Jeux de caractères et encodage (par Michel Michaud © 2014)

Les ordinateurs ne traitent que des données numériques. En fait, les codages électriques qu'ils conservent en mémoire centrale ne représentent que des 0 et 1. Pour les fichiers, sur disques magnétiques ou autres, c'est la même chose : tout est conservé sous forme de séries de 0 et de 1. C'est pourquoi on dit que l'ordinateur travaille en système binaire, et l'élément de base de toutes les données est le bit (*binary digit*). En utilisant un série de bits, on peut représenter tout le reste : si un bit ne permet de représenter que deux possibilités (0 et 1), avec deux bits on a deux fois plus de possibilités (00, 01, 10, 11) et ainsi de suite. Avec 32 bits par exemple, on a déjà plus de 4 milliards de possibilités, et chaque combinaison de bits pourrait représenter un nombre différent, selon le codage choisi.

Pour représenter des caractères ou d'autres données, l'ordinateur utilise habituellement des groupes de 8, 16 ou 32 bits, ce qui donne respectivement 256, 65 536 et 4 294 967 296 possibilités. Les groupes de 8 bits sont appellés octet (communément, mais erronément, *byte* en anglais). La plupart des mesures de capacité des ordinateurs sont données en octets, la manipulation de bits individuels étant plutôt rare.

On a longtemps conservé chaque caractère dans un octet. Les 256 possibilités étant suffisantes pour l'alphabet habituellement utilisé en anglais et en français, même en comptant les caractères en majuscules et minuscules, la plupart des caractères accentués et les caractères spéciaux comme le point ou l'astérisque. C'est encore une possibilité fréquemment utilisée, qu'il vaut donc la peine d'étudier.

Jeu de caractères, notation binaire et hexadécimale

Une fois qu'on a décidé d'utiliser un octet (ou tout autre regroupement de bits) pour représenter un caractère, il faut décider quelle série de bits représente quel caractère et s'assurer que toutes les composantes de l'ordinateur, ainsi que les périphériques, respectent ce choix. C'est ce qu'on appelle le jeu de caractères. Par exemple, on pourrait choisir que 01000001 représente le A et donc, en interprétant la série de bits comme un nombre binaire, que A porte le numéro 65 (en base 10). Si les caractères se suivent normalement, on aura le numéro 66 pour le B, et le numéro 90 pour le Z.

Il est parfois intéressant de regarder la valeur des bits, mais, pour simplifier, on utilise souvent la base 16, l'hexadécimal, pour représenter ces valeurs. Son avantage est qu'un chiffre hexadécimal représente une valeur sur 4 bits et qu'on peut alors facilement convertir en binaire au besoin, simplement en connaissant les 16 valeurs suivantes, qu'on peut aussi apprendre en comprenant le système binaire :

0 : 0000	1 : 0001	2 : 0010	3 : 0011	4 : 0100	5 : 0101	6 : 0110	7 : 0111
8 : 1000	9 : 1001	A : 1010	B : 1011	C : 1100	D : 1101	E : 1110	F : 1111

Puisque les chiffres de la base 16 doivent couvrir de 0 à 15, on utilise les lettres A à F pour représenter les chiffres hexadécimaux ayant les valeurs de 10 à 15. Ainsi le caractère Z aura le code hexadécimal 5A (donc 0101 1010 en binaire), on écrit aussi $(5A)_{16}$. On peut facilement convertir ce code en valeur décimale, par exemple, dans $(5A)_{16}$ le 5 représente 5×16 et le A représente 10, donc 5×16+10 qui donne $(90)_{10}$.

ASCII, EBCDIC et ISO-8859

Voici le jeu de caractères de base le plus connu, le *American Standard Code for Information Interchange*, le plus souvent appelé simplement ASCII :

Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex
	32	20	@	64	40	`	96	60
!	33	21	A	65	41	a	97	61
"	34	22	B	66	42	b	98	62
#	35	23	C	67	43	c	99	63
$	36	24	D	68	44	d	100	64
%	37	25	E	69	45	e	101	65
&	38	26	F	70	46	f	102	66
'	39	27	G	71	47	g	103	67
(	40	28	H	72	48	h	104	68
)	41	29	I	73	49	i	105	69
*	42	2A	J	74	4A	j	106	6A
+	43	2B	K	75	4B	k	107	6B
,	44	2C	L	76	4C	l	108	6C
-	45	2D	M	77	4D	m	109	6D
.	46	2E	N	78	4E	n	110	6E
/	47	2F	O	79	4F	o	111	6F
0	48	30	P	80	50	p	112	70
1	49	31	Q	81	51	q	113	71
2	50	32	R	82	52	r	114	72
3	51	33	S	83	53	s	115	73
4	52	34	T	84	54	t	116	74
5	53	35	U	85	55	u	117	75
6	54	36	V	86	56	v	118	76
7	55	37	W	87	57	w	119	77
8	56	38	X	88	58	x	120	78
9	57	39	Y	89	59	y	121	79
:	58	3A	Z	90	5A	z	122	7A
;	59	3B	[	91	5B	{	123	7B
<	60	3C	\	92	5C	|	124	7C
=	61	3D	]	93	5D	}	125	7D
>	62	3E	^	94	5E	~	126	7E
?	63	3F	_	95	5F			

Plusieurs codes ne sont pas montrés ici[1], les codes de 0 à 31 et le code 127, car ils ne sont pas attribués à des caractères ordinaires, mais bien à des caractères de contrôle des périphériques. Par exemple, le caractère 10 représente le saut de ligne, un code essentiel pour contrôler une imprimante par exemple. Le caractère 32, montré, est l'espace.

Ce jeu de caractères peut sembler incomplet, par exemple il n'y a aucun caractère accentué. En fait, le code ASCII est très limité parce que toutes les valeurs sont codées sur un maximum de 7 bits, donc 128 possibilités seulement, pour des raisons historiques de transmission de données (le huitième bit servait à la vérification des transmissions).

[1] Pour des tables plus complètes, voir par exemple http://www.michelmichaud.com/Tables.htm.

Ce jeu de caractères est quand même fondamental, car c'est celui qui s'est le plus répandu. En fait, c'est l'un des deux qui se sont répandus, avec le code EBCDIC qui était, et est encore, populaire sur les gros ordinateurs IBM. Tous les autres jeux de caractères courants, dans toutes leurs variantes, respectent le même codage de base qu'ASCII (A est toujours le caractère numéro 65 par exemple). Ce qui a varié grandement, c'est comment les autres caractères, ceux qui n'existent pas en ASCII, sont représentés. Comme on le verra, il y a maintenant une certaine uniformité, mais ça n'a pas toujours été le cas.

Le problème de base avec les caractères codés sur 8 bits est qu'il n'y a pas assez de codes possibles pour représenter les caractères particuliers de toutes les langues courantes. On peut alors décider que ce n'est pas utile de pouvoir mélanger plusieurs langues dans un même document, et s'entendre sur des jeux de caractères 8 bits particuliers, différents, pour les documents en différentes langues. Par exemple, voici le jeu de caractères ISO-8859-1, en principe pour le français, l'espagnol, etc. :

Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex
	32	20	@	64	40	`	96	60		160	A0	À	192	C0	à	224	E0
!	33	21	A	65	41	a	97	61	¡	161	A1	Á	193	C1	á	225	E1
"	34	22	B	66	42	b	98	62	¢	162	A2	Â	194	C2	â	226	E2
#	35	23	C	67	43	c	99	63	£	163	A3	Ã	195	C3	ã	227	E3
$	36	24	D	68	44	d	100	64	¤	164	A4	Ä	196	C4	ä	228	E4
%	37	25	E	69	45	e	101	65	¥	165	A5	Å	197	C5	å	229	E5
&	38	26	F	70	46	f	102	66	¦	166	A6	Æ	198	C6	æ	230	E6
'	39	27	G	71	47	g	103	67	§	167	A7	Ç	199	C7	ç	231	E7
(	40	28	H	72	48	h	104	68	¨	168	A8	È	200	C8	è	232	E8
)	41	29	I	73	49	i	105	69	©	169	A9	É	201	C9	é	233	E9
*	42	2A	J	74	4A	j	106	6A	ª	170	AA	Ê	202	CA	ê	234	EA
+	43	2B	K	75	4B	k	107	6B	«	171	AB	Ë	203	CB	ë	235	EB
,	44	2C	L	76	4C	l	108	6C	¬	172	AC	Ì	204	CC	ì	236	EC
-	45	2D	M	77	4D	m	109	6D	-³	173	AD	Í	205	CD	í	237	ED
.	46	2E	N	78	4E	n	110	6E	®	174	AE	Î	206	CE	î	238	EE
/	47	2F	O	79	4F	o	111	6F	¯	175	AF	Ï	207	CF	ï	239	EF
0	48	30	P	80	50	p	112	70	°	176	B0	Đ	208	D0	ð	240	F0
1	49	31	Q	81	51	q	113	71	±	177	B1	Ñ	209	D1	ñ	241	F1
2	50	32	R	82	52	r	114	72	²	178	B2	Ò	210	D2	ò	242	F2
3	51	33	S	83	53	s	115	73	³	179	B3	Ó	211	D3	ó	243	F3
4	52	34	T	84	54	t	116	74	´	180	B4	Ô	212	D4	ô	244	F4
5	53	35	U	85	55	u	117	75	µ	181	B5	Õ	213	D5	õ	245	F5
6	54	36	V	86	56	v	118	76	¶	182	B6	Ö	214	D6	ö	246	F6
7	55	37	W	87	57	w	119	77	·	183	B7	×	215	D7	÷	247	F7
8	56	38	X	88	58	x	120	78	¸	184	B8	Ø	216	D8	ø	248	F8
9	57	39	Y	89	59	y	121	79	¹	185	B9	Ù	217	D9	ù	249	F9
:	58	3A	Z	90	5A	z	122	7A	º	186	BA	Ú	218	DA	ú	250	FA
;	59	3B	[	91	5B	{	123	7B	»	187	BB	Û	219	DB	û	251	FB
<	60	3C	\	92	5C	|	124	7C	¼	188	BC	Ü	220	DC	ü	252	FC
=	61	3D	]	93	5D	}	125	7D	½	189	BD	Ý	221	DD	ý	253	FD
>	62	3E	^	94	5E	~	126	7E	¾	190	BE	Þ	222	DE	þ	254	FE
?	63	3F	_	95	5F				¿	191	BF	ß	223	DF	ÿ	255	FF

Encore une fois, on a omis les caractères spéciaux non affichables; le caractère 32 est l'espace, alors que le caractère 160 est l'espace insécable.

Ce jeu de caractères comporte presque tous les caractères normalement nécessaires pour les langues latines. Par contre, quelques années après sa création, l'Europe est passée à l'euro pour sa monnaie, et il

fallait trouver une façon de représenter ce symbole. On a donc créé un nouveau code, presque identique, appelé ISO-8859-15, avec les différences suivantes :

- le code 164 (A4) est le symbole de l'euro €;
- le code 166 (A6) est le Š;
- le code 168 (A8) est le š;
- le code 180 (B4) est le Ž;
- le code 184 (B8) est le ž;
- le code 188 (BC) est le Œ;
- le code 189 (BD) est le œ;
- le code 190 (BE) est le Ÿ;
- évidemment, on a perdu les anciens caractères de ces codes (¤¦¨´¸½¾).

Il y a plusieurs autres variantes du code ISO-8859 (par exemple, ISO-8859-2 pour le polonais, le hongrois, etc.). Par contre, il n'y a plus de travail sur ce standard, car on préfère maintenant Unicode…

Web et entités HTML

S'il faut transférer des données d'un ordinateur à l'autre, par exemple au travers d'un réseau ou par internet, il faut que tous les ordinateurs soient au courant du jeu de caractères utilisé et puissent le présenter correctement. C'est pourquoi les pages web (fichier html) devraient indiquer le type de caractères qu'elles utilisent (par exemple `<meta charset="ISO-8859-15">` si c'est le cas).

Par contre, on pourrait vouloir représenter, en mémoire centrale ou dans des fichiers de données, des caractères qui ne sont pas dans le même jeu de caractères. Par exemple, l'euro (€) *et* le ¾. Plusieurs possibilités s'offrent à nous. En HTML par exemple, quel que soit le jeu de caractères utilisés, on peut indiquer au fureteur qu'on veut voir le symbole de l'euro en écrivant l'entité `&euro;` dans le texte du fichier HTML. Le fureteur reconnaît l'entité (le code) et ne fait apparaître que le symbole équivalent. Il y a ainsi de nombreuses entités, en fait, on peut indiquer tous les caractères qui ne sont pas dans le jeu de caractères de base (ASCII). Par exemple, `&agrave;` (pour à) et `&copy;` (pour ©). Il est évidemment inutile d'utiliser une entité pour le à si ce caractère est dans le jeu de caractères que l'on utilise et qui a été spécifié dans `charset`; ça rendrait aussi le texte difficile à lire. Un autre problème de ce système est qu'il faut, par exemple, 8 octets pour représenter un à. S'il y a beaucoup de caractères spéciaux, non ASCII, un fichier de données pourrait devenir très gros, et sa transmission inutilement lente.

