Encodages de caractères

I. Introduction

1) Qu'est-ce qu'un fichier texte ?

En informatique, on distingue deux sortes de fichiers : les fichiers texte et les fichiers binaires.

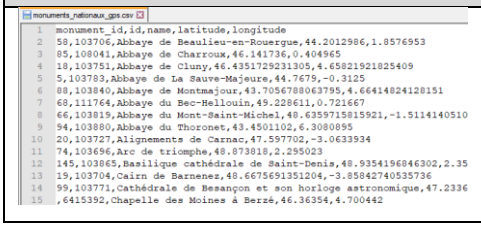
- Un fichier texte est un fichier dont le contenu représente *uniquement* une suite de caractères, il est donc interprétable sous forme d'une suite de caractères.
- Un fichier binaire est un fichier qui n'est pas un fichier texte : certaines séquences de bits ne représentent pas une suite de caractères imprimables.

Il faut remarquer qu'il n'existe pas de définition officielle de ce qu'est un fichier texte.
Un fichier texte peut s'ouvrir à l'aide d'un *éditeur de texte* tel que Notepad++, auquel cas son contenu est complètement compréhensible par un humain.
A contrario, un fichier binaire a très souvent besoin de s'ouvrir avec un logiciel dédié qui peut en interpréter le contenu (par exemple un logiciel de retouche d'image ou un logiciel de *traitement de texte*).

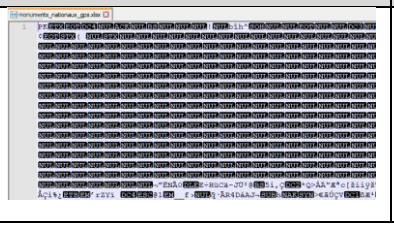
Finalité du fichier	Extension	Texte	Binaire	Commentaire
"bloc-notes"	.txt	X		
"document Libre Office"	.odt		X	
"tableau CSV"	.csv	X		
"tableau Excel"	.xlsx		X	
"image"	.jpeg		X	
"code python"	.py	X		
"page web"	.html	X		
"document"	.pdf		X	
"ROM de Game Boy"	.GB		X	

Exemples de fichiers ouverts dans *l'éditeur de texte* Notepad++

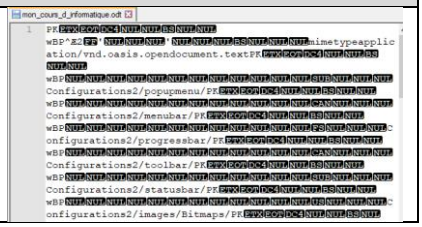
"tableau .CSV"



"tableau Excel .XLSX"



"document Libre Office .ODT"

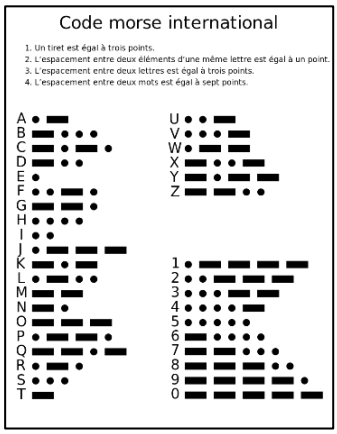


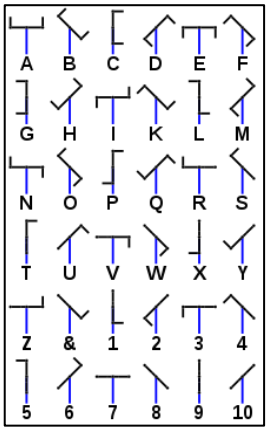
Dans la suite de ce cours nous nous intéresserons à la façon dont une machine informatique représente du texte en machine (notion d'*encodage*). Attention : ce dont nous parlerons ne s'appliquera a priori qu'à des fichiers texte, pas à des fichiers binaires.

