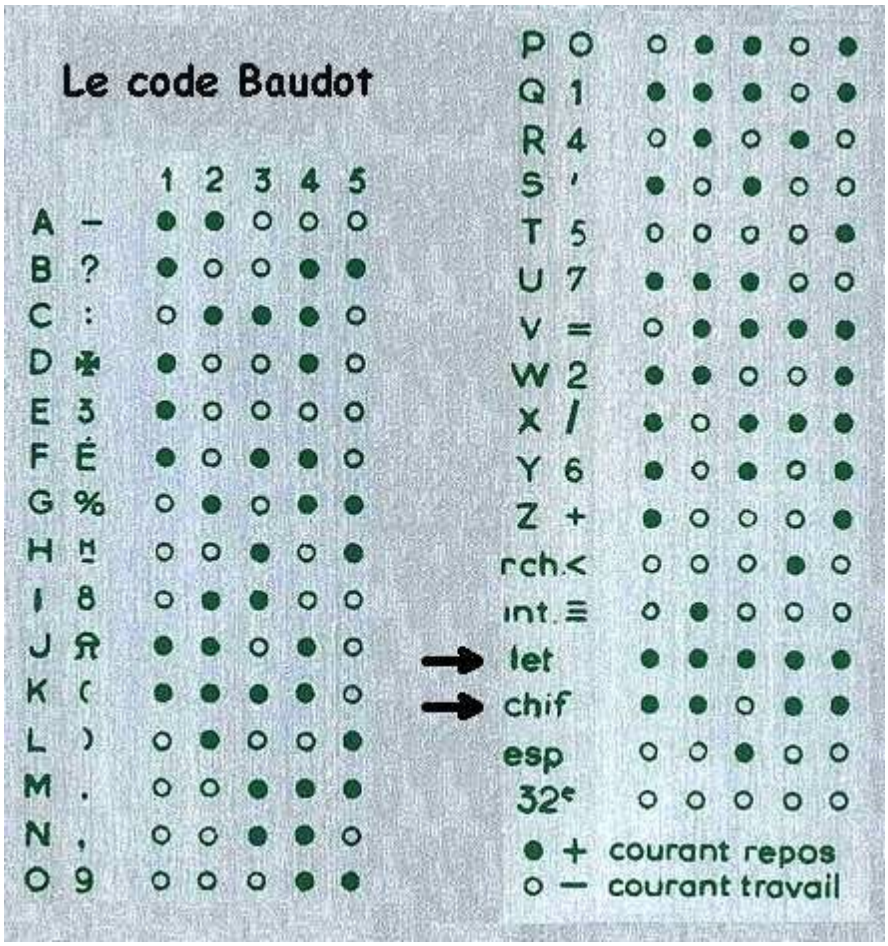
CT n°5 : Du Telex aux emojis

Les débuts de l'alphabet en binaire : (oh non!!!! pas encore le binaire!!!)

Comme son arrivée est un peu floue, on va commencer par l'étape marquante du Telex dans les années 30 et faisant appel à un code inventé 55 ans avant : le code Baudot.

Initialement il s'agissait d'une version très rudimentaire:
5 touches pouvant être utilisées simultanément et en un sens, encodant (avant l'heure) le texte sur 5 espèces de bits et permettant 32 combinaisons de commandes.



Je ne détaillerai pas son fonctionnement mais sa particularité est d'augmenter artificiellement le nombre de caractères et donc de combinaisons en utilisant 2 tables de correspondance dont le code est indiqué au début de la communication, puis au changement de table.

Ce changement de table est particulièrement contraignant d'autant plus qu'il est très fréquent.

illustration du principe:

LET: BONJOUR
CHIF: ,
(ESP)
LET: J
CHIF: '
LET: AI
(ESP)
CHIF: 30
(ESP)
LET: ANS

Ouais, c'est particulièrement chiant. Mais + rapide que le Morse.

Arrive donc en 1968 l'ASCII (*American Standard Code for Information Interchange*) codé sur 7 bits et permettant 128 combinaisons.

Cette norme gère ainsi les majuscules, les minuscules, les chiffres et la ponctuation sur une seule et même table :

USASCII code chart

b_7 b_6 b_5 → / Bits ↓					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
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

ce qui en lisible donne :
100 0001 pour A (65 en binaire)
100 0010 pour B (66 en binaire)
etc

En gros c'est facile, vous ignorez le plus gros bit (celui du début) et vous tombez sur la position dans l'alphabet, codé en binaire)

on saute ensuite à
110 0001 pour a
110 0010 pour b

Là aussi c'est facile, vous ignorez les 2 plus gros bits (ceux du début) et vous tombez sur la position dans l'alphabet, codé en binaire.

