

Sortemic Text Encoding

W Neville Holmes
University of Tasmania

0 Introduction

In a recent *Computer* column¹ I deplored the continuing use in the telecommunications and computing industry of the ASCII and EBCDIC characters sets, and condemned the incipient use of the Unicode scheme for text encoding, suggesting that something simpler was required to support, for example, simple interlingual text equipment applied to tasks such as e-mail.

In the following I describe and advocate a basis for uniform text encoding systems, and possibly only one is needed, which will support digital use of all the world's writing systems both for encoding of text by people, and for practical digital interlingual storage and exchange of text, such as is sorely needed for Internet. (Consideration of the last diagram included in this article will give the reader a preview of how such a system would be used.)

1 Standard Character Sets

The history of the development of standard character sets is rich and fascinating. The various aspects and benefits of the adoption of standard printing and publishing practices over the last half millenium or so in Europe² reflect the growth of modern society there.

As far as character sets are concerned, the roman style, and its associated italic, emerged in the sixteenth century from a morass of regional styles largely because of international trade in books in the Latin language, and the roman and italic styles both embody what is now called the Latin alphabet. Italic was favored early for its economy of space, and its use for quotations embedded within roman seems to have begun around 1510 in Basle. However, the *Umlaut* and *Schwabach* gothic alphabets, developed also in the sixteenth century, were widely adopted in the germanic world and survived well into the twentieth century. Fifty years ago, learning to read German in an Australian school meant learning another alphabet.

1.1 Coded Character Sets

Character sets have long been coded for transmission of data other than in handwriting or print. By the early nineteenth century there were two major data networks in Europe based on digital signalling by semaphore, one

based on shutters, the other on wigwags³. Napoleon's wigwags connected Italy and Belgium with France, and this system was later extended to Spain and the Netherlands. Similar systems became widely used throughout the world.

Morse code was developed a century ago, and displaced a variety of other codes for textual electrical and wireless telegraphy. It has only recently been dropped from official international wireless use. Morse coding was digital in the sense that it relied on distinguishing long pulses from short, and long spaces from short.

At about the same time that Morse code was developed, another digital code was adopted for the storage of data as holes in stiff paper cards, and later in paper tape. Early punched card codes used ten hole places (effectively binary digits, or *bits*) per decimal digit, then twelve bits per character, though there were many later variations on this. When digital computers came to be used to process these cards, a six-bit character set called BCD (Binary Coded Decimal) came into wide use⁴.

When punched paper tape became popular for electric telegraphy, an early international standard adopted in 1931, and long and widely used, was CCITT#2. This was an 5-bit code, for speed of transmission, and used a special coding to switch between numeric characters ("figure shift") and alphabetic characters ("letter shift").

The economies of the time, and the technology, dictated that both BCD and CCITT#2, and their like, used only a single case of alphabet and also used some of their codes to control machinery. BCD character sets of the 1950s typically coded for 48 graphic characters and extra coding provided within the 64 possible codes were used for instance to control magnetic tape. CCITT#2 nominated codings for answer back, to feed paper, to move the print-head of the typical teletypewriter, and to ring its bell!

In the late 1950s the international community moved to a new 7-bit standard code, now called ASCII (American Standard Code for Information Interchange), then primarily for the communications industry (AT&T officially requested and got an all-zeroes *null* and an all-ones *delete*), but later widely adopted for digital computing machinery. This standard provided many more telecommunications control characters, codes for both upper and lower case letters of the Latin alphabet, as well as more special characters.

At about the time ASCII was being agreed, IBM announced its move into 8-bit computers with the System/360, and took an each-way bet on character sets by providing an EBCDIC/USASCII control bit in the central PSW (Program Status Word) so that users could opt for either the USASCII (at the time an official alternative name to ASCII) or the EBCDIC (Extended BCD Interchange Code) character set. (sidebar here?) EBCDIC was an 8-bit code designed to be largely compatible with earlier BCD machinery, but it also included more control characters, upper and lower case alphabet, and more special characters. In the event, IBM's users all opted for EBCDIC, and many are still using it on their mainframe computers to this day.

However, the disadvantages of a variety of coexisting 7-bit ASCII and 8-bit EBCDIC systems was so obvious that great effort has been put into superceding them, most notably by the Unicode Consortium.

1.2 Unicode

The Unicode standard is an admirably supported and impressively extensive effort. "Unicode provides a consistent coding format for international character sets and brings order to a chaotic state of affairs that has made it difficult to exchange text files across language borders."⁵ Unicode is a 16-bit subset of a Universal Character Set (ISO/IEC 10646) which is a 32-bit standard.

