Jeux de caractères codés

Histoire

Structure

Manipulation

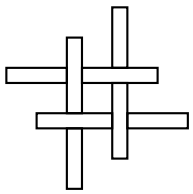
Jean-Marc Bourguet

Jeux de caractères codés

Histoire

Structure

Manipulation



Brouillon de la première édition 28 février 2009

Table des matières

Liste des tableaux

Ce qu'il y a de bien avec les normes,
c'est qu'il y en a tant parmis
lesquelles choisir. [1]

1 Introduction

Nous codons. Nous abstrayons. Nous codons les pensées que nous voulons communiquer sous forme de sons et nous appelons cela la parole. Nous abstrayons l'essentiel de la parole et nous appelons cela une langue. Nous codons les langues sous forme de dessins et nous appelons cela l'écriture. Nous abstrayons l'essentiel de ces dessins et nous appelons cela des lettres. Les codes sont utilisés partout dans notre vie. Les panneaux routiers en sont un. Les joueurs de bridge les utilisent pour donner des informations sur leur jeu à leur partenaire tout en essayant de les masquer à leurs adversaires.

Les codes permettent de donner un sens à des choses qui n'en n'ont pas par elles-mêmes. Ou d'en donner d'autres à ce qui en a déjà. Les codes évoluent et acquièrent des conventions supplémentaires qui permettent une expression au delà des limites de ce qu'ils étaient sensés coder au départ. Je change de paragraphe et mon lecteur sait que je vais aborder une nouvelle idée.

À quoi servent donc tous ces codes ? Ils ont été introduits avec trois buts principaux.

Tout d'abord transmettre des informations – les exemples donnés relevaient tous de cet aspect ; conserver l'information n'est pas autre chose qu'un type de transmission où la distance temporelle prime sur la distance spaciale.

Une fois l'information codée, elle est accessible à tout ceux qui ont accès à la représentation codée et qui connaissent le code, ce qui peut être inconvéniant comme pour nos joueurs de bridge. Surtout que l'expérience a montré que la connaissance du code n'était pas toujours nécessaire pour extraire l'information parce qu'on pouvait souvent le déduire avec un effort plus ou moins important. Ce qui mène au deuxième objectif dans lequel on a conçu des codes : dissimuler l'information. La *cryptographie* cherche à dissimuler l'information à ceux qui n'ont pas la clé nécessaire, même s'ils en connaissent la présence et la nature du code utilisé, la *stéganographie* cherche à dissimuler la présence même de l'information – qui peut en plus être codée avec les techniques de la cryptographie.

Le troisième objectif est le traitement automatisé de l'information.

Nous traiterons ici des codes utilisés pour la transmission et le traitement de l'information. Et plus précisément, ne seront traités que des codes basés sur l'écriture.

2 Structure

Pour décrire les jeux de caractères codés, il faut avoir un modèle permettant d'en donner la structure. En voici un, inspiré de ceux de [ISO2022], [RFC2130] et [UTR17]

1. J'ai vu cet aphorisme attribué à Andrew Tanenbaum, mais je n'ai pas de référence.

mais qui n'est identique à aucun d'eux ; les objectifs étant différents.

Le *caractère* est l'unité de base décrivant ce qui sera codé.

Un *jeu de caractères* ou encore *répertoire de caractères* est un ensemble de caractères, considéré complet pour l'usage auquel on destine le jeu de caractères.

Un *jeu de caractères codé*, c'est un répertoire de caractères auquel on ajoute une association d'un nombre à chaque caractère. Ce nombre est appelé *codet*.

Le *mécanisme de codage* permet de passer d'une suite de caractères, provenant d'un ou de plusieurs jeux de caractères codés, en une suite de *bytes*, l'unité de base de la représentation codée [2].

Il est possible de considérer que le problème du codage des caractères est terminé quand on a une suite de bytes. Mais il faut parfois tenir des des problèmes pratiques, qui existent chaque fois qu'on a à manipuler des données.

Premièrement, le byte du mécanisme de codage peut ne pas correspondre à l'unité de base du système cible. Il faut alors *sérialiser* la suite de bytes obtenus.

Deuxièmement, quand on veut transmettre le résultat, on peut avoir à effectuer un *surcodage* pour respecter les contraintes du protocole utilisé.

2.1 Caractères

Les caractères sont naturellement les différents symboles utilisés par l'écriture. Les lettres, les chiffres, les idéogrammes, les signes de ponctuation, ... Mais il ne faut pas confondre caractère et glyphe. Un *glyphe*, c'est un dessin. Le caractère, c'est la notion abstraite que certains glyphes désignent. « $\mathscr{A}$ », « A », « A » sont trois glyphes différents. Et ces trois glyphes sont des représentations du même caractère : la lettre A majuscule. Si le principe est simple, son application est plus difficile. Examinons quelques problèmes potentiels.

Il n'est pas toujours simple de savoir si on a affaire à un caractère ou à plusieurs. Par exemple, est-ce que les accents doivent être considérés comme des caractères séparés ? Est-ce que « à » est un caractère ou le résultat de la combinaison d'un « a » avec un « ` » ? Pour une langue comme le français où les combinaisons possibles sont relativement peu nombreuses, avoir un caractère pour chaque lettre accentuées a des avantages. Mais quand le nombre de combinaisons possibles augmente, il peut être intéressant de structurer plus le code et de considérer les accents comme des caractères en soi.

Les ligatures ont le même genre de problèmes. Les ligatures linguistiques comme « œ » sont fortement candidates à être des caractères, la substition de la paire de caractères « oe » par « œ » ne pouvant être faite automatiquement qu'en utilisant un dictionnaire ou une connaissance approfondie de la langue utilisée. Mais qu'en est-il des ligatures typographiques (« fi » plutôt que « fi ») dont l'utilisation résulte plus de considérations typographiques que linguistiques ? Et il ne faut pas oublier que les considérations linguistiques ne sont pas totalement absentes, si même on dispose de la ligature « ff » on peut préférer « shelfful » (sans ligature) à « shelfful » (avec une ligature).

2. Le mot *byte* est aussi utilisé avec au moins deux autres acceptions : la plus petite unité adressable par un ordinateur et une chaîne d'exactement 8 bits. Dans ce dernier sens, il faut préférer le mot *octet* dont c'est l'unique signification.

