Unicode

Florian “Florob” Zeitz

2014-09-25

1. About me
2. History
3. Encodings
4. Normalization
5. Confusables
6. Special Characters

1. About me
2. History
3. Encodings
4. Normalization
5. Confusables
6. Special Characters

About me

- Florian Zeitz
- M. Sc. Electrical Engineering (Computer Engineering)
- interests:
  - XMPP
  - Systems Programming
  - C
  - Objective-C
  - Rust

1. About me
2. History
3. Encodings
4. Normalization
5. Confusables
6. Special Characters

ASCII

- published in 1963 by the American Standards Association (today: ANSI)
- 7 bit character encoding
- based on encodings used for teletypes
- includes 128 characters, 33 control, 95 printable
- first commercial use in AT&T's TWX (TeletypeWriter eXchange) network

ASCII

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td>NUL</td><td>DLE</td><td>SP</td><td>0</td><td>@</td><td>P</td><td>‘</td><td>p</td></tr>
<tr><td>1</td><td>SOH</td><td>DC1</td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td></tr>
<tr><td>2</td><td>STX</td><td>DC2</td><td>”</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td></tr>
<tr><td>3</td><td>ETX</td><td>DC3</td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td></tr>
<tr><td>4</td><td>EOT</td><td>DC4</td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td></tr>
<tr><td>5</td><td>ENQ</td><td>NAK</td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td></tr>
<tr><td>6</td><td>ACK</td><td>SYN</td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td></tr>
<tr><td>7</td><td>BEL</td><td>ETB</td><td>’</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td></tr>
<tr><td>8</td><td>BS</td><td>CAN</td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td></tr>
<tr><td>9</td><td>HT</td><td>EM</td><td>)</td><td>9</td><td>I</td><td>Y</td><td>i</td><td>y</td></tr>
<tr><td>a</td><td>LF</td><td>SUB</td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td></tr>
<tr><td>b</td><td>VT</td><td>ESC</td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td>{</td></tr>
<tr><td>c</td><td>FF</td><td>FS</td><td>,</td><td><</td><td>L</td><td>\</td><td>l</td><td>|</td></tr>
<tr><td>d</td><td>CR</td><td>GS</td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td>}</td></tr>
<tr><td>e</td><td>SO</td><td>RS</td><td>.</td><td>></td><td>N</td><td>^</td><td>n</td><td>~</td></tr>
<tr><td>f</td><td>SI</td><td>US</td><td>/</td><td>?</td><td>O</td><td>_</td><td>o</td><td>DEL</td></tr>
</table>

ISO 646

- originally published 1972
- 7 bit character encoding
- largely equivalent to ASCII
- supports national variants
- variable characters are:
  #, $, @, [, \,], ^, ‘, {, |, }, ~

ISO 646 Variant 21

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	§	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
a	LF	SUB	*	:	J	Z	j	z
b	VT	ESC	+	;	K	Ä	k	ä
c	FF	FS	,	<	L	Ö	l	ö
d	CR	GS	-	=	M	Ü	m	ü
e	SO	RS	.	>	N	^	n	ß
f	SI	US	/	?	O	_	o	DEL

ISO 8859

- originally published 1987 (part 1 and 2)
- 8 bit character encoding
- 16 parts, tailored to different regions
- first 128 characters equivalent to ASCII

Unicode

- “universal character encoding”
- assigns “codepoints” to characters

1991 first version (1.0.0)

- 24 scripts
- 7161 characters
- limited to 16 bit numbers
- 65 536 characters should be enough for everyone

Unicode

1996 2.0 released

- 25 scripts
- 38 950 characters
- 21 bit encoding
- 1 114 112 possible codepoints
  17 planes, 65 536 codepoints each

2014 Unicode 7.0

- 123 scripts
- 113 021 characters

Unicode planes

Plane 0	Basic Multilingual Plane (BMP)
Plane 1	Supplementary Multilingual Plane (SMP)
Plane 2	Supplementary Ideographic Plane (SIP)
Plane 3–13	unassigned
Plane 14	Supplementary Special-purpose Plane
Plane 15–16	Supplementary Private Use Area

1 About me

2 History

3 Encodings

4 Normalization

5 Confusables

6 Special Characters

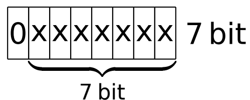
Encoding Forms

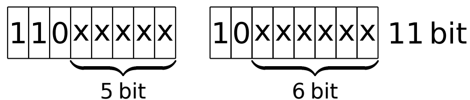
- Unicode specifies three encoding forms: UTF-8, UTF-16, and UTF32
- Unicode Transformation Format
- number specifies bit-size of the “code units”
- encodes “Unicode scalar values”, encompassing U+0000–U+D7FF, and U+E000–U+10FFFF
- the gap contains so called High-Surrogates (U+D800–U+DBFF), and Low-Surrogates (U+DC00–U+DFFF)

