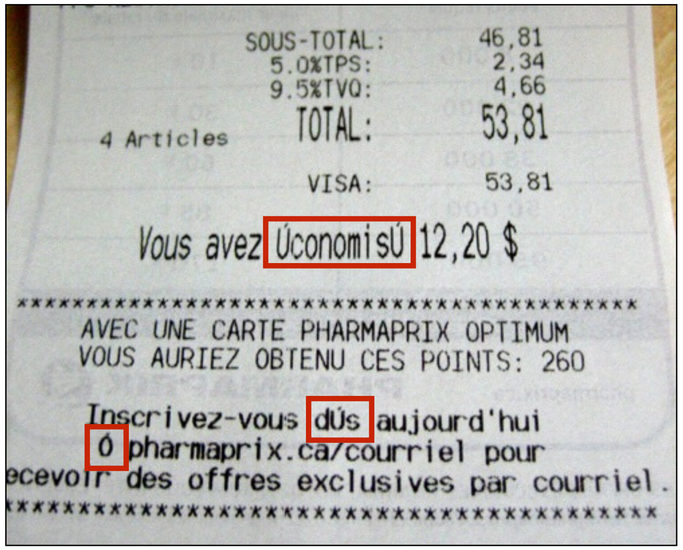


http://unicode-wall-of-shame.com

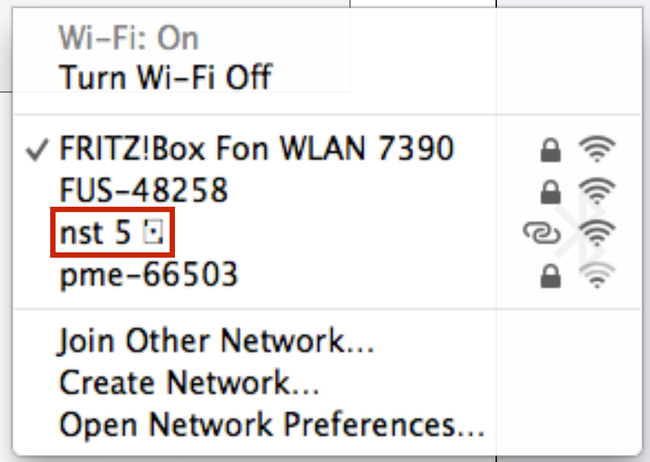
Unicode Hacks

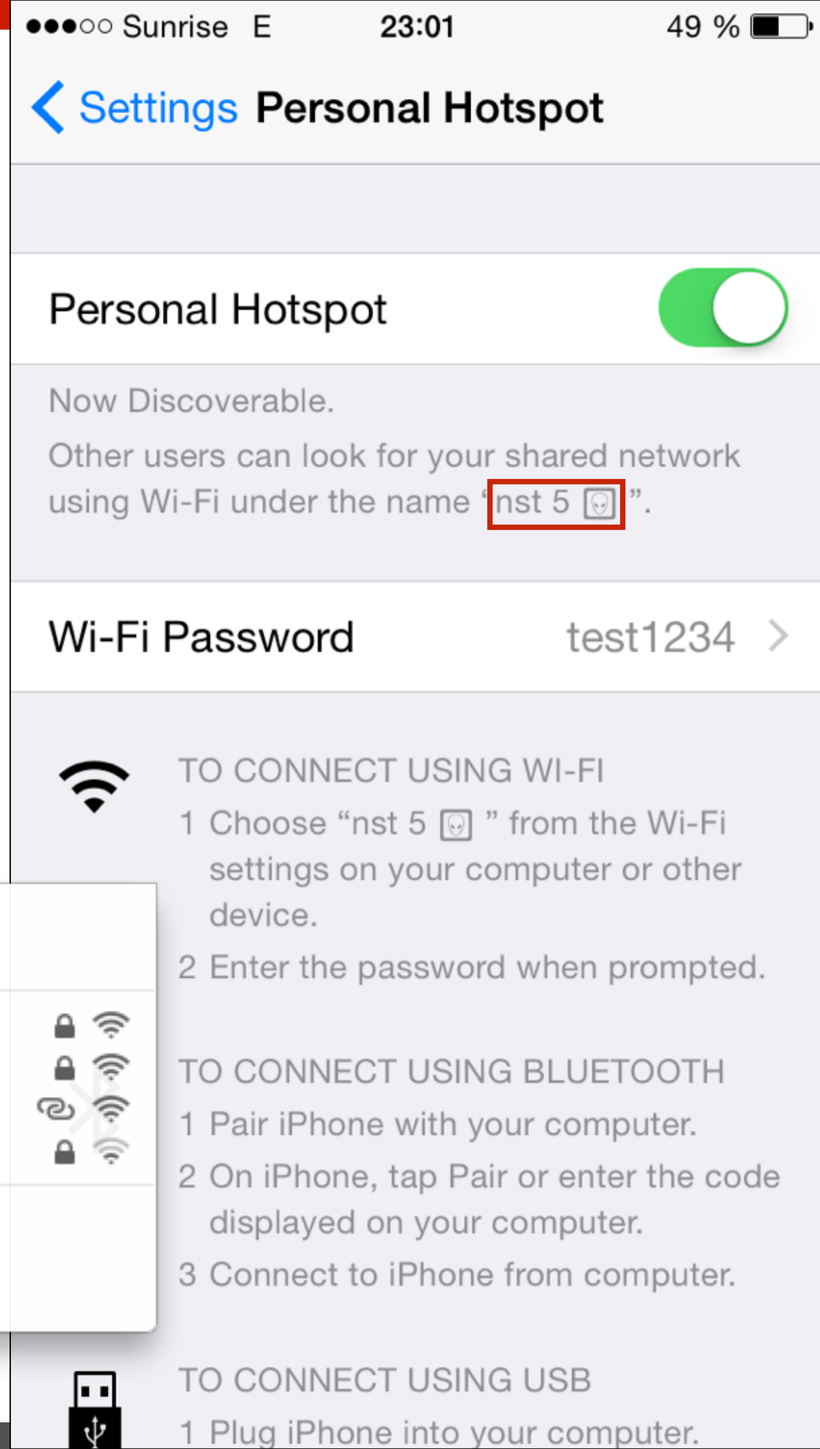
Nicolas Seriot

November 6th, 2014

Application Security Forum - 2014
Western Switzerland

5-6 novembre 2014
Y-Parc / Yverdon-les-Bains





- full presentation at SoftShake
- 10 min. / 38 slides –> 15.8 s. / slide
- an article is coming…

reddit PROGRAMMING comments related other discussions (1)

1149 I ❖ Unicode (seriot.ch)
submitted 9 days ago by nst021
554 comments share save hide delete nsfw retry thumb

Y Hacker News new | threads | comments | show | ask | jobs | submit

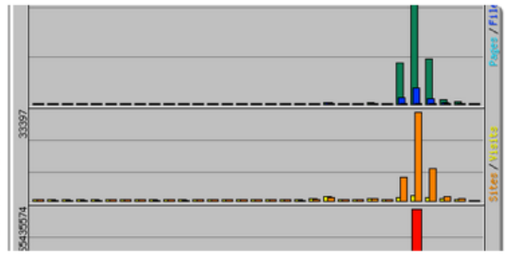
* I ❖ Unicode [pdf] (seriot.ch)
98 points by beefburger 10 days ago | comments

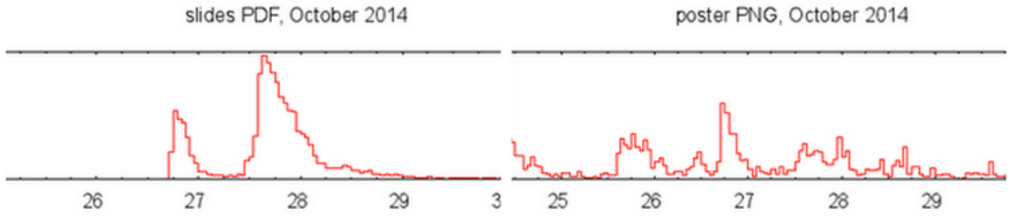
Nicolas Seriot
@nst021

Weird, huge stats for my Unicode slides.
1052k downloads, 59k unique (ip,day,agent).

Top 10 of 59088 Total Sites By KBytes

#	Hits		Files		KBytes		Visits		Hostname
1	1170	0.10%	1114	0.66%	4378369	0.89%	7	0.03%	199.16.156.124
2	1119	0.10%	1059	0.62%	4242725	0.86%	9	0.04%	199.16.156.126
3	1189	0.10%	1125	0.66%	4237841	0.86%	5	0.02%	199.16.156.125
4	540	0.05%	540	0.32%	4022767	0.81%	0	0.00%	ec2-54-185-96-103.us-west-2.compute.amazonaws.com
5	528	0.05%	528	0.31%	3933372	0.80%	0	0.00%	ec2-54-202-244-76.us-west-2.compute.amazonaws.com
6	528	0.05%	528	0.31%	3933372	0.80%	0	0.00%	ec2-54-244-154-122.us-west-2.compute.amazonaws.com
7	522	0.05%	522	0.31%	3888675	0.79%	0	0.00%	ec2-54-184-38-108.us-west-2.compute.amazonaws.com
8	519	0.04%	519	0.31%	3866326	0.78%	0	0.00%	ec2-54-212-245-232.us-west-2.compute.amazonaws.com
9	510	0.04%	510	0.30%	3799280	0.77%	0	0.00%	ec2-54-245-0-218.us-west-2.compute.amazonaws.com
10	508	0.04%	508	0.30%	3784381	0.77%	0	0.00%	ec2-54-212-229-209.us-west-2.compute.amazonaws.com