2) Représentation des lettres sous forme codée : avant l'informatique

Le codage des lettres afin de transmettre un texte n'est pas récent. Citons par exemple :

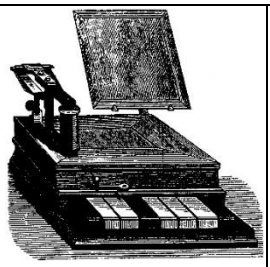
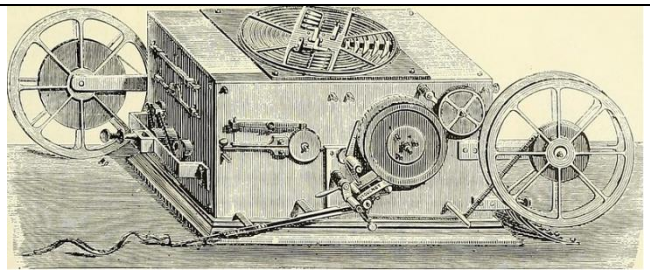
- Les signaux de fumée (dès l'Antiquité)
- Le code international des signaux maritimes issu d'une longue évolution (entamée dès 1738 par de La Bourdonnais)
- Le sémaphore ou télégraphe optique des frères Chappe (1790)
- Le code Morse pour la télégraphie électrique (1832)





Le code Baudot, créé en 1874, est le premier à utiliser un principe de codage des caractères par un système binaire abstrait. Ce codage est effectué sur cinq bits et utilise deux jeux de caractères : un jeu pour les lettres et un jeu pour les chiffres et symboles divers. On passe d'un jeu de caractères à l'autre grâce à deux codes spéciaux (27 et 31).

Ce code a d'abord été utilisé sur un appareil dédié permettant aux opérateurs d'être plus rapides qu'avec le code Morse pour la rédaction et pouvant "imprimer" un ruban perforé à la réception. On parle de *téléscripteurs*.
Puis ce code Baudot, comme l'encodage ASCII en informatique, connaîtra des variantes et des évolutions (code Western-Union, code Murray, code TTY).


Clavier de saisie et récepteur mis au point par Baudot (env. 1875) et téléscripteurs durant la 2nde guerre mondiale.

Une des normalisations de ce code (la CCITT#2) sera ensuite reprise dans le réseau de téléscripteurs Télex (en France, derniers abonnements Télex clôturés en janvier 2017).

Le principe du code Baudot est également repris en 1948 par le premier ordinateur électronique (le Manchester Mark I). Très peu de temps après, différents encodages de caractères sont mis au point sur les premiers ordinateurs : RADIX-50, SIXBIT, BCD, EBCDIC, ASCII, MARC-8, GOST 10859 (prenant en charge les caractères cyrilliques) …

L'encodage de caractères est nécessaire pour représenter du texte en machine. Sur des ordinateurs cet encodage conduit à une représentation binaire des caractères. D'un point de vue historique de nombreux formats d'encodage sont apparus. Très rapidement plusieurs problématiques ont émergé :
- comment gérer la multitude de caractères présents sur Terre (latins, chinois, arabes, cyrilliques etc.) ?
- comment gérer l'interopérabilité des fichiers texte (utiliser un même fichier texte sur différentes machines) ?
- comment gérer au mieux la compatibilité d'un nouveau format d'encodage avec l'existant ?

Nous allons voir, à travers l'exemple des encodages ASCII, ISO-8859-1 et Unicode, comment les acteurs du monde de l'informatique ont répondu à ces questions.
Surtout, l'étude de ces exemples nous permettra de comprendre les différents problèmes qui peuvent survenir lorsqu'on utilise des fichiers texte fournis par des tiers.

II. Trois exemples d'encodages

Dans tout ce qui suit nous utiliserons le préfixe 0x pour désigner un nombre écrit en hexadécimal.

1) L'encodage de caractères ASCII *(American Standard Code for Information Interchange)*

La première version publiée de l'encodage ASCII date de 1963. Il s'agit alors d'un *codage sur 7 bits* qui peut donc supporter 128 caractères.

