

# FieldWorks Sorting With ICU Rules

Ken Zook

September 4, 2019

## Contents

FieldWorks Sorting With ICU Rules .....	1
1 Introduction.....	1
1.1 Online resources .....	1
2 ICU sorting levels .....	2
2.1 Primary level .....	2
2.2 Secondary level .....	2
2.3 Tertiary level .....	3
2.4 Quaternary level .....	4
3 Additional features.....	4
3.1 Sort before primary characters .....	4
3.2 Sorting punctuation .....	4
3.3 Ignoring characters .....	5
3.4 Normalization issue.....	6
3.5 Sorting examples .....	7
3.6 Advanced help.....	8
4 Converting Toolbox sort rules to ICU rules .....	8

## 1 Introduction

There are many resources available online for FLEx sorting since FLEx integrates code from International Components for Unicode (ICU). ICU is an open-source implementation for processing Unicode using the Unicode Collation Algorithm (UCA) which is part of the Unicode standard that describes methods for collating Unicode characters.

The UCA attempts to provide ways for collating any language using Unicode. While their documentation tries to cover all aspects, it doesn't give adequate examples for many of the more advanced features and can be rather overwhelming. This document attempts to pull together some of the most common features that you'll find useful for sorting in FLEx.

### 1.1 Online resources

- International Components for Unicode (ICU) <http://site.icu-project.org/>
- Unicode Collation Algorithm (UCA) <http://www.unicode.org/reports/tr10/>.
- Default collation chart for UCA/ICU <http://unicode.org/charts/collation/>
- ICU tailoring rules <http://userguide.icu-project.org/collation/customization>.
- ICU provides a useful Web site for testing collation rules at <http://demo.icu-project.org/icu-bin/collation.html>.

**Note:** This demo uses the currently released version of ICU which may not be identical to what is currently available in FieldWorks, but they are usually very close. The 'strength' setting should be set to 'primary' rather than 'default' to emulate

FieldWorks sorting. Some elements such as caseFirst do not seem to work in the demo.

- LDML collation [http://www.unicode.org/reports/tr35/tr35-collation.html#Common\\_Settings](http://www.unicode.org/reports/tr35/tr35-collation.html#Common_Settings).
- Martin Hosken wrote an excellent tutorial on using ICU collation ([http://scriptsource.org/cms/scripts/page.php?item\\_id=entry\\_detail&uid=pnrnlhkrq9](http://scriptsource.org/cms/scripts/page.php?item_id=entry_detail&uid=pnrnlhkrq9)).
- This document <https://software.sil.org/fieldworks/wp-content/uploads/sites/38/2018/06/FieldWorks-Sorting-With-ICU-Rules.pdf>
- Unicode Character Browser <https://www.unicode.org/unibook/>

## 2 ICU sorting levels

ICU has default collation rules for all Unicode characters. In many cases the default collation is all that is needed to get correct sorting. So it's not necessary to list rules for anything that already sorts correctly. Rules are only needed to override the default collation.

Most ICU rules start with an ampersand followed by an anchor point. The rest of the rule is the reset value and specifies how characters are collated after the anchor point. There is no need to start a new line for each rule, but it makes it more readable.

### 2.1 Primary level

Here is a simple example of a rule using a primary level (single left wedge):

```
&c<k
```

This rule states that “k” comes immediately after “c” (e.g., cat, kite, dog).

**Note:** This does *not* affect uppercase “K”. See tertiary examples below. FLEx will only produce alpha headers for primary elements.

Diagraphs can be handled as well:

```
&n<ng
```

This rule states that the “ng” diagraph occurs after “n” (e.g., nab, nung, ngab).

**Note:** Again, this does not affect uppercase.

### 2.2 Secondary level

The Unicode default collation ignores diacritics unless the rest of the word is identical. In that case, words are sorted based on the diacritic (e.g., bad, bád, bàd, båd, båd, båd, båd).