On voit ce que fait la touche shift… elle remplace le 0 du 2e plus gros bit par un 1 pour faire passer de “A” à “a” ou le “!” à “1”

Et si on regarde un clavier QWERTY, on voit que le “1” et le “!” sont bien sur la même touche. **(enfin un début de concret)**

Pour rappeler le débat actuel sur le nouveau clavier, c’est de là que viennent les guillemets verticaux! **(encore du concret !)**

Pourquoi?
Parce qu’avoir 2 symboles (un pour ouvrir et un autre pour fermer les guillemets) c’était un caractère supplémentaire dont on pouvait se passer sans gêner la compréhension : donc optimisation.

Sinon plus largement, remarquez qu’il n’y a pas de caractères accentués (en même temps c’est des ricains)

Mais l’ASCII utilise 7 bits seulement, et les ordinateurs vont adopter un format: l’octet, contenant 8 bits.

Et c’est là le début des emmerdes… *jingle de drama*

Ce bit en plus, permet de créer une toute nouvelle table, vierge… Et que tout me monde va remplir selon ses propres besoins, à sa guise.

En Europe de l’ouest, on va y mettre ce dont on a besoin en Europe de l’ouest :

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_																
1_																
2_		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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	{	|	}	~	
8_																
9_																
A_		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B_	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Cette table s'appelle ISO8859-1 et vous la connaissez probablement sous le nom de "latin-1"

Mais les grecs, les russes, les scandinaves vont eux aussi la remplir selon leurs besoins.

On passe ainsi d'un standard à UNE PUTAIN DE CHIÉE DE STANDARDS

15 standards en 2001 mais 9 d'entre eux ont été introduits entre 1987 et 1989.

Évidemment j'ignore au passage les tables (ou charsets, character set) prévues pour les japonais, les indiens, les chinois, les thaïlandais…
(Un jour peut-être une mini-chronique pour expliquer comment on tape en chinois ou en arabe…)

Bref, nous sommes en 1989 et c'est déjà une jungle d'incompatibilité pour la base-même de nos communications.

Ce n'est pas si gênant que ça au quotidien quand la transmission se fait en faxant ses impressions.

Pour re-situer où nous sommes vis- à vis du déploiement d'internet:

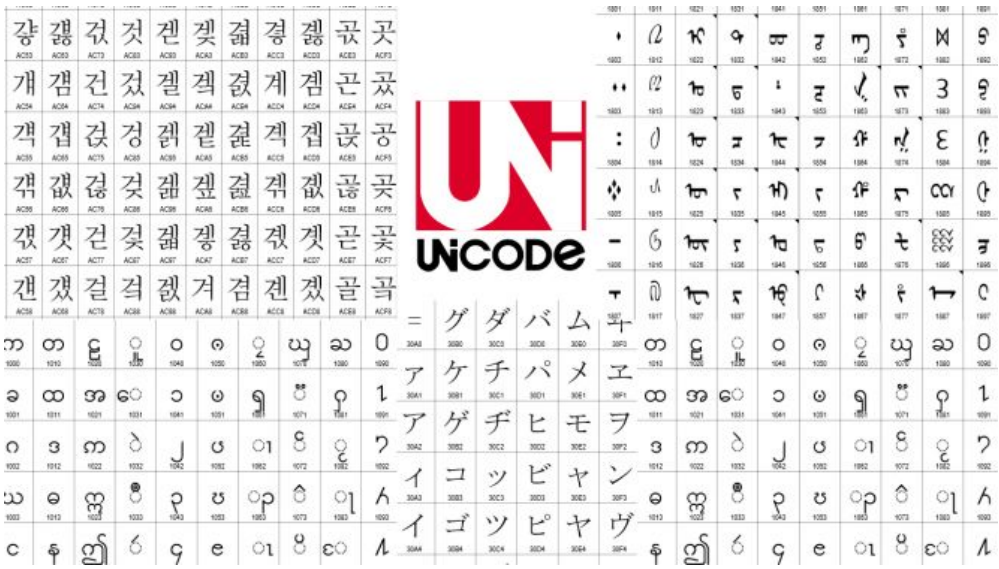
1989	100 000 ordinateurs inter-connectés (avec 9 charsets)
1990	Disparition d'ARPANET
1990	Annonce publique du World Wide Web
1992	1 000 000 ordinateurs connectés (10x plus en 3 ans… et une dizaine de charsets...)