Parfois, ce qui est considéré comme une lettre par les locuteurs d'une langue correspond à plusieurs glyphes, le choix du glyphe se faisant en tenant compte du contexte. Par exemple est-ce qu'il faut avoir un caractère pour la lettre grec sigma, en considérant les variantes « ς » et « σ » comme des glyphes différents que le moteur d'affichage doit choisir suivant qu'il s'agit de la dernière lettre d'un mot ou pas ou faut-il encoder chacune des variantes comme un caractère ? Et si on considère d'autres langues, le nombre de formes peut être beaucoup plus important, l'arabe par exemple a jusqu'à quatre formes pour ses lettres (version isolée, initiale, finale et en milieu de mot) . Si avoir deux caractères peut être sensé pour le grec, multiplier par quatre le nombre de caractères est exclus quand on a un nombre de codets limités.

Et si généralement plusieurs glyphes peuvent représenter un caractère, parfois, deux caractères distincts peuvent être représentés par un même glyphe. Par exemple, le « A » de tout à l'heure pourrait être aussi bien un alpha majuscule. Est-ce qu'il faut avoir un caractère pour la lettre A majuscule et un autre pour la lettre alpha majuscule ou bien n'avoir qu'un caractère ?

Ces questions ne sont pas simples. En fait, il n'y a pas de bonne réponse universelle à leur apporter. Suivant les contraintes et les applications prévues, on peut faire – et on a fait – des choix différents. Même quelque chose d'apparemment aussi simple que de dire que « $\mathscr{A}$ » et « A » sont des glyphes différents d'un même caractère peut être remis en question si on considère les mathématiques où ces deux glyphes peuvent désigner des choses différentes et donc leur différence ne pas simplement être un embellissement typographique.

En plus des caractères représentant les symboles de l'écriture, les jeux de caractères comportent souvent une série d'autres caractères sans correspondance graphique, qui ont été ajoutés parce qu'ils étaient utiles. Ce sont les *caractères de commande* (souvent appelés *caractères de contrôle*) car la plupart d'entre eux servent à commander les périphériques.

Entre les caractères de commande et les caractères graphiques, il y a l'espace, caractère dont on s'est demandé s'il est un caractère graphique complètement blanc ou un caractère de commande indiquant qu'il faut avancer la position d'écriture. La première alternative semble s'être imposée.

On peut diviser les caractères de commandes en plusieurs classes (voir [ISO2382-4]) :
- ceux servant à contrôler les transmissions ;
- ceux servant à contrôler la mise en page ;
- ceux servant à gérer les codes ;
- ceux servant à commander les périphériques.

Les fonctions que l'on désire encoder avec les caractères de contrôle sont tellement nombreuses qu'on se sert de combinaisons de caractères, commençant souvent par un caractère de commande nommé ESC (voir [ISO6429]).

2.2 Jeu de caractères codé

On emploie aussi les termes de *charset*, *codeset*, *page de code* (surtout chez IBM et Microsoft) pour designer les jeux de caractères codés.

Et codet n'est pas non plus dénué de synonymes : *valeur scalaire*, *élément de code*, *position de code*, *point de code* et souvent même *code* malgré l'ambiguïté possible.

Si chaque caractère du répertoire doit avoir un codet, un codet donné peut correspondre à plusieurs caractères. Les jeux de caractères associant plusieurs caractères à un codet sont des jeux modaux : pour les décoder il faut se souvenir dans quel mode on est. Le changement de mode s'effectuant souvent à l'aide de caractères de contrôle.

2.3 Mécanisme de codage

Le mécanisme de codage décrit donc comment passer des codets à une suite de bytes. Le plus simple, c'est la fonction identité (chaque codet produit un byte de la valeur du codet) et il est souvent utilisé implicitement. Certains mécanismes permettent de combiner des jeux de caractères codés différents. Dans ce cas, le mécanisme peut toujours être très simple : continuer à utiliser la fonction identité tout en imposant que les codets soient différents, ou beaucoup plus compliqué (voir 3.7) en utilisant des caractères de contrôle pour indiquer le changement de charset.

Les jeux de caractères comportant un nombre important de caractères (ce qui est le cas de ceux permettant de coder les écritures idéographiques) ont un problème avec l'utilisation de la fonction identité : ils exigeraient alors des bytes de grandes tailles. Ils prévoient donc des moyens d'utiliser des bytes plus petits, mais en utilisant parfois plusieurs bytes pour un caractère.

Il faut noter que la différence entre charset et mécanisme de codage n'est pas toujours faite. C'est souvent le cas quand on envisage l'utilisation de la fonction identité comme mécanisme de codage. Mais par exclusivement ; on parle parfois de charset multibyte pour désigner la combinaison d'un charset ayant un nombre important de caractères et un mécanisme de codage utilisant parfois plus d'un byte pour représenter un caractère.

2.4 Sérialisation

Si la taille du byte ne correspond pas à une unité manipulée naturellement par le système, il faut alors prévoir comment représenter ceux-ci.

On peut alors avoir à regrouper plusieurs bytes en un mot. Par exemple, le PDP-10 a des mots de 36 bits. Il a été utilisé avec des mécanismes de codage utilisant des bytes de 6 bits (on mettait 6 caractères par mot), de 7 bits (on mettait 5 caractères par mot, et un bit était inutilisé) et de 9 bits (4 caractères par mot). Savoir que plusieurs bytes se trouvent dans le même mot ne suffit pas, il faut encore connaître leur position respective. Dans le cas du PDP-10, c'était déterminé par le jeu d'instruction de ce processeur qui a une notion de pointeur vers bytes où les pointeurs en question contiennent le nombre de bits du byte, de 1 à 36. Naturellement, la position des bytes dans le mot correspondait à ce qu'il fallait pour utiliser ces instructions.

On peut aussi avoir le problème inverse et devoir découper un byte en plusieurs unités. Par exemple Unicode définit un mécanisme de codage appelé UTF-16 qui utilise des bytes de 16 bits. Pour les représenter sur un ordinateur ayant comme unité de base

l'octet on a le choix : si on place la partie la plus significative du byte en premier, la sérialisation est *gros-boutiste*, si la partie la moins significative est en premier, elle est *petit-boutiste*.