2:38 PM - 31 Oct 2014

Baudot Code

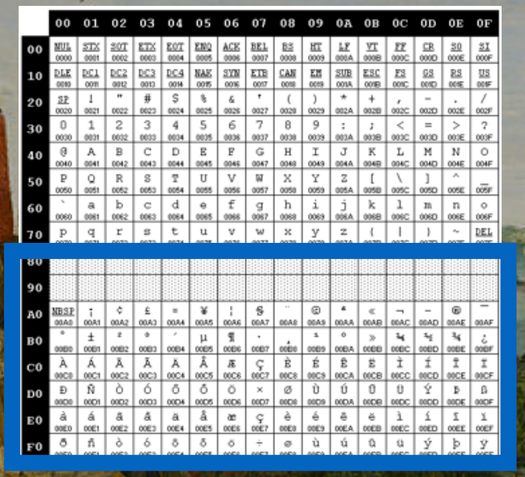
BCD

EBCDIC

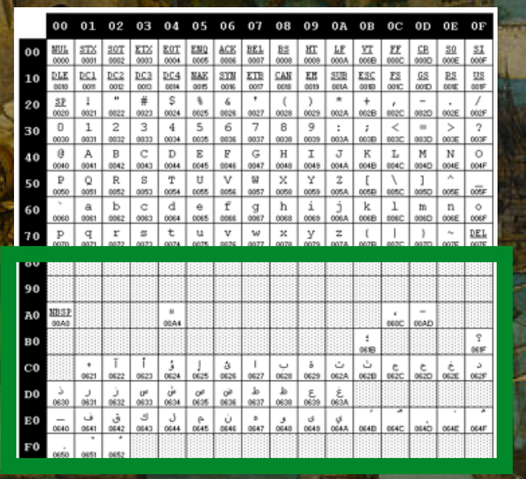
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d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

1963: ASCII

1990's: 8 bit encodings



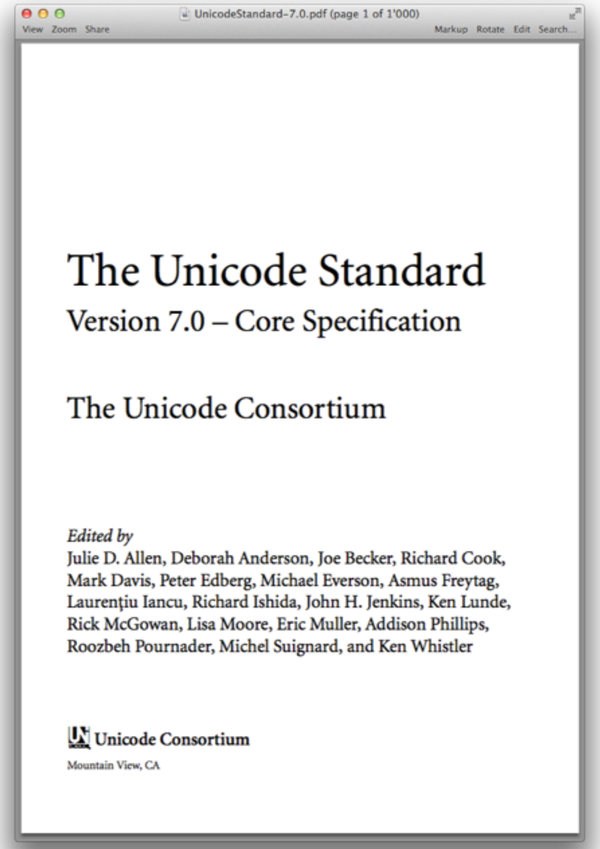
ISO/IEC 8859-1 (Latin 1)

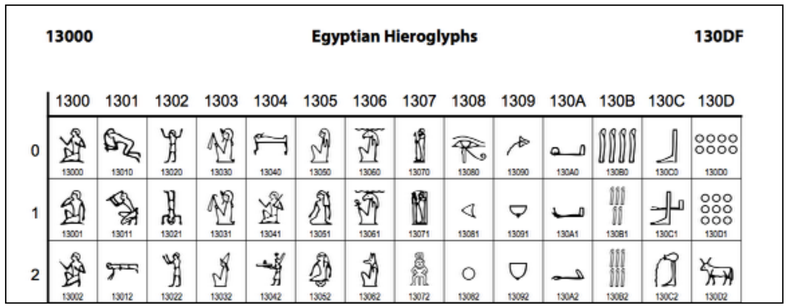


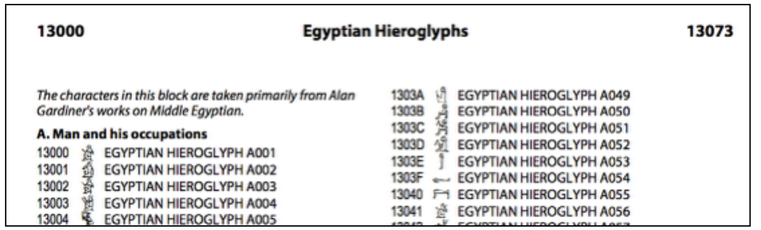
ISO/IEC 8859-6 (Arabic)

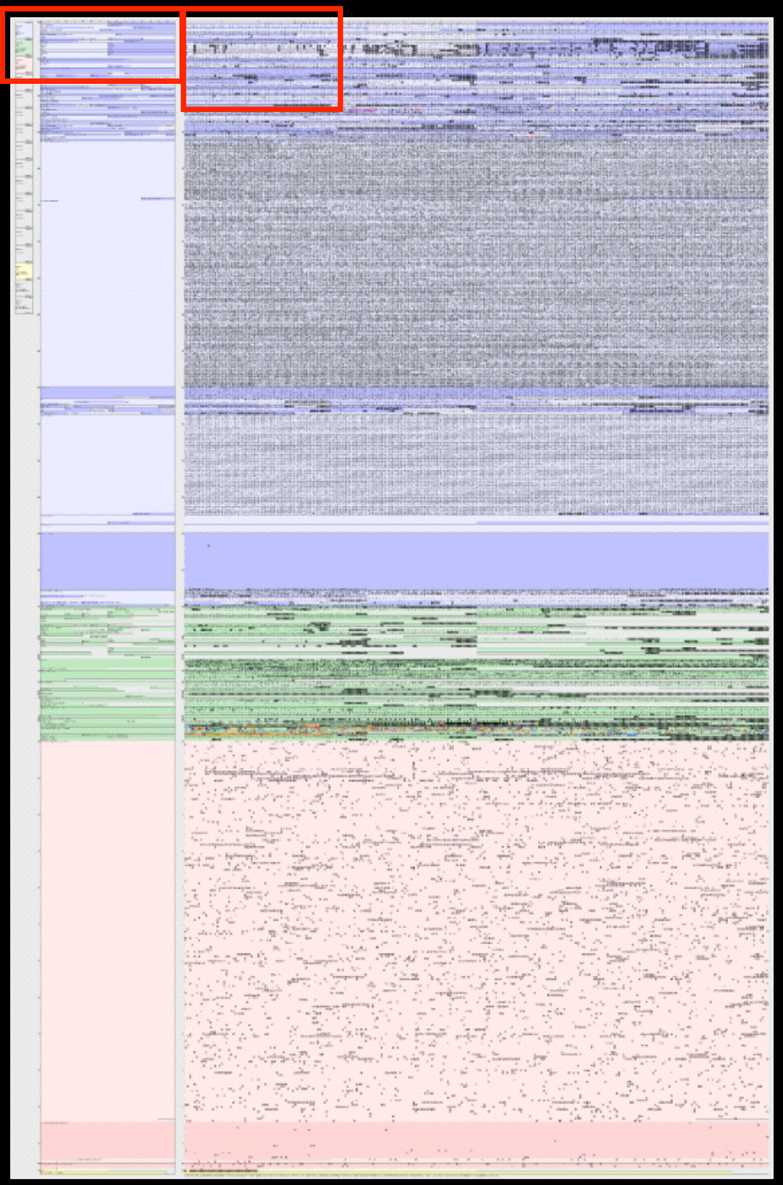
The Unicode Consortium

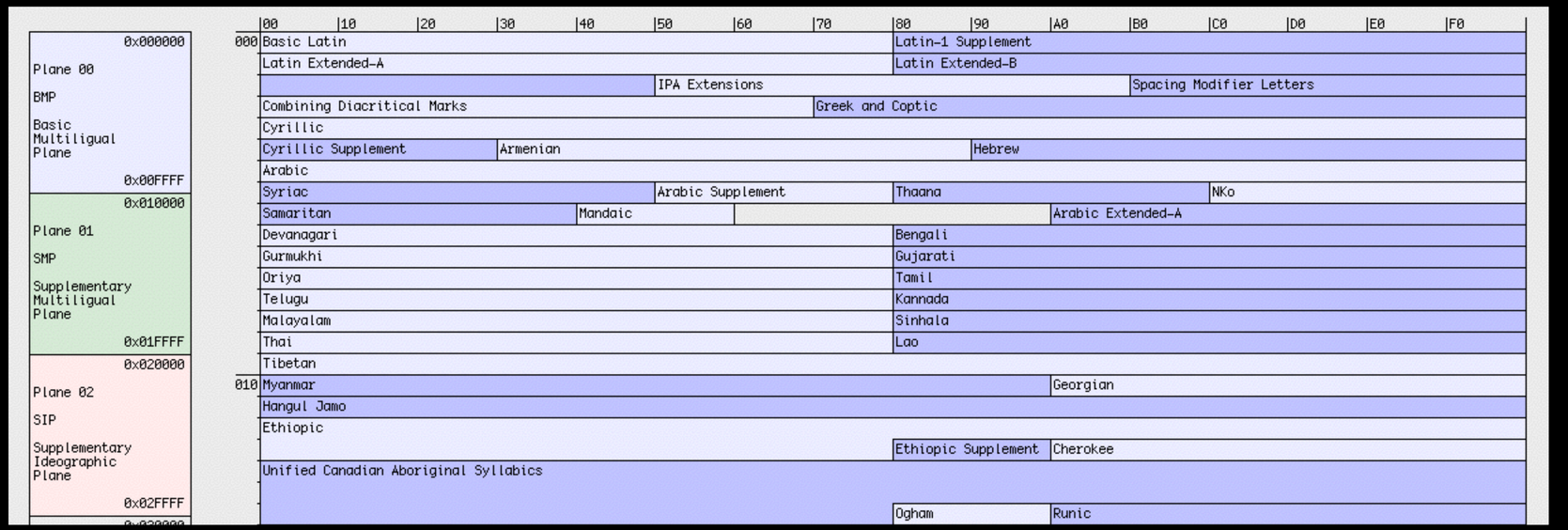
Microsoft YAHOO! ORACLE IBM SAP Google Adobe intel

