UTC Document Register **L2/14-127**

Standardizing the order of Arabic combining marks

Roozbeh Pournader, Google Inc.
May 2, 2014

Summary

The combining class of the combining characters used in the Arabic script has been creating problems for sequences with multiple combining marks since Unicode 2.0. The present document specifies some of the problems and proposes a solution.

Proposal

1. Specify that the order specified in Algorithm A or an equivalent algorithm should be used when displaying character sequences with multiple Arabic combining marks.
2. Specify that when the algorithm does not result in a preferred order, U+034F COMBINING GRAPHEME JOINER (CGI) should be used to request a different order.

Algorithm A

Definition: The Modifier Combining Marks (MCM), is the set of combining marks listed below:

U+0654 ARABIC HAMZA ABOVE
U+0655 ARABIC HAMZA BELOW
U+0658 ARABIC MARK NOON GHUNNA
U+06DC ARABIC SMALL HIGH SEEN
U+06E3 ARABIC SMALL LOW SEEN
U+06E7 ARABIC SMALL HIGH YEH
U+06E8 ARABIC SMALL HIGH NOON
U+08F3 ARABIC SMALL HIGH WAW

Input: the sequence of Arabic combining characters with ccc ≠ 0 with an optional base character or another character of ccc = 0.

Output: a canonically equivalent sequence of Arabic combining marks, in their real logical order, which could be used for rendering.

Steps:

1. Convert the input sequence to Normalization Form D (NFD).

2. Divide the resulting sequence into groups of the same combining class.
3. Output all the characters in MCM **at the beginning of** the group that has ccc=220 (*below marks*), keeping their relative order.
4. Output all the characters in MCM **at the beginning of** the group that has ccc=230 (*above marks*), keeping their relative order.
5. Output all the characters in the group that has ccc=33 (*shadda*), keeping their relative order.
6. Output the remaining characters, keeping their relative order.

Sample run

Here is a very artificial test case, just to demonstrate the algorithm:

```
inp: 0618 0619 064E 064F 0654 0658 0653 0654 0651 0656 0651 065C 0655 0650
ccc:   30   31   30   31  230  230  230  230   33  220   33  220  220   32
MCM:                     Yes  Yes       Yes                     Yes

out: 0654 0658 0651 0651 0618 064E 0619 064F 0650 0656 065C 0655 0653 0654
ccc:  230  230   33   33   30   30   31   31   32  220  220  220  230  230
MCM:  Yes  Yes                                          Yes       Yes
```

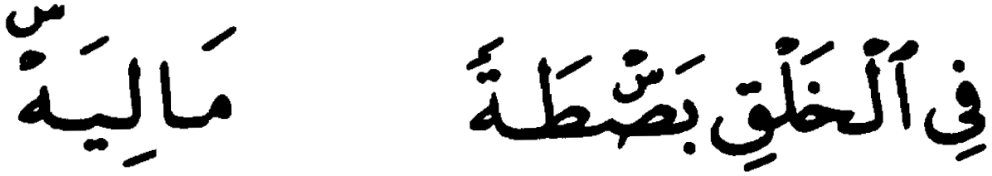
Background

The combining classes of Arabic combining characters in Unicode are “weird”. Here is a breakdown of the existing classes:

27: fathatan, open fathatan
28: dammatan, open dammatan
29: kasratan, open kasratan
30: fatha, small fatha
31: damma, small damma
32: kasra, small kasra
33: shadda
34: sukun
35: superscript alef
220: all other below combining marks
230: all other above combining marks

What that means, is when a sequence of combining characters from at least two different classes, the order is not specified in Unicode. This is what the Unicode Core Specification says on the subject: “The Unicode Standard does not specify a sequence order in case of multiple harakat applied to the same Arabic base character, as there is no possible ambiguity of interpretation.” (version 6.2, section 8.2, page 253).

Unfortunately, the last part of that sentence is not true. The order is indeed ambiguous, as the following examples show:



In the right example, the *small high seen* is seen below the *sukun*, while in the left example, it is seen over it. The examples are indeed from the same document (Al-Hilâlî and Khân 1996), just two pages away. The *small high seen* has different roles: in the right sample it is a hint that the base letter, *sad*, should be pronounced as if it was a seen; in the left sample, it is a pause-related hint.

Even if such contrastive examples didn't exist, the order would still have been unclear, since some characters of the combining class 230 (like *hamza*) are rendered below a *fatha* (ccc=30), while others of the same class (like most Quranic marks) are rendered above a *fatha*. This means that extensive research would be needed for rendering engines and/or fonts, easily leading to different findings and thus loss of interchangeability of Arabic text.

Some cases are "obvious" to speakers of languages written in the Arabic script. For example, a shadda always comes before a damma. But as we go further down the list of Arabic combining marks, less information is available. The best location to provide such information is indeed the Unicode Standard itself.