Unicode separately "defines codes for characters used in every major language used today", as well as "punctuation marks, diacritics, mathematical symbols, technical symbols, arrows, dingbats, etc.". It also encourages idiosyncratic use by reserving 6,000 codes to be permanently without standard significance.

Unicode makes a particular point of incorporating any relevant prior character set standard, particularly ASCII. One infers that the included ASCII includes the notorious control characters, though the Technical Introduction⁵ notes that the ASCII CR and LF characters "are often used ambiguously", and reports that Unicode "eliminates that ambiguity." Unicode offers precomposed characters (for example, diacritically marked letters) for compatibility with prior standards, but also provides at least some of those characters as "composed character sequence[s]".

Unicode assigns names to individual bit codes that represent characters, but does not define "glyph images", a task it leaves to the "software or hardware-rendering engine of a computer". However it does pay some attention to the complexities of presentation. For example it provides for switching between left-to-right and right-to-left presentation, and presumably or potentially top-to-bottom for Mongolian and bottom-to-top for Batak⁶. And there are "glyphs and ligatures encoded as characters for font compatibility reasons."⁷

In brief, Unicode seeks to represent separately, within one 16-bit code space, all the letters, all the logograms,

and all the special textual symbols, in current and historical use throughout the world.

1.3 Multicode

Multicode was proposed by Muhammad Mudawwar⁷ to overcome drawbacks that he saw Unicode as introducing. The problems he sought to counter with Multicode were inefficiency (the 16-bit overkill), specific language programming (different languages might use the same character different ways), language mixing (poor encoding for specific languages), and incompatibility with ASCII (having to prefix each ASCII code with a null byte).

"Multicode attempts to define a unique character set for each written language, rather than unifying characters across languages as Unicode does." It proposes a set of character sets, most for specific languages, but some for mathematical and graphical symbols. A *switch character* embedded in the text changes character sets in flight, much in the manner of the shift characters of CCITT#2, though with the ability even to shift between 7-bit, 8-bit and 16-bit character sets.

1.4 Discussion

The ASCII and EBCDIC character sets were too specifically oriented, and too technically constrained, to remain in satisfactory use for long. Their character sets were quite inadequate, both of them, for example lacking the symbols used for multiplication and division used in the primary schools of much of the world! Their use of control characters was obsolescent from the start, and relevant mainly to bisynchronous digital transmission protocols. The only wonder is that both have been tolerated for so long particularly when so much ad-hockery was needed to shoehorn other languages into them⁴.

Unicode seems to be an overreaction to the chaos flowing from the inadequacies of ASCII and EBCDIC. It has the flavor of not being sure whether it is a system which people use to encode text, or which machines use to transmit or display text. People almost always encode text within one particular writing system, and for many writing systems a 16-bit character set is grossly inefficient. Unicode acknowledges that the fine details of presentation are outside its scope, but unfortunately assumes that to be able to switch between writing systems is of crucial importance in everyday text encoding.

Multicode seems reasonably to react to some of the difficulties of Unicode, but the reaction is misdirected in trying to switch between languages rather than between writing systems. Languages which share writing systems should also be able to share or borrow words from each other (as English has done so frequently and so successfully), and to share equipment and techniques specific to their shared writing system.

The rest of this article therefore outlines a text encoding system *designed for people using specific writing systems*. An essential aim for this system is to support different writing systems in a consistent way so that text processing can be as independent as possible of whatever writing system happens to be in use.

2 Encoding Text

What is needed to encode text well?

The first point to be made is that using machinery to deal with text in fact compounds three distinct tasks—text encoding, text formatting or composition, and text presentation. When manuscript factories were all there was in the way of publishing these three tasks were usually merged, though often calligraphy was separated from illumination. When movable type came into use in Europe the distinction between typesetting, composition, and physical printing was obvious, and were separated into at least two trades. With typewriters, the three tasks merged back into one.

Early commercial data processing separated the three tasks again, insofar as commercial documents are regarded as text. Typically, text such as names and addresses was encoded into cards by keypunch operators, and the encoded text was composed by various operations until it was finally printed under control of a plugged panel program which also did some final formatting. More general text processing was done by using computers as “text processors” on time-sharing systems in the ’70s, and as “word processors” on personal computers in the ’80s. These systems completely separated the printing from the encoding and composition. The text processors also distinguished the text proper from the instructions for text composition, which were couched in “mark up” code. Unfortunately, the so-called WYSIWYG word processors backed off the explicit mark-up coding and became more WYSINWYW—what you see is not what you want. It is hard to understand why systems that combine the best of both the explicit mark-up and the WYSIWYG worlds⁸ have not become popular.

