Character encoding

Victor Eijkhout

August 2004

Have you ever wondered what goes on between the 'A' you hit on your keyboard, the 'A' stored in your file, and the 'A' that comes out of your printer? Why that letter still comes out of the printer if the file is printed by your friend in Greece who doesn't use the letter 'A'? Maybe you know that 'A' is character 41 in ascii; if you put it on a web page, and it's watched by someone in Japan, why don't they get character 41 in the Kanji alphabet? Do you remember the DOS days when your Mac owning colleague would send you a file and what were supposed to be accented characters would turn into smiley faces? Have you ever pasted text from MS-Word into Emacs, and Emacs wanted to save the document as UTF-8? Just what is that about?

All this, and more, will be explained in this article.

1 History in one byte

Somewhere in the depths of prehistory, people in the western world got to agree on a standard for character codes under 127, ASCII, the American Standard Code for Information Interchange. This standard declares that the letter 'A' is character number 41, so if your file contains the bit pattern for 41 (which is `00101001`), it will produce an 'A' when sent to the printer.

ASCII has a few nice properties, some of which were not shared by another encoding scheme, EBCDIC (which was used almost exclusively by IBM):

- All letters are consecutive, making a test 'is this a letter' easy to perform.
- Uppercase and lowercase letters are at a distance of 32; this means that the Shift key on your keyboard simply toggles the sixth bit in the pattern of whatever key you are holding down.
- The first 31 codes, everything below the space character, as well as position 127, are 'unprintable', and can be used for such purposes as terminal cursor control.

The ISO 646 standard codified 7-bit ASCII, but it left certain character positions (or 'code points') open for national variation. For instance, British usage put a pound sign (£) in the position of the dollar. The ASCII character set was originally accepted as ANSI X3.4 in 1968.

dec CHAR hex oct

ASCII CONTROL CODES

BITS b7 b6 b5 / b4 b3 b2 b1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SYMBOLS NUMBERS		UPPERCASE		LOWERCASE	
0 0 0 0	0 NUL 0 0	16 DLE 10 20	32 SP 20 40	48 0 30 60	64 @ 40 100	80 P 50 120	96 ‘ 60 140	112 p 70 160
0 0 0 1	1 SOH 1 1	17 DC1 11 21	33 ! 21 41	49 1 31 61	65 A 41 101	81 Q 51 121	97 a 61 141	113 q 71 161
0 0 1 0	2 STX 2 2	18 DC2 12 22	34 ” 22 42	50 2 32 62	66 B 42 102	82 R 52 122	98 b 62 142	114 r 72 162
0 0 1 1	3 ETX 3 3	19 DC3 13 23	35 # 23 43	51 3 33 63	67 C 43 103	83 S 53 123	99 c 63 143	115 s 73 163
0 1 0 0	4 EOT 4 4	20 DC4 14 24	36 $ 24 44	52 4 34 64	68 D 44 104	84 T 54 124	100 d 64 144	116 t 74 164
0 1 0 1	5 ENQ 5 5	21 NAK 15 25	37 % 25 45	53 5 35 65	69 E 45 105	85 U 55 125	101 e 65 145	117 u 75 165
0 1 1 0	6 ACK 6 6	22 SYN 16 26	38 & 26 46	54 6 36 66	70 F 46 106	86 V 56 126	102 f 66 146	118 v 76 166
0 1 1 1	7 BEL 7 7	23 ETB 17 27	39 ’ 27 47	55 7 37 67	71 G 47 107	87 W 57 127	103 g 67 147	119 w 77 167
1 0 0 0	8 BS 8 10	24 CAN 18 30	40 (28 50	56 8 38 70	72 H 48 110	88 X 58 130	104 h 68 150	120 x 78 170
1 0 0 1	9 HT 9 11	25 EM 19 31	41) 29 51	57 9 39 71	73 I 49 111	89 Y 59 131	105 i 69 151	121 y 79 171
1 0 1 0	10 LF A 12	26 SUB 1A 32	42 * 2A 52	58 : 3A 72	74 J 4A 112	90 Z 5A 132	106 j 6A 152	122 z 7A 172
1 0 1 1	11 VT B 13	27 ESC 1B 33	43 + 2B 53	59 ; 3B 73	75 K 4B 113	91 [5B 133	107 k 6B 153	123 { 7B 173
1 1 0 0	12 FF C 14	28 FS 1C 34	44 , 2C 54	60 < 3C 74	76 L 4C 114	92 \ 5C 134	108 l 6C 154	124 | 7C 174
1 1 0 1	13 CR D 15	29 GS 1D 35	45 – 2D 55	61 = 3D 75	77 M 4D 115	93] 5D 135	109 m 6D 155	125 } 7D 175
1 1 1 0	14 SO E 16	30 RS 1E 36	46 . 2E 56	62 > 3E 76	78 N 4E 116	94 ˆ 5E 136	110 n 6E 156	126 ˜ 7E 176
1 1 1 1	15 SI F 17	31 US 1F 37	47 / 2F 57	63 ? 3F 77	79 O 4F 117	95 ˙ 5F 137	111 o 6F 157	127 DEL 7F 177