Avec l'arrivée d'internet, il va donc falloir mettre de l'ordre pour pouvoir échanger et un consortium part en mission pour harmoniser cela en collaboration avec l'ISO (responsable des normes, comme l'AFNOR en France)

Les alphabets réunis : "Unicode, attrapez-les tous !"

Ce consortium a un nom, il s'appelle Unicode

En 1991, sort la première version d'Unicode : une table prévue pour contenir à terme TOUS les symboles du monde : passés, présents et futurs. (vaste programme)



On associe toujours un nombre à un caractère, comme en ASCII, et tout le monde (le monde entier) se met d'accord sur le caractère utilisé.

Pour info, on en est à la version 8.0 sortie en juin 2015 avec 110 000 caractères couvrant 100 écritures

Pour numéroter tout cela on ne compte plus sur 8 bits mais sur un nombre de bits bien plus élevé : 4x 8bits. soit 32 bit, ce qui fait un gros 4,2 milliards de caractères potentiels.

Ces 4,2 milliards de caractères ne sont pas inscrits au fur et à mesure, il y a des plages de caractères réservées à tel ou tel usage

ex:
les flèches sont dans une plage (complète, sans trou) allant de 8592 à 8703

Et là vous allez être déçus de la simplicité de la chose, mais c'est simplement ainsi qu'on a pu mettre en place les emojis.
On a juste ajouté des caractères spéciaux et prévu des plages dédiées à ces symboles sympathiques inventé par nos amis les Japonais.

Je détaillerai les emojis à la fin.

Le problème est qu'en informatique, le nombre de symbole, bits, est constant, qu'ils soient utilisés ou non.

Imaginez devoir au quotidien mettre autant de chiffres que le plus grand nombre connu au monde. Vous voulez écrire 42, vous devriez mettre 0000000….000000….00000042.

Bonjour la rédaction des chèques...

Donc le problème en informatique est qu'en passant d'un encodage de 8 bis à 32 bits, on a, pour les lettres usuelles, ajouté PLEIN de zéros inutiles au quotidien, ce qui fait que les caractères, et donc au global les fichiers, deviendraient 4 fois plus gros en unicode qu'en latin-1

Pas très catastrophique si vous avec 3 lignes, mais beaucoup plus si vous brassez des quantités impressionnantes de données…

Il y a un autre problème: la rétrocompatibilité.
Pour certaines machines, avoir 8 zéros d'affiler, c'est la fin d'un programme, un point final, générique de fin.

Ce qui fait qu'un caractère basique comme un "A" va faire planter l'ordinateur qui ne lira pas la suite, l'ignorant complètement.

Pour ces deux raisons a été inventé (en parallèle de l'Unicode) l'UTF ou Unicode Transformation Format qui permet de se passer de ce qui n'est pas utile et en plus d'assurer une espèce de rétro-compatibilité +/- correcte. (Ouais c'est un hack, mais un hack super bien vu)

Son fonctionnement est assez simple (mais accrochez-vous, c'est la partie un peu tendue de la chronique) :

Dans le cas de l'UTF-8 on a une base de 8 bits (un octet). On peut donc couvrir l'ASCII en mettant un zéro devant (l'ascii c'est 7 bit je rappelle)

Si nous sommes en présence d'un caractère qui requiert plus de 7 bits, on va découper ce caractère sur plusieurs octets avec un code au début de chacun des octets indiquant que le caractère est en morceaux.

Le principe est hyper simple :

- Si l'octet démarre par un zéro : il est seul et auto-contenu. (et en ASCII forcément puisque sur 7 bits réellement utilisés!)

Si l'octet démarre par un 1, c'est que le caractère est sur plusieurs octets :

- Si l'octet démarre par 110x, il faut s'attendre à 1 octet supplémentaire qui commencera par 10x
- Si l'octet démarre par 1110x, il faut s'attendre à 2 octets supplémentaire qui commenceront TOUS DEUX par 10x
- Si l'octet démarre par 11110x, il faut s'attendre à 3 octets supplémentaire qui commenceront TOUS TROIS par 10x

Le maximum de découpe est donc 6 morceaux parce que le premier octet indiquera 1111 110x

