

From: John Cowan <jcowan@reutershealth.com>

Date: 2002-04-10 10:53:42 -0700

To: unicoRe@unicode.org

Subject: Re: LS/02-134

Sender: unicore-bounce@unicode.org

I have sent the following as a private response to Mr. Simpson.
UTC, make what use of it you will.

Dear Mr Simpson:

I am writing in response to your paper entitled "Keyboards, Fonts, and the Unicode Standard". I must emphasize that I am not a member of the Unicode Technical Committee, and do not speak for the Unicode Consortium. This response is as correct as I can make it, but must be interpreted as purely personal.

It seems to me that your paper has one major and a variety of minor misunderstandings of the Unicode Standard, and I offer this response to help clear up some of the confusions.

The problem of mapping

"Latin-script fonts [...] appear to map non-ANSI characters such as Polish z or Turkish i on a somewhat ad-hoc basis."

There is actually quite a lot of standardization for the Latin as well as the Greek, Cyrillic, Hebrew, and Arabic scripts in the 8-bit world. There are ISO standards as well as vendor standards (sometimes derived from them, sometimes not) for all these scripts. Polish is normally rendered with ISO 8859-2 (where Western European languages use ISO 8859-1), and Turkish with ISO 8859-9. I agree that the Indic languages are much less commonly standardized, although there is an Indian standard called ISCII (on which the Unicode Standard's Indic scripts are directly based).

"This approach means that, except for western European languages, there will not in general be a one-to-one relationship between [...] the rendered text and the code stream which is stored or transmitted."

In fact essentially all languages using the Latin, Greek, Cyrillic, Hebrew, Armenian, and CJK scripts will tend to behave like this, with occasional exceptions.

"But a rendering engine can only select from the repertoire available in the chosen font, so any sign that has to be printed must be mapped to some code point."

This is a subtle error which infects most of the rest of your paper. A font is a combination of informational tables and a set of images. A rendering engine accepts a stream of characters (expressed in Unicode or some other character set) and uses the informational tables to choose which images should be displayed on the output device. It

is only in trivial cases that there are exactly as many images in the font as there are characters in the character set. In real-life situations there are typically many more.

Let us consider a simple case: a font meant for rendering English and a rendering engine that understands only the 96 ASCII printable characters. A dumb font would simply have 96 images and a trivial table that maps every character from 0x20 to 0x7F to precisely one of them, and the rendering engine would have little to do. If the rendering engine found a ligaturing table, though, it would be able to take advantage of ligatures present in the font to render the character sequence 66 69 using a fi-ligature. This would be entirely transparent to the application making use of the rendering engine: changing the font would in no way affect the application programmer.

It's true that the font has to have some internal code representing an fi-ligature, but that code is of no concern to anyone except the font and the rendering engine, and even the rendering engine discovers what the code is by looking at the ligaturing table. The entries in the table conceptually look like this:

66 69 -> 80

where 80 is the arbitrary font-assigned code for the fi-ligature image. There is neither need nor reason to standardize this code provided only that the ligature table is consistent with the numbering of the glyph images. The value 80 is not a "code point", but rather an internal image number.

"But if a sign has no code point specified in the Standard, it will have no 'natural home', as it were, in a font."

In principle no character has a "natural home" in any font. There is no reason other than convenience why the image number (as I have called it above) of "A" should be 41; it might perfectly well be 1 or 1001 as long as the CMAP table (which maps individual Unicode characters to image numbers) correctly records that fact.

"[...] constant need for rendering, 'derendering', and re-rendering [...]."

No derendering is required. Applications store characters and ask a rendering engine to render them as needed.

"Anything must be an advance if it simplifies text processing by reducing the number of stages required between [...] typing and [rendering]."

In fact text processing, except in the simplest case of a word processor, often involves complex transformations between typing and rendering, if rendering is a goal at all; indeed, much text is not typed by anyone, but automatically generated.

"[...] case is contextually determined [...]."

Context has influence on the choice of case, but does not *determine* it. It is easy to construct sentence pairs like:

He went to the north pole.
He went to the North Pole.

where the distinction is beyond the power of any computer to insert automatically. The same is not true of Arabic shaping, except in exceptional cases for which ZWJ and ZWNJ are provided. I discuss Arabic shaping further in I.4.

Note: The ARABIC LETTER HEH appears in the charts in initial rather than isolated form to avoid confusion with the identical-looking ARABIC-INDIC DIGIT FIVE, as noted on p. 195.

" [...] matras [...] not aksharas [...]."

Processes other than rendering have difficulty coping with characters that may be either combining marks or base characters. For example, the DOT ABOVE in Irish use is equivalent to a following LATIN SMALL LETTER H. However, creating a separate character SEIMHIU which could be rendered as a dot in Gaelic-style fonts and an "h" in Roman-style ones would entail difficulties in deciding on letter boundaries, an otherwise font-independent operation. Therefore, the Unicode Standard in every case implements matras using a separate codepoint (marked as a combining mark) from the corresponding independent vowel (marked as a letter). The design goals of the Unicode Standard require occasional compromises between rendering and non-rendering processes.