2.5 Surcodage

Le surcodage est nécessaire quand le résultat, éventuellement sérialisé en mémoire, du codage n'est pas utilisable comme tel. Par exemple parce qu'il faut le transmettre avec un protocole qui n'a pas la même unité de base, ou qui interdit certaines valeurs.

Le surcodage est généralement transparent pour l'utilisateur ; et le système qui l'effectue n'a pas nécessairement connaissance qu'il est en train de manipuler des caractères plutôt qu'un autre type de donnée.

L'exemple le plus commun de surcodage visible – heureusement de moins en moins – est avec le protocole SMTP [RFC821], utilisé pour transmettre les couriels. Il est définit comme opérant sur des octets, mais sans garantie de conservation du bit de poids fort. De plus les lignes sont de longueurs limitées. Ça fonctionne bien avec l'ASCII – qui est un code sur 7 bits – mais les données utilisant un code sur 8 bits peuvent être modifiées dans le transport.

Un protocole annexe, MIME [RFC1521], résouds ce problème – et d'autres, il permet d'indiquer dans le message le charset utilisé – tout en conservant une bonne compatibilité avec les systèmes antérieurs. Bonne compatibilité en ce sens qu'avec un logiciel qui ne comprend pas MIME (ce qui est maintenant rare, sauf quand les mécanismes de MIME sont utilisés là où ce n'est pas prévu), on se retrouve avec une partie du message dégradée, mais souvent encore compréhensible.

MIME définit deux surcodages qui peuvent être utilisés pour transmettre un code sur 8 bits : un (*quoted printable*) à utiliser plutôt quand les données sont principalement des codets avec les 8^{e} bit à 0, seuls ces codets sont transformés ce qui laisse souvent le message surcodé compréhensible. Par exemple si un couriel est écrit en utilisant le code ISO 8859-1, les lettres sans accents sont transmises sans changement et un « é », par exemple, devient « `=E9` » – le codet de « é » étant `E9` exprimé en hexadécimal. Le deuxième surcodage (*base64*) est à utiliser dans les autres cas, et le message est alors incompréhensible sans décodeur.

3 Quelques codes

Les codes utilisés en transmission ont une longue histoire. Au deuxième siècle avant notre ère, le général et historien grec Polybe (Πολύβιος, v. 200 – v. 120 av. J.-C.) propose un système pour coder l'alphabet grec en utilisant deux groupes de cinq torches. Ce système a eu un certain succès : au XX^{e} siècle les prisonniers de guerre américains au Viêt Nam auraient utilisé une variante de ce système, adapté à l'alphabet latin [Wika].

Les marins de la Grèce antique utilisaient déjà des drapeaux pour communiquer entre navires d'une même flotte. Ce moyen de communication était relativement inflexible mais en 1738, Bertrand François Mahé de la Bourdonnais a conçu un code plus complexe. Dix drapeaux différents codaient les dix chiffres, ensuite trois drapeaux per-

mettaient de coder 1000 messages, un dictionnaire donnait la correspondance entre les nombres et les messages. Ce système ne fut pas directement employé, mais influença vraisemblablement Lord Richard Howe qui inclu un mécanisme similaire dans son « The Howe Code ». Par la suite, ce code fut complèté par Sir Home Popham dans son « Telegraphic Signals or Marine Vocabulary » qui finit de le perfectionner en codant individuellement les lettres, permettant ainsi d'épeller ce qui ne se trouve pas dans le dictionnaire. Ce qui, en 1805 à Trafalgar, permit à l'amiral Horatio Nelson de transmettre le célèbre message « England expects that every man will do his duty » bien que « duty » ne soit pas dans le dictionnaire ([Mü]).

À la fin du dix-huitième siècle, les frères Chappe établirent un système de sémaphores composés de tours équipées de bras articulés. Comme le code de Popham, le code utilisé était aussi un système à dictionnaire disposant d'un moyen d'épeller ce qui n'était pas dans le dictionnaire. Ce système fut utilisé par Napoléon pour gouverner son empire et commander ses armées, lui donnant un avantage sur ses ennemis [Wikb], [Kat].

3.1 Le code Morse

A	·—	M	——	Y	—·——
B	—···	N	—·	Z	——··
C	—·—·	O	———	0	—————
D	—··	P	·——·	1	·————
E	·	Q	——·—	2	··———
F	··—·	R	·—·	3	···——
G	——·	S	···	4	····—
H	····	T	—	5	·····
I	··	U	··—	6	—····
J	·———	V	···—	7	——···
K	—·—	W	·——	8	———··
L	·—··	X	—··—	9	————·

TABLE 1: Le code Morse

En 1837 Samuel Morse brevette un télégraphe électrique. Le code qu'il propose alors est apparemment basé uniquement sur un dictionnaire, mais il n'a pas été utilisé. C'est son assistant, Alfred Vail, qui aurait travaillé sur le code et proposé par après ce qu'on appelle le code Morse (voir table 1). Ce code est basé sur cinq symboles (deux durées de courant sur la ligne – représentées par « · » et « — » dans la table – et trois durées d'absence de courant – séparant les points et les traits, les lettres et les mots). Dans ce code tous les caractères codés ne sont pas composés du même nombre de symboles, ce qui peut être un avantage dans une application de transmission – après tout, c'est un principe de compression bien connu que de coder avec moins de symboles ce qui est fréquent ; ce n'est pas un hasard si le « E », lettre la plus fréquente

en anglais comme en français, est codée par « · » – mais est plutôt une nuisance quand il s'agit de traiter automatiquement ces informations.

3.2 Le code Baudot

En 1870 et toujours pour les transmissions télégraphiques, Émile Baudot invente un système télégraphique multiplexant plusieurs messages sur une ligne. Ce système était composés de claviers à cinq touches permettant aux opérateurs de composer les messages, du système de multiplexage et du système de réception qui séparait les différents messages et les imprimait.

Ce système employait un code (voir table 2 [3]) à deux symboles (impulsion positive ou négative), c'est donc un code binaire, comme tout ceux que nous rencontrerons par la suite. Les caractères étaient codés en utilisant des motifs de cinq *bits* (*moment* dans la terminologie de l'époque), une longueur fixe donc, au contraire du code Morse. Le code aurait été conçu de manière à limiter la fatigue de l'opérateur, structure que la représentation tabulaire utilisée ici ne met pas en valeur.

