L2/03-044

Encoding Arabic extensions:
options for the future of Unicode

Date: February 11, 2003
Author: Jonathan Kew, SIL International
Address: Horsleys Green
High Wycombe
Bucks HP14 3XL
England
Tel: +44 1494 682306
Email: jonathan_kew@sil.org

Summary

Unicode currently encodes Arabic letters as indivisible units. However, the true structure of the script is better understood as a small set of underlying "skeleton" letter forms, to which patterns of dots ("nuktas") are added to differentiate between sounds or letters needed to write a particular language. The creation of *base letter + nukta* combinations is a living, productive process, still used today as Arabic script is adopted for writing additional languages.

To model and support this, it is proposed that a set of combining nuktas be encoded in Unicode, allowing new *base + nukta* coinages to be encoded as needed.

This offers advantages to users, who would be able to accurately represent text that currently cannot be encoded in Unicode, and would not face the hurdle and delay of having to negotiate the standardization process for each particular *base + nukta* combination. It also offers advantages to the standards bodies and to implementers, in that it deals in a single step with what will otherwise be an open-ended list of new characters appearing over the course of many years.

1. Background

Mark Davis and Kamal Mansour's document L2/02–021, presented at UTC90, proposed the addition to Unicode of a set of encoded characters for "generative" representation of Arabic letters. Kamal's document L2/02–161 is a follow-up progress report on related investigations.

The justification for encoding a full set of Arabic nuktas is (at least) twofold, as suggested by Davis and Mansour. First, there are various languages where combinations of nuktas have been used to create novel letters that are not currently encoded, but are in use by local language communities—often because their languages have sounds not present in the major languages with well-established and already-encoded orthographies. The difficulty and slowness of collecting examples of written material from such communities means that if each such letter is to be encoded in its own right, the Arabic repertoire can be expected to have a trickle of "ad hoc" additions for many years to come. Second, in scholarly and pedagogical materials, a need can arise to deal with nuktas separately from base letters, either to apply different markup or formatting or to show the nuktas in isolation. This cannot be represented in Unicode without encoding the nuktas in their own right.

(Although not all such marks would strictly speaking be "nuktas" or dots—others include the SMALL TAH, RING, and SMALL V shapes—I will use that term here as a convenient label for the various marks that are added to "skeleton" (dotless) Arabic letterforms to form new letters.)

It is clear as one looks at the use of Arabic script for non-Arabic languages around the world, in north Africa, south and central Asia, and even as far east as the Philippines, that the practice of extending the alphabet by the application of new nukta combinations to existing base letterforms is widespread and well-established. This should not be a surprise; indeed, Arabic itself did this at one time, adding the nuktas of today's script to earlier dotless letters in order to represent distinctions that were previously unwritten. (It may be noted that the nuktas carry far more critical linguistic information than is typical for diacritics in European languages. While French or German without diacritics is generally still readable, with perhaps occasional ambiguity, stripping the nuktas from, say, a Sindhi text would make it difficult or impossible for a typical reader to understand.)

As Arabic script has been adopted by many language communities with languages quite different from Arabic, belonging to a variety of families besides Semitic, users have often needed to write a wide variety of sounds that are absent from the standard Arabic writing system. Significantly, they rarely invent entirely new letter shapes to do this, nor do they borrow from other scripts; in virtually every case, they take existing Arabic letters and add different nukta combinations to represent novel sounds. It appears that this practice seems natural, even instinctive, to many of those who use the Arabic script.

Until now, there has been no way within Unicode to dynamically create new Arabic-script letters in this way; the standard encodes a particular repertoire of letters, based on existing standards for Arabic and other major languages, together with extra letters from a small selection of minority languages for which information has been available. However, it is clear that there are gaps in the repertoire; many conceivable *base + nukta* combinations are not present, and at least some of these can be shown to have been used by someone somewhere. Given that Arabic script is still being adopted by and adapted for additional languages, often in relatively remote parts of the world, we can also be confident that the information we have is incomplete, and will remain so for the foreseeable future.