Nowadays it is commonplace to mark text up for printing or display to include illustrations as graphs, pictures, movie clips, and hypertext links. The marking up also controls textual display aspects such as font, font size, text orientation, and page layout, as well as indexing and labelling. What can be done by mark-up needs to be extensible, and is so complex that to encode it specially within the character set would be wildly inappropriate. But marking up is not text encoding, though marking up is done by special encoded text.

The second point to be made is that text encoding is not just about character sets. Or even just about character sets and their binary representation. Text encoding is also about the machinery and conventions that allow people to

encode text, and to mark it up. However, text encoding is not about the embedding of control characters à la ASCII and EBCDIC to operate the equipment that might eventually display the text.

Decent text encoding, text encoding for everyday use, must therefore focus on the text encoding process and its associated machinery, and play down text composition and display.

2.1 Character Sets

Character sets are the means by which encoded text is made visible. But encoded text is not read as characters, it is read as words or phrases, particularly when the writing system used provides for word boundaries to be encoded, usually as gaps.

The words or phrases of encoded text are expressed in a particular language, and this is why Multicode proposes a distinct character set for each language. It is also why ASCII and EBCDIC were so grotesquely adapted to different languages even though many of those languages used basically the same writing system.

There are two problems with the Multicode approach, at least in its distinguishing of languages rather than of writing systems.

Firstly, many different languages use the same writing system—the Latin alphabet, the Cyrillic alphabet, and the Arab writing system are each widely used for many different languages. To have a different character set for every different language, and presumably for at least some dialects, is to be unduly conservative. And some languages, like Urdu/Hindi and Serbo-Croatian, use different writing systems in different places.

Secondly, words do not respect language boundaries, particularly within writing systems, witness *Mêlée*, *Schwärmerei*, and *Épée* in my Shorter Oxford English Dictionary. In any case it is an important mark of respect to reproduce personal names exactly across languages at least within the same writing system—in the Latin writing system this typically means full use of diacritical markings.

Now that the Latin alphabet with its distinct roman/italic styles is so widely used, it makes *every* kind of sense for cultures that use that writing system to be able freely to use it between cultures. No longer should we in Australia for example be unable properly to read the names and addresses of say German people who send e-mail to us. After all, the variations on the Latin alphabet are very simple, almost entirely confined to use of diacritical marks and, as in the case of Icelandic, a very few distinct extra letters.

2.2 Terminology

(This section, and its references, could usefully be put into a side-bar)

One of the problems with discussion of character sets is the diffuse meaning of the word *character* itself. For example, characters are often contrasted to punctuation marks. And is the character *G* the same character as *g*? To allow discussion of writing systems to be more formal, the science of linguistics has defined the term *grapheme* as the smallest written unit that can cause a contrast in meaning, by analogy with the term *phoneme*, which is the smallest spoken unit that can cause a contrast in meaning⁹.

The conveying of a contrast in meaning is determined by the discovery of minimal pairs of words. Thus the minimal pair of words *ban* and *pan* have distinct meanings, but differ in speech only in their /b/ and /p/ sounds, which sounds are therefore distinct phonemes. They also differ in writing only in their *b* and *p* letters which are therefore the distinct graphemes <p> and .

But in discussing character sets, at least in respect of ordinary use of the Latin alphabet, the term *grapheme* does not make the needed distinctions.

In the first place, The science of graphology considers *g* and *G* to be allographs, that is, different versions of the same grapheme. However upper case letters have long been used in Latin alphabet writings to mark differences in meaning, differences sometimes only conveyed in speech by subtle inflections. Most such writings mark names of places and people, and German marks all of its nouns, with upper case initial letters. Such frequent use mandates including both upper and lower case letters in character sets.

On the other hand, graphology considers *e* and *é* and *è* to be distinct graphemes, whereas it is necessary in character sets to consider *é* and *è* as having two components, one of which is an *e*, otherwise a huge number of special characters needs to be provided for the various usages of diacritical marks by different languages using the Latin alphabet.

It is also very useful to be able to encode characters of different alphabets the same way, to use binary encodings concurrently for different alphabets. Thus one encoding can stand for a particular letter in each of several alphabets. It is vital therefore to have a term which refers to a specific underlying encoding independently of whatever graphic symbol one writing system or another might equate to it.