ASCII Code Chart

	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		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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

Sur ces 128 caractères, 95 sont imprimables : lettres minuscules et majuscules, les chiffres arabes de 0 à 9, des symboles mathématiques et de ponctuation comme illustré par ce tableau :

D'autres caractères non-imprimables sont disponibles, ce sont des caractères de contrôle

Les caractères de contrôle vont du N°00 (0x00) au N° 31 (0x1F) et le N°127 (0xFF). Ils permettent de piloter différents périphériques de sortie (écran, imprimante) ou de communication.

On notera par exemple que le N°9 (0x09) correspond à une tabulation (touche [TAB] du clavier), le N°13 (0x0D) qui permet de faire un retour chariot, le N°10 (0x0A) qui fait passer à la ligne suivante, ou encore le N°7 (0x07) qui permet de faire retentir un bip sonore. Le caractère N°127 (0x7F) est celui qui permet d'effacer le dernier caractère saisi.

Remarques :

- On retiendra que dès les années 1970 les ordinateurs travaillent presque tous sur des multiples de huit bits. Par conséquent, chaque caractère est *souvent stocké dans un octet dont le huitième bit est 0*.

- L'encodage ASCII présente un défaut majeur : il n'offre aucune place aux divers besoins régionaux (par exemple les caractères accentués).

- Dès 1972 apparait ainsi la norme ISO 646 qui offre la possibilité de créer des *variantes régionales* issues de l'ASCII, *toujours codées sur 7 bits*. Cela se fait en rendant 18 symboles "modifiables".
  Par exemple le caractère N°92 (0x5C) "\" devient :
  - dans le jeu ISO 646 FR : "ç"
  - dans le jeu ISO 646 DK : "Ø"
  - dans le jeu ISO 646 ES : "Ñ"

  Remarque : l'encodage ASCII est intégralement préservé dans le jeu ISO 646 US. C'est pour cette raison que le jeu ISO 646 US est parfois appelé ASCII (à tort).

2) L'encodage de caractères ISO-8859-1

L'amélioration – par rapport à l'ASCII – apportée par l'ISO 646 s'est rapidement montrée insuffisante. De nombreux caractères européens ne pouvaient être affichés. Les encodages ISO-8859 sont venus résoudre partiellement ce problème en 1985 :

- *codés sur 8 bits* ce qui permet de disposer a priori de 128 caractères supplémentaires,

- *les caractères 0 à 127 sont les mêmes que ceux de l'ISO 646 US, et donc les mêmes que l'ASCII*,

- ils offrent de *nombreuses variantes régionales* :
  - *ISO-8859-1 : européen occidental (parfois appelé Latin-1 dans le cadre ISO 8859-1 non IANA)*
  - ISO-8859-2 : européen central
  - ISO-8859-6 : arabe
  - ISO-8859-7 : grec
  - ISO-8859-8 : hébreu etc.

ISO-8859-1																
	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Remarques :

- La norme ISO 8859-1 (aucun tiret entre le ISO et le 8859) associée à ce jeu est légèrement différente. En effet la norme ISO 8859-1 ne comporte pas les caractères de contrôle des plages 0x00 à 0x1F et 0x80 à 0x90.

3) Unicode (avec transformation UTF-8)

L'encodage Unicode, datant de 1991 et sans cesse mis à jour, a été développé dans le but de remplacer l'utilisation de *toutes* les pages de code nationales et de comporter ainsi l'ensemble des caractères existant ou ayant existé sur Terre.
Il a aussi été pensé pour gérer les règles sur la sémantique des caractères, leurs compositions et la manière de les combiner. Unicode permet ainsi de gérer les écritures de gauche à droite ou les ligatures.

```
1 print_HTML('Ailleurs, affiche, rafting, f, t, i')
```

