FieldWorks Sorting With ICU Rules

Ken Zook

June 26, 2025

Contents

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 https://icu4c-demos.unicode.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.
- Martin Hosken wrote an excellent tutorial on using ICU collation (http://scriptsource.org/cms/scripts/page.php?item_id=entry_detail&uid=pnrnlhkrq9).
- This document : https://downloads.languagetechnology.org/fieldworks/Documentation/FieldWorks%20Sorting%20With%20ICU%20Rules.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 letter headings for primary elements.

Digraphs can be handled as well:

```
&n<ng
```

This rule states that the “ng” digraph 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, bâd, båd, bäd, bãd).

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 “bàd”, you can add a secondary rule for these two diacritics.

```
&à<<á
```

The resulting order is bad, bàd, bád, bâd, båd, bäd, bãd

You can chain multiple characters or digraphs together in a single rule to define a new order (e.g., bad, båd, bãd, bâd, 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 after the same character with a grave accent (0300) (e.g., bàd, bád, bèd, béd). At least this works in FLEx because it normalizes all data to NFD internally so that normal diacritics are separate code point.

Note that \u is expecting 4 hex digits following the \u. For Unicode upper planes, you should use \U with 8 hex digits following the \U. (e.g., \U00010d00 for 𐴀 HANIFI ROHINGYA LETTER A).

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 letter headings 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 4th 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 á̧ ą̧ à̧. 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 << á̧ << a̧ << à̧
```

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

Note that FLEx letter heading code may not work correctly using the backslash before a single character, so it is safer to surround the character with apostrophes.

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<d͡ʒ
&e<ɛ<f<ɸ
&i<ɨ
&k<k''
&n<ŋ
&p<p''<r<ɾ
&s<ʃ<ʂ
&t<t''<t͡s<t͡s''<t͡ʃ<t͡ʃ''<ʈ͡ʂ<ʈ͡ʂ''
&z<ʒ<ʐ<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. You can also include more than one character to ignore a string (e.g., ='(-)') will ignore (-). 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' =
\u006e\u0067
```

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

```
[alternate shifted]
```

A character can be ignored, but still defined in a rule. An example is a project where apostrophes appeared before vowels, where they should be ignored in collation, but also occur as part of consonant digraphs where they were important (e.g., bode, b'ad, 'om). The following rules will work to produce this.

```
&b < b' <<< B'
[last tertiary ignorable]=''
```