For such use I derive the term *sorteme* from the word *sort* in its particular typographical use to describe the different letters or symbols available in a font of type. In old-time printeries, each sort had a standard place within its case (at least within any country)² much as each key on a typewriter keyboard has its standard place, and for the same reason. A *sorteme*, then, is a distinctive component of a text encoding system of the kind proposed here, basically one of the distinct codes. A *sort set* is consequently the set of graphic symbols used by a writing system to stand for the full set of sortemes used by the underlying

text encoding system.

2.3 Encoding Systems

A text encoding system should make it as easy and effective as possible for users to encode text. Typewriters provided an early though restricted text encoding system that could be used by relatively unskilled people. Personal computers, in their rôle as word processing systems, have almost wiped out the typewriter industry, but their text encoding capabilities have been shackled to the ASCII character set, a character set originally designed for improved textual telecommunications with a bare-bones Latin alphabet.

What is needed by the world-wide computer-using community is a text encoding system that allows for different writing systems, and allows for the richness of any particular writing system to be fully exploited both within and across any languages using that system. Such a text encoding system should encourage the cooperation of languages within any writing system by providing a full and rich sort set, and promote the movement of languages across writing systems by mapping sort sets compatibly onto the binary encoding which are the sortemes.

Such a text encoding system would allow development of text input machinery and systems that would be both versatile and adaptable, rather than tied to a particular language. It would also give particular support to automatic transliteration and translation of text.

2.4 Mark-up and Display

The issues in the design of text display systems are quite distinct from those for text encoding systems. The possibilities for display of text are so multifarious and extensible that they can only be expressed practically outside a specific character set through some kind of mark-up language.

The non-textual components of such display systems must be supported by non-textual standards, but the mark-up would be done in whatever writing system is in use. The mark-up would specify the inclusion and shaping of both textual and non-textual components, as well as the switching from one writing system to another. All of these considerations are external to any underlying text encoding system.

Given the likelihood if not certainty that any modern display system would be required to present any textual or non-textual data extracted from the Internet, appropriate display capabilities would be required. Many display systems, however, would need to double as monitors for text encoding systems. The capabilities in this case would need to be more specific, and would be relevant to the text encoding system used. Monitoring, though normally coëxistent with text display, will be discussed in the following as though it were independent of text display.

3 Sortemic Text Encoding

This proposal is for a family of text encoding systems that embrace different writing systems in as general a way as possible. Ideally, any text should be encodable and displayable on any equipment supporting the proposal, and text processing programs should be quite independent of any particular writing system.

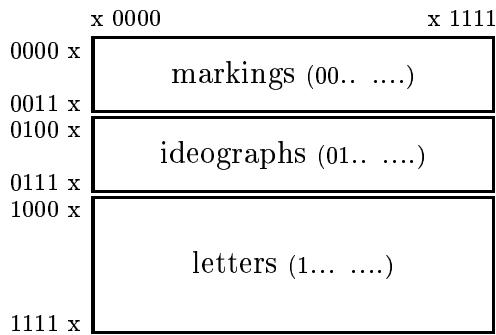
3.1 Writing Systems

Writing systems need different classes of sortemes. For example, some sortemes might be used alone or in sequence to compose the component words; other distinct sortemes will be used secondarily to mark or separate words or phrases or sentences.

In a sortemic text encoding system, the sortemes are the basic elements of the code, and any text processing depends on the classification of the sortemes into textually relevant groups such as lexical, punctuating, and numeric. This proposal seeks to constrain the functionally congruent character classes of different writing systems to the same sortemes. Punctuation marks in one writing systems should use the same sortemes as punctuation marks in another.

3.2 An Eight-bit System

The diagram shows an 8-bit sorteme scheme for a text encoding system suitable for the major alphabetic and syllabic writing systems. The suggested bit pattern allocations show that the leading bit suffices to distinguish letter sortemes from other sortemes, and a second bit to distinguish marking sortemes from ideographic sortemes.



The 64 *markings* are all the sortemes, and only the sortemes, that mark variations on letters and ideographs, or, at a higher level, words, phrases, and other lexical structures.

