The Problem with Unicode

Neville Holmes, University of Tasmania

One reader took my "Seven Great Blunders of the Computing World" column (*Computer*, July 2002, pp. 112, 110-111) as generally offensive to the memory of modern computing's great pioneers (Letters, Oct. 2002, pp. 9-10). Others questioned if more significant blunders than those I selected exist (Letters, Nov. 2002, pp. 8-9). But only one group, Unicode supporters, strongly denied a particular alleged blunder.

A relatively inconclusive e-mail exchange led me to offer this column to Unicode's most vocal supporter for a considered one-round debate on the issue, which he accepted. After he failed to respond to the half column I sent him supporting my claim, I suspected that the Unicode people had withdrawn from the debate when they realized that by *blunder* I did not mean *failure*.

However, a recently arrived issue of *Vector*, the British APL Association's quarterly journal, led off with an Adrian Smith editorial commenting on Unicode (www.vector.org.uk/v193/ed193.htm). This piece recalled Unicode's basic problem, which particularly afflicts technical symbolism. Thus provoked, I will now expand my case in the very faint hope that a much better approach to digital implementation of the world's writing systems might yet be adopted.

Unicode is a success, but would another approach have fared even better?

UNICODE

What is Unicode? The official Unicode site states that it is an encoding system that "provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language" (www.unicode.org/unicode/standard/WhatIsUnicode.html). That not all characters have unique numbers is one of Smith's complaints. A little further on, the text states that the "Unicode Standard has been adopted by … industry leaders," that it "is required by modern standards," and that it "is supported in many operating systems, all modern browsers, and many other products."

Even a brief study of the online material, impressive in both amount and detail, confirms that Unicode clearly is an admirable success. But all this does not avert my claim that Unicode is a blunder. A different approach would have worked much better for encoding text, documents, and writing systems.

TEXT AND DOCUMENTS

Text encoding and document encoding differ, although the two cover a spectrum of written-language uses. At the most populous end of the spectrum lie plain text messages such as I have to deal with every day: letters, e-mail, handwritten notes. Plain text of this kind, being mostly brief and personal, never mixes writing systems. Toward the spectrum's other end lie formal documents, beautifully typeset and replete with tables, indexes, and illustrations. Rarely do writing systems mix in such documents. Somewhere in the middle lie HTML documents.

Traditionally, we use *markup* to produce all but the simplest documents. This system of coding in situ instructions to a compositor, human or programmed, prescribes detailed aspects of the document's final form. The typical modern markup coding defines a hierarchy of detail by specification or default. For example, the coding first specifies a font, within the scope of that specification it specifies a size, then within the size it specifies a form: roman, italic, or small capitals.

The markup is encoded in plain text, as is the text it modifies. This text is coded within a single writing system—properly so for simplicity's sake. The resulting document almost never needs more than one writing system.

The writing system specification properly belongs at a level above the typographical. Font classes such as *typewriter, serif*, and *sans serif* have as little meaning in the Arab writing system as *diwani, kufic*, and *thuluth* have in the Latin writing system. The Arab allography has no relationship to Korean hanguel syllabary, and both starkly differ from the Latin writing system with its two cases and spaces between words.

Continued on page 114

Continued from page 116

Table 1. Possible eight-bit coding system for the Latin writing system.

Binary	Hexadecimal	Class
000x xxxx	00-1f	Modifiers
0010 xxxx	20-2f	Combiners
0011 xxxx	30-3f	Punctuation
0100 xxxx	40-4f	General symbols
0101 xxxx	50-5f	Arithmetic symbols
0110 xxxx	60-6f	Italic digits
0111 xxxx	70-7f	Roman digits
100x xxxx	80-9f	Italic smalls
101x xxxx	a0-bf	Italic capitals
110x xxxx	c0-df	Roman smalls
111x xxxx	e0-ff	Roman capitals

Documents are best marked up in a single writing system, with any mixing of writing systems specified through markup directly or, better, by using macrodefinitions or specifying an inclusion.

THE LATIN WRITING SYSTEM

By putting all writing systems and languages together, Unicode becomes much too complex and unstable. A far better approach would be to provide a standard for each writing system, with each standard providing the system's specific graphical characteristics. The Latin system of writing can, for example, be completely and effectively encompassed in an eight-bit coding system. Table 1 suggests this system's nature.

This approach treats the writing system as a generative graphical structure from which basic symbols can be selected, modified, or combined outside any particular font. Specific fonts can then vary the details of plain, modified, or combined forms in their own ways.

No attempt should be made to provide compatibility with ASCII or EBCDIC, especially with those extremely awkward control characters intended for use in telegraphy. The sooner we phase out these obsolescent systems the better.

In addition, no attempt should be made to implement any particular collating sequences. Not only are these complex, they also differ from culture to culture. For example, German treats *ä* as though it were *a*, while Finnish treats the two as distinct. English treats *rh* as two letters, while Welsh treats them as one. Thus, the placement of symbols within alphabets should be chosen to support transliteration.

By ignoring traditional alphabetic sequences, other alphabetic writing systems—particularly the Greek and Cyrillic—can probably be accommodated at the font level so that acceptable direct transliteration can be achieved across those systems. This would permit a single Cyrillic-Greek-Latin coding standard rather than three separate standards.

Modifiers and combiners

The modifiers and combiners provide for the numerous symbolic variations and distinctions needed both for different languages and for specialists such as mathematicians and phoneticists. Modifiers affect a single symbol, combiners affect two symbols, and the affected symbols can themselves be basic, modified, or combined.

