TO: Unicode Technical Committee L2/12-049
FROM: Deborah Anderson
RE: Encoding model options for Mende numbers
DATE: 29 January 2012

This document provides a summary of different possible approaches for encoding Mende numbers. It incorporates some discussion contained in the Mende proposal, L2/12-023 (pages 4-5, §8.1-§8.3), but includes additional options. The bulleted points under "Pros" or "Cons" are not evaluated here nor are they exhaustive, but are listed because they came up during discussion on the topic.

Note: If option **2** or **3** is chosen, the Roadmap Committee will need to remove 2 columns from the 1E900 row in the SMP which is currently set aside for Mende.

1 Atomic encoding
Description: This approach encodes the numbers atomically, requiring 72 code points. This is the model currently reflected in the Mende proposal, L2/12-023, and is the proposal authors' preferred method for handling Mende numbers.

Pros:

- By encoding the Mende numbers atomically, the character properties can have the correct values.
- Rendering will be simple, as is true for the main syllabary - no special ligation or OpenType behavior would be required.
- "Pre-composed" complex numbers have already been encoded for Cuneiform, Egyptian hieroglyphs, and the Aegean scripts, and many of these could, in principle, be "composed". The methods described below under **2** and **3** are not used elsewhere.

Cons:

- The Mende system is a *de novo* one, which works differently than Cuneiform, etc., so the argument which relies on the encoding models of Cuneiform, Aegean scripts, etc., is not analogous.
- This approach requires a total of 72 code points (filling one row, 1E8D0-1E8FF, plus two columns in the 1E900 row in the SMP). The alternative methods, described under **2** and **3** below, will require far fewer characters and be easier to type and map to a keyboard.

2 Encoding with Combining Characters
Description: There are two possibilities for combining characters with base characters, either (**a**) combining superscript units or (**b**) combining subscript units. Both (**a**) and (**b**) require a total of 16 code points.

In this model, the sequence of <base, subscript/superscript combining mark> would map to the fully constructed glyph. (When '10' is the subscript/superscript combining mark, the glyph without the dot would appear.)

a. Base characters and superscript units (pp. 4-5, §8.2 on L2/12-023)

Pros:

- Requires 9 combining superscript units and 7 base characters, far fewer than the atomic model, **1**.

Cons:

- The order for typing the characters would be less intuitive for users than use of subscript units (described in **2.b**, below).
- This method complicates the encoding and/or representation of the tens and twenties because the glyph for '10' 𞀰 has an inherent dot (or is it a second superscript unit? - *𞀰 does not occur and neither does the form without the dot). The teen mark "/" has no independent existence, and the numbers for "20" and above have no dot.
- Combining character encoding forces complex rendering requirements on Mende, which otherwise does not need it.
- If handled by dynamic composition, this method would require expert diacritic positioning in fonts, particularly over very wide bases like those of the hundreds and above. Such support may not be available in fonts used for display of filenames at an OS level.
  [Note: With a ligature table, fonts could still have fully-specified glyphs for the relevant combinations and would not require dynamic composition.]

EXAMPLE:

999 would be represented as <100, combining-9, 10, combining-9, 9>

b. Base character digits and combining subscript units (10s, 100s, etc.)

Pros:

- In this method, the bases would have roughly the same width, unlike (**2.a**, above).
- This method would require 16 characters, far fewer than atomic encoding in **1**.
- This method would better reflect the logical and speaking order for users.

Cons:

- Basically the same set of "Cons" arguments as above, **2.a**, except for first bullet

EXAMPLE:

999 would be represented as <9, combining-100, 9, combining-10, 9>, with the glyphs ligated as stacks for the 100s and 10s.

3 Atomically encode digits and units but represent stacks with ligated glyphs.
Description: This method would encode 16 characters (9 digits and 7 units) as atomic characters, and represent the elements which stack in Mende as sequences of a digit character followed by a unit character.

The presentation of the two-character sequences, <base, base>, would be as a ligated glyph. The font would require 63 ligature definitions, with the same number of display elements, but fewer characters than the atomic option described in **1**.

Pros:

- Easier to define a keyboard than the approach in **1**, and with fewer characters.
- The logical and typing order would be more natural for users than **2.a**.

Cons:

- This encoding model is not used elsewhere.
- Typographic ligatures are essentially optional, so it must be stipulated that legibility must not be compromised if the ligatures are broken.
- Use of ligatures forces complex rendering requirements on Mende, which otherwise does not need it.

EXAMPLE:

9,999 would be typed as <9, 1000s-character, 9, 100s-character, 9>, with the display ligating the 1000s-character to 9, and 100s-character to 9.