Questions & Answers

- **Why this algorithm?** The author believes it to be the simplest algorithm that keeps user expectations, stays within the stability requirements of Unicode, renders all canonically equivalent sequences the same way, and doesn't break orthographies yet unknown to the author.
- **Your algorithm puts a *damma* over a *hamza above*. What if my orthography needs to put the *hamza above* over *the damma*?** The algorithm provides the most expected output for most users of the Arabic script, which want the *damma* over the *hamza above*. You would need to encode your text as <damma, CGJ, hamza above>.
- **As you said, SMALL HIGH SEEN can happen both below a *sukun* or above a *sukun*. Why is it in MCM?** The author needed to make a decision here. The orthographies that use the character both ways would need to use in a CGJ in one of the cases anyway. But it appears that most Quranic orthographies use the character as an MCM only, so we put it in MCM.
- **Don't the *kasra* and *kasra*-like characters behave weird around *shadda* or *hamza*? Shouldn't the algorithm take care of that?** It should, and it does.

kasra-like character are ligated with a shadda or hamza in some styles and appear just below them instead of below the base letter, but they still logically follow the shadda or hamza.

- **What are the Modifier Combining Marks?** These are combining characters that modify the base letter before them, and because of that, should remain close to the letter when they are followed by *tashkil*. These are not exactly *i'jam*, but the next closest thing to *i'jam*.
- **Why NFD and not NFC?** To make sure sequences such as <superscript alef, madda> always results in the same ordering, independent of the base letter. If the algorithm used NFC, the sequence <alef, superscript alef, madda> would have resulted in a different order than <lam, superscript alef, madda>.
- **Is there anything controversial about this proposal?** This proposal received some early feedback in October 2013 on the HarfBuzz mailing list. There was some discussion about the difference in the identity of U+0653 ARABIC MADDAH ABOVE and U+06E4 ARABIC SMALL HIGH MADDA and if the normal madda should belong in MCM. After some offline discussion with Tom Milo, the author decided to keep them in the same group outside MCM, and consider the two characters size variants of each other.
- **What about the *sukun* alternate forms?** There are three sukun-like shapes encoded at U+06DF..06E1 that are used in some Quranic orthographies to denote different things which are not always a sukun. Fortunately, their canonical combining class is 230, so we don't need to worry much about their ordering in presence of other combining marks. Unfortunately, they won't be treated like a normal sukun in all theoretically possible cases. But this does not create a real world problem, as users who create the data simply have more flexibility with the alternate sukuns than with the normal sukun. The author prefers not to change the algorithm to make them equivalent, as that would make the algorithm unnecessarily complex and make the usage of CGJ more frequent.
- **Would your proposal break existing implementations?** Most probably not. The author has been very careful. Unless the implementation is not complying with Unicode, which we can't do much about anyway.
- **What about dotted circles in some implementations?** The author thinks it's not a very good idea to insert dotted circles in a text rendering engine, since something that appears invalid today to a rendering engine designer may actually be valid text in some lesser known orthography of a minority language or the Quran. (For example, Microsoft Windows's text rendering engine, described in Microsoft Typography 2014, appears to disallow combination of certain Quranic marks that are known to appear with each other in the Quran.)

  Such spell-checking processes are best implemented at a higher level than a rendering engine. Also, a dotted circle insertion algorithm that would display

all canonically equivalent sequences the same way is hard to design and the result may be counter-intuitive for its users.

Still, the algorithm could easily be extended for implementations that may wish to insert such marks (such as the one described Microsoft Typography 2014) by applying the algorithm first and then inserting the dotted circles.

- **Can we *improve* the algorithm in future versions of the standard?** We better not, or we may change the rendering of data that assumes its implementation. (There are ways to extend the algorithm for new characters while keeping the existing data stable. One would be adding new characters to the MCM. Another would be putting them in a new combining class or group like MCM and add a step to the algorithm to take care of them.)
- **But what if we missed some character that should really belong in MCM?** The author tried his best to handle all the cases he could find, across various minority languages and Quranic orthographies, but he may have missed some cases (and some cases are impossible to capture anyway). The best way to handle those interchangeably is to use a CGJ in the Unicode representation.
- **What are the best combining classes for Arabic combining characters encoded in the future?** For most characters, 220 and 230. For alternative versions of the basic tashkil, the same combining class as the tashkil could be used, but very very carefully.
- **How else can we use Algorithm A?** The algorithm is very useful for backspacing, when there is no external information available about the order used when entering the text.
- **Can we just generate our Arabic data in the same order as the output of Algorithm A would be and not worry about having such an algorithm in software?** You can do that, but then you can't correctly render or handle data coming from somebody else, including data in normalized forms.
- **Why do I need the CGJ again?** Because the default order of Algorithm A is not perfect (mostly for cases we don't know about, but still). You may want to modify the order of some marks that have different combining classes.

Bibliography

1. Muhammad Taqî-ud-Dîn Al-Hilâlî and Muhammad Muhsin Khân (translators) 1417 AH (=1996 CE). *The Noble Qur'an: English Translation of the meanings and commentary.* King Fahd Complex For The Printing of The Holy Qur'an. ISBN 9960-770-15-X.
2. Microsoft Typography 2014. "Developing OpenType Fonts for Arabic Script." http://www.microsoft.com/typography/OpenTypeDev/arabic/intro.htm
3. The Unicode Consortium. 2013. *The Unicode Standard Version 6.2 – Core Specification.*