Ce code est un code modal. Pour permettre plus de 32 caractères (ce qui ne permet même pas de coder 26 lettres plus 10 chiffres), il utilise deux modes : certains motifs sont utilisés pour coder deux caractères différents, le choix entre eux étant effectué en fonction du mode courant. Le changement de mode est notifié par l'envoi des caractères notés (FB) et (LB) dans le tableau.

Le code ne prévoit pas de représentation pour l'espace, les caractères utilisés pour changer de modes (FB) et (LB) ont aussi cette fonction. Ce qui présente un avantage : on est certain qui si un de ces caractères est corromptu – et donc que le changement de mode ne se fait pas –, il en viendra bientôt un autre qui remettra les choses en ordre.

3.3 CCITT#2

Vers 1900, Donald Murray développa un système télégraphique inspiré de celui de Baudot, mais remplaça le clavier à cinq touches par un clavier semblable à celui d'une machine à écrire. Il changea aussi le code, cette fois-ci avec pour objectif de faciliter la mécanique de son système. Son code, légèrement modifié, devint le l'*alphabet télégraphique international n° 2* [4] (voir table 3).

Aux États-Unis, une variante de ce code appelée USTTY a été utilisée.

À noter qu'il est d'usage d'appeler ce code « Baudot » même si ce n'est pas le code Baudot proprement dit.

			Lettres		Chiffres	
HEX			0	1	0	1
	BIN		0	1	0	1
		OCT	*0*	*2*	*0*	*2*
0	0000	*0*	0 (?)	16 (LB)	0 (?)	16 (LB)
1	0001	*1*	1 A	17 t	1 1	17 .
2	0010	*2*	2 E	18 X	2 2	18 ,
3	0011	*3*	3 É	19 Z	3 &	19 :
4	0100	*4*	4 Y	20 S	4 3	20 ;
5	0101	*5*	5 U	21 T	5 4	21 !
6	0110	*6*	6 I	22 W	6 o	22 ?
7	0111	*7*	7 O	23 V	7 5	23 ’
8	1000	*0*	8 (FB)	24 (ERA)	8 (FB)	24 (ERA)
9	1001	*1*	9 J	25 K	9 6	25 (
A	1010	*2*	10 G	26 M	10 7	26)
B	1011	*3*	11 H	27 L	11 h	27 =
C	1100	*4*	12 B	28 R	12 8	28 —
D	1101	*5*	13 C	29 Q	13 9	29 /
E	1110	*6*	14 F	30 N	14 f	30 N°
F	1111	*7*	15 D	31 P	15 0	31 %
		OCT	*1*	*3*	*1*	*3*

(?) pas utilisé
LS Letter-Blank (passage aux lettres + espace)
FS Figure-Blank (passage aux chiffres + espace)
ERA Erasure (annulation du caractère précédent)

TABLE 2: Le code Baudot

			Lettres		Chiffres	
HEX			0	1	0	1
	BIN		0	1	0	1
		OCT	*0*	*2*	*0*	*2*
0	0000	*0*	0 (2)	16 E	0 (2)	16 3
1	0001	*1*	1 T	17 Z	1 5	17 +
2	0010	*2*	2 (CR)	18 D	2 (CR)	18 (WRU)
3	0011	*3*	3 O	19 B	3 9	19 ?
4	0100	*4*	4 (SP)	20 S	4 (SP)	20 ’
5	0101	*5*	5 H	21 Y	5 (1)	21 6
6	0110	*6*	6 N	22 F	6 ,	22 (1)
7	0111	*7*	7 M	23 X	7 .	23 /
8	1000	*0*	8 (LF)	24 A	8 (LF)	24 -
9	1001	*1*	9 L	25 W	9)	25 2
A	1010	*2*	10 R	26 J	10 4	26 (Bell)
B	1011	*3*	11 G	27 (FS)	11 (1)	27 (FS)
C	1100	*4*	12 I	28 U	12 8	28 7
D	1101	*5*	13 P	29 Q	13 0	29 1
E	1110	*6*	14 C	30 K	14 :	30 (
F	1111	*7*	15 V	31 (LS)	15 =	31 (LS)
		OCT	*1*	*3*	*1*	*3*

(1) usage local (lettres accentuées par exemple)
(2) pas utilisé
WRU Who are you
CR Carriage-return (retour chariot)
LF Line-feed (passage à la ligne)
LS Letter-shift (passage aux lettres)
FS Figure-shift (passage aux chiffres)
SP Space (espace)

TABLE 3: Le code CCITT#2 ou alphabet télégraphique international n° 2

	HEX		0	1	2	3
	BIN		00	01	10	11
		OCT	*0*	*2*	*4*	*6*
0	0000	*0*	0 (SP)	16 ƀ	32 -	48 & ou +
1	0001	*1*	1 1	17 /	33 J	49 A
2	0010	*2*	2 2	18 S	34 K	50 B
3	0011	*3*	3 3	19 T	35 L	51 C
4	0100	*4*	4 4	20 U	36 M	52 D
5	0101	*5*	5 5	21 V	37 N	53 E
6	0110	*6*	6 6	22 W	38 O	54 F
7	0111	*7*	7 7	23 X	39 P	55 G
8	1000	*0*	8 8	24 Y	40 Q	56 H
9	1001	*1*	9 9	25 Z	41 R	57 I
A	1010	*2*	10 0	26 ǂ	42 !	58 ?
B	1011	*3*	11 # ou =	27 ,	43 $	59 .
C	1100	*4*	12 @ ou ’	28 % ou (	44 *	60 ¤ ou)
D	1101	*5*	13 :	29 γ	45]	61 [
E	1110	*6*	14 >	30 \	46 ;	62 <
F	1111	*7*	15 ✓	31 ǂǂ	47 Δ	63 ǂǂǂ
		OCT	*1*	*3*	*5*	*7*

SP	Space (espace)
ƀ	Substitute Blank
Δ	Mode Change
γ	Word Separator
ǂ	Record Mark
ǂǂ	Group Mark
ǂǂǂ	Segment Mark
✓	Tape Mark
& # @ % ¤	version commerciale
+ = ’ ( )	version FORTRAN