Ailleurs, affiche, rafting, f, τ, i

Unicode permet de gérer les ligatures typographiques et d'autres effets : selon leurs voisins, les caractères prennent des formes différentes.

```
1 heh, ya, lam = '\u0647', '\u064A', '\u0644'
2 print_HTML(heh+ ' ' + ya + ' ' + lam)
3
```

ه ي ل

```
1 print_HTML(heh+ya+lam + ' ' + lam+ya+heh + ' ' + ya+ya+ya + ' ' + heh+heh+heh + ' ' + lam+lam+lam)
```

هيل ليه ييي هههه للل

En arabe, en plus des ligatures, Unicode permet d'afficher les caractères de gauche à droite mais aussi de gérer le fait qu'un même caractère possède différentes formes selon son emplacement dans le mot.

a) Différence entre glyphes et caractères abstraits [Hors-Programme : pour culture générale]

Les glyphes sont les représentations d'un caractère. À un même caractère peuvent correspondre différents glyphes et à un même glyphe peuvent correspondre différents caractères.

Glyphe ou séquence de glyphes	Caractère ou séquence de caractères
a ou *a* ou a ou **a** ou **a** ou **a** selon police	"a latin minuscule"
fi ou f ı ou f i selon police ou logiciel	"f latin minuscule" suivi de "i latin minuscule"
ò ou o ^ ` ou o `	"o latin minuscule" suivi de "accent circonflexe" suivi de "accent grave"
ò ou o ^ ` ou o `	*ồ*
ه ou ھ ou ھ ou ھ selon contexte dans le texte	"heh arabe"

b) Du glyphe à la représentation mémoire : vue d'ensemble de l'encodage Unicode

Glyphes	⇜⇝	Caractères abstraits	⟺	Caractères codés (points de code)	⟺	Caractères encodés (exemple avec UTF-8)	⇜⇝	Séquences encodées
		LATIN SMALL LETTER F + LATIN SMALL LETTER I		U+0066 U+0069		01100110 01101001		Hors-programme
		LATIN CAPITAL LETTER L + LATIN SMALL LETTER A WITH GRAVE		U+004C U+00E0		00101100 11000011 10100000		Hors-programme

La partie qui est au programme est le passage d'un caractère abstrait à un caractère encodé (⟺) .
Les transformations (⇜⇝) sont hors programme.

- *Le passage des caractères abstraits aux caractères codés (points de code) est effectué grâce à des tableaux donnant des points de code*. Quelques exemples sont donnés en annexe. Ils ont tous disponibles sur le site d'Unicode.

- *Le passage des points de code à leur représentation encodée binaire est effectuée grâce à la transformation UTF (Universal Transformation Format). Il y a plusieurs variantes de cette transformation UTF-8, UTF-16 et UTF-32* où le nombre indique le nombre de bits minimal avec lesquels un point de code valide est représenté. *La méthode de transformation UTF-8 – la plus utilisée – est décrite en annexe.*
  Par exemple en UTF-16, les représentations des points de code prennent tous au moins 16 bits (y compris la représentation de U+0034).

c) Quelques remarques

- Il est important d'être au clair sur les cinq "objets" qui sont en jeu dans le modèle d'encodage Unicode. Et, par ailleurs, de ne pas confondre Unicode et UTF-8. UTF-8 est une "petite partie" d'Unicode, c'est uniquement la transformation qui permet de passer du point de code au caractère encodé.

- Les points de code de U+0000 à U+00FF sont conformes à l'encodage ISO-8859-1

- Il est impératif d'aller visiter le site d'Unicode pour se rendre compte de la richesses et de l'esthétique des caractères pris en charge : du chinois au tamoul en passant par le goudjerate, le télougou… On voit aussi une grande variété de symboles, les émoticônes, les cartes à jouer, les pièces d'échec etc.
  En tout, en mars 2019, ce sont 137 929 caractères qui sont décrits par Unicode.

