

ASCII CONTROL CODES										dec CHAR hex oct		
b7 b6 b5 BITS		0 0	0 1	1 0	1 1	1 0	1 0	1 1	1 0	1 1		
b4 b3 b2 b1		CONTROL			SYMBOLS NUMBERS		UPPERCASE		LOWERCASE			
0 0 0 0	0	NUL	DLE	SP	0	@	P	'	p			
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q			
0 0 1 0	2	STX	DC2	"	2	B	R	b	r			
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s			
0 1 0 0	4	EOT	DC4	\$	4	D	T	d	t			
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u			
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v			
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w			
1 0 0 0	8	BS	CAN	(8	H	X	h	x			
1 0 0 1	9	HT	EM)	9	I	Y	i	y			
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z			
1 0 1 1	11	VT	ESC	+	;	K	[k	{			
1 1 0 0	12	FF	FS	,	<	L	\	l				
1 1 0 1	13	CR	GS	-	=	M]	m	}			
1 1 1 0	14	SO	RS	.	>	N	^	n	~			
1 1 1 1	15	SI	US	/	?	O	.	o	DEL			

Table 1: The ASCII table

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	AB	AC	AD	AE	AF
ø	í	φ	£	κ	¥	!	§	..	@	≡	«	¬	-	®	-
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
±	²	³	³	⁴	⁵	μ	¶	·	¹	²	³	¼	½	¾	¿
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

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

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	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	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
Ѐ	Ђ	Ѓ	Ѕ	Ї	Ї	Ј	Љ	Њ	Ћ	Ќ	Ѝ	Ў	Џ	ѐ	ё
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
Ў	ѐ	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	ѝ	ў

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, ISO2022-JP is the standard ASCII character set, but the character 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 non-negative 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 non-negative 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 points, 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 transformation 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: the Unicode standard (originally 2-byte), and ISO 10646 (originally 4-byte). Unicode was extended further, so that it has all code points up to 10FFFFFF, 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 other 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 the ‘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/ISO10646, 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 likely heard of, such as Linear B, cuneiform, Aztec, and Mayan; 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 planes 1–16 by using ‘surrogate pairs’ of two-byte units.

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 an ‘escape’, so that higher values can be encoded. Like UTF-1 and UCSU, 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).

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

Table 2: UTF-8 encoding blocks

Note that 8 = 1000 and B = 1011, so the highest two bits are always 10, leaving six bits for encoding). All bytes in a multi-byte sequence have their high bit set. See table 2.

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 in [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]. See figure 1.

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-To-Left 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

أوروبا, برمجيات الحاسوب + انترنت :

تصبح عالميا مع يونيكود

تسجل الآن لحضور المؤتمر الدولي العاشر ليونيكود، الذي سيعقد في 10-12 آذار 1997 بمدينة ماينتس ألمانيا. وسيجمع المؤتمر بين خبراء من كافة قطاعات الصناعة على الشبكة العالمية انترنت ويونيكود. حيث ستم على الصعيدين الدولي والمحلي على حد سواء مناقشة سبل استخدام يونيكود في النظم القائمة وفيما يخص التطبيقات الحاسوبية الخطوط. تصميم النصوص والحوسبة متعددة اللغات.

عندما يريد العالم أن يتكلم فهو يتحدث بلغة يونيكود.

Figure 1: Right-to-left Arabic text containing left-to-right numerals

for character sets are standardized by IANA: the Internet Assigned Numbers Authority [9]. 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	RFC 1345, 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 online [8], 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 accommodate 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 wide 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 with 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, ` ` is a non-breaking space, etc.
- 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 (end with `</>` for XHTML):

```
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8">
```

It is a requirement of user agents that they 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 NUL 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 quoted-printable, 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 software) 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 (L)TeX

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 could not hyphenate 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 common 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 on the input side and in the font addressing.

5.2 L^ATeX input file access to fonts

If an input file for L^ATeX is allowed to contain all 8-bit octets, we get all the problems of compatibility

that plague regular text files. This is solved by the package `inputenc`:

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

where *code* is `applemac`, `ansinew`, `utf8`, 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 L^ATeX output encoding

The `inputenc` package does not solve the whole problem of producing a certain font character from certain keyboard input. It only maps a byte value to the TeX command for producing a character. To map such commands to an actual code point in a font file, the TeX and L^ATeX 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 L^ATeX 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 L^ATeX packages allow flexible handling of (8-bit) code pages, 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 first project that aimed at use of Unicode throughout TeX’s code base was Omega [13]; the modern TeX extensions X_YTeX (<http://scripts.sil.org/xetex>) and LuaTeX (<http://luatex.org>) also do so.

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. <http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF>.
- [4] Brief history of character codes in North America, Europe, and East Asia. <http://tronweb.super-nova.co.jp/characodehist.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] IANA character set names. <http://www.iana.org/assignments/character-sets>.
- [9] Internet Assigned Numbers Authority. <http://www.iana.org/>.
- [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/~jkorpe/latin1/index.html>.
- [12] Character entities for ISO Latin 1. <http://www.cs.tut.fi/~jkorpe/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/~jkorpe/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>.

◇ Victor Eijkhout
University of Texas at Austin
victor (at) eijkhout dot net