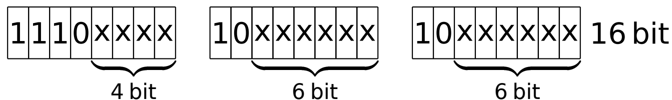
UTF-8

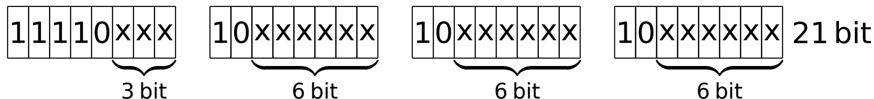
- code unit is 8 bit
- leading 1s in the first byte signify length
- “continuation bytes” start with 10
- backwards compatible with 7 bit ASCII (1 Byte)
- 2 Byte for the next 1920 characters
- 3 Byte for the rest of the BMP (most CJK is here)
- 4 Byte for all other planes

UTF-8









UTF-16

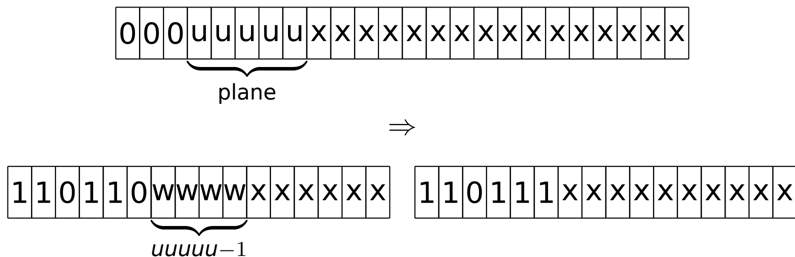
- code unit is 16 bit
- Unicode scalar values up to U+FFFF encoded as unsigned 16 bit integer

⇒ 2 Byte for the whole BMP (including most CJK)

- higher Unicode scalar values are split into a “surrogate pair”

⇒ 4 Byte for any Unicode scalar value outside the BMP

UTF-16



UCS-2

- 2 Byte Universal Character Set
- code unit is 16 bit
- fixed length encoding
- can *only* represent the BMP
- replaced by UTF-16 since Unicode 2.0 (1996)
- ECMAScript may use this as encoding ☹

UTF-32

- Unicode scalar values encoded as unsigned 32 bit integer
- the high byte is always zero
- only fixed-width encoding

BOM — Endianness

- U+FEFF ZERO WIDTH NO-BREAK SPACE
- BOM as first character of a file
- can be used to detect byte order for UTF-16 and UTF-32:
  - UTF-16BE: 0xFE 0xFF
  - UTF-16LE: 0xFF 0xFE
- can be used to detect encoding
  - UTF-8: 0xEF 0xBB 0xBF
  - UTF-16: 0xFE 0xFF
  - UTF-32: 0x00 0x00 0xFE 0xFF
- IETF: discourages BOM as a signature, always use UTF-8, or signal encoding

Summary

- UTF-8: smallest for Latin scripts, expensive indexing
- UTF-16: smallest for CJK, expensive indexing, less decoding overhead than UTF-8
- UTF-32: wasteful, efficient indexing, trivial to decode

1. About me
2. History
3. Encodings
4. **Normalization**
5. Confusables
6. Special Characters

The Problem

- different codepoint sequences are considered equivalent
- canonical equivalence: sequences should be displayed and interpreted the same
- compatibility equivalence: sequences may display differently and be interpreted differently in certain contexts

Canonical Equivalence

B + Ä ≡ B + A + ◌̈

0042 00C4 ≡ 0042 0041 0308

q + ◌̣ + ◌̇ ≡ q + ◌̇ + ◌̣

0071 0323 0307 ≡ 0071 0307 0323

Ω ≡ Ω

2126 ≡ 03A9

Compatibility Equivalence

ℌ ≡ H
210C 0048

② ≡ 2
2461 0032

dž ≡ d + z + ◌̌
01C6 0064 007A 030C

Normalization Forms

- 4 Normalization Forms
- Normalization Form D (NFD): Canonical Decomposition
- Normalization Form C (NFC): Canonical Decomposition, followed by Canonical Composition
- Normalization Form KD (NFKD): Compatibility Decomposition
- Normalization Form KC (NFKC): Compatibility Decomposition, followed by Canonical Composition