2. How extensive are the needs?

It is relevant to consider the extent of the need for additional Arabic-script letters. Many known extensions to the standard script have already been included in Unicode as characters in their own right. However, information from a variety of sources indicates that there are still a considerable number of novel letterforms that have been used or are being considered for use in lesser-known languages, and are therefore likely to need to be encoded at some point. In many cases, adequate documentation to present formal character proposals for the individual letters is difficult to obtain, but this does not mean that the letters should be entirely ignored.

Looking through just a limited selection of sources (see Appendix A), I have seen mention of around 50 possible extended Arabic letters that are not currently in Unicode, but have been used or suggested for use in minority languages where it is desired to use Arabic script:

BEH (skeleton) with:
- two dots vertically above the right-hand end (not centered)
- three dots above and one below
- three dots pointing upwards below
- three dots pointing upwards below and two horizontally above
- three dots in a horizontal line below
- four dots in a diamond shape below
- one dot above and one below
- two dots above and one below
- dot and filled diamond below
- dot and open diamond below
- dot and ring below
- dot and ‘circumflex’ (inverted V) below

JEEM skeleton (HAH) with:
- four dots in a diamond shape below
- three dots above and one below
- two dots above and one below
- one dot above and one below
- two dots horizontally above
- small TAH above
- dot below and small TAH above
- three dots below and small TAH above
- small TAH below

REH with:
- two dots and small TAH above
- three dots and small TAH above

SEEN with:
- two dots below
- two dots vertically above
- four dots above
- two dots and small TAH above

- three dots and small TAH above

AIN with:
- one dot below
- two dots above
- three dots pointing downwards above
- one dot above and DAMMA-like mark below

FEH with:
- two dots below

QAF with:
- one dot above and two below
- dot above and DAMMA-like mark below
- two dots above and one below

KAF with:
- two dots below

KEHEH with:
- three dots above

LAM with:
- small TAH above
- horizontal bar through the vertical stem

MEEM with:
- one dot below

NOON with:
- two dots below and one above
- dot and small TAH above
- ring above
- dot and small V above

WAW with:
- small TAH above

YEH with:
- small TAH above
- two dots and ‘circumflex’ below

YEH BARREE with
- small TAH above

Note that it is unlikely that every one of these letters will become established in wide use; some represent orthographic experiments that may well die out, and some might be able to be treated as glyph variants of each other or of already-encoded characters. However, this list serves to illustrate how the technique of adding new nukta combinations to existing letter skeletons is a productive feature of Arabic script. It is, unfortunately, very difficult to establish the exact status of many of these letters, or to collect examples of published materials, but we can be confident that some significant number of them do represent real encoding needs.

3. Limitations of the existing approach

At present, none of the characters listed above are included in the Unicode standard. Those wishing to use any of these to write their languages would have to encode them using Private Use codes; but the issues of directionality and contextual rendering mean that very few systems will actually support this in a useful way. Moreover, using PUA characters necessarily affects the interoperability of data, fonts, etc.

Potential users can, of course, propose the addition of the individual letters they need to the Standard, as has been done in the past. However, many of the would-be users and user communities are not well placed to do this, having limited access to and understanding of the technology and processes involved. It seems clear that following this model, it will be many years before the repertoire of Arabic-script letters in Unicode can be considered “complete” (if indeed this can ever be claimed!). Although there are technical experts interested in assisting the process, no one is in a position to gather and adequately document anything approaching an exhaustive set of the characters needed.

This means that font and software vendors aiming to support the Arabic block will be faced with a "moving target" for the foreseeable future, with sporadic addition of individual new letters whenever sufficient information becomes available to support their proposal.

Moreover, within a few years it is likely that the U+06xx block will be full, requiring expansion to a new Arabic Extensions block (but this will not belong with the Arabic Compatibility blocks, nor would it be appropriate to shunt it off to a supplementary plane).

Note that in addition to new Arabic-script letters such as those suggested above, there is also a need for a significant number of additional Arabic-specific diacritics. Many non-Semitic languages using Arabic script need to write more vowels than standard Arabic script provides, and various new diacritics have been suggested for this purpose. As with the consonant extensions, it is difficult to collect definitive information on which forms should be considered sufficiently established for standardization, but there will certainly be a number that merit encoding. This paper does not address these needs, which are for straightforward encoding of additional diacritics.