Table 1: The ASCII table

Since a computer organizes its bits in 8-bit bytes, and ASCII only codified the codes under 128, this left the codes with the high bit set (‘extended ASCII’) undefined, and different manufacturers of computer equipment came up with their own way of filling them in. These standards were called ‘code pages’, and IBM gave a standard numbering to them. For instance, code page 437 is the MS-DOS code page with accented characters for most European languages, 862 is DOS in Israel, 737 is DOS for Greeks.

Here is cp473:

	01 ☺	02 ☻	03 ♥	04 ♦	05 ♣	06 ♠	07 •	08 ◘	09 ○	0A ◙	0B ♂	0C ♀	0D ♪	0E ♫	0F ☼
10 ►	11 ◄	12 ↕	13 ‼	14 ¶	15 §	16 ▬	17 ↨	18 ↑	19 ↓	1A →	1B ←	1C ∟	1D ↔	1E ▲	1F ▼
20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	
80 Ç	81 ü	82 é	83 â	84 ä	85 à	86 å	87 ç	88 ê	89 ë	8A è	8B ï	8C î	8D ì	8E Ä	8F Å
90 É	91 æ	92 Æ	93 ô	94 ö	95 ò	96 û	97 ù	98 ÿ	99 Ö	9A Ü	9B ¢	9C £	9D ¥	9E ₧	9F ƒ
A0 á	A1 í	A2 ó	A3 ú	A4 ñ	A5 Ñ	A6 ª	A7 º	A8 ¿	A9 ⌐	AA ¬	AB ½	AC ¼	AD ¡	AE «	AF »
B0 ░	B1 ▒	B2 ▓	B3 │	B4 ┤	B5 ╡	B6 ╢	B7 ╖	B8 ╕	B9 ╣	BA ║	BB ╗	BC ╝	BD ╜	BE ╛	BF ┐
C0 └	C1 ┴	C2 ┬	C3 ├	C4 ─	C5 ┼	C6 ╞	C7 ╟	C8 ╚	C9 ╔	CA ╩	CB ╦	CC ╠	CD ═	CE ╬	CF ╧
D0 ╨	D1 ╤	D2 ╥	D3 ╙	D4 ╘	D5 ╒	D6 ╓	D7 ╫	D8 ╪	D9 ┘	DA ┌	DB █	DC ▄	DD ▌	DE ▐	DF ▀
E0 α	E1 β	E2 Γ	E3 π	E4 Σ	E5 σ	E6 µ	E7 τ	E8 Φ	E9 Θ	EA Ω	EB δ	EC ∞	ED ø	EE ∈	EF ∩
F0 ≡	F1 ±	F2 ≥	F3 ≤	F4 ⌠	F5 ⌡	F6 ÷	F7 ≈	F8 °	F9 ∙	FA ·	FB √	FC ⁿ	FD ²	FE ■	FF

MacRoman:

80 Ä	81 Å	82 Ç	83 É	84 Ñ	85 Ö	86 Ü	87 á	88 à	89 â	8A ä	8B ã	8C å	8D ç	8E é	8F è
90 ê	91 ë	92 í	93 ì	94 î	95 ï	96 ñ	97 ó	98 ò	99 ô	9A ö	9B õ	9C ú	9D ù	9E û	9F ü
A0 †	A1 °	A2 ¢	A3 £	A4 §	A5 •	A6 ¶	A7 ß	A8 ®	A9 ©	AA ™	AB ´	AC ¨	AD ≠	AE Æ	AF Ø
B0 ∞	B1 ±	B2 ≤	B3 ≥	B4 ¥	B5 µ	B6 ∂	B7 ∑	B8 ∏	B9 π	BA ∫	BB ª	BC º	BD Ω	BE æ	BF ø
C0 ¿	C1 ¡	C2 ¬	C3 √	C4 ƒ	C5 ≈	C6 ∆	C7 «	C8 »	C9 …	CA	CB À	CC Ã	CD Õ	CE Œ	CF œ
D0 –	D1 —	D2 “	D3 ”	D4 ‘	D5 ’	D6 ÷	D7 ◊	D8 ÿ	D9 Ÿ	DA ⁄	DB ¤	DC ‹	DD ›	DE ﬁ	DF ﬂ
E0 ‡	E1 ·	E2 ‚	E3 „	E4 ‰	E5 Â	E6 Ê	E7 Á	E8 Ë	E9 È	EA Í	EB Î	EC Ï	ED Ì	EE Ó	EF Ô
F0 ♥	F1 Ò	F2 Ú	F3 Û	F4 Ù	F5 ı	F6 ˆ	F7 ˜	F8 ¯	F9 ˘	FA ˙	FB ˚	FC ¸	FD ˝	FE ˛	FF ˇ

and Microsoft cp1252:

80 €		82 ‚	83 ƒ	84 „	85 …	86 †	87 ‡	88 ˆ	89 ‰	8A Š	8B ‹	8C Œ		8E Ž	
	91 ‘	92 ’	93 “	94 ”	95 •	96 –	97 —	98 ˜	99 ™	9A š	9B ›	9C œ		9E ž	9F Ÿ
A0	A1 ¡	A2 ¢	A3 £	A4 ¤	A5 ¥	A6 ¦	A7 §	A8 ¨	A9 ©	AA ª	AB «	AC ¬	AD -	AE ®	AF ¯
B0 °	B1 ±	B2 ²	B3 ³	B4 ´	B5 µ	B6 ¶	B7 ·	B8 ¸	B9 ¹	BA º	BB »	BC ¼	BD ½	BE ¾	BF ¿
C0 À	C1 Á	C2 Â	C3 Ã	C4 Ä	C5 Å	C6 Æ	C7 Ç	C8 È	C9 É	CA Ê	CB Ë	CC Ì	CD Í	CE Î	CF Ï
D0 Ð	D1 Ñ	D2 Ò	D3 Ó	D4 Ô	D5 Õ	D6 Ö	D7 ×	D8 Ø	D9 Ù	DA Ú	DB Û	DC Ü	DD Ý	DE Þ	DF ß
E0 à	E1 á	E2 â	E3 ã	E4 ä	E5 å	E6 æ	E7 ç	E8 è	E9 é	EA ê	EB ë	EC ì	ED í	EE î	EF ï
F0 ð	F1 ñ	F2 ò	F3 ó	F4 ô	F5 õ	F6 ö	F7 ÷	F8 ø	F9 ù	FA ú	FB û	FC ü	FD ý	FE þ	FF ÿ

(Find more code pages are displayed on [5].)

The international variants were standardized as ISO 646-DE (German), 646-DK (Danish), et cetera. Originally, the dollar sign could still be replaced by the currency symbol, but after a 1991 revision the dollar is now the only possibility.

The different code pages were ultimately standardized as ISO 8859, with such popular code pages as 8859-1 (‘Latin 1’) for western European,

A0	A1 ¡	A2 ¢	A3 £	A4 ¤	A5 ¥	A6 ¦	A7 §	A8 ¨	A9 ©	AA ª	AB «	AC ¬	AD -	AE ®	AF ¯
B0 °	B1 ±	B2 ²	B3 ³	B4 ´	B5 µ	B6 ¶	B7 ·	B8 ¸	B9 ¹	BA º	BB »	BC ¼	BD ½	BE ¾	BF ¿
C0 À	C1 Á	C2 Â	C3 Ã	C4 Ä	C5 Å	C6 Æ	C7 Ç	C8 È	C9 É	CA Ê	CB Ë	CC Ì	CD Í	CE Î	CF Ï
D0 Ð	D1 Ñ	D2 Ò	D3 Ó	D4 Ô	D5 Õ	D6 Ö	D7 ×	D8 Ø	D9 Ù	DA Ú	DB Û	DC Ü	DD Ý	DE Þ	DF ß
E0 à	E1 á	E2 â	E3 ã	E4 ä	E5 å	E6 æ	E7 ç	E8 è	E9 é	EA ê	EB ë	EC ì	ED í	EE î	EF ï
F0 ð	F1 ñ	F2 ò	F3 ó	F4 ô	F5 õ	F6 ö	F7 ÷	F8 ø	F9 ù	FA ú	FB û	FC ü	FD ý	FE þ	FF ÿ

8859-2 for eastern European, and 8859-5 for Cyrillic.

