Character Sets

J H Jenkins, Apple Computer, Inc., Cupertino, CA, USA

Everything is a number to a computer, and so, in order to represent text, it is necessary to create a numerical representation for it. The most common technique for doing so is to assign numbers to individual characters in a writing system; the term 'character set' is used to refer to such a mapping.

More specifically, everything is a binary number of a fixed size to a computer. A single binary digit is referred to as a bit. Because the binary representation of a number is difficult for humans to use, computer scientists usually use base 8 (octal) or base 16 (hexadecimal) numbers. In octal numerals, only the digits 0 through 7 are required; hexadecimal uses 0 through 9 and A through F as digits. When it is unclear which base is being used, octal numerals are written with a leading 0, and hexadecimal numerals with a leading 0x.

The smallest-size number a computer can readily manipulate is called a byte. Earlier systems used bytes that varied in size from computer to computer; today, an eight-bit byte or octet is standard. Bytes are the basis of data interchange between different computers.

We can therefore say that a character set is a mapping between a set of text elements such as characters to a series of bytes. Conceptually, this process can be broken down into stages (see **Figure 1**):

1. the abstract character repertoire, which is the collection of characters to be encoded;
2. the coded character set, which maps the elements of an abstract character repertoire to a set of non-negative integers;
3. the character encoding form, which maps the elements of a coded character set to code units of a specific size; and
4. the character encoding scheme, which maps code units from one or more character encoding forms to an actual sequence of bytes.

History

The first efforts to create numerical encodings for text date from the 19th century with the advent of telegraphy. As the technology advanced and developed, so did increasingly sophisticated techniques to coordinate the transmission of textual information via numerical or other codes and the technology used for transmission. This dovetailed with the pioneering work in computers to create the first computer character sets.

The most direct descendant of the telegraphy codes is EBCDIC (Extended Binary Coded Decimal Interchange Code), used in IBM mainframe computers from the early 1960s onwards. Its chief rival was the American Code for Standard Information Interchange (ASCII), developed at about the same time by what is now the American National Standards Institute (ANSI). ASCII has proven far more popular and influential; use of EBCDIC has been restricted largely to IBM mainframes and closely related systems.

ASCII represents characters using units of seven bits, which provides for a set of $2^7 = 128$ characters. The first 32 and last one are used for control codes – numbers that do not represent text but are used to give instructions to the machines displaying or printing it. (Character 0x08, for example, was frequently used to instruct a terminal to ring its bell.) The remaining 95 are the characters for display. In its current form, ASCII includes the upper- and lowercase letters of the Latin alphabet as used for English, the numerals 0 through 9, basic punctuation, and some accents. The accents were used for non-English languages. A French 'é' would be represented by instructing the terminal to type 'e', physically move backwards one place, and then type the accent.

ASCII is sufficient for use with standard modern English as typed on typewriters, but not for anything else. Nor is it actually sufficient even for English typesetting, as it is missing important symbols such as em and en dashes.

On systems based on the eight-bit byte, the use of ASCII seems wasteful, since that eighth bit is unused. Originally it was intended that the extra bit should be used as a parity bit as a redundancy check. By the 1980s, however, eight-bit extensions to ASCII had become common. The first 128 characters, 0x00 through 0x0F, were as in ASCII, and the remaining 128 were used for extensions such as dingbats or additional letters. Most notable of the eight-bit character sets are MacRoman, developed by Apple Computer for use on its Macintosh computers, and the ISO 8859 series of standards developed by the International Organization for Standardization.

Eight-bit character sets are inadequate for languages requiring more than 256 characters. Extensions were developed, therefore, for use in East Asia. The most common are referred to as double-byte character sets. In these character sets, the bytes 0x00 through 0x7F are used as ASCII. Bytes 0x80 through 0xFF are available to signal the start of

Abstract character repertoire:	A, B, C, D,…
Coded character set:	A → 65, B → 66, C → 67, D → 66,…
Character encoding form:	A → 0x0041, B → 0x0042, C → 0x0043, D → 0x0044,…
Character encoding scheme:	ABCD → 00 41 00 42 00 43 00 44, or ABCD → 41 00 42 00 43 00 44 00

Figure 1 Stages in the character-encoding process.

two-byte units. The Latin letter 'A' would therefore be represented by the single byte 0x41, and the Chinese character 井 by the two-byte sequence 0xA4AB. Such techniques provide enough room for the daily needs of modern East Asian languages.

By 1990, there were dozens of character sets in common use, with varying repertoires and architectures. This created problems for software companies, whose products were originally written with ASCII or one of its eight-bit extensions in mind. Getting software to work with multiple character sets was a long and expensive process. This led to the desire for a universal character set, adequate for the representation of all languages.

Two efforts to produce a universal character set converged in the early 1990s. One is known as ISO/IEC 10646, produced by the international standardization community. The other is Unicode, produced by the Unicode Consortium, an industry consortium consisting largely (but not entirely) of companies headquartered in the United States. The convergence of the two character sets means that the terms 'Unicode' and 'ISO/IEC 10646' can be used almost interchangeably in most circumstances. We will generally use the term 'Unicode', as Unicode is formally an implementation of ISO/IEC 10646 with additional specifications.