- Si vous avez un problème d'affichage du contenu texte d'un fichier cela peut avoir plusieurs origines :
  - format de décodage différent du format d'encodage (vous décodez en ASCII un fichier texte encodé en ISO-8859-3)
  - fichier binaire que vous essayez d'éditer comme un fichier texte
  - fichier encodé en Unicode et décodé en Unicode mais pour lequel certains caractères ne sont pas présents dans les polices utilisées par votre logiciel.

- Unicode avec la transformation UTF-8 (souvent abrégé en UTF-8) est LE format d'encodage à utiliser dorénavant.

Annexe 1 : quelques plans Unicode

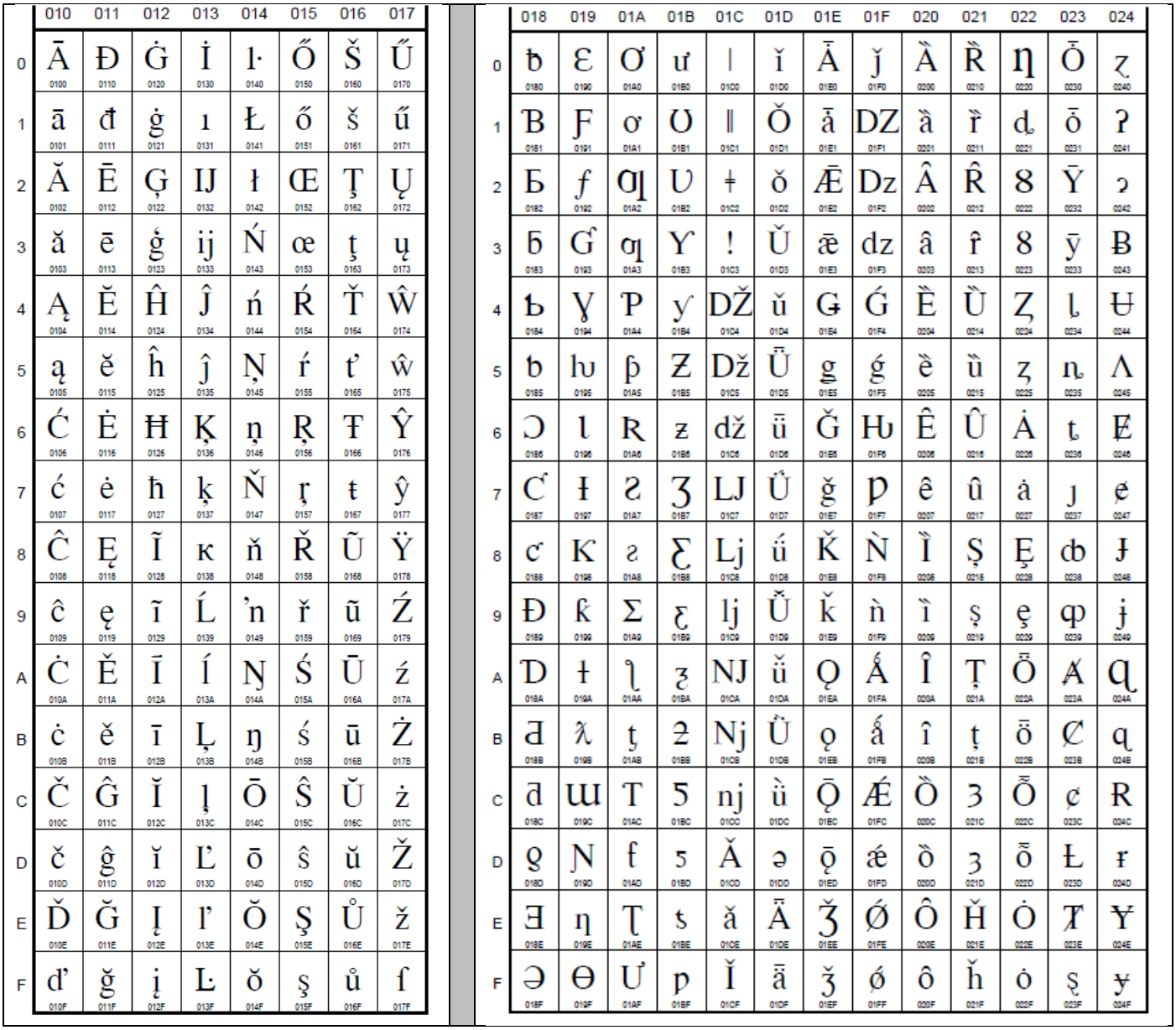
Les plans reprenant ASCII et ISO-5589

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HTJ 0089	XXX 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	VTS 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SS3 008F	APC 009F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

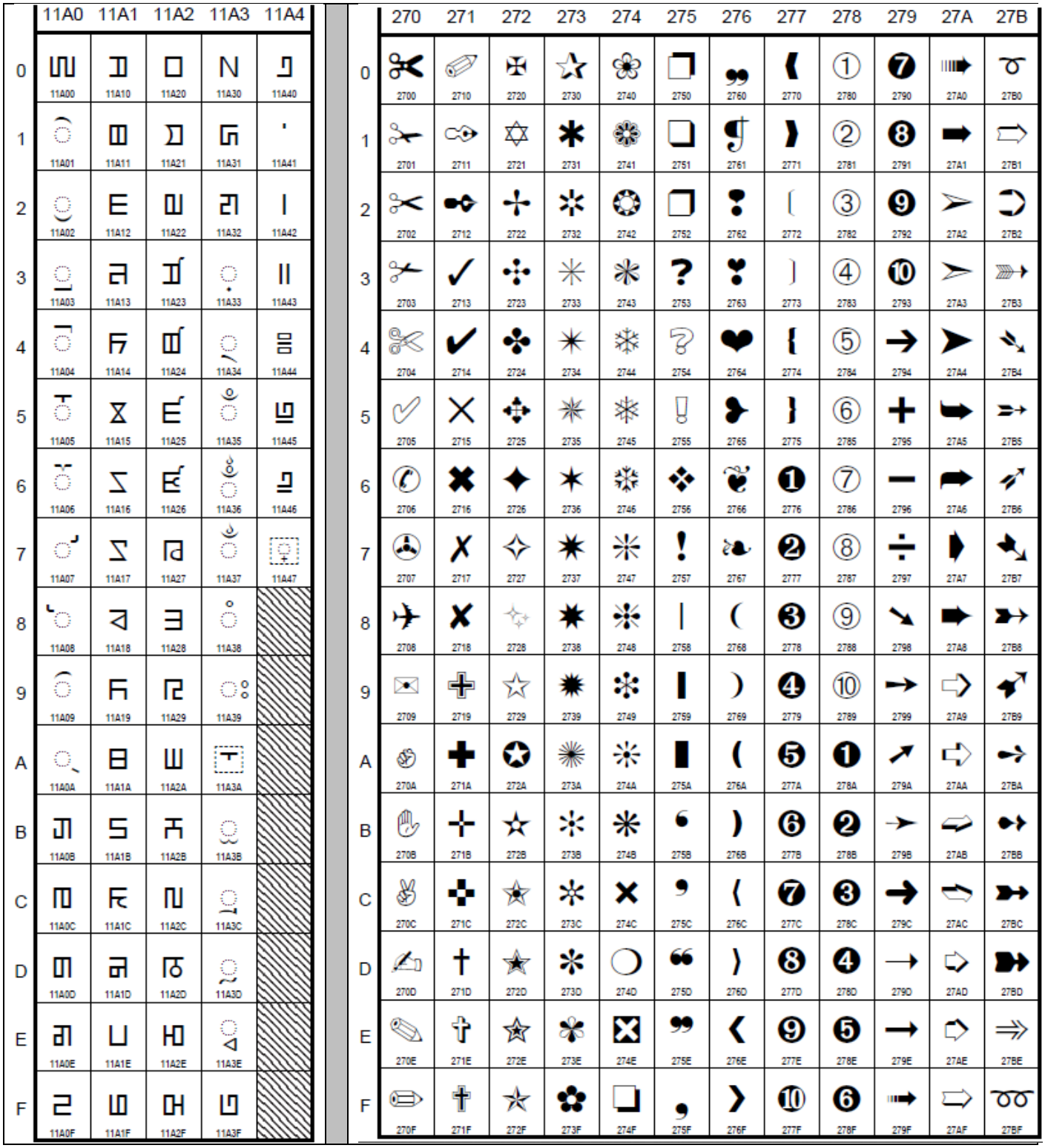
Annexe 1 : quelques plans Unicode (suite)