4. A flexible solution: combining Arabic nuktas

The alternative to encoding individual new Arabic-script letters as and when adequate supporting evidence becomes available for each one is to adopt the approach proposed by Davis and Mansour (L2/02–021). Encoding a set of combining Arabic nuktas would allow users to represent whatever novel combinations of letter skeleton plus nukta they may choose to write. This would remove the need to encode each individual letter when its case becomes strong enough, and would in addition allow users the flexibility to experiment with orthographic ideas while still working within the confines of the Standard.

Note that this encoding model would be entirely within the spirit of the script, in that users clearly understand that the skeleton letterforms are the most fundamental part of the script, and can carry various nukta patterns. These combinations of *base + nuktas* are well established for major languages, and constitute their established alphabets, but there is also the possibility of "coining" new combinations where needed, as illustrated by the list given above.

Encoding such a set of combining nukta characters for use in Arabic would amount to a new encoding model for the script, and as such cannot be undertaken lightly. The existing Arabic characters in the standard cannot be changed or removed, so the two models would have to coexist, raising questions of canonical equivalence and normalization, in addition to any inherent complexity of the model itself. However, given that nuktas are used productively within Arabic script, in a manner that is not supported by the existing encoding model, it is worth seriously considering whether the generative-nukta model could indeed be adopted as an addition to the existing standard.

Note that there is a real possibility of multiple nuktas being used with a single base character. As well as the obvious examples already in Unicode, where nuktas occur both above and below the same base, such as U+068B, U+069A and others, we have seen examples such as NOON WITH DOT AND SMALL TAH that stack several nukta-type marks. Given the difficulty of predicting what combinations might be used, it seems wiser to allow the "stacking" of nuktas and other similar marks than to try and encode each such combination as a "composite nukta" in its own right.

The exact set of nukta characters required still needs to be finalized; I will be happy to contribute to this work in the event that there is agreement in principle to encode such a set of characters.

4.1 Rendering

Adding combining nuktas to Unicode would entail some additional work for rendering systems and fonts to support the new approach to encoding. Many Arabic letters that are presently "shaped" by simple substitution of a joined glyph form would require both a glyph substitution step (to choose appropriately-joined letter skeleton forms) *and* a positioning step to place the nuktas correctly on the skeletons.

However, this would actually be a natural extension of the work already needed to support the various kinds of Arabic-script diacritics; it does not introduce fundamentally new problems of rendering. Any implementation that can efficiently render cursive Arabic script including diacritics should equally well be able to render such "decomposed Arabic", with only an incremental space and time cost.

Systems that lack provision for accurate dynamic diacritic positioning, relying on simple fixed-position overstrikes, would perform poorly with decomposed Arabic text; the text might often be barely legible, with nuktas mispositioned and sometimes colliding with base letters. However, most text in major Arabic-script

languages would probably still use the precomposed letters, so this does not imply that the behavior for text that is currently encodable would actually degrade. As such systems can in any case not handle Arabic script with full use of diacritics, the fact that they would also handle decomposed text poorly does not really weaken the case for such an encoding. Users with a serious interest in Arabic script will simply require implementations with good mark-positioning support.

Given that encoding a full set of combining nuktas would allow the Unicode Arabic repertoire to become stable much more quickly than gradually adding more precomposed letters as evidence becomes available, it actually offers an advantage to implementers in that fonts and rendering implementations would not need to be revised on such a frequent basis to keep up with the ever-growing repertoire of Arabic letters.

4.2 Normalization issues

A greater concern relates to normalization. With combining nuktas in the standard, there would be two possible ways to encode many Arabic letters: using a dotless letter and nuktas, or using the current precomposed letter. Clearly, these must be treated as canonically equivalent, and must therefore be treated identically under normalization. This seems at first glance to suggest that canonical decompositions for the existing Arabic letters should be added to the Unicode Character Database, so that U+062A ARABIC LETTER TEH would decompose to <U+066E ARABIC LETTER DOTLESS BEH, U+06xx ARABIC NUKTA TWO ABOVE>.