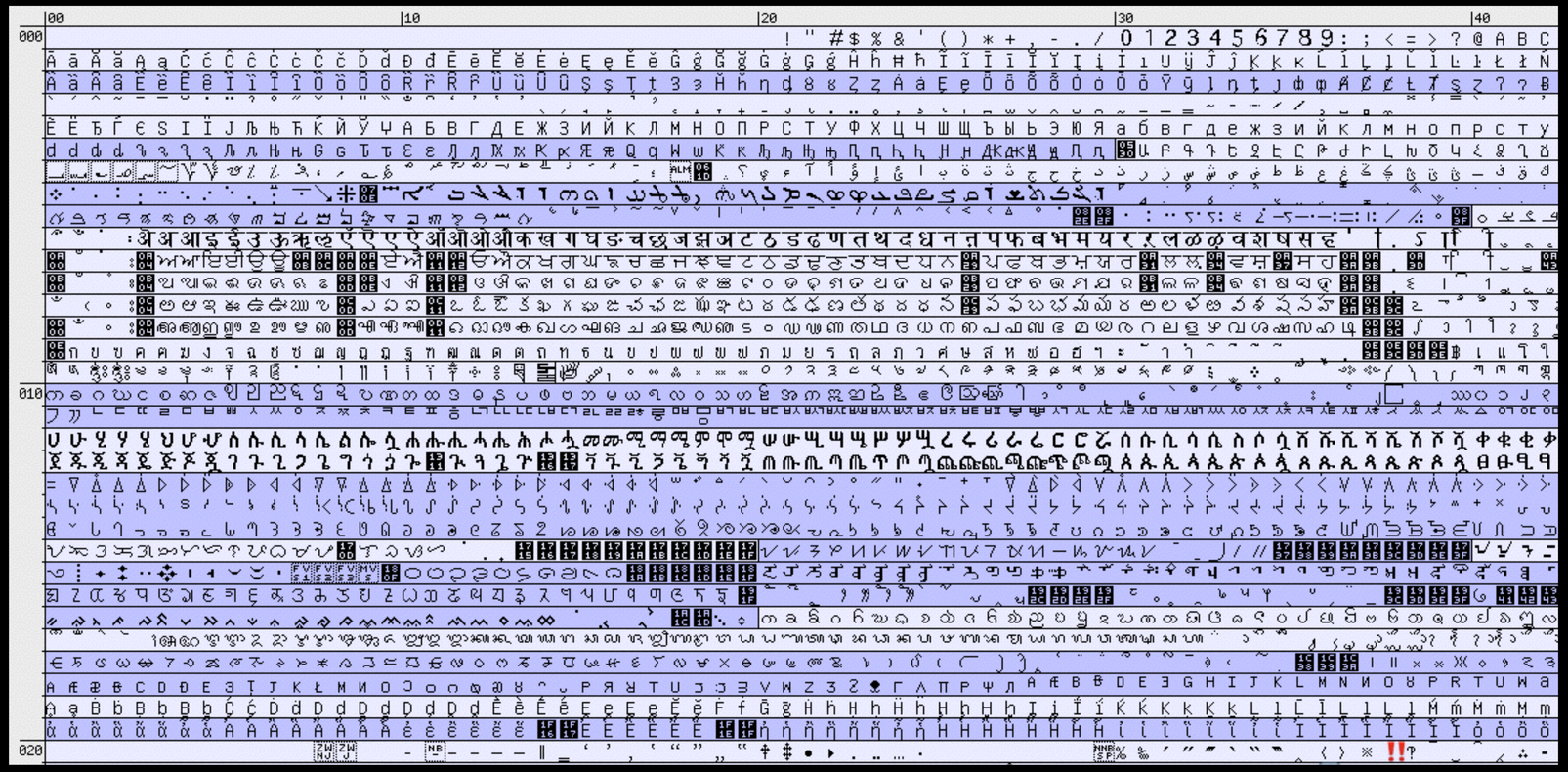


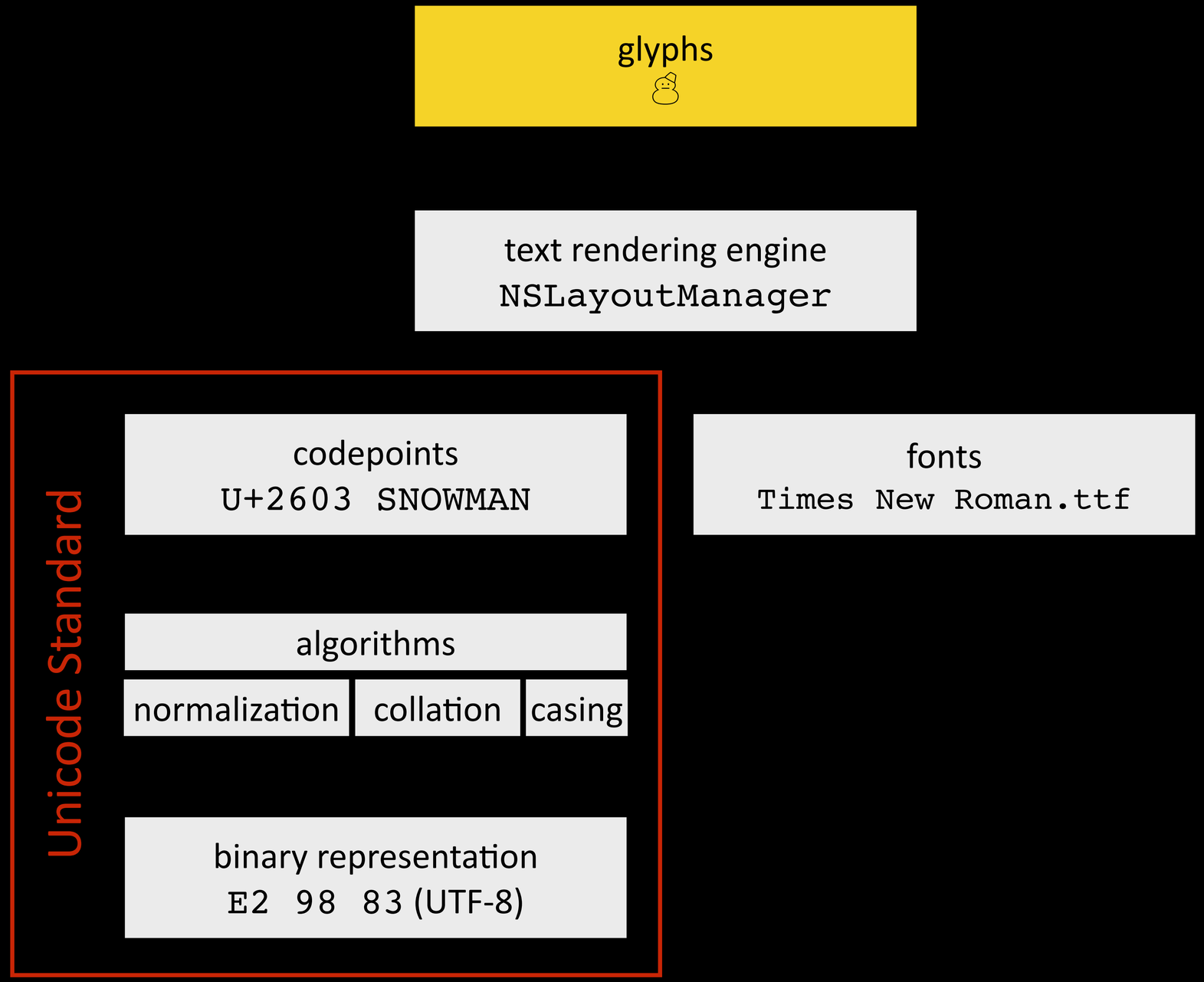







0x000000
Plane 00
BMP
Basic Multiligual Plane
0x00FFFF
0x010000
Plane 01
SMP
Supplementary Multiligual Plane
0x01FFFF
0x020000
Plane 02
SIP
Supplementary Ideographic Plane
0x02FFFF
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
000 Basic Latin
Latin-1 Supplement
Latin Extended-A
Latin Extended-B
IPA Extensions
Spacing Modifier Letters
Combining Diacritical Marks
Greek and Coptic
Cyrillic
Cyrillic Supplement
Armenian
Hebrew
Arabic
Syriac
Arabic Supplement
Thaana
NKo
Samaritan
Mandaic
Arabic Extended-A
Devanagari
Bengali
Gurmukhi
Gujarati
Oriya
Tamil
Telugu
Kannada
Malayalam
Sinhala
Thai
Lao
Tibetan
010 Myanmar
Georgian
Hangul Jamo
Ethiopic
Ethiopic Supplement
Cherokee
Unified Canadian Aboriginal Syllabics
Ogham
Runic


00 10 20 30 40
000
010
020


glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)
fonts
Times New Roman.ttf

Visual Similarities

AA A A Δ ᴀ A

www.google.com – U+0067 LATIN SMALL LETTER G
www.google.com – U+0261 LATIN SMALL LETTER SCRIPT G