The existence of a universal character set has proven vital in the rise of the Internet. So long as documents are created and printed on a single computer, it doesn't make much difference which character set is used. When a user in Nome, Alaska is using his Windows system to access a Japanese Web page hosted by a computer in Calcutta running Linux, however, it becomes important to make sure that both systems can handle the same character set. And when that computer in Calcutta is used to host the records of a multinational corporation requiring support for English, French, Chinese, Japanese, Hindi, and Thai, it becomes even more important to avoid the accounting headache of dealing with multiple character sets.

By the turn of the 21st century, Unicode and ISO/IEC 10646 had become the focus of all efforts to extend the computer representation of human languages. They are also increasingly supported by modern operating systems and software.

Characteristics of Unicode

Unicode was originally based on a simple 16-bit architecture: there were $2^{16} = 65\,536$ code points available from 0x0000 through 0xFFFF. The first 128 code points are identical with ASCII; indeed, the first 256 code points are identical with the popular eight-bit standard, ISO 8859-1. The remaining space included characters of well over a dozen other scripts ranging from Greek to Thai. Numerals and punctuation marks were included, together with mathematical symbols and popular dingbats.

It soon became clear that this was insufficient room for the full set of characters people actually wanted to use. Sixteen additional planes each of 65 536 code points each were added and multiple encoding forms created, as described below, to accommodate them.

The number assigned a character is referred to as its Unicode scalar value (USV), and is usually written as 'U+' followed by four or more hexadecimal digits. Each character is also assigned a name. The Latin letter 'A' has U+0041 for its USV and is named LATIN CAPITAL LETTER A. In addition to standalone characters, Unicode includes a number of combining marks. These are included to allow support for on-the-fly generation of new accented forms. This makes it unnecessary to actually catalogue every accented Latin letter, for example, before it can be represent on computers.

Several thousand code points are reserved for private use; that is, the standard does not define what they are to represent but leaves that to private agreements between different users. This allows users to interchange data using unencoded characters or characters inappropriate for encoding, such as corporate logos.

One of the most controversial aspects of Unicode is its inclusion of a unified set of ideographs for all of East Asia: Chinese (both its simplified and traditional forms), Japanese, and Korean. (Unicode and ISO/IEC 10646 use the linguistically incorrect term 'ideograph' for historical reasons. The authors of both standards are aware that other terms better describe the function of these characters in the languages that use them.) The process of producing a single set of characters for East Asian languages is referred to as

Character	Chinese (PRC)	Chinese (Taiwan)	Japanese	Korean
U+4E95	井	井	井	井
U+753B	画	画	画	
U+76F4	直	直	直	直
U+8435	葛	葛	葛	葛

Figure 2 Han unification.

Han unification (see Figure 2). Because individual characters are often written in visually distinct ways in different parts of East Asia, there was some concern that this would force, for example, a Japanese user to see their name written with culturally inappropriate Chinese glyphs; and because the Unicode Consortium itself and the bulk of its officers are American, it was also felt that Americans were being insensitive to the needs of people in East Asia.

Both fears proved to be overstated. In point of fact, most Japanese users use systems with fonts designed specifically for Japanese and only rarely are confronted by Japanese text written using Chinese glyphs. Even where they are, the glyphs they see would generally be perfectly acceptable, or, at most, about as odd-looking as the spelling 'colour' is to an American or 'color' to a Briton. The actual work of Han unification, moreover, has been done by an international group. This group, now called the Ideographic Rapporteur Group or IRG, currently has delegations representing the People's Republic of China (PRC), the Special Administrative Regions of Hong Kong and Macao, Taiwan, North and South Korea, Japan, Vietnam, Singapore, and the United States, with a liaison representing the Unicode Consortium. The current head of the IRG is Dr Lu Qin of Hong Kong Polytechnic University, and its remaining officers are all from the PRC.

The issue of Han unification serves to illustrate a fundamental distinction made by Unicode and ISO/IEC 10646, that between character and glyph. A character is a unit of meaning in a writing system, while a glyph is what one actually sees on the page or on the screen. Thus 'a', 'b', and 'c' are different characters, while 'a' and '*a*' are different glyphs for the same character (see Figure 3).

The distinction is an important one. Characters are the fundamental units where meaning is concerned, and are what are used in processes such as searching, sorting, pattern matching, or text to speech. They are insufficient for more visually oriented processes such as printing or optical character recognition. Scripts such as Arabic, for example, make extensive use of contextual forms, where a letter changes its shape depending on its position in a word. If each of the different forms for each letter were separately encoded, not only would that make text input and

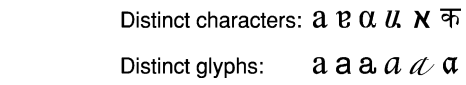

Figure 3 Characters and glyphs.