Unicode

Plusieurs autres solutions ont été trouvées et mises en application, mais la plupart supposent l'utilisation du jeu de caractères Unicode, ou l'équivalent ISO/IEC 10646. Ce jeu de caractères a été mis au point afin d'uniformiser le codage des caractères de toutes les cultures, et de nombreux symboles spéciaux. En Unicode, chaque signe est décrit, et un numéro (appelé point de code) lui est assigné. Par exemple, le é, « Latin small letter e with acute », porte le numéro 233, alors que le ῥ, « Greek small letter rho with psili », est 8164 et le « Horizontal black hexagon » (⬢) est 11043. Les numéros Unicode sont écrits en hexadécimal préfixés par U+. Les trois caractères précédents sont donc U+00E9, U+1FE4 et U+2B23.

Il y a actuellement plus de 100 000 caractères répertoriés dans la plus récente version d'Unicode. Les 256 premiers correspondent à ceux d'ISO-8859-1. En principe, Unicode n'indique pas comment seront représentés les numéros dans les ordinateurs ou les fichiers de données, mais la norme indique que les numéros nécessiteront au plus 6 chiffres hexadécimaux ou 21 bits. Le maximum est en réalité U+10FFFF, en binaire 1 0000 1111 1111 1111 1111, ce qui donne tout de même plus d'un million de possibilités, dix fois plus que ce qui est actuellement utilisé. Les ordinateurs auraient donc besoin de trois octets au maximum, mais comme ils utilisent plutôt les puissances de 2, quatre octets pourraient être nécessaires.

S'il fallait encoder tous les caractères sur 4 octets au lieu d'un, les fichiers de données deviendraient inutilement gros. En effet, la plupart des caractères sont entre 0 à FFFF, et, pour les caractères latins communs, la plupart sont même entre 0 et FF comme en ISO-8859-1. Le codage sur quatre octets est donc rarement utilisé. Les solutions les plus courantes sont d'utiliser soit un ou soit deux octets (8 ou 16 bits) pour la plupart des caractères, et un code spécial, utilisant plus d'octets, pour les autres. Ce sont les codages UTF-8 et UTF-16. On peut considérer qu'UTF signifie *Unicode Transformation Format*.

UTF-8

L'idée première du format, ou plutôt de l'encodage UTF-8, est que tous les fichiers ASCII usuels sont déjà dans ce format. Un logiciel qui travaille en UTF-8 pourra donc utiliser ces fichiers directement, sans autres formalités. Par contre, dès que le numéro d'un caractère à conserver dépasse 127 (le dernier code ASCII valide), il faut un encodage particulier, qui prendra entre deux et quatre octets. Le code maximum, 127, étant 0111 1111 en binaire, tous les caractères normaux ont le bit de gauche à 0. Dès qu'un octet commence par un bit à 1, on est en présence d'un codage UTF-8 que le logiciel doit décoder.

Si le numéro nécessite entre 8 et 11 bits, donc entre 128 et 4095, ce qui est le cas de la plupart des caractères courants et latins, qui tiennent même dans 8 bits, les bits en question sont codés dans les bits marqués d'un x dans une séquence de deux octets ayant la forme suivante :

110xxxxx 10xxxxxx

Par exemple, le é, code 233/U+E9 donc 1110 1001, ou 00011101001 sur 11 bits :

11000011 10101001

Ce qui donne les octets C3 A9. On peut donc dire que é est codé, en UTF-8, par C3 A9...

Si le numéro nécessite entre 12 et 16 bits, donc entre 4096 et 65535, le codage utilise trois octets :

1110xxxx 10xxxxxx 10xxxxxx

Par exemple, le ῤ, U+1FE4, est 0001111111100100 sur 16 bits, et sera codé ainsi :

11100001 10111111 10100100

Ce qui donne les octets E1 BF A4. On peut donc dire que ῤ est codé, en UTF-8, par E1 BF A4...

