L2/15-194

Title: A Response to L2/15-184 "Addressing SignWriting Collation in DUCET"
Author: Stephen E Slevinski Jr
Date: July 22nd, 2015
Status: Response and counter

Background
The specification for SignWriting has drifted from the actual community use towards a committee draft by people who have experience using script that are fundamentally different than SignWriting.

These differences have cause several issues. The issue at hand is sorting.

The problem arose with the invent of inherent values for Fill-1 and Rotation-1. This model is not used in current software or historical. There are 37,811 glyphs. They are referenced by either a 16-bit code (plane 16 codepoint with 6 character ASCII name of symbol key) or by a ligature definition of 3 characters.

The concept of inherent Fill-1 and Rotation-1 replaces a ligature defined with 3 characters for a single character ligature. This is a very significant break that has many ramifications. Primarily, it break sorting.

Secondly, it causes ambiguity. A simple search and replace can break a string. This affect can be seen in the Unicode 8 ligatures used to create the TrueType font as defined in the feature file.
https://raw.githubusercontent.com/Slevinski/signwriting_2010_tools/master/source/signwriting_2010_unicode8.fea

Notice that the symbols are decreasing from S380b7 down to S10000. This reverse order is required because inherent characters creates ambiguity that complicates otherwise simple routines.

Analysis
>> The feedback points out that a *binary* sort of the cited 4 strings will end up not in expected order. But a binary sort of *any* Unicode data seldom produces expected order for any script. So that was never in the cards.

I understand that is not a design requirement, but it is a very useful feature in environments without the full array of Unicode add-ons. Sorting a dictionary with simple SQL or with a basic JavaScript sort is very important for SignWriting's wide spread adoption and support. To throw it away with distain is troubling.

Additionally, Ken does not understand how we sort. Let's consider his example:

>> 1D800 SIGNWRITING HAND-FIST INDEX (HFI)
>> 1DAA1 SIGNWRITING ROTATION MODIFIER-2 (R2)
>> 1DA9B SIGNWRITING FILL MODIFIER-2 (F2)
>>
>> Now giving them some arbitrary weights, per the recommendation above:
>>
>> HFI = 100
>>
>> Next the rotation modifiers, say 410.., so R2 = 410

>>
>> Next the fill modifiers, say 420.., so F2 = 420
>>
>>1. HFI
>>   100
>>
>>2. HFI R2
>>   100 410
>>
>>3. HFI F2
>>   100 420
>>
>>4. HFI F2 R2
>>   100 420 410
>>
>>5. HFI HFI
>>   100 100
>>
>>6. HFI R2 HFI
>>   100 410 100
>>
>>7. HFI F2 HFI
>>   100 420 100
>>
\>>The expected order of the full strings is as shown:
>> #1, #2, #3, #4, #5, #6, #7, but because the weight for the base (HFI)
>> is lower than either the weight for the fill *or* the rotation modifiers,
>> the actual order that would be calculated based on the weights is:
>> #1, #5, #2, #6, #3, #7, #4, which is completely out of order.

Unfortunately, I disagree with his analysis. The correct sort order is
#1, #5, #2, #6, #3, #7, #4. If you care to meet on Skype of Google Hangouts,
I can share my screen and show you how it works.

Here is the list properly sorted with F1 and R1 to make explicit the correct sorting.

1. HFI F1 R1
5. HFI F1 R1 HFI F1 R1
2. HFI F1 R2
6. HFI F1 R2 HFI F1 R1
3. HFI F2 R1
7. HFI F2 R1 HFI F1 R1
4. HFI F2 R2

Regarding the rest of Ken's analysis, it doesn't seem to apply.