Two left wedges are used for secondary level collation which only comes into effect if the primary levels are identical. The secondary level (two left wedges) is typically used for diacritics. If you just need to change the order of “bád” and “båd”, you can add a secondary rule for these two diacritics.

```
&à<<á
```

The resulting order is bad, bàd, bád, båd, båd, båd, båd

You can chain multiple characters or diagraphs together in a single rule to define a new order (e.g., bad, båd, bãd, båd, bàd, bäd, bád). White space around the wedge markers is optional.

```
&a << å << ã << â << à << ä << á
```

In addition to inserting the actual character in a rule, you can also give the Unicode code point. The following commands are identical:

```
&a<<à
&\u0061<<\u00e0
```

In the above examples, we specified the diacritics over “a”. As a result, these diacritic rules only apply when the diacritic is over an “a”. We can generalize diacritic behavior by specifying only the diacritic in the rule.

```
&\u0300<<\u0301
```

This rule will cause any character with an acute accent (0301) to come before the same character with a grave accent (0300) (e.g., bàd, bád, béd, béd). At least this works in FLEx because it normalizes all data to NFD internally so that normal diacritics are separate code point.

## 2.3 Tertiary level

Three left wedges are used for tertiary level collation which is typically used for case. Tertiary level sorting only affects strings that are identical through the secondary level. By default, ICU would sort words in this order (nab, Nab, NAB, ngab, Ngab, NGAB, nung). If we want “ng” to come after “n” at a primary level, we could use this rule:

```
&n<ng
```

The resulting order is now nab, Nab, NAB, Ngab, NGAB, nung, ngab. Note “NGAB” and “Ngab” are not affected by this rule. To include these words as well, we need to use the tertiary level (three left wedges).

```
&n<ng<<<Ng<<<NG
&c<k<<<K
```

The first rule moves “ng” (regardless of case) to follow “n” (e.g., nab, Nab, NAB, nung, ngab, Ngab, NGAB). The second moves “k” (regardless of case) to follow “c” (e.g., cat, kite, Kite, dog).

Note this orders uppercase letters after lowercase letters. If you prefer to sort with uppercase letters first, you could specify a series of rules such as

```
&A<<<a
&B<<<b
```

However, this does not work as well with alpha headers in FLEx, so it’s better to add the caseFirst collation element as the first collation rule.

```
[caseFirst upper]
&n<ng<<<Ng<<<NG
```

With the caseFirst element we can use the default collation as well as our previous rule definitions the normal way, but ICU will automatically switch everything to lowercase following uppercase (e.g., NAB, Nab, nab, nung, NGAB, Ngab, ngab).

To sort “á” at a primary level after all other “a”s, use this rule:

```
&a<á<<<Á
```

This sort order gives ade, ãde, apple, Azure, áde, Áde.

## 2.4 Quaternary level

There is a 4<sup>th</sup> level for rules using “<<<<” that can be used when the first 3 levels are all identical. This is rarely needed.

## 3 Additional features

### 3.1 Sort before primary characters

If you need to sort a character before a normal predefined character instead of after, (e.g., āb, Āb, aa, Aa) the preferred way is to use the before element if the code points are generally considered equal.

```
&[before 1]a<ā<<<Ā
```

In this case, the right-hand side goes before the “a” anchor instead of after. The digit 1 indicates this is a primary level which corresponds to the single left wedge after “a”.

Suppose we want to sort diacritics in this order: á a à á ą à. We can do this using the secondary “before” element to specify the one that goes before the regular “a”, and then the standard secondary elements for ones following the “a”.

```
&[before 2]a << á
&a << à << á << ą << à
```

Another way that will work but isn’t as clear would be to put the target character after the character before the anchor. For example, if “ng” comes before “n” rather than after “n” you could use this rule

```
&m<ng
```

Since “m” comes before “n” this rule says that “ng” follows “m”, which means it comes before “n”. But &[before 1]n<ng makes it obvious that ng is related to n, but precedes it.