TABLE 4: Le code BCDIC

3.4 BCDIC

L'automatisation du traitement des données est lui plus récent. C'est durant les années 1880 qu'Herman Hollerith développe les premières tabulatrices qui seront utilisées pour le traitement du recensement américain de 1890. Il fonde alors une compagnie qui deviendra après quelques fusions IBM. Dans ces systèmes, les informations sont codées par des trous dans des cartes. Les premiers codes pour les cartes perforées avaient 12 caractères : les 10 chiffres et deux caractères de contrôle. Les cartes étaient composées d'un certain nombre de colonnes, chaque colonne ayant 12 positions, une pour chaque caractère possible. Les rangées portaient comme nom le chiffre qu'elle codait. Le nom des deux rangées supplémentaires varie suivant les sources – 10 et 11, 11 et 12, A et B, X et Y – ce texte utilise X et Y. Une des utilisations des caractères de contrôle X et Y fut d'indiquer si le nombre était positif ou négatif.

Ce code fut étendu à 40 caractères dans les années 30 puis à 48 caractères dans les années 50. Les caractères correspondent alors à des combinaisons de plusieurs trous. Les trous sont séparés en deux groupes : ceux sur des rangées dites *de zones* et ceux sur des rangées *de chiffres* permettant d'adresser un tableau a double entrée – avec souvent quelques exceptions. Au départ les rangées de zones était les rangées X et Y, et les rangées de chiffres... celles des chiffres. Mais par la suite les rangées 0 et 9 ont été utilisées comme rangées de zones. Jusqu'alors, nous avons un code défini purement pour les cartes perforées. On pourrait dire qu'il s'agit d'un code utilisant un byte de 12 bits ; mais il n'est pas perçu comme tel. D'autres types de codes ont été utilisés avec les cartes perforées, certains plus proches d'un codage en binaire.

Pour faciliter le traitement par les ordinateurs, on ajoute au code Hollerith, défini par des combinaison de trous dans les 12 rangées des cartes, un codage numérique sur 6 bits : le code BCDIC *Binary Coded Decimal Interchange Code* (voir table 4). Le code BCDIC tel que présenté ici est le résultat final de l'évolution chez IBM. Ce code respecte la structure du code pour carte perforée : les colonnes correspondent à une combinaison donnée dans les rangées de zones et les rangées correspondent à une combinaison de trous dans les rangées de chiffres [5].

Il y a eu quelques variantes de ce code. Deux sont données ici : la variante commerciale, utilisée pour ce genre d'application, et la variante FORTRAN, utilisée pour programmer dans ce langage.

Si tous les caractères ont une représentation graphique, certains caractères étaient aussi utilisés comme caractère de commande. En particulier, certains étaient interprétés dans tous les contextes comme des commandes par les lecteur de bandes magnétiques ; ils n'étaient donc pas stockables sur bande sans un surcodage, ce qui n'était en pratique pas mis en œuvre.

Pour plus de renseignements sur les codes pour cartes, voir [Mac80, Win, Jona, Jonb].

3. Certains codets étaient utilisés pour d'autres caractères sur certaines lignes – les caractères duaux et variantes localisées, ça ne date pas de l'ASCII ! La variante donnée ici est celle pour les lignes internationales.

4. Mes sources divergent quant à savoir quel code était l'*Alphabet télégraphique international n° 1*, certaines donnent le code Morse, d'autre le code Baudot original ou une variante proche.

5. Il y a quelques exceptions causées par le fait que 0 est à la fois une rangée de zone et de chiffres.

HEX			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	BIN		00				01				10				11			
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		OCT	*0*	*2*	*4*	*6*	*10*	*12*	*14*	*16*	*20*	*22*	*24*	*26*	*30*	*32*	*34*	*36*
0	0000	*0*	0 (NUL)	16 (DLE)	32 (PAD)	48 (DCS)	64 (SP)	80 &	96 -	112	128	144	160	176	192 {	208 }	224 \	240 0
1	0001	*1*	1 (SOH)	17 (DC1)	33 (HOP)	49 (PUL)	65	81	97 /	113	129 a	145 j	161 ~	177	193 A	209 J	225	241 1
2	0010	*2*	2 (STX)	18 (DC2)	34 (BHP)	50 (SYN)	66	82	98	114	130 b	146 k	162 s	178	194 B	210 K	226 S	242 2
3	0011	*3*	3 (ETX)	19 (DC3)	35 (NBH)	51 (STS)	67	83	99	115	131 c	147 l	163 t	179	195 C	211 L	227 T	243 3
4	0100	*4*	4 (ST)	20 (OC)	36 (IND)	52 (CCH)	68	84	100	116	132 d	148 m	164 u	180	196 D	212 M	228 U	244 4
5	0101	*5*	5 (HT)	21 (NEL)	37 (LF)	53 (MW)	69	85	101	117	133 e	149 n	165 v	181	197 E	213 N	229 V	245 5
6	0110	*6*	6 (SSA)	22 (BS)	38 (ETB)	54 (SPA)	70	86	102	118	134 f	150 o	166 w	182	198 F	214 O	230 W	246 6
7	0111	*7*	7 (DEL)	23 (ESA)	39 (ESC)	55 (EOT)	71	87	103	119	135 g	151 p	167 x	183	199 G	215 P	231 X	247 7
8	1000	*0*	8 (EPA)	24 (CAN)	40 (HTS)	56 (SOS)	72	88	104	120	136 h	152 q	168 y	184	200 H	216 Q	232 Y	248 8
9	1001	*1*	9 (RI)	25 (EM)	41 (HTJ)	57 (SGC)	73	89	105	121 `	137 i	153 r	169 z	185	201 I	217 R	233 Z	249 9
A	1010	*2*	10 (SS2)	26 (PU2)	42 (VTS)	58 (SCI)	74 ¢	90 !	106 ¦	122 :	138	154	170	186	202	218	234	250
B	1011	*3*	11 (VT)	27 (SS3)	43 (PLD)	59 (CSI)	75 .	91 $	107 ,	123 #	139	155	171	187	203	219	235	251
C	1100	*4*	12 (FF)	28 (IS4)	44 (PLU)	60 (DC4)	76 <	92 *	108 %	124 @	140	156	172	188	204	220	236	252
D	1101	*5*	13 (CR)	29 (IS3)	45 (ENQ)	61 (NAK)	77 (	93)	109 _	125 ’	141	157	173	189	205	221	237	253
E	1110	*6*	14 (SO)	30 (IS2)	46 (ACQ)	62 (PM)	78 +	94 ;	110 >	126 =	142	158	174	190	206	222	238	254
F	1111	*7*	15 (SI)	31 (IS1)	47 (BEL)	63 (SUB)	79 |	95 ¬	111 ?	127 "	143	159	175	191	207	223	239	255 (APC)
		OCT	*1*	*3*	*5*	*7*	*11*	*13*	*15*	*17*	*21*	*23*	*25*	*27*	*31*	*33*	*35*	*37*