A0	A1 Ё	A2 Ђ	A3 Ѓ	A4 Є	A5 Ѕ	A6 І	A7 Ї	A8 Ј	A9 Љ	AA Њ	AB Ћ	AC Ќ	AD -	AE Ў	AF Џ
B0 А	B1 Б	B2 В	B3 Г	B4 Д	B5 Е	B6 Ж	B7 З	B8 И	B9 Й	BA К	BB Л	BC М	BD Н	BE О	BF П
C0 Р	C1 С	C2 Т	C3 У	C4 Ф	C5 Х	C6 Ц	C7 Ч	C8 Ш	C9 Щ	CA Ъ	CB Ы	CC Ь	CD Э	CE Ю	CF Я
D0 а	D1 б	D2 в	D3 г	D4 д	D5 е	D6 ж	D7 з	D8 и	D9 й	DA к	DB л	DC м	DD н	DE о	DF п
E0 р	E1 с	E2 т	E3 у	E4 ф	E5 х	E6 ц	E7 ч	E8 ш	E9 щ	EA ъ	EB ы	EC ь	ED э	EE ю	EF я
F0 №	F1 ё	F2 ђ	F3 ѓ	F4 є	F5 ѕ	F6 і	F7 ї	F8 ј	F9 љ	FA њ	FB ћ	FC ќ	FD §	FE ў	FF џ

These ISO standards explicitly left the first 32 extended positions undefined.

Reading material: The history of ASCII out of telegraph codes [1]. A history, paying attention to multilingual use [4]. Bob Bemer, the ‘father of ascii’ [2]. A detailed discussion of ISO 8859, Latin-1 [11].

2 Character sets and encodings

As you can tell from the introduction, there is quite a bit of confusion possible between characters and representations or encodings. Let us clear up the concepts a little.

Informally, the term ‘character set’ (also ‘character code’ or ‘code’) used to mean something like ‘a table of bytes, each with a character shape’. With only the English alphabet to deal with that is a good enough definition. These days, much more general cases are handled, mapping one octet into several characters, or several octets into one character. The definition has changed accordingly:

> A *charset* is a method of converting a sequence of octets into a sequence of characters. This conversion may also optionally produce additional control information such as directionality indicators.

(From RFC 2978) A conversion the other way may not exist, since different octet combinations may map to the same character. Another complicating factor is the possibility of switching between character sets; for instance, ISO 2022-JP is the standard ASCII character set, but the escape sequence `ESC $ @` switches to JIS X 0208-1978.

To disentangle the concepts behind encoding, we need to introduce a couple of levels:

ACR Abstract Character Repertoire: the set of characters to be encoded; for example, some alphabet or symbol set. This is an unordered set of characters, which can be fixed (the contents of ISO 8859-1), or open (the contents of Unicode).

CCS Coded Character Set: a mapping from an abstract character repertoire to a set of nonnegative integers. This is what is meant by ‘encoding’, ‘character set definition’, or ‘code page’; the integer assigned to a character is its ‘code point’.
There used to be a drive towards unambiguous abstract character names across repertoires and encodings, but Unicode ended this, as it provides (or aims to provide) more or less a complete list of every character on earth.

CEF Character Encoding Form: a mapping from a set of nonnegative integers that are elements of a CCS to a set of sequences of particular code units. A ‘code unit’ is an integer of a specific binary width, for instance 8 or 16 bits. A CEF then maps the code points of a coded character set into sequences of code point, and these sequences can be of different lengths inside one code page. For instance ASCII uses a single 7-bit unit; UTF-8 uses one to four 8-bit units. We will discuss the UTF encodings below.

CES Character Encoding Scheme: a reversible transformation from a set of sequences of code units (from one or more CEFs to a serialized sequence of bytes. In single-byte cases such as ASCII and UTF-8 this mapping is trivial. With the two-byte scheme UCS-2 there is a single ‘byte order mark’, after which the code units are trivially mapped to bytes. On the other hand, ISO 2022, which uses escape sequences to switch between different encodings, is a complicated CES.

Additionally, there are the concepts of

CM Character Map: a mapping from sequences of members of an abstract character repertoire to serialized sequences of bytes bridging all four levels in a single operation. These maps are what gets assigned MIBenum values by IANA; see section 4.1.

TES Transfer Encoding Syntax: a reversible transform of encoded data. This data may or may not contain textual data. Examples of a TES are base64, uuencode, and quoted-printable, which all transform a byte stream to avoid certain values.

3 Unicode and UTF encodings

The systems above functioned quite well as long as you stuck to one language or writing system. Poor dictionary makers. More or less simultaneously two efforts started that aimed to incorporate all the world's character sets in one standard: Unicode standard (originally 2-byte), and ISO 10646 (oringally 4-byte). Unicode was extended further, so that it has all code points up to `10FFFFF`, which is slightly over a million.

Two international standards organizations, the Unicode Consortium and ISO/IEC JTC1/SC2, started designing a universal standard that was to be a superset of all existing character sets. These standards are now synchronized. Unicode has elements that are not in 10646, but they are compatible where it concerns straight character encoding.

ISO 10646 defines UCS, the 'Universal Character Set'. This is in essence a table of official names and code numbers for characters. Unicode adds to this rules for hyphenation, bi-directional writing, and more.