Modifiers can shrink or expand, thicken or thin, raise or lower, rotate or reflect, double or treble, with many of these permutations offering horizontal, vertical, or other variations. A horizontal reflector would let ([{< be generated from)] }>, for example.

Combiners can juxtapose, ligate, or overlay the two symbols they affect in various ways. For example, an accenting combiner would usually place a punctuation mark over a letter of the alphabet. Traditional symbols such as @#$£¥% are overlays, and even the & originated as an E ligated to a subscripted t after the Latin *et*, and thus would not need a basic code. The combiners generalize the kerning first used in the 16th century to accommodate the Greek writing system, as Figure 1 shows, as well as the coding similarly used in TeX to effect accents.

The generative capability of this approach provides for complex use of accents as in Vietnamese and for the stable generation of new transliterations and symbols, thanks to typography's ability to provide esthetically pleasing forms of newly popular compound symbols such as the euro.

Other basic letters

Punctuation codes allow for very simple symbols, useful in combination, such as accents. These symbols, which should include the blank and some rulings, also allow versatility in modification, particularly when doubled, as in “ and =. The general and arithmetic symbol codes provide basic shapes chosen for their usefulness in combinations, as most of the familiar symbols can conveniently be generated as compounds.

The codes for the two sets of digits provide for 10 numerals as well as for all the signs needed to represent decimal values, such as a negative sign and decimal point. This allows compressed four-bit coding for numbers in special numeric applications.

Keyboards

Given that different languages use the Latin writing system differently, using this coding system would require different keyboards and keyboard drivers, with some single keystrokes generating several bytes of code. In particular, because the alphabets will provide roman and italic **i** and **j** without the superposed dot, tapping the **i** and **j** keys will generate a compound code. Coding these letters without their customary dots is not only much more general—accommodating the full Turkish alphabet and allowing accenting in a more general way—but also provides a range of useful ligatures such as those needed for some standard phonetic symbols.

Being able to mix roman and italic forms in plain text without recourse to markup will benefit expressiveness greatly. Better still, it can be easily introduced on any keyboard by providing an extra shift key.

Keyboards should also allow for the keying in of modifiers and combiners in their own right and not just to generate ad hoc symbols. English would benefit particularly from the reintroduction of accents on borrowed words to preserve distinctive pronunciations. Not only could words like *café*, *résumé*, and *zoölogy* be confidently spoken, but foreign names as well. For example, in addition to fostering better pronunciation, adding tone marks to Chinese names would also show some long-overdue cultural respect and sensitivity.

Existing keyboards could be driven to provide this generality, but adoption of the kind of coding system I've described would lead to keyboards and drivers that let the system's full capabilities be evoked simply. Among these capabilities would be alternative codings for some symbols, for example by reflection of a symmetric symbol. In contrast to Unicode, these codings would be obvious given the small codespace. At the font level, such alternatives would allow fine distinctions. Notice that, if Greek and Cyrillic fonts use the same coding scheme as Latin fonts, letter forms symmetric in one family may not be symmetric in another.

Clearly, Unicode is far too complex. Partly, this stems from its attempt to accommodate all the world's languages and their writing systems in one gigantic codespace. Further, Unicode does not take full advantage of the systematic graphical features of the various writing systems.

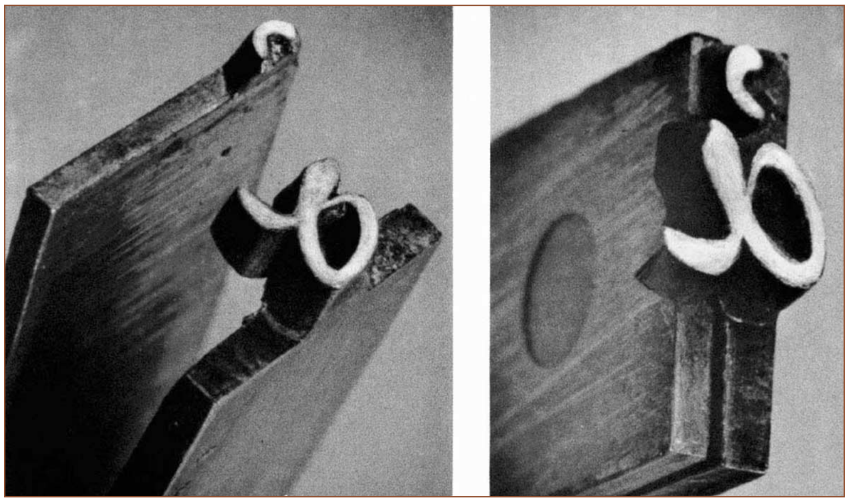

Figure 1. A piece of Greek type—the vowel alpha—kerned for use with separate accents, shown here in combination with a smooth breathing. [From* A New Introduction to Bibliography*, Philip Gaskell, 1972. Reprinted by permission of Oxford University Press.]

In contrast, using graphical synthesis, the alternative that I propose could accommodate the entire Latin writing system within an eight-bit codespace.

In the unlikely event the profession accepts my argument unanimously, replacing Unicode with separate standards for each writing system would still be quite impractical, particularly as Unicode is still in the process of replacing older systems. However, if even a majority of computing professionals consider my argument merely persuasive, an orderly transition to separate writing system encodings could be made possible and not mandatory by including them within Unicode as subsets alongside existing subsets, giving us the best of both worlds. ■

Neville Holmes is an honorary research associate at the University of Tasmania's School of Computing. Contact him at neville.holmes@utas.edu.au. Details of citations in this essay, links to further material, and possibly further discussion of issues are at www.comp.utas.edu.au/users/nholmes/prfsn.