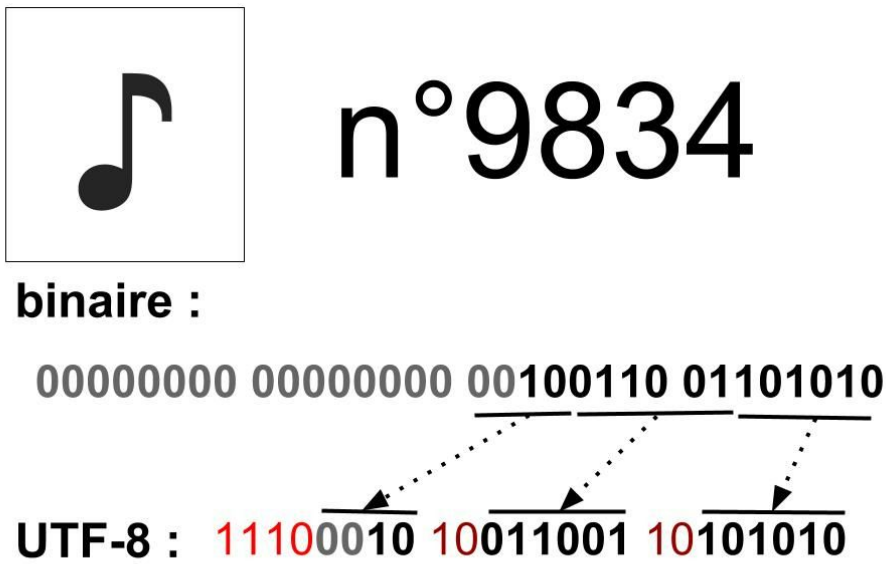
Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

L'avantage de faire cela est qu'un ancien ordinateur, ou un programme lisant du texte sur 8 bits ne va pas planter à la lecture et devrait afficher correctement la majorité du texte, le rendant relativement lisible (entre les bugs).

Et c'est pour cela que votre fichier de sous-titre encodé en UTF-8, lorsqu'il va y avoir un petit malin qui va mettre une croche de musique, l'utf-8 va écrire son numéro
(9834 en binaire soit 00000000 00000000 00100110 01101010)

et il va le faire de la façon suivante (ça se fait de droite à gauche):
=> entrer les 6 derniers bits et mettre "10" devant
=> prendre les 6 précédents et mettre "10" devant

=> prendre les 2 bits significatifs restants, mettre 1110 devant pour annoncer qu'il y a 3 morceaux et intercaler des 0 entre ces 2 morceaux pour former un octet.
résultat :
11100010 **10**011001 **10**101010



Pour ceux que j'ai perdus, en gros ça découpe en morceaux, ici 3 morceaux, 3 octets.

Cela va être compris sur votre ordinateur qui attend du latin-1 comme étant 3 octets représentant non pas 1 caractère mais 3 caractères latin-1 (en europe) non-ASCII, parce que ça démarre par un 1, et ça donne ça : **â™ ª**

Pour un **é**, vous aurez **Ã©**
pour un **ç** , vous aurez **Ã§**

etc...

Voilà, j'ai livré ce qui était promis: vous savez pourquoi les sous-titres partent parfois en sucette.

Et donc je vais tenir mon autre promesse et terminer sur les emoji avec des précisions et surtout une révélation :

Instant wikipedia:

> Les premiers *emoji* ont été créés entre 1998 et 1999 par Shigetaka Kurita, de l'équipe i-mode de NTT DoCoMo *(le Orange Japonais)*.
> Le premier jeu de 172 *emoji* de 12x12 pixels a été conçu comme une caractéristique spécifique de la messagerie i-mode, pour faciliter la communication électronique et *(surtout?)* pour la démarquer de services concurrents.

La révélation choquante : les emoji n'existent pas en Unicode.

Nulle part il n'est fait mention d'eux. Un acte de dissimulation? (jingle "théorie du complot")

Rien de tout ça :
Nous comprenons entre nous "emoji" au sens de "l'ensemble des symboles colorés allant de l'emoji-immeuble, pardon "l'édifice à bureau" (n°127 970) à l'emoji-caca, pardon "tas de crotte" (n°128 169)".

En réalité, les emojis se répartissent sur 7 plages unicode appelées blocs :

- Divers symboles et pictogrammes
- Émoticônes (les smileys)
- Symboles du transport et cartographiques
- Symboles alchimiques
- Formes géométriques étendues
- Supplément de flèches - C
- Supplemental Symbols and Pictographs

(et j'ai du en oublier...)

Mais bon, on va continuer à appeler ce joyeux bordel d'icônes "emoji" , parce que qu'on se comprend et c'est le principal.

La majorité d'entre eux commence donc à n°127 744 (avec un cyclone… TU LA SENS L'ORIGINE JAPONAISE? La vague du tsunami étant juste 10 numéros plus loin^^) et finit à n°131 071 avec… rien (un caractère non attribué).

D'ailleurs, et malgré que la grande majorité des symboles de la planète y soient inscrits, rien n'est encore prévu au-delà de cette limite de 131 071, pour dire si la limite de 4 milliards est une limite éloignée.