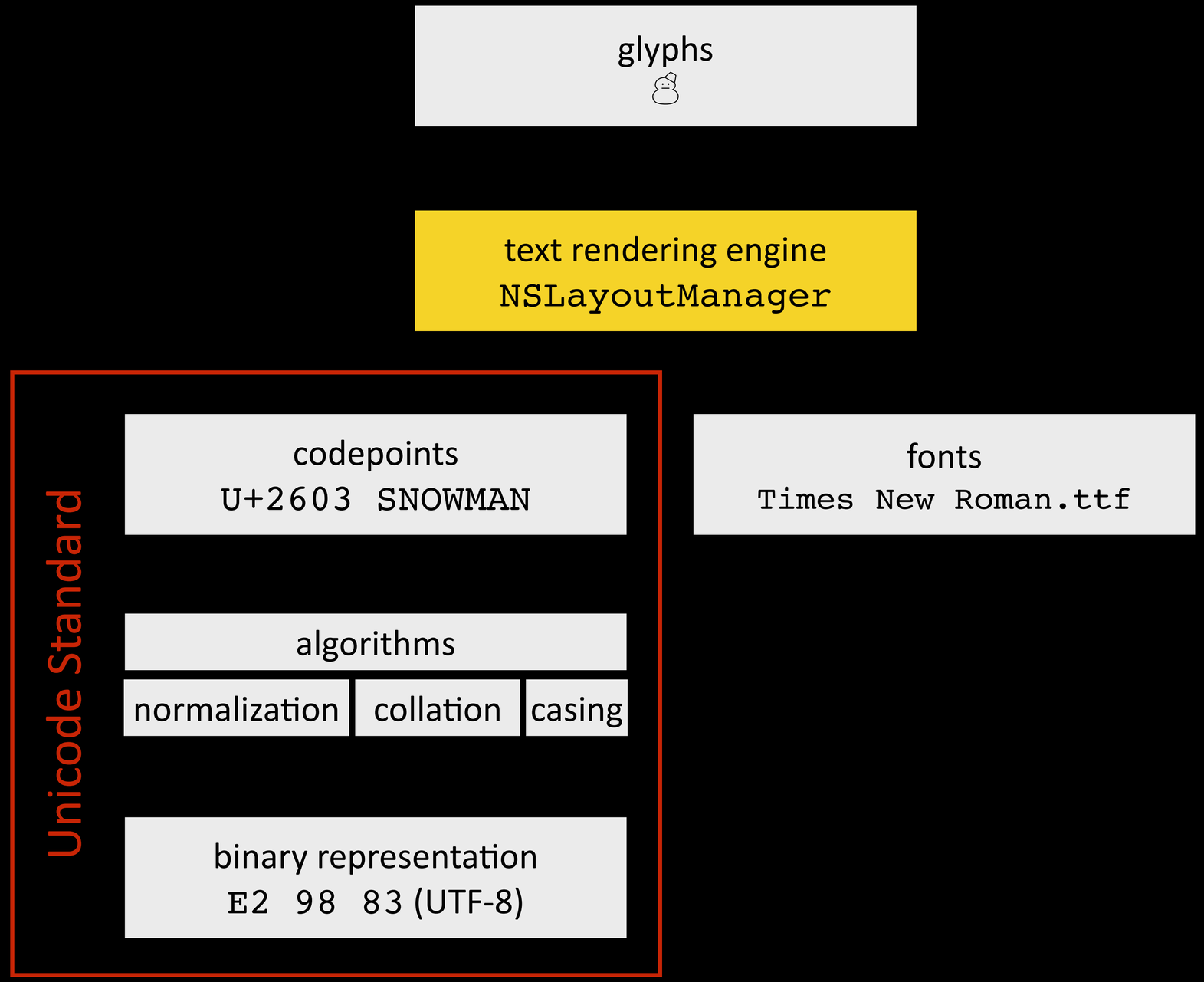
৪ – U+09EA BENGALI DIGIT FOUR
୨ – U+0B68 ORIYA DIGIT TWO

Country Flags

U+1F1E6 + U+1F1E7	🇦🇧
U+1F1E8 + U+1F1F3	🇨🇳
U+1F1E9 + U+1F1EA	🇩🇪
U+1F1EA + U+1F1F8	🇪🇸
U+1F1EB + U+1F1F7	🇫🇷
U+1F1EC + U+1F1E7	🇬🇧
U+1F1EE + U+1F1F9	🇮🇹
U+1F1EF + U+1F1F5	🇯🇵
U+1F1F0 + U+1F1F7	🇰🇷
U+1F1F7 + U+1F1FA	🇷🇺
U+1F1FA + U+1F1F8	🇺🇸

Bi-directinal Text

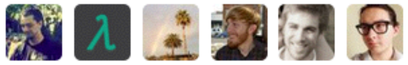
```
# U+202E RIGHT-TO-LEFT OVERRIDE
# double click a .jpg, open an .exe
$ python3 -c "print('s\u202Egpj.exe')"
sexe.jpg
```


glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)
fonts
Times New Roman.ttf

Nicolas Seriot
@nst021

Here is a nice little Core Text crasher for OS X: $ python -c "print u'\u0647\u0020\u0488\u0488\u0488'"

RETWEETS 38 FAVORITES 34



10:49 AM - 25 Mar 2013

```
$ gdb Twitter

(gdb) r
Starting program: /Applications/Twitter.app/Contents/MacOS/Twitter

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x00000001084e8008
0x00007fff9432ead2 in vDSP_sveD ()

(gdb) bt
#0  0x00007fff9432ead2 in vDSP_sveD ()
#1  0x00007fff934594fe in TStorageRange::SetStorageSubRange ()
#2  0x00007fff93457d5c in TRun::TRun ()
#3  0x00007fff934579ee in CTGlyphRun::CloneRange ()
#4  0x00007fff93466764 in TLine::SetLevelRange ()
#5  0x00007fff93467e2c in TLine::SetTrailingWhitespaceLevel ()
#6  0x00007fff93467d58 in TRunReorder::ReorderRuns ()
#7  0x00007fff93467bfe in TTypesetter::FinishLineFill ()
#8  0x00007fff934858ae in TFramesetter::FrameInRect ()
#9  0x00007fff93485110 in TFramesetter::CreateFrame ()
#10 0x00007fff93484af2 in CTFramesetterCreateFrame ()
...
```

ars technica

MAIN MENU · MY STORIES: 25 · FORUMS · SUBSCRIBE · JOBS

INFINITE LOOP / THE APPLE ECOSYSTEM

Rendering bug crashes OS X, iOS apps with string of Arabic characters (Updated)

CoreText bug crashes any iOS 6 and OS X programs that use the API.

by Andrew Cunningham and Dan Goodin Aug 29 2013, 9:30pm CEST

Share Tweet 149

LATEST FEATURE STORY

FEATURE STORY (1 PAGE)

The Register®

Biting the hand that feeds IT

Data Centre · Software · Networks · Security · Business · Hardware · Science · Bootnotes · Video · Forums · Weekend Edition · Search site

Operating Systems · Applications · Developer · Verity Stob

SOFTWARE > OPERATING SYSTEMS

Anatomy of a killer bug: How just 5 characters can murder iPhone, Mac apps

What evil lurks in the Unicode of Death ... oh, a buffer overrun

By Chris Williams, 4 Sep 2013

93

RELATED STORIES

Analysis There has been much sniggering into sleeves after wags found they could upset iOS 6 iPhones and iPads, and Macs running OS X 10.8, by sending a simple rogue text message or email.

A bug is triggered when the CoreText component in vulnerable Apple operating

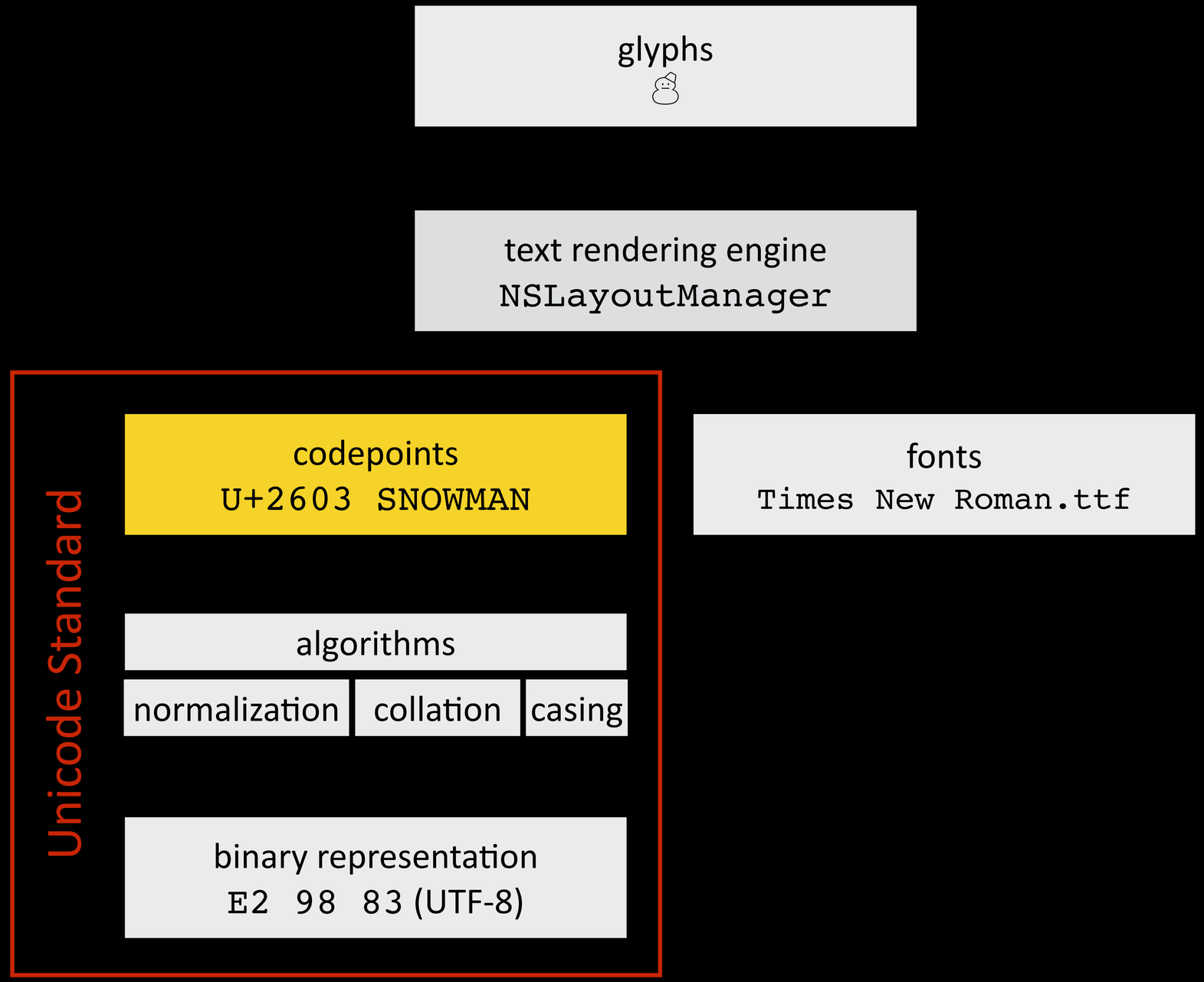
MOST READ · MOST COMMENTED

- YARR! Pirates walk the plank: DMCA magnets sink in Google results
- Whisper tracks its users. So we tracked down its LA office. This is what happened next
- Xperia Z3: Crikey, Sony – ANOTHER flagship phondleslab?
- Ex-US Navy fighter pilot MIT prof: Drones beat humans - I should know
- Apple flings iOS 8.1 at world+dog: Our AMAZEBALLS 9-step installation guide