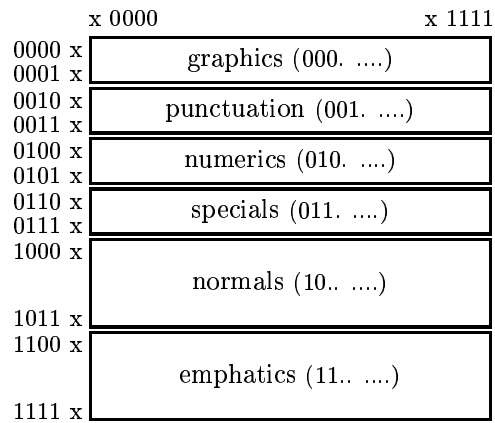
The 64 *ideographs* are all the sortemes, and only the sortemes, that have independent significance by themselves as words. In the Latin alphabetic writing system these could range from decimal digits to the asterisk and the ampersand.

The 128 *letters* are all the sortemes, and only the sortemes, that are the alphabetic or syllabic constituents of words in any writing system using this scheme.

These distinctions are sufficient for basic text processing (and for distinguishing names in networking and operating systems), and possibly this system would serve for all writing systems that can be accommodated within an 8-bit scheme. But another level of classification is required, at least for typical alphabetic and syllabic writing systems.

3.3 A Six-class Subsystem

In the next level of classification each of the groups in the system above is split into halves, in other words, one discriminating bit further in each case. This provides six classes of sortemes on which basis much more detailed text processing may be carried out.



The 64 *normal* letters are the letters used in ordinary text, and these would be the roman letters under the Latin, hiragana under the Japanese, and naskh style under the Arab writing system. The 64 *emphatic* letters are the corresponding letters used for emphasis or quotation, and these would be the italic letters under Latin, katakana under Japanese, and diwani under the Arab writing system.

The 32 *special* ideographs are the stand-alone sortemes that are abbreviations for words or ideas. The specials do not combine with letters, and would be parsed by a text analyser each as a separate word. In the Latin writing system two kinds would be needed, the conventional ideographs like & and * and the arithmetical ideographs like = and ÷. The 32 *numeric* ideographs are the sortemes used to construct numbers, and would combine under text analysis to parse as numbers. Two distinct sets of decimal digits would be useful to have, for example so that the exact or certain part of a number could be distinguished from the inexact or uncertain part. Various signs, such as for fraction parts, denominator parts, negative numbers and monetary amounts, are needed.

The 32 *punctuation* markings are the sortemes used

mark, enclose, or separate sorteme clusters, particularly clusters of letters. For writing systems that use them, and most do, these are basically reading aids, and so are parsed as relatively unimportant text components or else are ignored. A simple group of sortemes might be included here to allow tables and boxing to be encoded. The 32 *graphics* are the sortemes used to vary systematically details of the sort or character of a preceding (as in Unicode) sorteme, for example by applying a diacritical mark. In parsing, such sortemes do not have an independent significance but would usually bind to the sortemes they modify and take on their significance. Graphic sortemes are used to adapt writing systems, especially the Latin one, to different languages.

It is convenient at this point to distinguish *sorts* from *characters*. A writing system will have its characteristic sorts (its sort set), one for each and every sorteme. The writing system's character set will comprise the sorts together with all the characters that can be composed by applying graphic sortemes.

Some writing systems also involve a variation in shape or arrangement of characters depending on context, and because it is defined by context, such variation does not have to be encoded in the sortemes. For example, many sorts of the Arab writing system have four character variants—isolated, initial, medial, and final.

3.4 Monitoring Text

A good text encoding system would make it easy for people to encode text. Whatever encoding equipment might be used, the encoding process requires the feeding back of what is being encoded so that continuity can be maintained and corrections made. This feedback should be explicit and complete.

Supposing a keyboard of some kind is used. Each keystroke should be echoed as its sort, or its separate sorts where a key stands for a composed character. At the same time the user should have an option to see the effect of all graphic markings and any automatic ligatures or contextual morphic changes of the kind that the Arab and Korean writing systems abound in. In other words, anyone encoding text should be able to see either the sorts relating directly to the sortemes, or the characters produced from the sorts by the writing system, if only to make correction and editing easy. This implies the need for monitoring software or hardware that provides a display of the sorts equispaced like `this font` so that small sorts aren't overlooked, and, concurrently or alternately, a display of the characters of the writing system complete with contextual variations.

3.5 The Latin System

(This specific description for the Latin writing would probably be better placed in a sidebar, or in an Ap-

pendix.)

An outline of how the Latin writing system would fit under the 8-bit text encoding system outlined above is given here as an illustration, using the writing system best known to the author. Similar outlines for other writing systems would more appropriately be contributed by people who customarily use them.