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 < ëë.

```
&e < ee <<< Ee <<< EE < ë <<< Ë < ëë <<< Ëë <<< ËË
&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 oö, and 02bc MODIFIER LETTER APOSTROPHE sorted after z. For some reason, FW9.0.7 and earlier was adding a bogus O letter heading 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 letter heading 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 letter headings.

3.5 Sorting examples

To sort phonetic characters in “p p^h b ɸ β m ʍ w” order, you could use either of the following identical rules:

&p < p^h < b < ɸ < β < m < ʍ < 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 Ā ā A a Ɓ ɓ ’B ’b B b Ch ch Ɗ ɗ Đ đ D d Dh dh É é Ê ê Ē ē E e F f Ɠ ɠ G g H h Í í Î î Ī ī I i Ƈ ƈ J j K k L l M m N n Ny ny Ng' ng' Ó ó Ô ô Ō ō O o R r S s Sh sh T t Ú ú Û û Ū ū U u V v W w Y y

Here is a set of rules that will produce this:

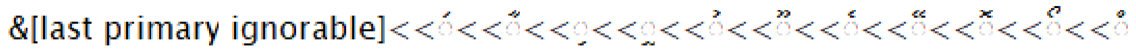
```
[caseFirst upper]
&[before 2]a<<á<<<Á<<â<<<Â<<ā<<<Ā
&[before 1]b<ɓ<<<Ɓ<’b<<<’B
&c<ch<<<Ch
&[before 1]d<ɗ<<<Ɗ<đ<<<Đ
&d<dh<<<Dh
&[before 2]e<<é<<<É<<ê<<<Ê<<ē<<<Ē
```

&[before 1]g<ɠ<<<Ɠ
&[before 2]i<<í<<<Í<<î<<<Î<<ī<<<Ī
&[before 1]j<ƈ<<<Ƈ
&n<ny<<<Ny<ng"<<<Ng"
&[before 2]o<<ó<<<Ó<<ô<<<Ô<<ō<<<Ō
&s<sh<<<Sh
&[before 2]u<<ú<<<Ú<<û<<<Û<<ū<<<Ū

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):

&[last primary ignorable]<<ަ<<ާ<<ި<<ީ<<ު<<ޫ<<ެ<<ޭ<<ޮ<<ޯ<<ް

The ZEdit program that is installed in the FieldWorks directory does not use a bi-directional algorithm for displaying text. It just shows characters in logical order, so here's how it would look in this program.


&[last primary ignorable]<<ަ<<ާ<<ި<<ީ<<ު<<ޫ<<ެ<<ޭ<<ޮ<<ޯ<<ް

3.6 FLEx letter headings

In dictionary views, FLEx will automatically insert letter headings as long as the dictionary is sorted on headwords, lexeme form, or citation form fields. If it is sorted on any other field, letter headings will not be inserted.

Letter headings are based on the primary collation level. Words that differ on secondary or tertiary levels will all be grouped together under a primary letter heading.

If you find that letter headings are not coming out right, it is normally an indication that you need to improve your collation.

For example, suppose 'lh' is a digraph that comes after 'l' and we use this rule:

&l < lh

This is what happens with these words: Lâa, Lhàchèdâ, lìá', lhàch:

L l
Lâa
Lh lh
Lhàchèdâ
L l
lìá'
Lh lh
lhàch

Notice we are getting extra letter headings. The first thing to notice is that the words are sorting the upper case 'L' before the lowercase 'l' at a primary level. The FLEx code that puts in letter headers sees a switch between 'L' and 'Lh', so it adds a letter heading at that point, then does the same when it goes from 'l' to 'lh'. But we don't want to consider upper and lower case separately at the primary level. The problem can be solved when we add lowercase and uppercase code points to the collation rule including tertiary information for the uppercase code points.

&l <<< L < lh <<< Lh <<< LH

However, if you are just looking at the Dictionary view, even after making this change to the collation, including Refresh it still gives the wrong results. The problem is you need to actually resort the records. To do that, you need to go to the Lexicon Edit view and resort the Headword column. Then when you go back to the Dictionary view, it will give the correct results.

L l
Lâa
lìá'
Lh lh
lhàch
Lhàchèdâ

Any time your orthography uses more than one code point for a letter, you need to specify the full primary and tertiary elements in the collation to get sorting and letter headings to come out correctly. For example, if your orthography has ‘a’, ‘à’, and ‘àꞌ’ (where the last character is a78c Saltillo) that each need to generate letter headings, if you use this rule, it will work if there are no uppercase letters, but it will not work correctly if you also have capitalized or uppercase words in your headwords.

&a < à < àꞌ

To get correct sorting and letter headings you would need to use this rule which handles uppercase including uppercase Saltillo.

&a <<< A < à <<< À < àꞌ <<< Àꞌ <<< ÀꞋ

3.7 Advanced help

Refer to the references at the beginning of this section for more complex sorting issues. ICU provides for complex collation far beyond the basics I’ve covered here. For example, Martin Hoskin has some Python code for complex reordering in LDML files for scripts such as TaiLue. In this language the order is determined first by the initial consonant, then by the final consonant, then by the vowel, and finally by the tone mark. So the collation needs to reorder the final syllable consonant to immediately following the initial syllable consonant. This is possible, but is far from trivial.

4 FieldWorks ICU sorting

FLEx stores collation information in Unicode Locale Data Markup Language (LDML) files in the project WritingSystemStore directory. The Sort tab in the Writing System Properties dialog uses the collation information in the ldml file for that writing system. Each time you make a change, it writes the change to the project WritingSystemStore directory and also writes the same information into the global writing system repository at C:\ProgramData\SIL\WritingSystemRepository\3.

The official Unicode site for storing LDML files is the Unicode Common Locale Data Repository (CLDR, http://cldr.unicode.org/). The SIL Writing Systems Technology Department maintains an SIL Locale Data Repository (SLDR, https://github.com/silnrsi/sldr), which is very similar to the CLDR, but can be customized for SIL use when needed. A special API is available for programs to access the SLDR. You can download information using this API in a browser. For example, https://ldml.api.sil.org/hi will download the hi.xml file which can be renamed to ldml. ICU uses the CLDR. If interested, you can access the raw ICU data from http://cldr.unicode.org/index/downloads. Follow the Data link, download the cldr-common zip file and look in common\collation for the desired xml file. The master list of rules for ICU code points is in the common\uca\UCA_Rules.txt file in the cldr-common zip file. The collations element in the ldml file may contain multiple collation elements. FieldWorks only uses the standard collation element which can be found by searching for <collation type="standard".

In FieldWorks Version 9, the first time you add a new writing system (e.g., Hindi), on your machine, and you are connected to the Internet, it will download the hi.xml file from the SLDR and store it in C:\ProgramData\SIL\SldrCache as an ldml file. It then stores this information in the project’s WritingSystemStore directory, and will also save a copy in the global repository for use when other projects or programs need the information. The writing system property dialog displays the collation from the ldml file in the Sort tab. When you make changes in this tab, it copies your changes into the current project hi.ldml and also updates the copy in the global repository, but will not affect the original in SldrCache.

In FLEx, you can choose an option in the Sort tab to use a collation from a standard language, which uses the default definitions built into ICU. If you need to make any adjustment to the standard collation, you need to get the ICU rules for the language and paste them into the Custom ICU rules pane in the Sort tab. Then you can customize these rules in any way you want for your purposes. If FLEx downloaded the file via the SLDR, the rules for that language will already be in the Custom ICU rules. If you have an existing ldml file that does not have the correct rules, you can use one of the resources above to get the current file and copy the rules to your Custom ICU rules dialog.

5 Converting Toolbox sort rules to ICU rules

Toolbox stores collations in a .lng file in the main Toolbox directory. Here is a fairly complex sort specification, where cco is the code for the language:

```
\+srt cco order
\primary A a Á á A̱ a̱ Á̱ á̱
B b
C c
Ch ch
D d
E e É é E̱ e̱ É̱ é̱
F f
G g gu gü
'G 'g
```

H h
I i Í í I̱ i̱ Í̱ í̱
Ɨ ɨ Ɨ́ ɨ́ Ɨ̱ ɨ̱ Ɨ̱́ ɨ̱́
J j
K k
L l
'L 'l
M m
'M 'm
N n
'N 'n
Ñ ñ
'Ñ 'ñ
O o Ó ó O̱ o̱ Ó̱ ó̱
Ø ø Ǿ ǿ Ø̱ ø̱ Ǿ_ ǿ_
P p
Q q
R r
S s
T t
U u Ú ú U̱ u̱ Ú̱ ú̱ ü
Ʉ ʉ Ʉ́ ʉ́ Ʉ̱ ʉ̱ Ʉ̱́ ʉ̱́
V v
W w
X x
Y y
Z z
'
0
1
2
3
4
5
6
7
8
9
\SecFollowing ˋ ¯ ˆ ˊ ˜ 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 letter headings 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 digraph 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 ˋ ¯ˆ ˊ ˜ 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] << ˋ << ¯ << ˆ << ˊ << ˜ << 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 << a̱ <<< A̱ << á̱ <<< Á̱
& c < ch <<< Ch
& e << é <<< É << e̱ <<< E̱ << é̱ <<< É̱
& g << gu << gü < 'g <<< 'G
& i << í <<< Í << i̱ <<< I̱ << í̱ <<< Í̱ < ɨ <<< Ɨ << ɨ́ <<< Ɨ́ << ɨ̱ <<<ˊ_
& l < 'l <<< 'L
& m < 'm <<< 'M
& n < 'ñ <<< 'Ñ
& o << ó <<< Ó << o̱ <<< O̱ << ó̱ <<< Ó̱ < ø <<< Ø << ǿ <<< Ǿ << ø̱ <<< Ø̱ << ǿ_
<<< Ǿ_
& u << ú <<< Ú << u̱ <<< U̱ << ú̱ <<< Ú̱ << ü < ʉ <<< Ʉ << ʉ́ <<< Ʉ́ << ʉ̱ <<< Ʉ̱
<< ʉ̱́ <<<ˊ_
&[last regular] < ' < 0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9
& [first secondary ignorable] << ˋ << ¯ << ˆ << ˊ << ˜ << ˇ
& [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.