"The sign [ks.a] can hardly be regarded as a variant of either of its constituents [...] since its form is quite different."

Form is not the issue. Rather, non-rendering processing is improved if the fact that [ks.a] contains two letters is clearly recognizable. In addition, exactly which conjuncts are required in Devanagari writing is language-sensitive: a well-tuned Sanskrit font requires hundreds of conjuncts that are not used (and would scarcely be recognized) in modern Hindi writing.

"[...] the font designer must assign a specific code to the conjunct [...]."

As discussed above in the consideration of ligatures, this code need not be of concern outside the font. Conjunct tables are in principle the same as ligaturing tables, and may be identified with them in particular font standards (I am no expert in font internals).

"[T]he Standard then goes on to assign code points to numerous Latin, Armenian, Hebrew, Arabic, and Tibetan combinations, ligatures, and combining blocks [...]"

Unicode is an evolving standard that is over ten years old, and its

predecessor XCCS goes back almost a decade before that. Certain compromises for backward compatibility or political reasons were inevitable in the long process of its adoption as an industry as well as an international standard. The Latin ligatures, for example, were required for one-to-one transcodability between Unicode and Macintosh character sets. An absolute consistency in such a large intellectual creation over such a long stretch of time is not to be looked for, although certainly desirable. The presence of a character in the Standard does not mean, especially in the case of compatibility characters, that it is desirable to use them.

"[...] as well as to the Ethiopic syllabic characters and thousands of precomposed Hangul syllable blocks [...]"

A preliminary encoding of Ethiopic used a decomposed model with abstract consonant and vowel characters. However, this is substantially more artificial than its Arabic-style or Indic-style analogues. Although Ethiopic syllables consist visually of modifiers attached to consonant forms, the consonant forms never appear without the modifiers (unlike the Hebrew and Arabic situations, where consonant-only text is common). Nor is there any analogue of virama in the Ethiopic system. Encoding Ethiopic in such a way would be equivalent to encoding Canadian Syllabics with consonant characters and "rotate 90", "rotate 180", "rotate 270" and "shrink" characters for the vowels.

As for the Hangul Syllables block, the original intention was to encode only the Jamo and a smaller set solely for compatibility with the Korean national standard, which encoded only the composed syllables actually in use in modern Korean. In Unicode 1.0, two such sets were encoded, requiring a large mapping table to convert between jamo and syllabic form. When the Korean standards body produced another standard containing every possible combination of (modern) jamo, whether the resulting syllable was in modern use or not, the Unicode Consortium and ISO WG2 bit the bullet, removed all the existing Hangul syllables, and added the new "universal" set (although Middle Korean cannot be rendered using it, but only with the Jamo). It was the reverberations of this "Korean mess" that determined UTC and WG2 never to move any characters again, no matter what.

"In Tibetan, however, subjoined forms are encoded [...]"

This particular model of Tibetan was adopted only after a great deal of consideration. Unicode 1.0 had a preliminary implementation of Tibetan on the virama model, which was withdrawn in Unicode 1.1. Unicode 2.0 reintroduced Tibetan using the current model, and extended it in Unicode 3.0. Among the Brahmi-derived scripts, Tibetan is an extreme case for conjunct formation: no other script has such deep stacks of consonants. Overall, it seemed that the allocation of a small number of subjoined characters in this one case would be particularly useful, especially in the expression of Sanskrit in the Tibetan script.

"Arabic joining forms are encoded [...] yet Syriac joining forms are not."

Arabic joining forms are provided solely for backward compatibility with existing standards, as you note. No such standards were in place for Syriac, so no such characters were encoded.

"[T]here are no separate codes for Arabic-script combining dot patterns [...]"

The combining characters in the U+0300 block, despite their rather Latin appearance, are meant for use with all scripts. In general, the Unicode Standard (tracking various national standards) decided not to get into the business of decomposing Arabic letters into basic shapes plus various dots. If such characters are needed, they can be added as new Arabiform letters, or created ad hoc with the existing combining characters.

Ampersand and Tamil abbreviations

Abbreviations are in general encoded only if they are distinct from the letters that form them. "&" is not truly a ligature of "et" any more:

it would be incorrect to write "AT et T" for "AT&T". A number of Tamil date abbreviations are being added to Unicode.

"The Hangul Jamo block includes 'final consonants', which might legitimately be regarded as 'joining forms' [...]."

It was probably a mistake not to classify Hangul vowels and finals as combining marks, but it's too late to change their status. However, the natural dividing point in Hangul is between the final and the following initial, rather than after the vowel as in Indic scripts, so some distinction must be made between initial and final consonants.

Alternative placement forms

Alternative placements of diacritics can be managed by the same old method of contextual glyph image selection within a font. Managing the two variants of seghol can be done with a table like this:

```
aleph seghol -> aleph-image centered-seghol-image  
yod seghol -> yod-image right-displaced-seghol-image
```

Again, the codes used for these seghol images are internal to the font and its tables, and need not be standardized.

