS GENDER

2

The MEANS Procedure

Analysis Variable : age

gender	N		Mean
	Obs	N	
Female	1	1	31.0000000
FEMALE	1	1	23.0000000
Male	1	1	27.0000000
MALE	1	1	25.0000000

CLASS GENDER with \$UPCASE format

3

The MEANS Procedure

Analysis Variable : age

gender	N		Mean
	Obs	N	
FEMALE	2	2	27.0000000
MALE	2	2	26.0000000

Appendix B: Where to Turn for More Information

More information on collating sequence options, collation rules, the sortKey function and or National Language Support in general can be obtained from the online documentation and or the National Language Support (NLS) Reference Guide.

ⁱ E.g. ISO/IEC 14651:2001 Information technology -- International string ordering and comparison -- Method for comparing character strings and description of the common template tailorable ordering.

ⁱⁱ Accents or other marks modifying a character.

-
- iii This is because accents at the end of a French word are considered more important for the understanding than other accents; the order is no accent, acute accent, grave accent, circumflex accent, and diaeresis.
- iv All names in this paper are purely fictitious. Any resemblance to actual persons is completely accidental.
- v In 1994, the Royal Spanish Academy agreed to alphabetize ch and ll as ordinary pairs of letters in the dictionary, and not as separate letters as in the past. This means there are now two ways to sort Spanish data: according to the "traditional" sort order and the "modern" one.
- vi Swedish alphabet. (2007, September 4). In Wikipedia, The Free Encyclopedia. Retrieved 08:55, October 2, 2007, from http://en.wikipedia.org/w/index.php?title=Swedish_alphabet&oldid=155619192.
- vii PHONEBOOK works only in conjunction with a German locale.
- viii Select TRADITIONAL only with a Spanish locale.
- ix Collation. (2007, September 14). In Wikipedia, The Free Encyclopedia. Retrieved 10:39, October 2, 2007, from <http://en.wikipedia.org/w/index.php?title=Collation&oldid=157934736>
- x Select these keywords only with the Chinese language.
- xi Correct alphabetization in Danish and Norwegian places Aa along with Å as the last letter in the alphabet, the sequence being Æ, Ø, Å/Aa. (Å. (2007, August 24). In Wikipedia, The Free Encyclopedia. Retrieved 08:31, October 4, 2007, from <http://en.wikipedia.org/w/index.php?title=%C3%85&oldid=153346567>