The full Unicode list of code points can be found online, broken down by blocks [14], and downloadable [17].

3.1 BMP and earlier Unicode subplanes

Characters in Unicode are mostly denoted hexadecimally as `U+wxyz`, for instance `U+0041` is 'Latin Capital Letter A'. The range `U+0000`–`U+007F` (0–127) is identical to US-ASCII (ISO 646 IRV), and `U+0000`–`U+00FF` (0–255) is identical to Latin 1 (ISO 8859-1).

The original 2-byte subset is now called 'BMP' for Basic Multilingual Plane, or plane 0. These are the Unicode code points that are nonzero in the last two bytes. Other 'planes' have been defined that have one or more bits set outside the last two bytes.

BMP (Basic Multilingual Plane) The first plane defined in Unicode/ISO 10646, designed to include all scripts in active modern use. The BMP currently includes the Latin, Greek, Cyrillic, Devangari, hiragana, katakana, and Cherokee scripts, among others, and a large body of mathematical, APL-related, and other miscellaneous characters. Most of the Han ideographs in current use are present in the BMP, but due to the large number of ideographs, many were placed in the Supplementary Ideographic Plane.

SMP (Supplementary Multilingual Plane; plane 1) This contains mostly ancient writing systems. Some of these you'll have heard of, such as Linear B, cuneiform, Aztec, and Maya; others are fairly obscure, such as Tangut, a language used in Central China between 1000 and 1500.

SIP (Supplementary Ideographic Plane) The third plane (plane 2) defined in Unicode/ISO 10646, designed to hold all the ideographs descended from Chinese writing (mainly found in Vietnamese, Korean, Japanese and Chinese) that aren't found in the Basic Multilingual Plane. The BMP was supposed to hold all ideographs in modern use; unfortunately, many Chinese dialects (like Cantonese and Hong Kong Chinese) were overlooked; to write these, characters from the SIP are necessary. This is one reason even non-academic software must support characters outside the BMP.

3.2 Unicode encodings

Unicode is basically a numbered list of characters. When they are used in a file, their numbers can be encoded in a number of ways. To name the obvious example: if only the first 128 positions are used, the long Unicode code point can be truncated to just one byte. Here are a few encodings:

UTF-32 Little used: this is a four-byte encoding. (UTF stands for 'UCS Transformation Format'.)

UTF-16 A two-byte encoding. Its precursor, UCS-2, encoded the BMP; UTF-16 has a way of going beyond that to encode the whole of Unicode.

UTF-8 A one-byte scheme; details below.

UTF-7 Another one-byte scheme, but now the high bit is always off. Certain byte values act as 'escape', so that higher values can be encoded. Like UTF-1 and SCSU, this encoding is only of historical interest.

There is an important practical reason for a one-byte encoding such as UTF-8. Multi-byte encodings such as UCS-2 are wasteful of space, if only traditional ASCII is needed. Furthermore, they would break software that is expecting to walk through a file with `s++` and such. Also, they would introduce many zero bytes in a file, which would play havoc with Unix software that uses null-termination for strings.

Then there would be the problem of whether two bytes are stored in low-endian or high-endian order. For this reason it was suggested to store `FE FF` or `FF FE` at the beginning of each file as the 'Unicode Byte Order Mark'. Formally, `FEFF` is the Unicode 'zero width nobreak space' character, which can innocently be inserted anywhere. Conversely `FFEF` is defined to be illegal, so encountering this is a sign that bytes should be interpreted little-endian. Of course this plays havoc with files such as shell scripts which expect to find `#!` at the beginning of the file.

3.3 UTF-8

UTF-8, standardized as RFC 3629, is an encoding where the positions up to 127 are encoded 'as such'; higher numbers are encoded in groups of 2 to 6 bytes. (Tim Bray describes this as 'kind of racist' [3]: the further east a language comes from, the more overhead is involved in its encoding.) In a multi-byte group, the first byte is in the range 0xC0–0xFD (192–252). The next up to 5 bytes are in the range 0x80–0xBF (128–191, bit pattern starting with `10`). Note that `8` = `1000` and `B` = `1011`, so the highest two bits are always `10`, leaving six bits for encoding).

`U-00000000 - U-0000007F`	7 bits	`0xxxxxxx`		
`U-00000080 - U-000007FF`	$11 = 5 + 6$	`110xxxxx`	`10xxxxxx`	
`U-00000800 - U-0000FFFF`	$16 = 4 + 2 \times 6$	`1110xxxx`	`10xxxxxx`	`10xxxxxx`
`U-00010000 - U-001FFFFF`	$21 = 3 + 3 \times 6$	`11110xxx`	`10xxxxxx (3 times)`	
`U-00200000 - U-03FFFFFF`	$26 = 2 + 4 \times 6$	`111110xx`	`10xxxxxx (4 times)`	
`U-04000000 - U-7FFFFFFF`	$31 = 1 + 5 \times 6$	`1111110x`	`10xxxxxx (5 times)`	

All bites in a multi-byte sequence have their high bit set.