OS X Finder

```
$ echo -e "\xFF\xFE" > x.txt # UTF-16LE BOM
$ xattr -w com.apple.TextEncoding "utf-16le" x.txt
$ qlmanage -p x.txt # or QuickLook with Finder
```

```
[ERROR] An uncaught exception was raised outside of any generator: *** -[NSConcreteTextStorage attribute:atIndex:longestEffectiveRange:inRange:]: Range or index out of
bounds
2014-10-24 10:53:08.474 qlmanage[5268:11f] *** Terminating app due to uncaught exception 'NSRangeException', reason: '*** -[NSConcreteTextStorage
attribute:atIndex:longestEffectiveRange:inRange:]: Range or index out of bounds'
*** First throw call stack:
(
      0   CoreFoundation                      0x00007fff89ebe25c __exceptionPreprocess + 172
      1   libobjc.A.dylib                     0x00007fff87934e75 objc_exception_throw + 43
      2   CoreFoundation                      0x00007fff89ebe10c +[NSException raise:format:] + 204
      3   AppKit                              0x00007fff81a83a7a -[NSConcreteTextStorage attribute:atIndex:longestEffectiveRange:inRange:] + 118
      4   AppKit                              0x00007fff81951ded -[NSMutableAttributedString(NSMutableAttributedStringKitAdditions) fixGlyphInfoAttributeInRange:] + 204
      5   AppKit                              0x00007fff81951cd8 -[NSMutableAttributedString(NSMutableAttributedStringKitAdditions) fixAttributesInRange:] + 39
      6   AppKit                              0x00007fff81a838e1 -[NSTextStorage processEditing] + 109
      7   AppKit                              0x00007fff81a7f742 -[NSTextStorage endEditing] + 110
      8   AppKit                              0x00007fff81c5db4f _NSReadAttributedStringFromURLOrData + 14525
      9   AppKit                              0x00007fff81c5e3a5 -[NSAttributedString(NSAttributedStringKitAdditions) initWithURL:options:documentAttributes:
```


glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
fonts
Times New Roman.ttf
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)

Weird Code Points May Bypass Filters

- Non-characters: eg. `U+FFFE, U+FFFF, U+1FFFE, U+10FFFF`
  Unassigned code points: eg. `U+2073`

- Must not be **deleted** (as allowed by Unicode < 5.2 C7) but **replaced** with `U+FFFD REPLACEMENT CHARACTER`.

```
<a href="java\uFEFFscript:alert("XSS")>
```

Non-Characters and OS X Bash / HFS+

```
$ mkdir /tmp/test
$ cd /tmp/test
$ touch `printf "a\xef\xbb\xbfb"`
# or "a\uFFFEb".encode('utf-8')
# which is a non-character
$ ls a*
a?b
$ touch ab
$ ls a*
a?b
# where did ab go?!
```

Regex

```
$ python3
>>> import re
>>> reg = re.compile("\d")
>>> gen = ( chr(c) for c in range(0, 0xFFFF) if re.match(reg, chr(c)) )
>>> print(''.join(gen))
0123456789٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹߀߁߂߃߄߅߆߇߈߉०१२३४५६७८९০১২৩৪৫৬৭৮৯੦੧੨੩੪੫੬੭੮੯૦૧૨૩૪૫૬૭૮૯୦୧୨୩୪୫୬୭୮୯௦௧௨௩௪௫௬௭௮௯౦౧౨౩౪౫౬౭౮౯೦೧೨೩೪೫೬೭೮೯൦൧൨൩൪൫൬൭൮൯๐๑๒๓๔๕๖๗๘๙໐໑໒໓໔໕໖໗໘໙
၀၁၂၃၄၅၆၇၈၉႐႑႒႓႔႕႖႗႘႙꘠꘡꘢꘣꘤꘥꘦꘧꘨꘩꣐꣑꣒꣓꣔꣕꣖꣗꣘꣙꤀꤁꤂꤃꤄꤅꤆꤇꤈꤉꧐꧑꧒꧓꧔꧕꧖꧗꧘꧙꧰꧱꧲꧳꧴꧵꧶꧷꧸꧹꩐꩑꩒꩓꩔꩕꩖꩗꩘꩙꯰꯱꯲꯳꯴꯵꯶꯷꯸꯹
0123456789𐒠𐒡𐒢𐒣𐒤𐒥𐒦𐒧𐒨𐒩𑁦𑁧𑁨𑁩𑁪𑁫𑁬𑁭𑁮𑁯𑃰𑃱𑃲𑃳𑃴𑃵𑃶𑃷𑃸𑃹𑄶𑄷𑄸𑄹𑄺𑄻𑄼𑄽𑄾𑄿𑇐𑇑𑇒𑇓𑇔𑇕𑇖𑇗𑇘𑇙𑋰𑋱𑋲𑋳𑋴𑋵𑋶𑋷𑋸𑋹𑑐𑑑𑑒𑑓𑑔𑑕𑑖𑑗𑑘𑑙𑓐𑓑𑓒𑓓𑓔𑓕𑓖𑓗𑓘𑓙𑙐𑙑𑙒𑙓𑙔𑙕𑙖𑙗𑙘𑙙𑛀𑛁𑛂𑛃𑛄𑛅𑛆𑛇𑛈𑛉𑜰𑜱𑜲𑜳𑜴𑜵𑜶𑜷𑜸𑜹𑣠𑣡𑣢𑣣𑣤𑣥𑣦𑣧𑣨𑣩𑥐𑥑𑥒𑥓𑥔𑥕𑥖𑥗𑥘𑥙𑱐𑱑𑱒𑱓𑱔𑱕𑱖𑱗𑱘𑱙𑵐𑵑𑵒𑵓𑵔𑵕𑵖𑵗𑵘𑵙𑶠𑶡𑶢𑶣𑶤𑶥𑶦𑶧𑶨𑶩𖄰𖄱𖄲𖄳𖄴𖄵𖄶𖄷𖄸𖄹𖩠𖩡𖩢𖩣𖩤𖩥𖩦𖩧𖩨𖩩𖫀𖫁𖫂𖫃𖫄𖫅𖫆𖫇𖫈𖫉𖭐𖭑𖭒𖭓𖭔𖭕𖭖𖭗𖭘𖭙𖵰𖵱𖵲𖵳𖵴𖵵𖵶𖵷𖵸𖵹𖺀𖺁𖺂𖺃𖺄𖺅𖺆𖺇𖺈𖺉𝟎𝟏𝟐𝟑𝟒𝟓𝟔𝟕𝟖𝟗𝟘𝟙𝟚𝟛𝟜𝟝𝟞𝟟𝟠𝟡𝟢𝟣𝟤𝟥𝟦𝟧𝟨𝟩𝟪𝟫𞅀𞅁𞅂𞅃𞅄𞅅𞅆𞅇𞅈𞅉𞋰𞋱𞋲𞋳𞋴𞋵𞋶𞋷𞋸𞋹𞓰𞓱𞓲𞓳𞓴𞓵𞓶𞓷𞓸𞓹𞗱𞗲𞗳𞗴𞗵𞗶𞗷𞗸𞗹𞗺𞗻𞣇𞣈𞣉𞣊𞣋𞣌𞣍𞣎𞣏𞥐𞥑𞥒𞥓𞥔𞥕𞥖𞥗𞥘𞥙🄀🯰🯱🯲🯳🯴🯵🯶🯷🯸🯹
0 1 2 3 4 5 6 7 8 9
>>> reg = re.compile("\d", re.ASCII)
```

Regex

```
$ jsc
>>> /a.c/.test('abc')
true
>>> /a.c/.test('a🐛c')
false
>>> /a....c/.test('a🐛c')
true
```