The letter sorts would need to be expanded to four alphabets of thirty two corresponding sorts, effectively defining thirty two graphemes. There are various contenders for the extra grapheme, among which the dotless *i* and *j* should figure prominently, both to provide a clean sort for taking diacritical marks, and to accommodate languages like Turkish that use *i* and *ı* as separate graphemes.

The 32 special ideographic sorts should be split equally between conventional sorts like & and ← and arithmetic sorts like + and ×. The provision may seem frugal, particularly for the arithmetics, but the graphic sortemes will allow vastly more ideographs to be composed. By contrast, providing 32 numeric ideographs (with two distinct sets of decimal digits) might seem particularly generous, but it is justified by providing a clean and simple 5-bit subset that would be effective for purely numeric applications.

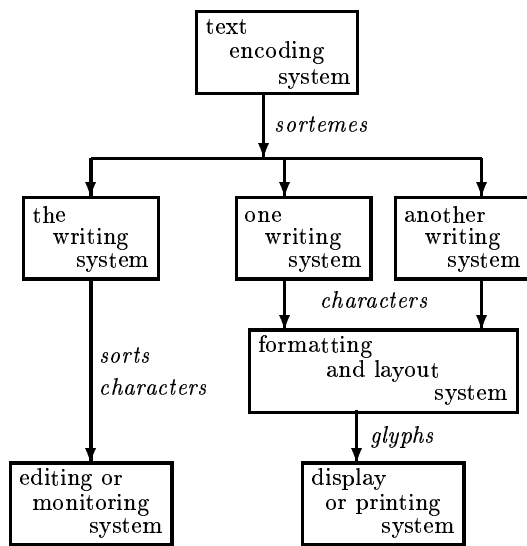
The punctuation sortemes are relatively few, considering that it is in this subclass of sorts that most inter-language variation is found. Some cultural compromise would be necessary here, though the need would be lessened if only say the closing versions of lunules), brackets], braces }, diples >, and any other enclosure sortemes, were provided. Then the graphic sorteme that reflects its preceding character about a vertical axis would compose the opening version of the sort. A doubling graphic sorteme could also be used to double a diple to get a guillemet « or », or a quote to get a double quote. A few spacing sortemes would be needed, and it should be noted that these would have visible sorts which convert to the appropriate level of gaps when they are converted to characters.

Two main groups of graphic sortemes would be needed, one group simply applying to an immediately preceding character to produce a modification such as a diacritical mark as in *Á* and *Ē* or a reflection as in *∇* and *∃*, and one group to apply to two preceding characters to produce compositions such as *œ* and *ø*. The diacritical marks would need to be able to stack, in particular to accommodate Vietnamese, which uses diacritical marks both to produce distinct characters and to mark tonal features. An all-zeroes *null* could be very useful, and also a cancellation that allows an immediately preceding graphic or other sort (all sortemes must have a visible sort) to become a character in its own right.

4 Discussion

The following diagram shows how a text encoding system of the kind advocated here might allow *sortemes* to be processed by programs that would either respect one writing system, or combine more than one.

The writing systems, unlike the other systems shown, remain purely notional, defining *sorts* when the graphic *sortemes* are not applied, and *characters* when the graphic *sortemes* are applied and the writing system's morphic changes such as ligatures (characteristic of the Arab writing system) and rearrangements (characteristic of Hangül) and proportional spacing are applied to the *sorts*. So while the diagram shows where *sortemes*, *sorts*, *characters*, and *glyphs* become relevant, the conversion from *sortemes* to *sorts* to *characters* is carried out within the systems the *sorts* and *characters* are shown feeding into. (The term *glyph* is nowadays being used to refer to a character after it has had a type font and size applied to it.)



4.1 How Many Text Encoding Systems?

For text encoding, how many systems are required for the world's many writing systems?

This is not the same thing as asking how many there will be, since making decisions about the adoption of such standards is more a social and political manoeuvre than a technical one. This second question can have no answer until the owners/users of all those writing systems have decided whether the advantages of sharing a text encoding system outweigh the benefits of developing a different one.

Two subquestions are worth considering. *Firstly*, how many 8-bit text encoding systems are needed, and, *secondly*, how many non-8-bit ones?

The answer to the first subquestion depends on how many writing systems fit into either the six subclasses described above, or, failing that, into the three classes,

where by *fit* is meant both that there are enough *sortemes*, and that the classification is appropriate.

Writing systems that have fewer than 65 normal letter *sortemes* have at least some prospect of fitting into the 6-subclass system described above with the opportunity of including the corresponding *emphatics*.