However, the requirement that existing normalizations cannot be changed means that this is unacceptable; it would break existing systems that rely on normalization form D. Existing data that is in NFD would suddenly become unnormalized by the new definition, breaking compatibility. Because the current NFD form of the Arabic letters is the precomposed letters themselves, it is essential that NFD continues to use this form, rather than a fully-decomposed *base + nukta* representation.

There appear to be three possible approaches that could be taken to the problem of normalization forms, in the event that agreement is reached that a generative approach should be used to encode extensions to Arabic script. We will consider each in turn, with their advantages and drawbacks.

4.2.1 Restrict usage of nuktas

Unicode could restrict the usage of the combining nuktas in such a way that letters that already exist in their own right cannot be encoded as sequences. Thus, the sequence <DOTLESS BEH, NUKTA BELOW> would be defined to *not* combine and form a letter looking like BEH. No new ambiguities are therefore introduced; any given Arabic letter still only has one Unicode representation. There is no impact whatsoever on normalization.

However, there is no obvious place within the existing standard to specify such constraints on the usage of characters, and indeed it runs counter to the general principle that any combining mark can be used on any base character. It is completely at odds with the design of the rest of Unicode, where (for example) the existence of a code for LATIN CAPITAL LETTER A WITH ACUTE does not "magically" make it illegal to use the sequence <LATIN CAPITAL LETTER A, COMBINING ACUTE>.

This solution makes the combining nuktas less helpful to those who would like to represent all Arabic text in a "fully decomposed" form, in order to regularize and simplify aspects of historical, linguistic or calligraphic processing, as all the standard Arabic letters could not actually be represented in their decomposed forms; these would be the very sequences that would be defined to be illegal. (It might be possible to use these "illegal" sequences for purely internal processing purposes, although one would have to be careful to distinguish such usage from truly illegal sequences in the original data—they should not be treated the same!)

It also requires implementers to deal with a specific "exclusion list" of apparently-typical sequences that must not be rendered "normally", nor interpreted as if they meant what they "ought" to mean. This would represent an unwelcome burden on every implementation that wants to handle Arabic script in any way.

I consider, then, that this option is impractical; it evades the normalization issue only by requiring implementers to respect arbitrary constraints on character co-occurrence. It seems unlikely that such a scheme would ever be widely and correctly implemented.

4.2.2 Allow multiple, non-canonically-equivalent representations

We could accept the fact that there will be something like 100 Arabic-script letters that can be encoded using existing single codes *or* composed using dotless letters plus nuktas, but normalization will not deal with these cases. Processes relying on Unicode normalization to determine string equivalency will fail to recognize these alternatives as equivalent even though they should be visually indistinguishable.

This option leaves normalization untouched, which is attractive to those who have to maintain normalization code. It can be argued that there are already cases where, for various reasons, the normalization algorithm does not deal with every equivalence that it should; I believe there are cases in Burmese, for example, where visually indistinguishable "spellings" are not canonically equivalent. (This is aside from the issue of possible "character spoofing" when unrelated characters, e.g., from different scripts, happen to look identical.) Introducing combining nuktas without addressing normalization would simply add a new class of such ambiguities.

While this might be regarded as an acceptable outcome—it merely adds instances of an already-existing class of problem, not creates an entirely new problem—it seems unfortunate, in my opinion, to create this many new ambiguities at a stroke. Most of the Arabic alphabet would suddenly become subject to "character spoofing", as opposed to a few instances at present. Operations such as searching and sorting that typically rely on normalization to deal with multiple possible representations of the same text would fail to work reliably unless implementations included some additional "special knowledge" about equivalencies in the Arabic block—which would surely be done infrequently and inconsistently, if at all.

There is a reasonable expectation among implementers and users that code sequences that are visually indistinguishable and semantically synonymous should be canonically equivalent, and exceptions to this may be considered flaws in the standard. We cannot fix every existing "problem", but knowingly adding this many new "flaws" would seem unfortunate.