Il y a aussi énormément de numéro de caractères encore libres entre 0 et 131 071.
On a 110 000 caractères pour 131 071 cases, soit + de 20 000 cases vierges. Cette table unicode est un véritable gruyère.

Dans ce cas il se passe quoi?
Le numéro existe mais rien n'y est inscrit encore.

Mais c'est con, autant tout coller… non?
Non, c'est bien vu : cela permettra ultérieurement d'intercaler sans soucis des symboles au sein des plages qui les catégorisent le mieux et limiter des plages "suppléments de trucmuche"

C'est pour cela que les pétitions pour introduire de nouveaux emoji ont un sens:

- Quitte à redéfinir l'alphabet 2.0, autant donner son avis.
- Il y a de la place, on a prévu le coup alors pourquoi pas ?

Oui et non : ces places étant +/- délimitées par les plages de catégorisation, on ne va pas non plus mettre tout et n'importe quoi. Mais nous ne sommes plus limités comme au temps de l'ASCII. Alors pourquoi pas mettre des tacos violets avec un arc en ciel derrière?

Sachez d'ailleurs que les emojis n'ont pas de couleur déterminées par l'Unicode.
Tout est en noir et blanc.

La polémique sur la couleur de peau d'une émoticone n'est pas du ressort d' Unicode mais du logiciel que vous utilisez.
Unicode a quand même émis une recommandation sur le sujet, le fait proposer un choix de couleur, comme le ferait une police de caractère, mais aucune obligation.

Plus important: le caractère unicode est défini par un nom uniquement… et non une image.
Ce qui veut dire qu'il peut exister des polices différentes, comme toute lettre classique.

La preuve est l'emoji cookie, qui n'existe pas.
Il s'appelle "biscuit sec" en français, et peut s'afficher sous une forme de Tuc sur un samsung (merci Pof pour l'anecdote).
Et ce n'est pas une erreur de la part de Samsung. C'est conforme à la description.

Une autre info intéressante: sachez que les emojis ne peuvent être utilisés en UTF-8 (il faudrait les découper en plus de 6 et on ne peut pas) et requièrent donc de l'UTF-16 (il fonctionne pareil que l'UTF-8 mais les bouts font 16 bit de long au lieu de 8).

Phil, putain, laisse tomber les détails tu nous embrouilles

Vous vous en foutez?
Essayez de mettre un emoji dans un SMS et la limite baissera à 70 caractères au lieu de 140.
Pourquoi? parce que l'encodage de votre SMS passera en UTF-16 et chaque lettre tapée prendra 2x plus de place… 140/2 = 70.

D'ailleurs, les plus futés d'entre vous me diront si on accepte les caractères spéciaux, rien n'empêcherait à terme d'avoir des mots de passe emoji.
Et ils auront raison… Mais ce n'est pas encore tout à fait conseillé.
Je peux citer pour exemple un pauvre bougre qui avait un mot de passe emoji, qui a upgradé de yosemite à El Capitan , s'est rendu compte que le clavier emoji était indisponible au login… et ne pouvait plus accéder à son ordinateur...

Allez, une dernière pour la route : les drapeaux prennent 2 caractères…
voilà, @+

ah, vous voulez savoir pourquoi?
En fait, c'est parce qu'ils ne sont pas représentés dans l'Unicode.

C'est quoi le délire ?
Les drapeaux peuvent changer, les pays apparaître ou disparaître, certains indépendantistes souhaitent un drapeau (salut amis bretons, corses et basques) et le consortium Unicode ne souhaitait pas prendre part à ce sujet épineux.

Ils ont donc botté en touche avec un "indicateur régional" composé de lettres (2 = F&R en france) mais des lettres "dédiées" ce ne sont pas les lettre de l'ascii, elles sont dédiées à ce code régional qui ressemble à s'y méprendre à la fin d'un nom de domaine.
Libre aux appareils de le comprendre ou non, Unicode s'en lave les mains et rejette le traitement aux développeurs de logiciels.

Pour conclure, je dirais que l'Unicode est une réalisation que je connaissais de nom, mais c'est en préparant cette chronique que sa beauté presque poétique m'a frappé.

C'est un symbole de l'unification de l'humanité, cette volonté de communiquer entre nous.

Créer cette base qui se veut solide et impartiale, aidant tous les langages à être lisibles partout… C'est… je sais pas... beau?
Peut-être que de ce travail déjà titanesque jaillira un jour cette fameuse langue mondiale ? Le premier pas en tout cas vers une union des nations via la communication.

Tout cela, toute cette énergie bienfaisante, pour que finalement vous envoyiez des "tas de crotte" (n°128 169) à vos copains…