Decomposition

- entries in the Unicode Character Database
- one for canonical, one for compatibility decomposition
- maps one codepoint to one or more codepoints
- the result of a mapping may still be decomposable $\Rightarrow$ iterate
- compatibility decomposition has a type:
  font, noBreak, initial, medial, final, isolated, circle, super, sub, vertical, wide, narrow, small, square, fraction, compat
- includes sorting of combining characters, according to their “canonical combining class”

Composition

- only canonical
- works on pairs of codepoints
- reverses canonical decomposition
- a composed codepoint can be composed with another codepoint
- string can change due to reordering

ḋ + ◌̣ $\overset{NFC}{\Longrightarrow}$ ḍ + ◌̇

1E0B 0323 1E0D 0307

Decomposition enlarges strings

"صلى الله عليه وسلم": Arabic "Peace be upon him"

- one codepoint: U+FDFA
- compatibility decomposition has 18 codepoints:
  U+0635, U+0644, U+0649, U+0020, U+0627, U+0644, U+0644, U+0647, U+0020, U+0639, U+0644, U+064A, U+0647, U+0020, U+0648, U+0633, U+0644, U+0645

Hangeul

한글

- Korean Script
- 24 letters (Jamo)
- 14 consonants, 10 vowels
- written in blocks (Syllables)
- would require huge tables
- can we do better?

Hangeul (De)Composition

- Solution: order Jamo, and Syllables to get an arithmetic relation
- differentiate Leading, Vowel, and Trailing Jamo

```
SBase = 0xAC00
LBase = 0x1100
VBase = 0x1161
TBase = 0x11A7
LCount = 19
VCount = 21
TCount = 28
NCount = VCount * TCount = 588
SCount = LCount * NCount = 11172
```

Hangeul Decomposition

```
% Decomposition of syllable s
SIndex = s - SBase
LIndex = SIndex / NCount
VIndex = (SIndex % NCount) / TCount
TIndex = SIndex % TCount
LPart = LBase + LIndex
VPart = VBase + VIndex
TPart = TBase + TIndex if TIndex > 0
```

Hangeul Composition

```
% Compose <LPart, VPart>
LIndex = LPart - LBase
VIndex = VPart - VBase
LVIndex = LIndex * NCount
          + VIndex * TCount
s = SBase + LVIndex
```

```
% Compose <LVPart, TPart>
TIndex = TPart - TBase
s = LVPart + TIndex
```

1. About me
2. History
3. Encodings
4. Normalization
5. **Confusables**
6. Special Characters

The Problem

- Unicode does not encode glyphs
- many characters/graphemes look similar or identical
- enables impersonation: nicks, domains, localparts, etc.

Some Examples

same script (l vs. 1)	juliet	ju1iet
mixed script (Cyrillic P vs. Latin P)	Peter	Peter
whole script (Cherokee vs. Latin)	STPETER	STPETER

Solutions

- no straightforward solution
- possibilities:
  - restrict deployment to a single/few scripts
  - manually vet all identifiers
  - Unicode confusable detection

Unicode confusable detection

- create a “skeleton” of each string
- two strings with the same skeleton are considered confusable
- tables to map characters to confusable ones:
  single-script/mixed-script lowercase/anycase

Skeleton

skeleton(X) is defined as:

1. convert X to NFD
2. map each character in X according to the chosen table
3. apply NFD to the result

Examples

$$
\begin{aligned}
\text{skeleton}(\text{juliet}) &= \text{juliet} \\
\text{skeleton}(\text{ju1iet}) &= \text{juliet} \\
\text{skeleton}(\text{STPETER}) &= \text{STPETER} \\
\text{skeleton}(\text{STPETER}) &= \text{STPETER}
\end{aligned}
$$

1 About me
2 History
3 Encodings
4 Normalization
5 Confusables
6 Special Characters

ZWNJ

- Zero-Width non-joiner
- invisible in many situations
- used to break up ligatures

German	Auflage	Auf‌lage
Arabic	أيبيإم	أي‌بي‌إم
Persian	ميخواهم	می‌خواهم

ZWJ

- Zero-width joiner
- used to force ligatures, or variants

Ba	ﺏ
ZWJ - Ba	ﺐ
Ba - ZWJ	ﺑ
ZWJ - Ba - ZWJ	ﺒ

Half-/Fullwidth characters

- CJK computing traditionally uses CJK characters
  twice as wide (fullwidth) as other characters halfwidth)
- printable ASCII reproduced in a fullwidth form
- some Katakana and Hangeul reproduced in a haflwidth form

Thank you

Questions?