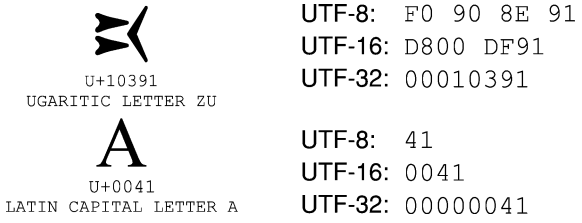

Figure 4 Unicode encoding forms.

spelling checking more complex, it would also lock the visual representation of Arabic into one calligraphic style. By separating the encoding of the characters from the drawing of the glyphs, it's possible for the same text to have substantially different appearances, depending on additional information such as font and point size.

Unicode is thus aimed at plain text rendering; its goal is minimal legibility – the minimum amount of information needed to guarantee legibility. Pure Unicode should be legible to users regardless of the details of the rendering process.

Unicode defines three character encoding forms, named after the size of the code unit they use: UTF-8, UTF-16, and UTF-32 (see Figure 4). ('UTF' can be taken to stand either for 'universal transformation format' or 'Unicode transformation format.') In UTF-8, each character is represented by one to four bytes. The 128 characters from ASCII are, in fact, represented by the same bytes as they are for ASCII itself. This is an enormous advantage for software originally written for ASCII or other character sets sharing its basic architecture. UTF-16 corresponds to the original definition of Unicode. Characters U+0000 through U+D7FF and U+E000 through U+FFFF are represented by one 16-bit unit. Characters above U+FFFF are represented by two 16-bit units, using a combination of two units in the ranges 0xD800 through 0xDFFF. (The scalar values U+D800 through U+DFFF are therefore not used for character encoding.) In UTF-32, each character is represented by its Unicode scalar value, padded to 32 bits.

Finally, Unicode defines five character-encoding schemes based on these three encoding forms. These are necessitated because computers vary in how they combine two bytes into one 16-bit unit.

The Future

Unicode and ISO/IEC 10646 can be considered to have rightfully won their title of **the** universal

character set. Although other character sets continue in use, all work on computer representation of text is now done either in Unicode or in a fashion compatible with it. All major operating systems are being written around Unicode, and most software in development supports it.

Major issues still remain. One is the distinction between character and glyph. Although in most cases, the line between the two is clear, in some it is not. Many of the ideographs being considered for encoding by the IRG, for example, are so obscure that even the experts are not sure which ones are distinct characters. In scripts that are yet to be fully deciphered, such as the Indus Valley script, there is the same problem.

Another issue is diachronicity. With a few exceptions, the scripts encoded in Unicode are modern scripts in current use, which means that the repertoires are relatively well defined, as are the ranges of acceptable glyphs for the characters. When an effort is made to represent a script in use over the course of centuries or millennia, things are not always clear cut. Oracle-bone Chinese is not fully understood, for example, and it's unclear how to coordinate it with modern Chinese.

And there is also the issue of scripts that do not follow the linear approach to layout common to all modern writing systems. Unicode has already expressly disavowed any intention of fully encoding music or mathematics because they are so two-dimensional in layout. Egyptian hieroglyphics, however, are a written representation of human language and should be encoded, although the details are far from clear.

But in the meantime, computer technology has advanced considerably in its ability to represent human languages, over the course of just the past decade. Prior to 1990, it would have been unthinkable to exchange data in virtually any human language virtually anywhere in the world. That, however, is now becoming a reality.

See also: Asia, Ancient Southwest: Scripts, Modern Semitic; China: Writing System; Digital Fonts and Typography; Japan: Writing System.

Bibliography

Gillam R (2003). *Unicode demystified: a practical programmer's guide to the encoding standard*. Boston: Addison-Wesley.

Graham T (2000). *Unicode: a primer*. Foster City, CA: M & T Books.

Lunde K (1999). *CJKV information processing*. Beijing: O'Reilly & Associates.

Character versus Content

C Spencer, Howard University, Washington, DC, USA

David Kaplan introduced the content/character distinction in his monograph *Demonstratives* (1989a) to distinguish between two aspects of the meaning of (1) indexical and demonstrative pronouns (e.g., 'I', 'here,' 'now,' 'this,' and 'that') and (2) sentences containing them. Roughly, the content of an occurrence of an indexical or demonstrative is the individual to which it refers, and its character is the rule that determines its referent as a function of context. Thus, an indexical has different contents in different contexts, but its character is the same in all contexts. For instance, the character of 'I' is the rule, or function, that maps a context of utterance to the speaker of that context. This function determines that the content of Sally's utterance of 'I' is Sally.

Content/Character Distinction and Semantics

Sentences containing indexicals or demonstratives are context-dependent in two ways. First, contexts help to determine what these sentences say. Second, contexts determine whether what is said is true or false. For instance, suppose Sally says, 'I'm cold now' at time t. The context supplies Sally as the referent for 'I' and time t as the referent for 'now,' so it helps to determine what Sally said. Other facts about the context, specifically whether Sally is cold at time t, determine whether she said something true or false. Different contexts can play these different roles, as they do when we ask whether what Sally said in one context would be true in a slightly different context. A central virtue of Kaplan's semantics is that it distinguishes between these two roles of context. For Kaplan, a *context of use* plays the first role, of