glyphs

text rendering engine
`NSLayoutManager`

Unicode Standard

codepoints
`U+2603 SNOWMAN`

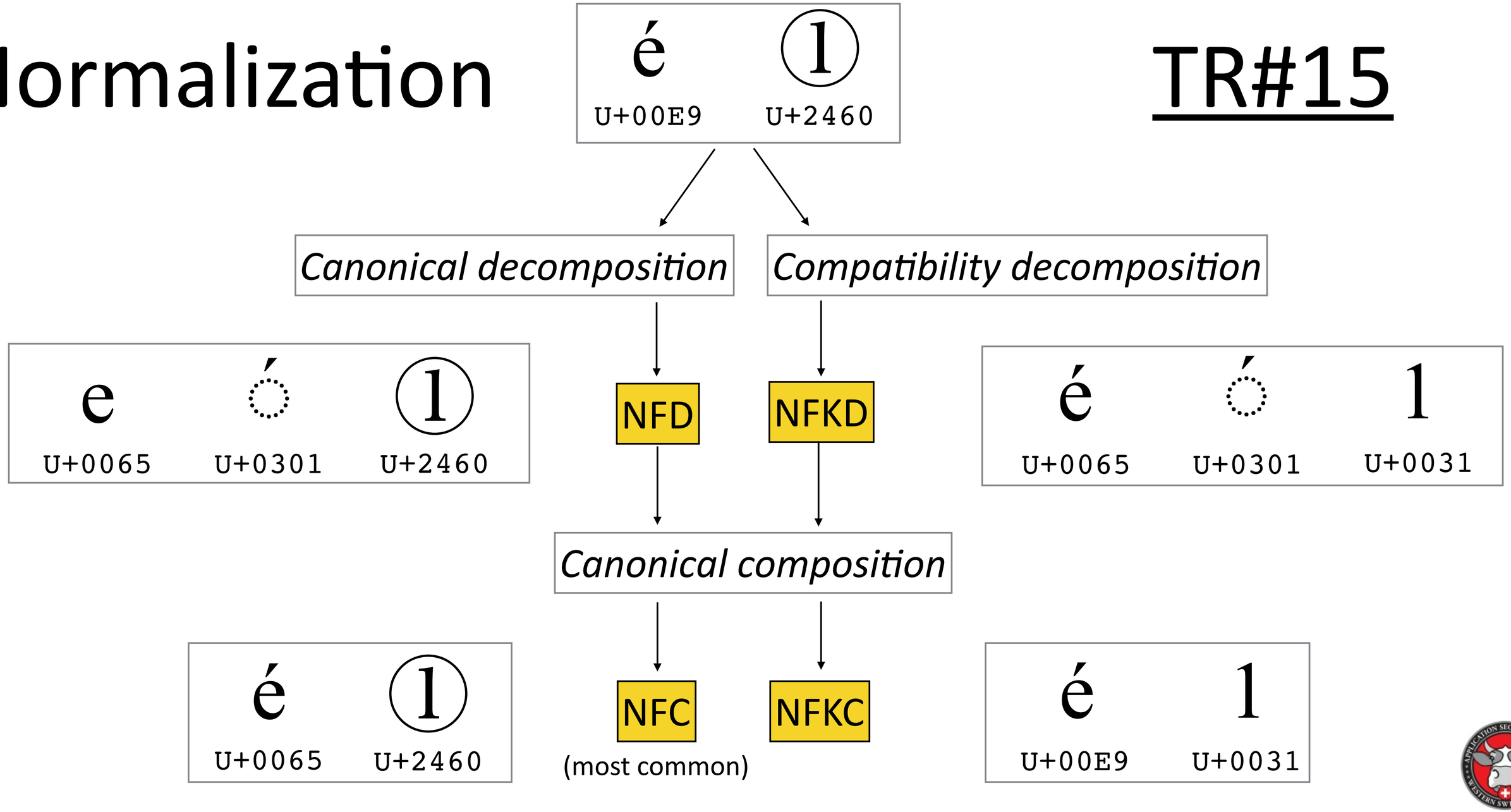
algorithms

normalization | collation | casing

binary representation
`E2 98 83` (UTF-8)

fonts
`Times New Roman.ttf`

Normalization



TR#15

NFC doesn’t Always Compose

שׁ U+FB2C HEBREW LETTER SHIN WITH DAGESH AND SHIN DOT

NFC(U+FB2C)

buffer overflow

ש U+05E9 HEBREW LETTER SHIN WITH DAGESH AND SHIN DOT + ◌ּ U+05BC HEBREW LETTER SHIN + ◌ׁ U+05C1 HEBREW LETTER SHIN DOT

NFKD Expands Up to 18x

U+FDFA
ARABIC
LIGATURE
SALLALLAHOU
ALAYHE
WASALLAM

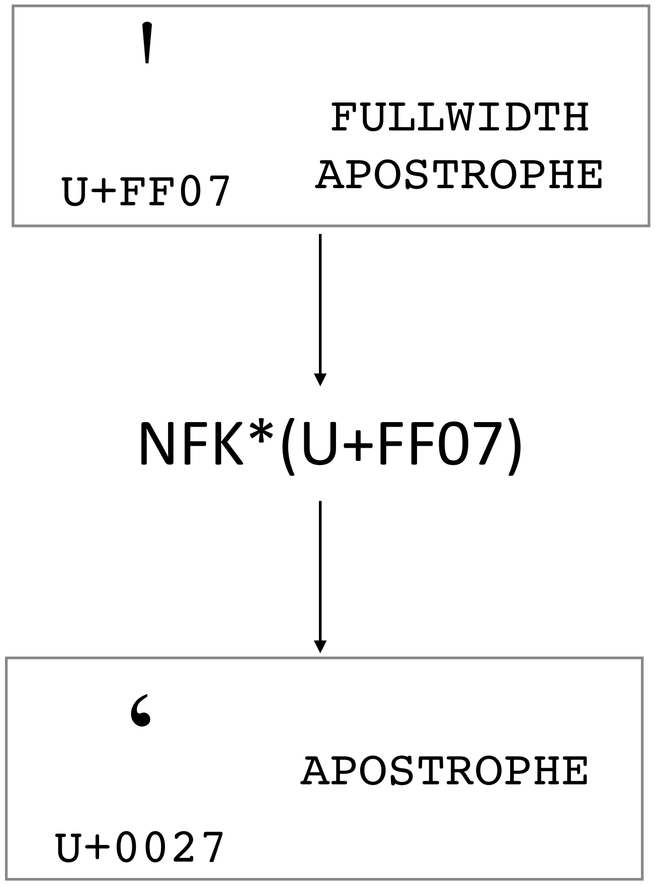
```
>>> import unicodedata

>>> s = '\uFDFA'
>>> len(s)
1

>>> s_nfkd = unicodedata.normalize('NFKD', s)
>>> s_nfkd.encode('unicode-escape')
b'\\u0635\\u0644\\u0649 \\u0627\\u0644\\u0644\\u0647 \\u0639\
\u0644\\u064a\\u0647 \\u0648\\u0633\\u0644\\u0645'
>>> len(s_nfkd)
18
```

buffer overflow

NFK* May Bypass Filters



SQL injection

Spotify Labs

Think it. Build it. Ship it. Tweak it. Blog it.

Search

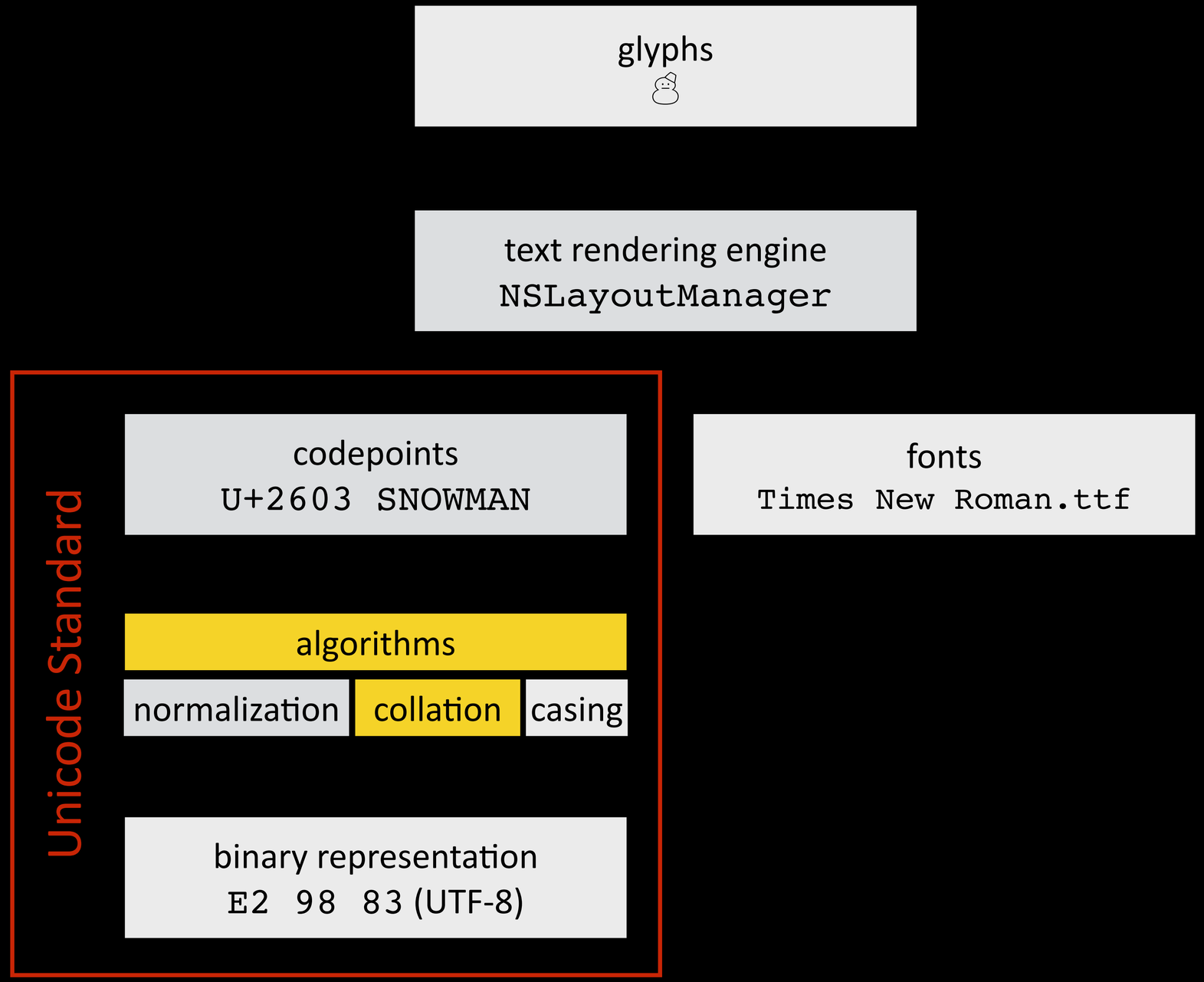
Home About Puzzles

Posted on June 18, 2013 by Mikael Goldmann

Creative usernames and Spotify account hijacking

1. Find a user account to hijack. For the sake of this example let us hijack the account belonging to user bigbird.
2. Create a new spotify account with username BIGBIRD (in python this is the string u'\u1d2e\u1d35\u1d33\u1d2e\u1d35\u1d3f\u1d30').
3. Send a request for a password reset for your new account.
4. A password reset link is sent to the email you registered for your new account. Use it to change the password.
5. Now, instead of logging in to account with username BIGBIRD, try logging in to account with username bigbird with the new password.
6. Success! Mission accomplished.