Likewise the following rule could be used to specify āb, Āb, aa, Aa ordering. The “first regular” element indicates the beginning of alphabetical characters, so is actually before “a”. So it places āb before “a”.

```
&[first regular]<ā<<<Ā
```

### 3.2 Sorting punctuation

In an ICU rule, any non-alphanumeric ASCII character is reserved for syntax characters. If you need to control collation of any of these characters, you must quote them with a backslash (\) or enclose them in apostrophes. A single apostrophe can also be represented

as two apostrophes. Here are some examples of alphanumeric and punctuation characters with or without the \u syntax.

a	letter a
\u0061	letter a
3	digit 3
ng	digraph ng
'ng'	digraph ng (quotes are optional for alphanumeric characters)
\u006e\u0067	digraph ng
\-	hyphen
'-'	hyphen
' '	space
\	space (there is a space following the \)
'\u0020'	space
\\u0020	space
'\''	apostrophe
''	apostrophe (two single quotes)
\u0027\u0027	apostrophe

To control the collation of an apostrophe you would thus add two apostrophes (not a double quote). To sort t' after t, you would use the rule

```
&t<t''
```

The following rules would be one way to handle IPA sorting rather than listing all characters in one rule. This only lists characters that do not follow default ordering.

```
&d<d̄
&e<ɛ<f<ϕ
&i<i
&k<k''
&n<ŋ
&p<p''<r<ɾ
&s<ʃ<ʂ
&t<t''<t̄<t̄''<t̄̄''<t̄̄̄''<t̄̄̄̄''
&z<ʒ<z<ʔ
```

### 3.3 Ignoring characters

Suppose you want to ignore an apostrophe after “m” and “n”, but you want “ng” to sort after “n”, and “ng” to sort after “ng”. The following rules allow for this.

The “=” syntax states that the right side is identical to the left side.

```
&m=m''
&M=M''
&n=n''
&N=N''
&n<ng<<<Ng<<<NG<ng''<<<Ng''<<<NG''
```

Suppose you want to totally ignore 02BC;MODIFIER LETTER APOSTROPHE in sorting. The way to do this is to use the following command.

```
&[last tertiary ignorable] = \u02BC
```

This would result in the following order ba, ba'd, bad, b'ad, bäd, bade, ba't, bat, b'at, bät, bate. Since bad, ba'd, and b'ad all have identical sort keys, their order is random.

If you need to ignore more than one character, use “=” to separate the list of characters. The following rule would ignore an apostrophe, a question mark, a hyphen, a space, and the ng digraph

```
&[last tertiary ignorable] = " = '?' = '-' = ' ' = ng
```

or

```
&[last tertiary ignorable] = \ = \? = \- = \ = ng
```

This could also be represented as

```
&[last tertiary ignorable] = \u0027\u0027 = '\u003f' = '\u002d' = '\u0020' =
\u0006e\u0067
```

If you simply want to ignore all punctuation as well as white space, you can use the following rule

```
[alternate shifted]
```

### 3.4 Normalization issue

ICU sorting always uses NFD normalization which means that eë is seen as ee followed by \u0308. When ee is sorted after e, this means that instead of sorting based on e followed by ë it is sorted on ee followed by \u0308, which puts words in the wrong order. To get around this problem we use this Expansion rule.

```
&e = eë/ë
```

The ICU Expansion rule uses somewhat confusing syntax. What this is actually saying is that for eë (following the =), use the collation for e (following &) followed by ë (following the /). This would be a complete collation including case for sorting e < ee < ë < eë.

```
&e < ee <<< Ee <<< EE < ë <<< Eë < eë <<< Eë <<< Eë
```

```
&e = eë/ë
```

```
&E = Eë/ë
```

```
&E = Eë/ë
```

```
&e = eëë/ëë
```

```
&E = Eëë/ëë
```

```
&E = Eëë/ëë
```