Si le numéro nécessite entre 17 et 21 bits (le maximum), le codage est sur 4 octets et sous la forme :

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Le codage d'un texte latin en UTF-8 occupe donc plus d'espace qu'en ISO-8859-1, mais moins qu'en simples caractères sur 16 ou 32 bits chacun. Voici par exemple, le contenu d'un fichier UTF-8 contenant la première phrase de cette section[2] :

```
00000000   4C 27 69 64 C3 A9 65 20 70 72 65 6D 69 C3 A8 72   L'id..e premi..r
00000010   65 20 64 75 20 66 6F 72 6D 61 74 2C 20 6F 75 20   e du format, ou
00000020   70 6C 75 74 C3 B4 74 20 64 65 20 6C 27 65 6E 63   plut..t de l'enc
00000030   6F 64 61 67 65 20 55 54 46 2D 38 2C 20 65 73 74   odage UTF-8, est
00000040   20 71 75 65 20 74 6F 75 73 20 6C 65 73 20 66 69    que tous les fi
00000050   63 68 69 65 72 73 20 41 53 43 49 49 20 75 73 75   chiers ASCII usu
00000060   65 6C 73 20 73 6F 6E 74 20 64 C3 A9 6A C3 A0 20   els sont d..j..
00000070   64 61 6E 73 20 63 65 20 66 6F 72 6D 61 74 2E      dans ce format.
```

Signature

Les logiciels qui acceptent les fichiers UTF-8 peuvent reconnaître les octets représentant un caractère codé, car ils ont une valeur supérieure à 127, donc entre $(80)_{16}$ et $(FF)_{16}$. En principe, un fichier purement ASCII fait aussi l'affaire, puisque ses caractères ne sont pas supérieurs à 127. Par contre, pour bien s'assurer que le codage est reconnu, dans les cas où plusieurs formats de fichiers pourraient être manipulés par le même logiciel, il est possible de signer les fichiers UTF-8 pour qu'ils soient toujours reconnus comme tels. Les trois premiers octets du fichier seront alors EF BB BF, qui représentent en principe le caractère U+FEFF, un code qui ne devrait pas se trouver en tête de fichier. Reconnaissant cette séquence, le logiciel identifie le fichier UTF-8 et continue la suite de l'analyse sans se préoccuper de ces trois octets. Voici le contenu d'un fichier semblable au précédent, mais avec la signature :

```
00000000   EF BB BF 4C 27 69 64 C3 A9 65 20 70 72 65 6D 69   ...L'id..e premi
00000010   C3 A8 72 65 20 64 75 20 66 6F 72 6D 61 74 2C 20   ..re du format,
00000020   6F 75 20 70 6C 75 74 C3 B4 74 20 64 65 20 6C 27   ou plut..t de l'
00000030   65 6E 63 6F 64 61 67 65 20 55 54 46 2D 38 2C 20   encodage UTF-8,
00000040   65 73 74 20 71 75 65 20 74 6F 75 73 20 6C 65 73   est que tous les
00000050   20 66 69 63 68 69 65 72 73 20 41 53 43 49 49 20    fichiers ASCII
00000060   75 73 75 65 6C 73 20 73 6F 6E 74 20 64 C3 A9 6A   usuels sont d..j
00000070   C3 A0 20 64 61 6E 73 20 63 65 20 66 6F 72 6D 61   .. dans ce forma
00000080   74 2E                                             t.
```

En principe, il vaut mieux ne pas utiliser de signature, car les logiciels qui ne traitent que l'ASCII de base ne sauront quoi en faire, mais tous les fureteurs doivent accepter et acceptent ces fichiers. Plusieurs logiciels ajoutent d'ailleurs une signature lorsqu'on demande d'enregistrer en UTF-8 sans autre précision.

UTF-16 et UCS-2

Le codage UTF-16 est semblable au UTF-8 : il utilise 16 bits par caractère au départ, et 32 seulement si nécessaire. Étant donné l'organisation des caractères Unicode, il n'est pas très répandu, pour le moment, car la majorité (totalité ?) des numéros de caractères utilisés en Occident ne dépassent pas 16 bits.