https://labs.spotify.com/2013/06/18/creative-usernames/


glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
fonts
Times New Roman.ttf
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)

Unicode Collation Algorithm – TR#10 (UTS)

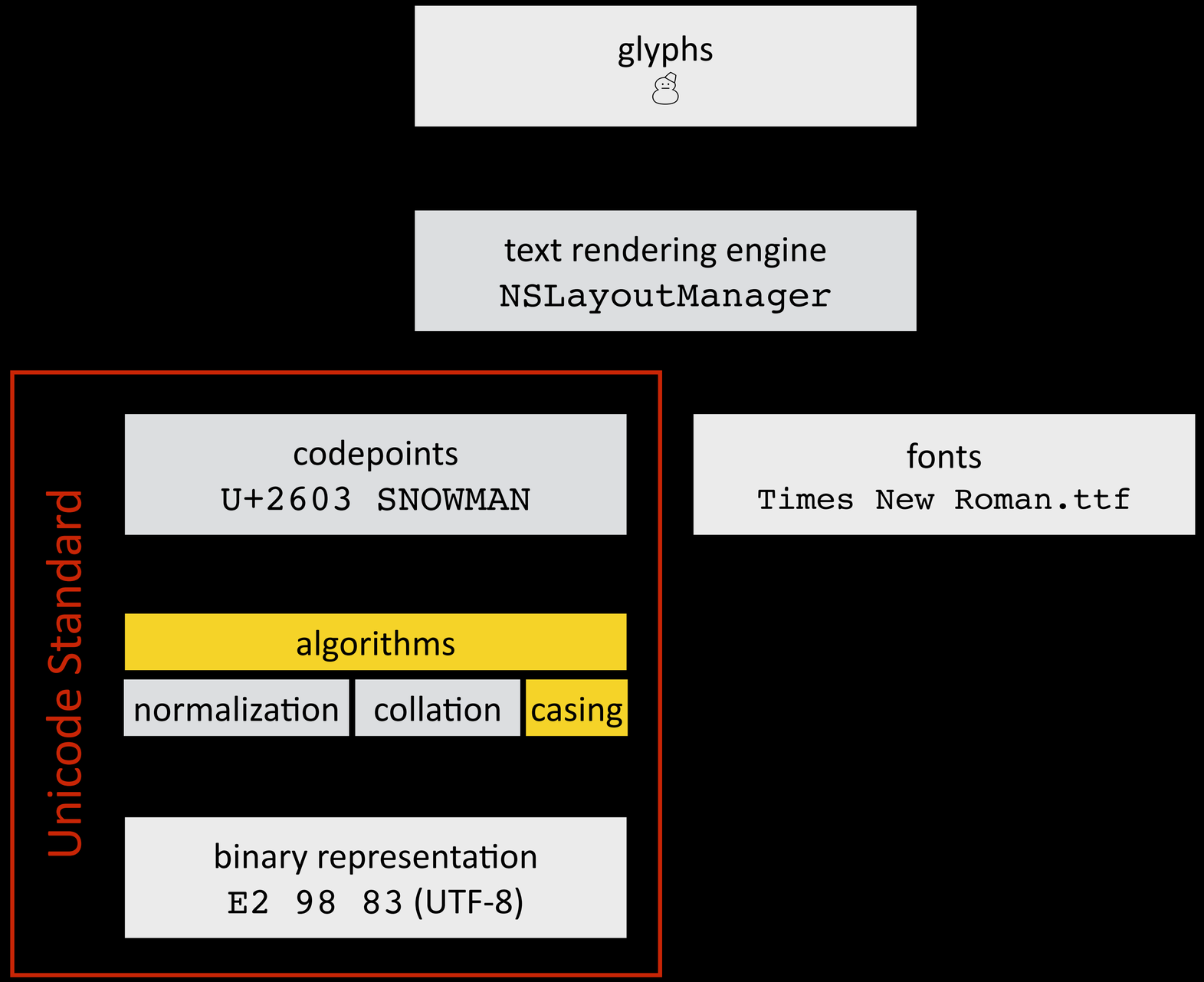
- **Text comparison**
  `café < cafe ?`
  `cafe < café ?`

- **Usage dependent**
  German dictionary: öf < of
  German phonebook: of < öf

- **Unstable over time**
  Sorted lists should be versioned

German			Swedish
Åkersberga	1	2	Alingsås
Alingsås	2	4	Oskarshamn
Äpplebo	3	7	Utting
Oskarshamn	4	6	Üttfeld
Östersund	5	8	Zwickau
Üttfeld	6	1	Åkersberga
Utting	7	3	Äpplebo
Zwickau	8	5	Östersund

(Steven R. Loomis, Mark Davis)

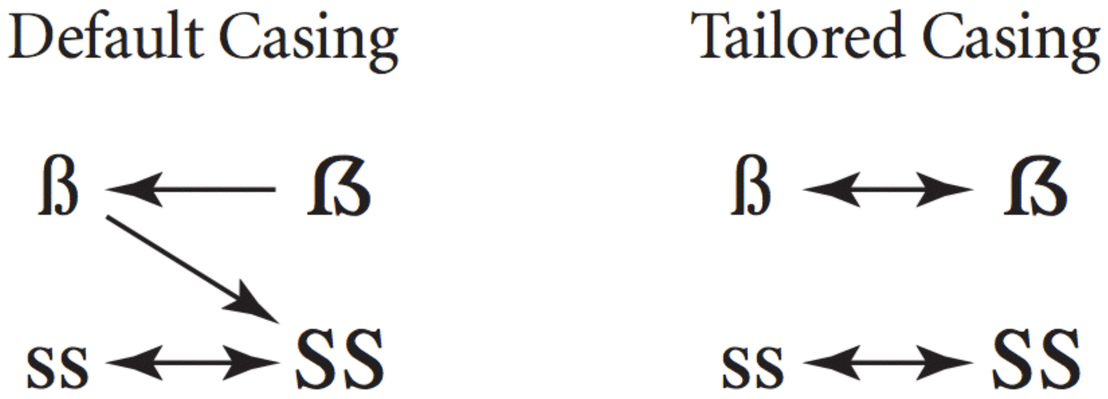

glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
fonts
Times New Roman.ttf
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)

Case Folding

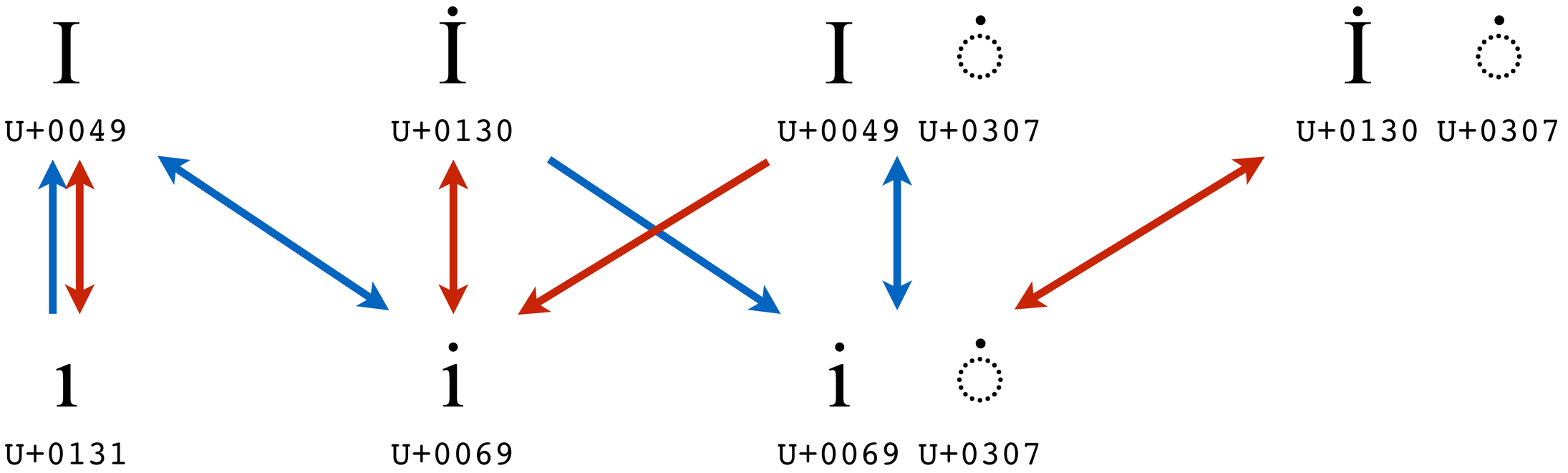
http://www.unicode.org/Public/UNIDATA/CaseFolding.txt

```
# The data supports both implementations that require simple case foldings
# (where string lengths don't change), and implementations that allow full case folding
# (where string lengths may grow). Note that where they can be supported, the
# full case foldings are superior: for example, they allow "MASSE" and "Maße" to match.
```

Figure 5-16. Casing of German Sharp S



Case Conversion



Posix Locale

Turkish Locale

Case Conversion – Locale

```
NSString *s = [NSString stringWithFormat:@"istambul"];

NSLocale *locale = [NSLocale localeWithLocaleIdentifier:@"tr_TR"];

NSString *s2 = [s uppercaseStringWithLocale:locale];

// İSTAMBUL ✅
```