"Unicode fonts"

"However, for many scripts a 'pure' Unicode font, i.e. one in which only Unicode-encoded characters were provided, would be manifestly insufficient, because it would lack vital conjuncts and combining forms."

This is true, and could be reworded thus: "[...] on in which only a

single image is provided for each Unicode-encoded character [...]"

"The alternative method requires the use of [ZWNJ] and [ZWJ] to indicate where joining characters are to be connected [...]."

This is not correct, and is based on a misunderstanding. These characters need to be used, and should be used, *only* where the normal rules of Arabic shaping are being violated. In your example, none of the ZWJ characters are required and none would normally be used: the substitution of contextually determined forms (initial, medial, and final) would be entirely automatic.

The implementation of this behavior requires a different kind of table in the font. This one specifies what images are used for initial, medial, final, and isolated forms of each Arabic or Syriac character. The font rendering engine applies a small set of rules, laid out on pp. 192-93, to decide which particular contextually determined form to use.

In your example, "what the user types" when rendered into Unicodes is essentially what is stored. Rendering is then done, as on all modern GUI systems, from the stored form rather than directly from the keystrokes. No derendering is necessary or even possible.

Note that in Farsi, the ZWJ and ZWNJ (especially the latter) are much more commonly required, and actually do appear on keyboards. The Farsi plural suffix, for example, is never joined to the word preceding it, so a ZWNJ is used to separate the two. In Arabic itself, the main use of ZWJ is to display contextual forms in isolation for exemplary purposes.

Conclusions

"If there are going to be such things as 'Unicode fonts', they must still provide a complete range of printed signs."

Absolutely correct; however, I hope to have shown that the internal numerical codes used for these printed signs (the glyphs of the Unicode Standard's definitions) need not be standardized for interchange of plain text, which is the primary purpose of Unicode.

Note, however, that a "Unicode font" in the sense of a font able to render all of the Unicode Standard is an unwieldy object and not really necessary or desirable except as a fallback. A well-tuned Spanish font will contain the character O WITH ACUTE, and so will a well-tuned Polish font; but the Polish glyph will look rather different, as the acute-accent part of the glyph will be more steeply inclined. This is normal and acceptable, since bare legibility is preserved even if this distinction is ignored. In fine typography, however, it is still necessary to use separate fonts for separate linguistic and typographical traditions; what cannot happen in a Unicode regime is that the bare lexical meaning of a text is incomprehensible to a non-rendering process because it uses the "wrong" character encoding.

"Thus the rendering engine must accompany the text at all times."

Not at all. It is merely necessary that the recipient have a rendering engine that is aware of the semantics of the Unicode characters used by the text. A rendering engine that can handle all of Unicode, unlike a universal font, is a perfectly reasonable component of modern operating systems, though there are still many legacy systems with old engines. Both Mozilla and Internet Explorer, for example, have built-in engines to compensate for deficiencies in the operating systems under which they may be run.

"Unicode format files will in general be larger than rendered-format files [...]."

Not at all. In UTF-8, standard Arabic will be more compact (two bytes per character) than pre-shaped Arabic (three bytes per character); in UTF-16, they occupy the same space except where the standard rules are being violated, i.e. very occasionally in Arabic text, and rather more often (but still fairly rarely) in Farsi text. I don't know about the other languages using the Arabic script.

"Unicode format files will require rendering or derendering whenever they are displayed, printed, modified, or saved"

As noted, no derendering is required. Unicode files, like other files, must be rendered when displayed or printed. Internal operations can and should be done in terms of Unicode characters rather than rendering-specific glyphs.

Part II

Coptic:

There is a proposal on the table (with which I personally agree) to disunify Coptic and Greek. Indeed, Michael Everson has shown that Coptic is actually rather more readable when rendered with Cyrillic letters than with ordinary modern Greek minuscules. So far, there has not been enough pressure from either the Coptic or the typography community to get this proposal acted on.

Georgian:

More recent versions of the Unicode Standard clarify that the intent is to use the range U+10D0 to U+10F0 whenever Georgian is written in the usual monocameral way, and to reserve U+10A0 to U+10C5 solely for uppercase letters (whatever their form) when Georgian is written in the archaic or archaizing bicameral style. Thus monocameral asomtavruli or khutsuri text would be written using the former range rather than the latter, and the actual distinction between the two, and between either of them and the modern mkhedruli script, is left to fonts.

Analogously, the distinction between Fraktur and Antiqua in German

typography is also left to fonts, despite the fact that Fraktur can be very hard to read for those who only know Antiqua.

--

John Cowan <jcowan@reutershealth.com> <http://www.reutershealth.com>
I amar prestar aen, han mathon ne nen, <http://www.ccil.org/~cowan>
han mathon ne chae, a han noston ne 'wilith. --Galadriel, _LOTR:FOTR_