The problem here is what a normal letter *sorteme* is considered to be. For the Latin alphabet, and any other writing system with two cases of letters and fewer than 33 letters, the two cases can both be included in the normal letter *sorteme* class. The Cyrillic alphabet has two cases but unfortunately doesn't come into this category because, although Russian only uses 32 letters, other languages use more. So the Cyrillic writing system must, if it is to use the proposed text encoding system, put its lower case letters into the normal *sorteme* class, and its upper case into the *emphatics*. Otherwise Cyrillic would need either a completely different text encoding system or some kind of reform agreed to by all cultures using Cyrillic.

The answer to the *second* subquestion embraces much more complex issues. The most obvious candidate for a text encoding system of more than eight bits is the Chinese system of what are sometimes called ideographs, sometimes pictographs, sometimes logographs, popularly *characters*, and, more recently and descriptively, *sinograms*. This system is used in its own right for various Chinese languages, and as an auxiliary and prestigious system most prominently for Japanese and, in South Korea, for Korean. Obviously, the use of a mixed system for Japanese and Korean presents problems which are beyond the scope of this article.

The Chinese writing system is perhaps the best in the world for reading, as it takes up less space for the same amount of text and can be read significantly faster than, say, text presented in the Latin alphabet. Since it encodes meanings more than it encodes sounds, it has often been advocated as a global written language, though there are many arguments against this, most amusingly presented by John DeFrancis¹⁰. However, it is not easy to learn and it is far from easy to encode with.

It seems practical, though not necessarily politic, to encode Japanese entirely using their syllabaries, and Korean their alphabet, and to treat the conversion of *sortemes* to *sinograms* as defined by the writing system and not by the text encoding system. The *sortemes* might well in this case fit within the 6-class subsystem. It's significant, therefore, that an 8-bit text encoding system has been introduced in South Korea within the last few years in competition with a variety of previous 16-bit systems, and has been well received¹¹.

As far as the Chinese use of *sinograms* is concerned, it is interesting that romanization and cyrillicization have long been proposed, and that a Latin alphabet system called Pīnyīn has been officially adopted to represent the official spoken language, Pūtōnghuà. Pīnyīn is not a replacement for the *sinograms*, but a supplement to them. That sino-

grams could be encoded using Pīnyīn, either by themselves or supplemented by radical marking, is attested by articles in the sadly isolated special issue of IEEE Computer devoted to Chinese/Kanji text processing¹².

4.2 Text Encoding Devices

The physical devices, particularly the keyboards, that have traditionally been used to encode text are often designed within a particular language area, and have been adapted to suit one language or combination of languages. Since there is no necessity for one keystroke to produce only one sorteme, these keyboards should be able to be adapted easily to the kind of text encoding system under discussion.

Indeed, now that Latin alphabet keyboards have been extended for use with personal computers to have what are effectively three shift keys, the case shift from the typewriter tradition and the *Alt* and *Ctrl* keys, one or both of the last two could be adopted to shift into the emphatic part of the sorteme letter set.

But much of the benefit of the kind of text encoding systems proposed here would come from new devices and techniques. The sortemes are bit-encoded, and this suggests chording keyboards of the kind found effective in stenography and music. This technique would map fingers onto bit keys, and could be used with data gloves as well as with keyboards. For an 8-bit text encoding system, each finger could be allocated to its specific bit key, and could be used without the kind of lateral movement needed for conventional keyboards. A thumb could be allocated an extra key used to invert the bits of the fingers of its hand. One way of using such a keyboard would be to specify that both makes and breaks of every key would be used, with bits read out only when and just as one key switch is breaking after a sequence of makes. Such a scheme would allow whole hand movements for beginners, but would also allow very high speed keying to be achieved as, for example, what would effectively be shift keys (the bits designating classes or subclasses) could be held down to produce ones over a sequence of sortemes.

The sortemic bit encoding could also be exploited to allow the kind of shorthand needed to make the use of a stylus on a tablet practical. Sortemes could be encoded by a sequence of simple strokes. The six low-order bits could be encoded by gently curved strokes in eight different directions (3 of the bits represented by NNE ENE ESE SSE SSW WSW WNW and NNW), of either increasing curvature (hump at end) or decreasing (hump at beginning, 1 bit), above or below the straight line direction (1 bit), and either short or long (the sixth bit). Two shift bits could be encoded by the tablet quadrant in which a continuous sequence of strokes started. But there are many other possibilities, and extensions, all of which couldn't fail to be more effective than ordinary handwriting recognition, and would be independent of any particular prior writing

system.