TABLE 5: Le code EBCDIC

3.5 EBCDIC

Au début des années 60, les insuffisances des jeux de caractères 6 bits tels que le BCDIC était bien connues : d'une part, ils n'avaient pas assez de caractères graphiques, donnant donc naissance à des variantes pour des usages particuliers, d'autre part ils n'avaient pas assez de caractères de contrôle, donnant donc naissance à des codes particuliers pour communiquer avec les équipements, et à la nécessité de conversions.

Une fois choisi de baser le System/360 autour d'une unité d'adressage de 8 bits, c'était l'occasion de concevoir un jeu de caractères codé qui s'affranchissait de ces limitations en profitant des 256 points de code disponibles. C'était a peu près au même moment que le développement de l'ASCII commençait, mais avec des contraintes

différentes.

Pour les clients d'IBM – et donc pour IBM – la compatibilité avec BCDIC était d'une importance majeure. Ils avaient déjà beaucoup d'information codée sur des cartes perforées. Avoir une conversion simple était une nécessité. Et respecter la structure globale aussi : une technique habituelle à l'époque était d'utiliser indépendamment les rangées de zones et de chiffres. C'est-à-dire que les caractères – ou plutôt les combinaisons de trous correspondant à ces caractères – ? et A à I étaient utilisés pour les chiffres 0 à 9 combinés avec un drapeau, de même que les caractères ! et J à R. Conserver le même code pour les cartes perforées pour les lettres étaient une des contraintes que les concepteurs d'EBCDIC s'étaient imposé, et conserver une équivalence simple entre les chiffres combinés avec d'autres informations en était une autre. On a l'explication de pourquoi les lettres sont ainsi codées en EBCDIC – mais par rapport au BCDIC, elles sont au moins dans l'ordre alphabétique.

La prise en compte de ces considérations et d'autres considérations a fini, après quelques compromis parce que les contraintes étaient incompatibles, par donner le code montré en table 5.

Malgré l'objectif de supprimer les variantes, celles d'EBCDIC sont nombreuses. Mais un progrès par rapport à BCDIC, un site donné pouvait généralement trouver une variante adéquate à l'ensemble de ses applications : les variantes étant plutôt dues à la localisation. Dans la variété des codes EBCDIC, il est à remarquer que certains caractères courants n'ont pas le même code dans toutes les versions.

3.6 ASCII

Ce code (voir table 6) est un code 7 bit développé aux États-Unis durant les années 60 (avec des étapes en 63, 65, 67 et 68 ; la version présentée ici est celle de 68 d'après [Mac80] [6]. Ses concepteurs étaient conscients qu'il était peu probable que des systèmes utilisent exactement 7 bits de manière interne [Mac80, P. 217]. Le choix d'un code sur 7 bits a cependant été fait car il satisfaisait les besoins de la plupart des utilisations prévues, que donc un code à 8 bits aurait été perçu comme imposant un coût inutile à la majorité et finalement que l'ajout d'un huitième bit ne permettait plus l'utilisation aisée de certaines techniques comme les rubans perforés (qui permettaient jusqu'à 8 bits mais un de ceux-ci était utilisé comme bit de parité).

Par la suite, il fut normalisé par l'ISO (sous le nom ISO 646, ECMA a publié une norme techniquement équivalente sous le nom ECMA 6 qui a l'avantage d'être disponible publiquement [ECMA6]). Cette norme prévoit la possibilité pour des variantes nationales de changer certains caractères pour correspondre à des nécessités nationales (par exemple l'ajout de lettres accentuées). De même, les variantes nationales peuvent prévoir des séquences combinantes (par exemple la variante française prévoyait l'utilisation de la séquence « B̂S a » pour le caractère « â »).

Les caractères pouvant avoir une définition nationale sont #, $, @, [, \,], ^, `, {, |, }, ~ et de plus les choix pour # et sont restreints (# peut être remplacé par £, et $ par ¤ ; ce dernier caractère – le symbole monétaire – a d'ailleurs été présent dans certaines versions de référence).

6. Voir 4.2 pour l'utilisation de ¦ plutôt que de |.

	HEX		0	1	2	3	4	5	6	7
	BIN		000	001	010	011	100	101	110	111
		OCT	*0*	*2*	*4*	*6*	*10*	*12*	*14*	*16*
0	0000	*0*	0 (NUL)	16 (DLE)	32 (SP)	48 0	64 @	80 P	96 `	112 p
1	0001	*1*	1 (SOH)	17 (DC1)	33 !	49 1	65 A	81 Q	97 a	113 q
2	0010	*2*	2 (STX)	18 (DC2)	34 "	50 2	66 B	82 R	98 b	114 r
3	0011	*3*	3 (LTX)	19 (DC3)	35 #	51 3	67 C	83 S	99 c	115 s
4	0100	*4*	4 (EOT)	20 (DC4)	36 $	52 4	68 D	84 T	100 d	116 t
5	0101	*5*	5 (ENO)	21 (NAK)	37 %	53 5	69 E	85 U	101 e	117 u
6	0110	*6*	6 (ACK)	22 (SYN)	38 &	54 6	70 F	86 V	102 f	118 v
7	0111	*7*	7 (BEL)	23 (ETB)	39 ’	55 7	71 G	87 W	103 g	119 w
8	1000	*0*	8 (BS)	24 (CAN)	40 (	56 8	72 H	88 X	104 h	120 x
9	1001	*1*	9 (HT)	25 (EM)	41)	57 9	73 I	89 Y	105 i	121 y
A	1010	*2*	10 (LF)	26 (SUB)	42 *	58 :	74 J	90 Z	106 j	122 z
B	1011	*3*	11 (VT)	27 (ESC)	43 +	59 ;	75 K	91 [	107 k	123 {
C	1100	*4*	12 (FF)	28 (FS)	44 ,	60 <	76 L	92 \	108 l	124 ¦
D	1101	*5*	13 (CR)	29 (GS)	45 -	61 =	77 M	93]	109 m	125 }
E	1110	*6*	14 (SO)	30 (RS)	46 .	62 >	78 N	94 ^	110 n	126 ~
F	1111	*7*	15 (SI)	31 (US)	47 /	63 ?	79 O	95 _	111 o	127 (DEL)
		OCT	*1*	*3*	*5*	*7*	*11*	*13*	*15*	*17*