Extensions de l'alphabet latin



Annexe 1 : quelques plans Unicode (suite)

Zanabazar Square et Symboles Divers



Annexe 2 : Transformation UTF-8

- Les points de code ayant une valeur de U+0000 à U+007F (attribués aux caractères du jeu codé sur 7 bits dans l'ASCII) sont codés sur un seul octet dont le bit de poids fort est nul.

- Les autres points de code ayant une valeur supérieure à U+007F sont codés sur plusieurs octets :

    - les bits de poids fort du premier octet de la séquence codée forment une suite de 1 de longueur égale au nombre total d'octets (au moins 2) utilisés pour la séquence entière suivie d'un 0
    - les octets suivants nécessaires ont leurs deux bits de poids fort positionnés à 10.

Caractères codés (Points de code)	Nombre de bits nécessaires	**Caractères encodés (après transformation UTF-8)**
U+0000 à U+007F	0**0000000** à 0**1000000** 7 bits suffisent	**0**XXXXXXX
U+0080 à U+07FF	00000**000 10000000** à 00000**111 11111111** 11 bits suffisent	**110**XXXXX **10**XXXXXX
U+0800 à U+FFFF	**00001000 00000000** à **1111111 11111111** 16 bits suffisent	**1110**XXXX **10**XXXXXX **10**XXXXXX
U+10000 à U+10FFFF	000**00001 00000000 00000000** à 000**10000 00000000 00000000** 21 bits suffisent	**11110**XXX **10**XXXXXX **10**XXXXXX **10**XXXXXX

Exemple d'encodage :

On souhaite encoder les caractères suivants ❼ Æ L soit les points de code : U+277C U+00C6 U+004C

- 0x277C = 00100111 01111100$_2$ a besoin de 14 bits. On va donc encoder sur 3 octets.
  Ce qui va donner : **1110**0010 **10**011101 **10**111100
- 0x00C6 = 11000110$_2$ a besoin de 8 bits. On va donc encoder sur 2 octets.
  Ce qui va donner : **110**00011 **10**000110
- 0x004C = 01001100$_2$ a besoin de 7 bits. On va donc encoder sur 1 octet.
  Ce qui va donner : **0**1001100
- Au final :
  11100010 **10**011101 **10**111100 **110**00011 **10**000110 **0**1001100
  0xE2 0x9D 0xBC 0xC3 0x86 0x4C

Exemple de décodage :

On souhaite décoder la séquence suivante de caractères encodés en UTF-8, et représentée en hexadécimal :
0xC3 0x91 0x59 0xE2 0x9C 0xB0

1. On la représente en binaire : 11000011 10010001 01011001 11100010 10011100 10110000

2. De gauche à droite on repère les blocs d'octets :
   11000011 10010001 ***0***1011001 ***1110***0010 10011100 10110000

3. On enlève les bits qui sont associés à la transformation UTF-8 :
   ~~***110***~~00011 ~~10~~010001 ~~***0***~~1011001 ~~***1110***~~0010 ~~10~~011100 ~~10~~110000
   00011010001 1011001 0010011100110000

4. On convertit en points de code :
   U+00D1 U+0059 U+2730 soit les caractères : ÑY☆