IETF documents such as RFC 2277 require support for this encoding in internet software. Readable introductions can be found all over the internet [19]; see also the history of UTF-8 [20].

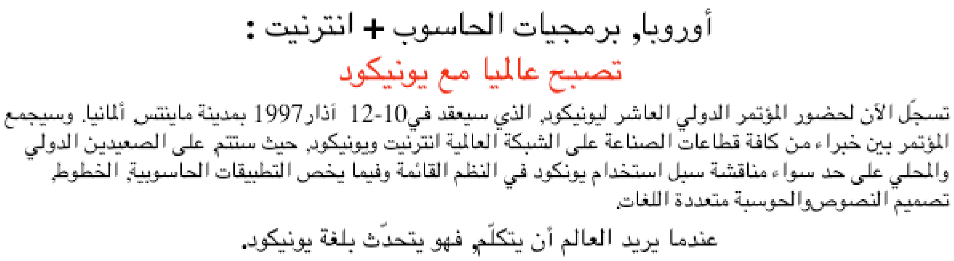
3.4 Unicode tidbits

3.4.1 Line breaking

The Unicode standard describes line breaking: it has a mechanism for specifying tables of character pairs between which line breaks are allowed or forbidden [15, 18].

3.4.2 Bi-directional writing

Most scripts are left-to-right, but Arabic and Hebrew run right-to-left. Characters in a file are stored in ‘logical order’, and usually it is clear in which direction to render them, even if they are used mixed. Letters have a ‘strong’ directionality: unless overridden, they will be displayed in their natural direction. The first letter of a paragraph with strong direction determines the main direction of that paragraph [16].



أوروبا, برمجيات الحاسوب + انترنيت :

تصبح عالميا مع يونيكود

تسجّل الآن لحضور المؤتمر الدولي العاشر ليونيكود, الذي سيعقد في10-12 آذار1997 بمدينة ماينتس ألمانيا. وسيجمع المؤتمر بين خبراء من كافة قطاعات الصناعة على الشبكة العالمية انترنيت ويونيكود, حيث ستتم على الصعيدين الدولي والمحلي على حد سواء مناقشة سبل استخدام يونكود في النظم القائمة وفيما يخص التطبيقات الحاسوبية, الخطوط, تصميم النصوص والحوسبة متعددة اللغات.

عندما يريد العالم أن يتكلّم, فهو يتحدّث بلغة يونيكود.

However, when differently directional texts are embedded, some explicit help is needed. The problem arises with letters that have only weak directionality. The following is a sketch of a problematic case.

> Memory: he said ”I NEED WATER!”, and expired.
> Display: he said ”RETAW DEEN I!”, and expired.

If the exclamation mark is to be part of the Arabic quotation, then the user can select the text ‘I NEED WATER!’ and explicitly mark it as embedded Arabic (`<RLE>` is Right-Left Embedding; `<PDF>` Pop Directional Format), which produces the following result:

> Memory: he said ”<RLE>I NEED WATER!<PDF>”, and expired.
> Display: he said ”!RETAW DEEN I”, and expired.

A simpler method of doing this is to place a Right Directional Mark <RLM> after the exclamation mark. Since the exclamation mark is now not on a directional boundary, this produces the correct result.

> Memory: he said ”I NEED WATER!<RLM>”, and expired.
> Display: he said ”!RETAW DEEN I”, and expired.

3.5 Unicode and oriental languages

‘Han unification’ is the Unicode strategy of saving space in the oriental languages (traditional Chinese, simplified Chinese, Japanese, Korean: ‘CJK’) by recognizing common characters. This idea is not uncontroversial [6].

4 Further tidbits

4.1 A bootstrapping problem

In order to know how to interpret a file, you need to know what character set it uses. This problem also occurs in MIME mail encoding (section 4.5), which can use many character sets. Names and numbers for character sets are standardized by IANA: the Internet Assigned Names Authority [8]. However, in what character set do you write this name down?

Fortunately, everyone agrees on (7-bit) ASCII, so that is what is used. A name can be up to 40 characters from us-ascii.

As an example, here is the iana definition of ASCII:

name `ANSI_X3.4-1968`
reference RFC1345,KXS2
MIBenum 3
source ECMA registry
aliases `iso-ir-6`, `ANSI_X3.4-1986`, `ISO_646.irv:1991`, `ASCII`, `ISO646-US`, `US-ASCII (preferred MIME name)`, `us`, `IBM367`, `cp367`, `csASCII`

The `MIBenum` (Management Information Base) is a number assigned by IANA[1]. The full list of character sets can be found oneline [9], and RFC 3808 is a memo that describes the IANA Charset MIB.