TABLE 6: Le code ASCII

Il y eut plusieurs révisions de cette norme. La plus importante fut vraisemblablement de la mettre en conformité avec la structure de ISO 2022 (voir 3.7) en enlevant la description des caractères de contrôle pour la confier à d'autres normes (ISO 6429 alias ECMA 48 [ECMA48] par exemple). Dans cette forme, ce jeu de caractères ne comporte plus que 94 caractères, ceux de codes 33 à 126 (l'espace et DEL font partie de ISO 2022).

L'existence de variantes nationales – parfois plusieurs variantes par pays – a posé suffisemment de problèmes pratiques pour que la France ait fini par retirer ses variantes et adopter la version de référence.

Ce jeu de caractères a servi de base à beaucoup d'autres qui ont rempli les 128 positions laissées libres si on le code avec une unité faisant 8 bits ; ces jeux sont parfois appelé ASCII étendu.

3.7 ISO 2022

La norme ISO 2022 (disponible plus faciment en tant que norme ECMA 35 [ECMA35]) décrit un mécanisme de codage permettant de coder plusieurs jeux de caractères codé respectant une certaine structure dans un même flux.

Trois types de charset sont codables :

– des jeux de 32 caractères de commandes ayant les codets 0 à 31.
– des jeux de 94 caractères graphiques ayant les codets 33 à 126.
– des jeux de 96 caractères graphiques ayant les codets 32 à 127.

Plus les jeux de caractères à $\mathbf{94}^n$ ou $\mathbf{96}^n$ positions.

Dés qu'on utilise une autre langue que l'anglais – en fait, même dès qu'on utilise un anglais cultivé – on a besoin de caractères absents du répertoire de l'ASCII. Les variantes nationales d'ISO 646 étaient une tentative de répondre à ces besoins ; mais étant *nationales*, elles posaient des problèmes dès qu'il fallait communiquer entre nations. Elles posaient aussi d'autres problèmes quand on avait besoin des caractères remplacés, ce qui est le cas par exemple pour écrire des programmes dans un langage de programmation qui faisait usage des ces caractères.

L'ASCII est un jeu de caractères sur 7 bits. Au moment de son introduction, les ordinateurs pouvaient se classer dans deux grandes familles : les ordinateurs commerciaux, destinés à manipuler des caractères représentés sur 6 bits ; et les ordinateurs scientifiques, destinés principalement au calcul et manipulant des mots de taille variable (36 bits était une taille courante). Entre son introduction et le moment où les problèmes évoqués ci-dessus sont devenus criant, les ordinateurs basés sur un byte de 8 bits sont devenus monnaie courante. Pour ces systèmes l'utilisation d'extensions de l'ASCII à 8 bits était naturel.

3.8 ISO 8859-1

La normes ISO 8859 en normalise une série de jeu de caractères à 8 bits, valables pour des ensembles de langues différents. Deux sont pertinentes pour le français :

– ISO 8859-1 (appelé aussi latin-1) qui était sensé être le charset à utilisé pour le français mais qui n'a pas 3 caractères nécessaires : æÆŒŸ. Il était cependant le meilleur choix jusqu'à l'introduction du second.
– ISO 8859-15 (appelé aussi latin-9 et latin-0) qui corrige ce défaut et quelques autres de ISO 8859-1 et qui introduit le caractère €. C'est le charset sur 8 bits à préferer pour le français.

3.9 CP 1252

> The term "ANSI" as used to signify Windows code pages is a historical reference, but is nowadays a misnomer that continues to persist in the Windows community. The source of this comes from the fact that the Windows code page 1252 was originally based on an ANSI draft, which became ISO Standard 8859-1 [Wis02].

HEX			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	BIN		00				01				10				11			
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		OCT	*0*	*2*	*4*	*6*	*10*	*12*	*14*	*16*	*20*	*22*	*24*	*26*	*30*	*32*	*34*	*36*
0	0000	*0*	0	16	32 (SP)	48 0	64 @	80 P	96 `	112 p	128	144	160 (NBS)	176 °	192 À	208 Ð	224 à	240 ð
1	0001	*1*	1	17	33 !	49 1	65 A	81 Q	97 a	113 q	129	145	161 ¡	177 ±	193 Á	209 Ñ	225 á	241 ñ
2	0010	*2*	2	18	34 "	50 2	66 B	82 R	98 b	114 r	130	146	162 ¢	178 ²	194 Â	210 Ò	226 â	242 ò
3	0011	*3*	3	19	35 #	51 3	67 C	83 S	99 c	115 s	131	147	163 £	179 ³	195 Ã	211 Ó	227 ã	243 ó
4	0100	*4*	4	20	36 $	52 4	68 D	84 T	100 d	116 t	132	148	164 ¤	180 ´	196 Ä	212 Ô	228 ä	244 ô
5	0101	*5*	5	21	37 %	53 5	69 E	85 U	101 e	117 u	133	149	165 ¥	181 µ	197 Å	213 Õ	229 å	245 õ
6	0110	*6*	6	22	38 &	54 6	70 F	86 V	102 f	118 v	134	150	166 ¦	182 ¶	198 Æ	214 Ö	230 æ	246 ö
7	0111	*7*	7	23	39 ’	55 7	71 G	87 W	103 g	119 w	135	151	167 §	183 ·	199 Ç	215 ×	231 ç	247 ÷
8	1000	*0*	8	24	40 (	56 8	72 H	88 X	104 h	120 x	136	152	168 ¨	184 ¸	200 È	216 Ø	232 è	248 ø
9	1001	*1*	9	25	41)	57 9	73 I	89 Y	105 i	121 y	137	153	169 ©	185 ¹	201 É	217 Ù	233 é	249 ù
A	1010	*2*	10	26	42 *	58 :	74 J	90 Z	106 j	122 z	138	154	170 ª	186 º	202 Ê	218 Ú	234 ê	250 ú
B	1011	*3*	11	27	43 +	59 ;	75 K	91 [	107 k	123 {	139	155	171 «	187 »	203 Ë	219 Û	235 ë	251 û
C	1100	*4*	12	28	44 ,	60 <	76 L	92 \	108 l	124 |	140	156	172 ¬	188 ¼	204 Ì	220 Ü	236 ì	252 ü
D	1101	*5*	13	29	45 -	61 =	77 M	93]	109 m	125 }	141	157	173 (SHY)	189 ½	205 Í	221 Ý	237 í	253 ý
E	1110	*6*	14	30	46 .	62 >	78 N	94 ^	110 n	126 ~	142	158	174 ®	190 ¾	206 Î	222 Þ	238 î	254 þ
F	1111	*7*	15	31	47 /	63 ?	79 O	95 _	111 o	127 (DEL)	143	159	175 ¯	191 ¿	207 Ï	223 ß	239 ï	255 ÿ
		OCT	*1*	*3*	*5*	*7*	*11*	*13*	*15*	*17*	*21*	*23*	*25*	*27*	*31*	*33*	*35*	*37*