Python 3

- ❌ Collation: still compare codepoints

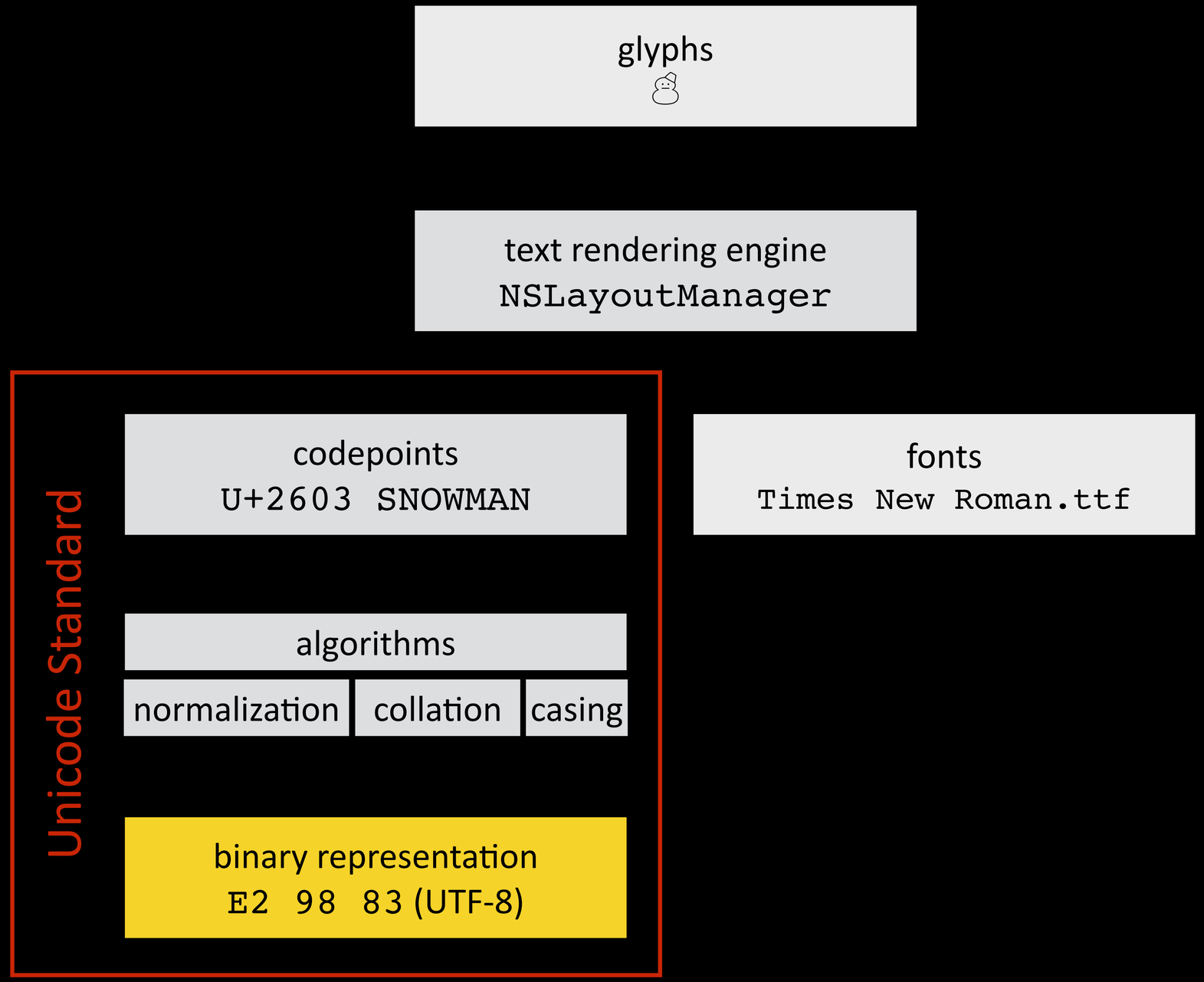
```
>>> 'café' < 'caff'
False
```

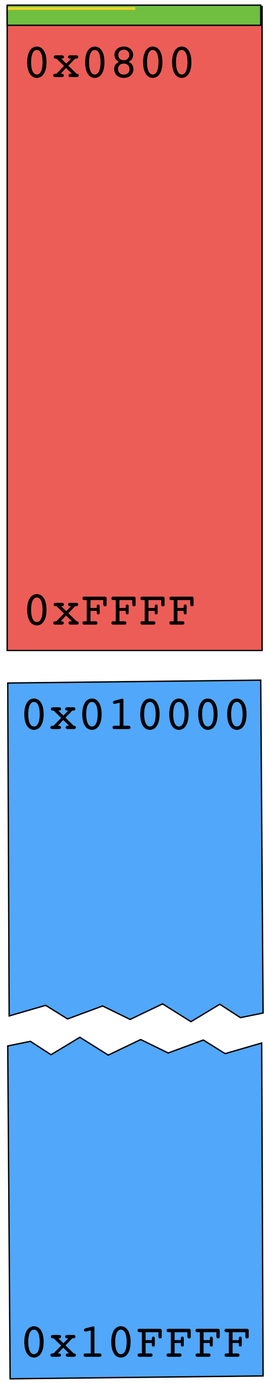
- ❌ Case Conversion restricted to 1:1 case mappings

```
>>> 'ß'.upper()
'ß'
```

- ❌ Case conversion ignores locale
  ❌ Additionaly, locale is global

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'tr_TR')
>>> s = "istanbul"
>>> s.upper()
'ISTANBUL'
```


glyphs
text rendering engine
NSLayoutManager
Unicode Standard
codepoints
U+2603 SNOWMAN
fonts
Times New Roman.ttf
algorithms
normalization
collation
casing
binary representation
E2 98 83 (UTF-8)



UTF-8

	Bits	Hex Min	Hex Max	Byte Sequence in Binary
1	7	00000000	0000007f	**0**vvvvvvv
2	11	00000080	000007FF	**110**vvvvv **10**vvvvvv
3	16	00000800	0000FFFF	**1110**vvvv **10**vvvvvv **10**vvvvvv
4	21	00010000	001FFFFF	**11110**vvv **10**vvvvvv **10**vvvvvv **10**vvvvvv
5	26	00200000	03FFFFFF	**111110**vv **10**vvvvvv **10**vvvvvv **10**vvvvvv **10**vvvvvv
6	31	04000000	7FFFFFFF	**1111110**v **10**vvvvvv **10**vvvvvv **10**vvvvvv **10**vvvvvv **10**vvvvvv

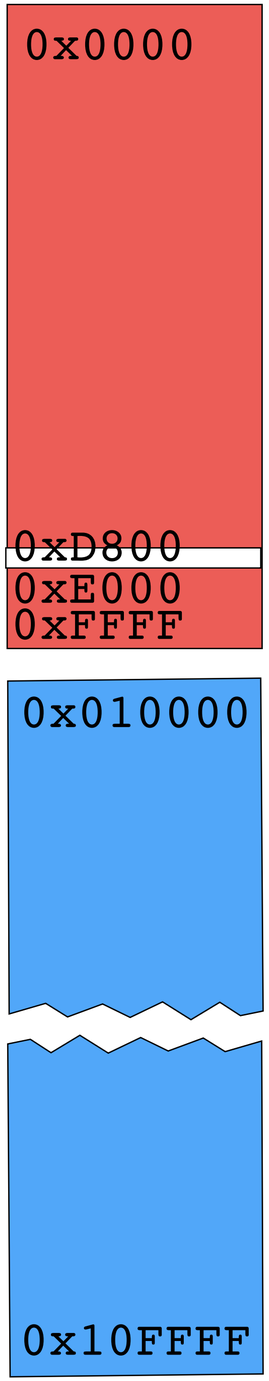
Malformed UTF-8 sequences include:

- overlong encoding, 0x1 on 2 bytes

11000000 **10**000001 0xC0 0x41

- unexpected continuation byte

11000000 00000000 0xC0 0x00



UTF-16

	Bits	Hex Min	Hex Max	Byte Sequence in Binary
2	16	00000000	0000FFFF	vvvvvvvv vvvvvvvv
4	21	00010000	001FFFFF	**110110**ww wwwwwwww **110111**ww wwwwwwww

www.. is (vvv.. - 0x10000) to map a 20 bits value

Malformed sequences include unpaired surrogates such as:

- **110110**ww wwwwwwww not followed by **110111**ww wwwwwwww
- **110111**ww wwwwwwww not preceded by **110110**ww wwwwwwww

Wide Characters

- Unicode code points were first defined on 16 bits (UCS-2)
- and now Java **`char`** / Objective-C **`unichar`** are 16 bits
- code points > `0xFFFF` defined as a pair of 16 bits values
- **`sizeof(wchar_t)`** is generally
  16 bits on Windows, 32 bits on Linux

Objective-C / Cocoa

```objc
NSString *s1 = @"abc";
NSString *s2 = @"\U0001F600bc";

NSLog(@"s1 %@", s1); // s1 abc
NSLog(@"s2 %@", s2); // s2 😀bc

NSLog(@"s1[0] -> %C", [s1 characterAtIndex:0]); // s1[0] -> a
NSLog(@"s2[0] -> %C", [s2 characterAtIndex:0]);
// nothing printed because
// s2 = [0xD83D, 0xDE00], and U+D83D is a high surrogate
// and NSLog() ignores nil strings
```

HFS+

Important: The terms used in this Q&A, precomposed and decomposed, roughly correspond to Unicode Normal Forms C and D, respectively. However, most volume formats do not follow the exact specification for these normal forms. For example, HFS Plus (Mac OS Extended) uses a variant of Normal Form D in which U+2000 through U+2FFF, U+F900 through U+FAFF, and U+2F800 through U+2FAFF are not decomposed (this avoids problems with round trip conversions from old Mac text encodings). It's likely that your volume format has similar oddities.

Apple Technical Q&A QA1173

HFS+

```
# what you write…
$ echo ü; echo ü | xxd
ü
0000000: c3bc 0a # NFC

# is not what you read
$ touch ü; ls; ls | xxd
ü
0000000: 75cc 880a # NFD
```

```
# watch your Finder go nuts!!!
$ cd; touch `printf "\x41\xe9"`
# NFC("Aé")
$ open .
# fixed in OS X 10.10
```

Conclusion

- Unicode is cool. Unicode is hard. Unicode is ubiquitous.
- How well do you know your framework of choice?
- Everything dealing with Unicode is a bug nest.
- Under-studied topic. Tons of low-hanging fruits.
- See Chris Weber's http://websec.github.io/unicode-security-guide/

« Unicode is just too complex to ever be secure. »
– Bruce Schneier, 2000