4.2 Unicode in programming languages

Before Unicode, a system called the 'Double Byte Character Set' was invented to accomodate Asian languages, where some characters were stored in one, others in two bytes. This is very messy, since you can not simply write `s++` or `s--` to traverse a string. Instead you have to use functions from some library that understands these encodings. While this system is now only of historical interest, the string handling problem is back in force with UTF-8.

Many modern languages (Python, C99) have support for Unicode. In C99 (which is the new standard for C) this is done through so-called 'wide characters'. For instance, `L'x'` is a wide character and `L"xyz"` is a string of side characters. Such strings can be manipulated through equivalents of the normal string library. For instance, `wcscpy` acts like `strcpy` but on wide strings. General Unicode characters can be represented as `\u0000` for 4-byte and `\U00000000` for up to 8-byte characters.

The two-byte UTF-16 encoding is popular among programmers, since it can handle almost any practically encountered character without extensions to longer byte sequences.

4.3 Character codes in HTML

HTML can access unusual characters in several ways:

- With a decimal numerical code: ` ` is a space token. (HTML 4 supports hexadecimal codes.)

1. Apparently these numbers derive from the Printer MIB, RFC 1759.

- With a vaguely symbolic name [12, 7]: `©` is the copyright symbol.
- The more interesting way is to use an encoding such as UTF-8 (section 3.2) for the file. For this it would be nice if the server could state that the file is
  `Content-type: text/html;charset=utf-8`
  but it is also all right if the file starts with
  `<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=utf-8">`

```
Description                          Char Code           Entity name
==================================== ==== ============   ==============
non-breaking space                          -->         -->
inverted exclamation                 ¡    &#161; --> ¡   &iexcl;  --> ¡
cent sign                            ¢    &#162; --> ¢   &cent;   --> ¢
pound sterling                       £    &#163; --> £   &pound;  --> £
general currency sign                ¤    &#164; --> ¤   &curren; --> ¤
yen sign                             ¥    &#165; --> ¥   &yen;    --> ¥
broken vertical bar                  ¦    &#166; --> ¦   &brvbar; --> ¦
```

It is requirement that user agents can at least parse the `charset` parameter, which means they have to understand us-ascii.

Open this link in your browser, and additionally view the source: `http://www.unicode.org/unicode/iuc10/x-utf8.html`. How well does your software deal with it?

4.4 Keyboards and control characters

Unprintable ascii codes are accessible through the control modifier key; for this reason they are also called ‘control codes’ or control characters. The control key, combined with a regular key, zeros bits 2 and 3 of the ascii code of that key. For instance, you can hit `Ctrl-[` to get `Esc`.

The way key presses generate characters is typically controlled in software. This mapping from keyboard scan codes to 7 or 8-bit characters is called a ‘keyboard’, and can be changed dynamically in most operating systems.

Using the modifier keys, one can generate more keystrokes than can be described in 8 bits, so keyboards can send an ‘escape sequence’: one escape character followed by one or more regular characters. The escape character is mostly ascii NULL or ESC [10].

4.5 Characters in email

The protocols for internet mail are based on ‘7-bit ASCII’, that is, the high bit in every byte transmitted is supposed to be off. This is a problem for any message that has text outside of ASCII, such as when accented characters from the various ISO 8859 character sets are used. It also makes transmitting binary data such as images impossible. For this reason the ‘Multipurpose Internet Mail Extensions’ (MIME) were designed. MIME uses several encoding schemes, such as base64 or printed quotable, to turn arbitrary data into 7-bit ascii.

The email standard RFC 822 states that anything outside 7-bit ascii has to be encoded with `uuencode`. This means that the sender and recipient need decoding program; it is decidedly overkill if a message is plain ASCII apart from a few accented characters.

The MIME protocol (RFC 2045 and 2046) inserts headers in a mail message, stating for each message section the content type and the encoding that is used for that section of the message. These encodings are also used for attachments, in which case the content type should give an indication what application can handle the attachment after its decoding. ‘Helpful’ mail programs that automatically invoke such applications have been a source of Trojans (malicious softwares) in the past.

4.6 FTP

FTP is a very old ARPA protocol for transferring files from one computer to another. It knows ‘binary’ and ‘text’ mode: in binary mode bytes are transferred without further interpretation, but the text mode is concerned with files that contain lines of text. Unfortunately, line ends are different between operating systems, and their transfer in text mode is not well defined. Some ftp programs adjust line ends; others, such as `Fetch` on the Mac, actually do code page translation.