In the actual project where this sorting was necessary, similar rules were used for öö, and 02bc MODIFIER LETTER APOSTROPHE sorted after z. For some reason, FW9.0.7 and earlier was adding a bogus O alpha header before a word starting with ö'. It turns out that the following Expansion rule solved this problem.

```
&ö=ö'/'
```

This seems like a real hack. It is saying that ö' should sort as ö followed by ' which is what it would do without the rule. The sort keys used by ICU are identical with or without this rule, so it has no effect on sorting, but somehow it allowed the Flex alpha header code to work. Since it doesn't change the sort in any way, it should not hurt to include it until Flex does a better job of generating alpha headers.

### 3.5 Sorting examples

To sort phonetic characters in “p p<sup>h</sup> b φ β m ɱ w” order, you could use either of the following identical rules:

```
&p < ph < b < φ < β < m < ɱ < w
&p < \u0070\u02b0 < b < \u0278 < \u03b2 < m < \u028d < w
```

To sort uppercase and lowercase in “c C b B a A” order, use these two rules :

```
&c < b < < < B
&b < a < < < A
```

This can also be combined into a single rule:

```
&c < b < < < B < a < < < A
```

Suppose you need to have this order:

Á á Â â Ã ã A a 'B b 'B 'b B b Ch ch 'D d 'D d Dh dh É é Ê ê Ë ë E e F f G' g' G g H h  
 Í í Î î Ï ï I i C' c' J j K k L l M m N n Ny ny Ng' ng' Ó ó Ô ô Õ õ O o R r S s Sh sh T t Ú ú  
 Û û Ü ü U u V v W w Y y

Here is a set of rules that will produce this:

```
[caseFirst upper]
&[before 2]a < á < < < Á < â < < < Â < ã < < < Ã
&[before 1]b < b < < < B < 'b < < < 'B
&c < ch < < < Ch
&[before 1]d < d < < < D < đ < < < Đ
&d < dh < < < Dh
&[before 2]e < é < < < É < ê < < < Ê < ë < < < Ë
&[before 1]g < g' < < < G'
&[before 2]i < í < < < Í < î < < < Î < ï < < < Ï
&[before 1]j < c' < < < C'
&n < ny < < < Ny < ng' < < < Ng'
&[before 2]o < ó < < < Ó < ô < < < Ô < õ < < < Õ
&s < sh < < < Sh
&[before 2]u < ú < < < Ú < û < < < Û < ü < < < Ü
```

The ICU default collation of Thaana characters, used by Divehi, treats diacritics as primary instead of secondary characters, making it impossible in Flex to filter on base characters without diacritics. This can be solved, however, by using this collation rule (unfortunately it looks mixed up because of RTL characters, but if you copy it and paste it into Flex, it will work):





Q q  
 R r  
 S s  
 T t  
 U u Ú ú Ū ū Ŭ ŭ Ů ů  
 Ū ū Ŭ ŭ Ů ů  
 V v  
 W w  
 X x  
 Y y  
 Z z  
 '  
 0  
 1  
 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 \SecFollowing ` ^ ˇ ~ ˇ  
 \ignore \_ - =  
 \SecAfterBase  
 \-srt  
  
 \SecPreceding  
 \SecAfterBase

The \primary section lists characters with primary distinctions starting on a new line and secondary distinctions on the same line separated by spaces. Every character and digraph needs to be included.

Flex uses ICU for specifying collation, and ICU defines a default Unicode order for every character. Often this order is correct. You could specify rules for every character if you want, but it's simpler to only list rules for characters that do not sort properly by default. Suppose Toolbox has this order:

```

\primary A a Á á
B b
C c
Ch ch

```

a, b, c is already the default order, so we don't need to specify rules for that. We don't need to list composed character/diacritic letters in FLEx because internally it uses decomposed form (NFD) where the diacritic is a separate code point after the vowel. Since diacritics sort after vowels using secondary ordering by default, we don't need to include them in collation rules unless you need an order different from the ICU default.