4.2.3 Revise the normalization algorithm

It would be possible to revise the normalization algorithm such that precomposed and decomposed (*base + nukta*) forms of Arabic letters *are* canonically equivalent, but with the normalized form (even under NFD) being the existing precomposed letter where available. This allows normalization to give the expected equivalencies, while protecting the integrity of existing data.

To achieve this, it is proposed that rather than adding the decompositions of the current precomposed Arabic letters to the UCD as canonical decompositions (which seems natural, but contravenes published Unicode stability policy), a new property that could be named "required compositions" should be defined. The existing precomposed Arabic letters would have their "decomposed forms" defined here. The intent is that the required composition property gives compositions that must *always* be used during normalization—even in NFD.

The decomposition step of NFD would then be defined to make use of both canonical decompositions *and* required compositions (in order to *fully* decompose before canonical ordering). Then, after applying canonical ordering, the required composition (but *not* decomposition) mappings would be used to recombine base characters with nuktas. This step will operate just like the composition step used in NFC, except that it will use only the new required composition mappings, not the canonical decompositions from the composition version of the UCD.

The end result is that there is no change to the NFD form of any existing data; the amended algorithm simply ensures that wherever a precomposed Arabic character exists, it is always used as the canonical form, even in NFD. Although this seems counter-intuitive (as it means NFD is not as decomposed as possible), it is required to maintain stability.

In fact, the simplest way to implement this revised normalization algorithm is probably to add the required compositions to the current decomposition mappings, as used by the NFD algorithm. (This could be done for internal implementation purposes even though statement 3.a in the stability policy rules out adding new decomposition mappings to existing characters in the UCD itself.) The algorithm itself then needs no change except the addition of a composition step at the end. This would be identical to the composition step at the end of NFC, except using the new required compositions in place of the canonical decompositions from the composition version of the UCD. Thus, it should be possible to reuse existing code to implement the updated algorithm, minimizing the burden on implementers.

If implementers take the approach of adding the required compositions to the canonical decomposition mappings, the implementation of NFC should not require any change; existing NFC algorithms should continue to give the correct result. The increased complexity of NFD is only the same as that already present in NFC, so the performance impact on processes should not be excessive. Note also that existing implementations based on the current version of the UCD would not be invalidated; they would continue to be correct for the version they support. Only implementations wishing to support a newer UCD version would need the enhanced algorithm (and they must in any case be updated with new data tables for a new UCD version).

An additional "normalization form", perhaps NFF (normalization form *fully*-decomposed) could easily be defined, comparably to the existing NFD; this might be useful to implementers of various processes. But it

would *not* be a replacement for NFD, because of the requirement to maintain stability for this form. Processes and specifications that depend on the stability of NFC and/or NFD would simply ignore this new form.

I consider this the preferred approach, and suggest that it be given serious consideration. It requires all normalization implementations to be updated for the Unicode version, but updates (at least to data tables) are required to support new Unicode versions anyway. No new algorithms are needed; all that is required is a new final step in NFD, equivalent to the final step of NFC but using the new "composition" property instead of the current canonical decompositions.

In particular, the stability guarantees given by the Unicode consortium are not compromised by the proposed changes; data that is currently in one of the normal forms remains normalized under future versions of the Standard.

4.3 Combining classes

The question of appropriate combining classes for the nuktas requires some attention. Given that the nuktas are closely associated with the base letter, it seems natural to assign them a low combining class value; this would keep them close to the base letter in NFF, which could benefit analytical processes and rendering systems. It could also tend to help the efficiency of the NFC/NFD algorithms which need to recombine *base + nukta* sequences.

There is already a combining class, 7, used for "nuktas" in Indic scripts; these are consonant-modifiers that go below the basic consonant, and thus very similar to the proposed Arabic nuktas. I suggest, therefore, using this same combining class value for the nuktas that are positioned below the base letter, and 6 (8 is already in use) for those that go above. Nukta-like marks that actually attach to the base letter (ring, as seen on U+067C and others; stroke through, as seen on U+06C5) could have combining class 1 (also used for combining overlays in the U+03xx and U+20xx blocks).