5 Character issues in TEX / LATEX

5.1 Diacritics

Before 1990, TEX was a 7-bit system: only characters 0–127 in the input could be recognized, and fonts were also limited to 127 positions. This meant that there was not enough space in fonts for letters with accents, so accents (diacritics) were implemented as things to put on top of characters, even when, as with the cedilla, they are under the letter. This leads to the problem that TEX can not hypenate a word with accents, since the accent introduces a space in the word (technically: an explicit kern).

Both problems were remedied to a large extent with the ‘Cork font encoding’, which contains most accented letters as single characters. This means that accents are correctly placed by design, and also that the word can be hyphenated, since the kern has disappeared.

These fonts with accented characters became possible when TEX version 3 came out around 1990. This introduced full 8-bit compatibility, both in the input side and in the font addressing.

5.2 LATEX input file access to fonts

If an input file for LATEX is allowed to contain all 8-bit octets, we get all the problems of compatibility that plagued regular text files. This is solved by the package `inputenc`:

```
\usepackage[code]{inputenc}
```

where `codes` is `applemac`, `ansinew`, or various other code pages.

This package makes all unprintable ASCII characters, plus the codes over 127, into active characters. The definitions are then dynamically set depending on the code page that is loaded.

5.3 LaTeX output encoding

The `inputenc` package does not solve the whole problem of producing a certain font character from certain keyboard input. It only mapped a byte value to the TeX command for producing a character. To map such commands to actual code point in a font file, the TeX and LaTeX formats contain lines such as

```
\chardef\i="10
```

declaring that the dotless-i is at position 16. However, this position is a convention, and other people – type manufacturers – may put it somewhere else.

This is handled by the 'font encoding' mechanism. The various people working on the LaTeX font schemes have devised a number of standard font encodings. For instance, the `OT1` encoding corresponds to the original 128-character set. The `T1` encoding is a 256-character extension thereof, which includes most accented characters for Latin alphabet languages.

A font encoding is selected with

```
\usepackage[T1]{fontenc}
```

A font encoding definition contains lines such as

```
\DeclareTextSymbol{\AE}{OT1}{29}
\DeclareTextSymbol{\OE}{OT1}{30}
\DeclareTextSymbol{\O}{OT1}{31}
\DeclareTextSymbol{\ae}{OT1}{26}
\DeclareTextSymbol{\i}{OT1}{16}
```

5.4 TeX beyond 8 bits

The above LaTeX packages allow flexible handling of (8-bit) codepages, essentially the ISO 8859 standard. For handling of other alphabets, a number of styles have been written over the years. However, their continued support is often uncertain. The only project that aims at use of Unicode throughout TeX's codebase is Omega [13].

Contents

References

[1] Annotated history of ascii. `http://www.wps.com/projects/codes/index.html`.
[2] Bob bemer homepage. `http://www.trailing-edge.com/~bobbemer/`.
[3] Tim Bray. Characters vs bytes. urlhttp://www.tbray.org/ongoing/When/200x/2003/04/26/UTF.
[4] Brief history of character codes in north american, europe, and east asia. `http://tronweb.super-nova.co.jp/characcodehist.html`.
[5] Codepage & co. `http://aspell.net/charsets/codepages.html`.
[6] Han unification. `http://en.wikipedia.org/wiki/Han_unification`.
[7] Character entity references in HTML4. `http://www.w3.org/TR/html401/sgml/entities.html`.
[8] `http://www.iana.org/`.
[9] IANA character set names. `http://www.iana.org/assignments/character-sets`.
[10] IBM PC keyboard scan codes. `http://jimprice.com/jim-asc.shtml#keycodes`.
[11] The ISO latin 1 character repertoire. `http://www.cs.tut.fi/~jkorpela/latin1/index.html`.
[12] Character entities for ISO latin 1. `http://www.cs.tut.fi/~jkorpela/HTML3.2/latin1.html`.
[13] Omega project home page. `http://omega.enstb.org/`.
[14] Unicode. `http://www.fileformat.info/info/unicode/index.htm`.
[15] Unicode standard annex 14, line breaking properties. `http://www.unicode.org/reports/tr14/`.
[16] Unicode standard annex 9, the bidirectional algorithm. `http://www.unicode.org/reports/tr9/`.
[17] Unicode code chart and scripts. `http://www.unicode.org/charts/`.

[18] Unicode line breaking rules: explanations and criticism. http://www.cs.tut.fi/~jkorpela/unicode/linebr.html.
[19] UTF-8 and Unicode FAQ for Unix/Linux. http://www.cl.cam.ac.uk/~mgk25/unicode.html.
[20] UTF-8 history. http://www.cl.cam.ac.uk/~mgk25/ucs/utf-8-history.txt.