So the only exception to defaults in this case is the ch digraph. So all we need is a rule to specify that the “ch” digraph comes after “c” with primary ordering.

```
&c < ch <<< Ch
```

One difference in the typical Toolbox sorting and ICU sorting is that Toolbox typically has uppercase letters coming before lowercase letters while ICU defaults to uppercase letters following lowercase letters using tertiary ordering. There is a single rule that tells ICU to put uppercase first regardless of how it is specified in the tertiary rules. By using this rule, you don't have to specify each combination of UC/LC as is needed in Toolbox. In Flex, the alpha headers work more reliably if we use default LC first rules and add this command at the beginning to tell ICU to use the opposite case order. So now our final collation for the above simple sample is

```
[caseFirst upper]
&c < ch <<< Ch
```

If Toolbox has a secondary sort for a diagraph as with this line

```
G g gu
```

then this can be replaced with this ICU rule which specifies gu is a secondary ordering after g instead of primary.

```
&g << gu
```

If Toolbox has a combination of secondary and primary ordering following a standard alphabetic character like this

```
G g gu gü
'G 'g
```

then you could use this ICU rule

```
& g << gu << gü < 'g <<< 'G
```

This uses “<” for every primary relation (new line in a Toolbox specification) and “<<” for secondary (following space in a Toolbox specification), and “<<<” for tertiary to handle uppercase.

Toolbox specifies diacritic ordering like this.

```
\SecFollowing ` ^ ˇ ~ ˇ
```

ICU defaults to secondary ordering of all diacritics using a fixed order. However, these characters are not true diacritics, but modifier letter diacritics that do not combine with a vowel. Toolbox specifies these as secondary ordering, so we can replace this with the equivalent ICU rule for secondary ordering.

```
& [first secondary ignorable] << ` << ^ << ˇ << ~ << ˇ
```

Toolbox specifies characters to be ignored in this way.

```
\ignore _ - =
```

This can be turned into an ICU rule that ignores these characters.

```
& [last tertiary ignorable] = '_' = '-' = '='
```

ICU defaults to sorting digits before alphabetical characters. If you want to place digits at the end of the alphabet as in this case, you can use this ICU rule:

```
&[last regular]< 0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9
```

Putting all of this together, here is the complete ICU collation.

```
[caseFirst upper]
& a << á <<< Á << a <<< A << á <<< Á
& c < ch <<< Ch
& e << é <<< É << e <<< E << é <<< É
& g << gu << gü < 'g <<< 'G
& i << í <<< Í << i <<< I << í <<< Í < i <<< I << í <<< Í << í <<< '
& l < 'l <<< 'L
& m < 'm <<< 'M
& n < 'ñ <<< 'Ñ
& o << ó <<< Ó << o <<< O << ó <<< Ó < ø <<< Ø << ø <<< Ø << ø <<< Ø << ó <<<
Ó_
& u << ú <<< Ú << u <<< U << ú <<< Ú << ü < u <<< U << ú <<< Ú << ü <<< Ü << ü <<< Ü << ü
<<<_
&[last regular] < ' < 0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9
& [first secondary ignorable] << ` <<< ^ << ' <<< ~ << ~
& [last tertiary ignorable] = '_' = '-' = '='
```

In general, you can take a Toolbox lng file and convert it to ICU rules quite easily. Anything that sorts by default in ICU can be dropped. Characters that start a new line are primary, so use the primary “<” ICU relation. Characters that appear between spaces on a line are secondary, so use the secondary “<<” ICU relation. If they specify uppercase before lowercase, just add the [caseFirst upper] rule at the top and do everything else as though it were the standard ICU putting uppercase after lowercase using tertiary “<<<” in rules.