---

[2] Cette figure présente l'affichage classique de forme « vidange de mémoire » (dump en anglais). La colonne de gauche numérote les octets en hexadécimal, ceux-ci sont représentés en hexadécimal dans la colonne du centre et en ASCII dans la colonne de droite (un point est mis pour les caractères non imprimables). Il y a 16 octets par ligne. Cet affichage a été obtenu par le petit logiciel VoirHexa (voir http://www.michelmichaud.com/VoirHexa).

Le codage UCS-2 est le simple codage des caractères sur deux octets. Ses codes sont les mêmes que ceux d'UTF-16 sur 16 bits, mais UTF-16 permet des codes impossibles en UCS-2 (ceux qui ont un numéro supérieur à U+FFFF). Dans beaucoup de logiciels et de langages de programmation, UCS-2 est le format utilisé en mémoire, même s'il ne permet pas formellement tout le jeu de caractères Unicode.

Autant UTF-16 qu'UCS-2 présentent un problème d'ordonnancement des octets dans les fichiers : une valeur sur deux octets pouvant avoir son octet de poids fort écrit avant ou après son octet de poids faible (XXYY ou YYXX)[3] selon l'architecture de l'ordinateur. Au besoin, pour les différencier, la signature U+FEFF peut être ajoutée : selon que le logiciel détecte FF FE ou FE FF, il saura l'ordre des octets du reste du fichier. On remarque que c'est le même caractère en signature que pour UTF-8, mais l'encodage UTF-8 (EF BB BF) permet directement de le différencier des deux versions d'UTF-16.

En résumé

- La norme ASCII est un peu la norme de base sur laquelle sont basées les autres. Il s'agit d'un code de 128 caractères seulement, ne donnant pas tous les caractères dont on a besoin.
- Par défaut, un fichier sans indication de contenu pourrait être pris pour un fichier ASCII ou plus probablement pour un fichier UTF-8. Aucun caractère de numéro supérieur à 127 ne devrait s'y trouver directement, car un octet supérieur à 127 identifie un encodage UTF-8 et non le caractère lui-même.
- L'emploi de ISO-8859-1 ou ISO-8859-15 permet d'avoir les plus petits fichiers possibles pour des contenus utilisant l'alphabet latin. Par contre, aucun caractère Unicode de numéro supérieur à 255 ne peut être utilisé directement[4]. En HTML, on peut passer par les entités HTML pour avoir les autres caractères, mais ceci présente un problème de lisibilité et affectera à la hausse la taille du fichier. Les 128 premiers caractères des codes ISO-8859 sont les mêmes que ceux du code ASCII. Les codages ISO-8859 n'évoluent plus et sont peu à peu abandonnés au profit de la norme Unicode.
- La norme Unicode définit des caractères numérotés, sans préciser comment ils seront conservés en mémoire et sur disque. Les 256 premiers caractères sont identiques à ceux d'ISO-8859-1.
- Le format UTF-8 est celui qui offre le meilleur compromis d'espace et de flexibilité pour l'encodage des caractères Unicode, car la plupart d'entre eux, dans un texte d'une langue latine, seront codés sur un seul octet, mais on peut employer tous les numéros de caractères d'Unicode au besoin[5].
- Les fichiers UTF-8 peuvent avoir une signature, mais elle n'est pas recommandée.[6]
- Les données de caractères usuels peuvent être conservées dans des blocs de 16 bits (UCS-2), qui est aussi compatible avec le format UTF-16 qui, lui, permet de coder tous les caractères Unicode.

Notons finalement que ce document parle de données (et de fichiers) contenant du texte seulement...

---

[3] Le nom anglais pour ce principe est *endianness*, avec les deux versions *little endian* et *big endian*. En français, on voit parfois boutisme : petit-boutiste et gros-boutiste.

[4] Ce qui arrive parfois accidentellement, lors d'un copier-coller du web ou d'un document Word, vers un fichier HTML.

[5] Mais plusieurs caractères de numéros moins communs ne sont pas dessinés correctement par tous les fureteurs.

[6] Le validateur HTML5 donne d'ailleurs un avertissement à ce propos. C'est un peu exagéré puisque les fureteurs, d'après les normes web, doivent accepter l'UTF-8 (et ils le font tous correctement, signature ou non).