4.3 Sequencing

Much attention, and compromise, has been given in character set development to the maintenance of the sequence of letters to conform to those in prior character sets. The main justification for this is to maintain so-called alphabetical order, partly because it's familiar, partly because that's the way dictionaries for alphabetically and syllabically written languages keep their words, and partly because it would make codings easier to figure out should you need to do so.

But dictionaries and word indexes are not kept in strict alphabetical order, at least not the alphabetical order of strict code sequencing. For example, the words *cat Cat dog Dog* would be sequenced *cat dog Cat Dog* in EBCDIC and *Cat Dog cat dog* in ASCII. So any lexical sequencing algorithm must depart from canonical sorteme sequencing, and must be applied beyond the text encoding process. This leaves it open to define a canonical lexical sequencing algorithm for a writing system, and to allow individual languages using the system to have their own sequencing algorithms.

4.4 Internet Names

There is one class of text that, for historical reasons, is much more constrained than normal text, and that is the encoding in the names for computers and files of data on the Internet. Because the present international data network known as Internet was pioneered in and long dominated by the English language, the English language usage of the Latin alphabetic writing system was used for these names for network entities.

As a result such names are typically case insensitive. If text encoding were reformed, then maybe Internet naming could also be reformed. In any case to have only one text encoding system for Internet-wide naming would seem to be the only practical convention, but if the sortemic encoding proposed here were adopted, and the Latin writing system was kept as a basis, then case-insensitivity would need to be retained. This might lead to difficulties with other writing systems, and ways around these difficulties would be needed. Such ways would be more easily found under a single widely-used text encoding system.

5 Conclusion

The approach to text encoding advocated here seeks to support writing systems by the simplest possible binary codes (sortemes) within as few as possible different code classification schemes, ideally within a single such scheme. Different writing systems are then united by sharing the

classifications, but distinguished when sorts and characters are generated from the sortemes for display or formatting.

Such a text encoding system seeks to separate processes for the presentation or display of text from the process of encoding text. By focussing on the problems of text encoding, and by designing systems suited to that task based on various writing systems, the task of text encoding should be made easier and more effective than it could be either by persisting with traditional text encoding systems, or by struggling to adapt them to new and more widely used display or printer technology.

The approach to text encoding advocated here rejects the complex Unicode approach which aims to gather all writing systems under a single umbrella, and to deal with a number of display problems as well. It also rejects the Multicode approach which focusses on text encoding but separates language systems, rather than writing systems, unnecessarily hindering word-borrowing, cross-language quotation and equipment sharing.

The separation of text encoding from formatting, layout, and illustration, provides a stable and versatile base for the development of various kinds of mark-up systems, for much improved programming and command languages, for gradual but much-needed reform of arithmetical and mathematical notations, for spelling reform and orthographical innovation, and for standard equipment to exploit the benefits of such a standard text encoding system or systems.

References

1. W.N.Holmes, "Toward Decent Text Encoding," *Computer*, Aug. 1998, pp.108–109.
2. L.Febvre, H-J.Martin, *The Coming of the Book*, 1958, tr.D.Gerard, Verso, London, 1997.
3. G.J.Holzmann, B.Pehrson, "The First Data Networks," *Scientific American*, Jan. 1994, pp.112–117.
4. C.E.Mackenzie, *Coded Character Sets, History and Development*, Addison-Wesley, Reading, Massachusetts, 1980.
5. *The Unicode Standard: A Technical Introduction*, Unicode Consortium, <http://www.unicode.org/>, 1998 (unicode/standard/principles.html).
6. R.Woodard, "Writing Systems," *The Atlas of Languages*, Quarto, New York, 1996, pp.162–209.
7. M.F.Mudawwar, "Multicode: A Truly Multilingual Approach to Text Encoding," *Computer*, April 1997, pp.37–43 (see also, "Letters," *op.cit.*, June 1997, pp.6,9).
8. K.P.Brooks, "Lilac: A Two-View Document Editor," *Computer*, June 1991, pp.7–19.
9. D.Crystal, *The Cambridge Encyclopedia of Language*, Cambridge University Press, Cambridge, 1987.
10. J.F.DeFrancis, *The Chinese language: fact and fantasy*, University of Hawaii Press, Honolulu, 1984.
11. Y.J. Choi, *personal communication*.
12. Y. Chu (ed.), "Chinese/Kanji Text and Data Processing," *Computer*, Jan. 1985, pp.11–66.