The one possible disadvantage I see with using these combining class values is that they differ from the class of the combining HAMZA marks that are already in Unicode. U+0681 shows that the HAMZA form has been used as a nukta-like mark to create a new letter in at least one instance, in addition to its conventional use on ALEF, WAW, and YEH. It therefore seems unfortunate for it not to share the combining class value of the other nuktas. However, stability considerations prevent us changing the class of the combining HAMZA marks, so this is an anomaly that will apparently have to remain.

The alternative, using the same classes as the HAMZA marks for the new nukta characters, would make for an extremely unnatural canonical order for fully decomposed Arabic (with vowels, etc., interposed between the skeleton and its nuktas), and would force the "required recomposition" step of all normalization processes to routinely look further ahead.

5. Conclusion

There are a substantial but unknown number of extended Arabic letters still to be encoded in Unicode. Obtaining accurate data on these characters is difficult given the remoteness and lack of technical sophistication of many user communities, the lack of standardization, and the fact that extension of Arabic is a live process even today.

The existing model for encoding Arabic script is cumbersome for users in that it fails to reflect the generative nature of the nuktas and related marks, making it impossible to represent written forms that do not yet happen to have been encoded as letters in their own right even though they are reasonable applications of the script. The socio-linguistic realities in many potential user communities mean that encoding all needed extensions as individual letters, under the current model, will be a protracted process taking many years. During this time, implementers will be faced with a constant need to revise systems as a few new letters are added to the Arabic block each year; or perhaps more likely, many implementations will be slow to fully support new letters, creating ongoing difficulties for users.

Supporting a new, generative model of Arabic script would benefit users by allowing them to encode new or even experimental written forms, in accordance with their understanding of the structure of the writing system, which allows such innovations. It would also benefit implementers in that it would greatly reduce the ongoing need to keep adding letters to the Arabic block.

This model would put Arabic script, with its productive use of nuktas and similar marks, on an equal footing with Latin and its diacritics. Latin users have ready access to a variety of precomposed *base + diacritic* combinations, but they also have the freedom to use new combinations without an onerous requirement of

documenting and justifying each combination and waiting for updates to the Standard and to implementations. Arabic users should be able to use the generative marks of their script with the same flexibility.

Potential obstacles to supporting the generative model (in addition to, not in place of, the existing Arabic characters in the standard) are twofold. First, implementers may be concerned at the added complexity of rendering with dynamically-placed nuktas. However, this is comparable to the already-existing issue of dynamically positioning vowel marks and other diacritics; it does not introduce new degrees of complexity but could be implemented using the same mechanisms as are already needed for any good Arabic rendering system.

Second, there are implications for normalization, in that existing "composed" Arabic letters could also be encoded in a "decomposed" form. These alternatives should clearly be canonically equivalent. This can be addressed by an extension to the NFD algorithm, which would maintain the stability of normalized data that Unicode has promised. The updated NFD algorithm would be of the same complexity as NFC (it is currently one step simpler); indeed, it would probably be possible for many implementations to use the same code, just with a different (smaller) data table at the final composition stage.

It would be helpful to all for the UTC to take a clear position on whether a generative Arabic model will be supported. This will allow users and communities with extended-Arabic character needs to know which path they should pursue to get their characters supported in Unicode.

Appendix A: sources

Sources for the Arabic-script letters listed in section 2, representing possible new encoding needs, include:

Chtatou, Mohamed. 1992. Using Arabic script in writing the languages of the peoples of Muslim Africa. Rabat: Institute of African Studies.

Addis, R. T. 1963. A study on the writing of Mandinka in Arabic script.

Mansour, Kamal. 2003. http://www.bisharat.net/A12N/Afro-Arabic_Symbols.pdf.

Baart, Joan L. G. and Muhammad Zaman Sagar. 2002. The Gawri language of Kalam and Dir Kohistan (http://www.geocities.com/kcs_kalam/gawri.pdf).

In addition to the published sources listed, a few characters have been mentioned in personal communication with linguists studying South and South-east Asian languages.