TABLE 7: Le code ISO 8859-1 ou Latin 1

La normalisation des ISO 8859 n’est pas ANSI puis ISO mais ISO puis organismes nationaux dont l’ANSI – c’est d’abord des normes internationales comme le sont le C et le C++ par exemple et elles ont été établies par des comités internationaux. En prime, cette normalisation s’est faite sous le double auspice de l’ISO et de l’ECMA et l’ECMA me semble avoir joué un rôle moteur (en passant, je donne souvent des références ECMA car ses normes sont publiques et leur site http ://www.ecma-international.org contient un certain nombre d’éditions qui ont été remplacées, par exemple ECMA 94, 1st edition).

J’ai du mal à penser que quelqu’un au courant des processus de normalisation aurait pu s’imaginer que l’ISO aurait pu accepter l’empiètement de la zone C1, cet

HEX			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	BIN		00				01				10				11			
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		OCT	*0*	*2*	*4*	*6*	*10*	*12*	*14*	*16*	*20*	*22*	*24*	*26*	*30*	*32*	*34*	*36*
0	0000	*0*	0	16	32 (SP)	48 0	64 @	80 P	96 `	112 p	128	144	160 (NBS)	176 °	192 À	208 Ð	224 à	240 ð
1	0001	*1*	1	17	33 !	49 1	65 A	81 Q	97 a	113 q	129	145	161 ¡	177 ±	193 Á	209 Ñ	225 á	241 ñ
2	0010	*2*	2	18	34 "	50 2	66 B	82 R	98 b	114 r	130	146	162 ¢	178 ²	194 Â	210 Ò	226 â	242 ò
3	0011	*3*	3	19	35 #	51 3	67 C	83 S	99 c	115 s	131	147	163 £	179 ³	195 Ã	211 Ó	227 ã	243 ó
4	0100	*4*	4	20	36 $	52 4	68 D	84 T	100 d	116 t	132	148	164 €	180 Ž	196 Ä	212 Ô	228 ä	244 ô
5	0101	*5*	5	21	37 %	53 5	69 E	85 U	101 e	117 u	133	149	165 ¥	181 µ	197 Å	213 Õ	229 å	245 õ
6	0110	*6*	6	22	38 &	54 6	70 F	86 V	102 f	118 v	134	150	166 Š	182 ¶	198 Æ	214 Ö	230 æ	246 ö
7	0111	*7*	7	23	39 ’	55 7	71 G	87 W	103 g	119 w	135	151	167 §	183 ·	199 Ç	215 ×	231 ç	247 ÷
8	1000	*0*	8	24	40 (	56 8	72 H	88 X	104 h	120 x	136	152	168 š	184 ž	200 È	216 Ø	232 è	248 ø
9	1001	*1*	9	25	41)	57 9	73 I	89 Y	105 i	121 y	137	153	169 ©	185 ¹	201 É	217 Ù	233 é	249 ù
A	1010	*2*	10	26	42 *	58 :	74 J	90 Z	106 j	122 z	138	154	170 ª	186 º	202 Ê	218 Ú	234 ê	250 ú
B	1011	*3*	11	27	43 +	59 ;	75 K	91 [	107 k	123 {	139	155	171 «	187 »	203 Ë	219 Û	235 ë	251 û
C	1100	*4*	12	28	44 ,	60 <	76 L	92 \	108 l	124 |	140	156	172 ¬	188 Œ	204 Ì	220 Ü	236 ì	252 ü
D	1101	*5*	13	29	45 -	61 =	77 M	93]	109 m	125 }	141	157	173 (SHY)	189 œ	205 Í	221 Ý	237 í	253 ý
E	1110	*6*	14	30	46 .	62 >	78 N	94 ^	110 n	126 ~	142	158	174 ®	190 Ÿ	206 Î	222 Þ	238 î	254 þ
F	1111	*7*	15	31	47 /	63 ?	79 O	95 _	111 o	127 (DEL)	143	159	175 ¯	191 ¿	207 Ï	223 ß	239 ï	255 ÿ
		OCT	*1*	*3*	*5*	*7*	*11*	*13*	*15*	*17*	*21*	*23*	*25*	*27*	*31*	*33*	*35*	*37*

TABLE 8: Le code ISO 8859-15 ou Latin 9

empiètement ne pouvant pas être perçu comme un complément (la charte était de développer des charsets respectant les structures d'ISO 2022 et ISO 4873 ; ISO 2022 en particulier permettant d'intégrer plusieurs charsets définis séparément, ne pas respecter la structure est fortement problématique).

Windows-1252 et ISO 8859-1 font plus que se ressembler. Windows-1252 c'est ISO 8859-1 plus des caractères dans la zone C1 (ou ISO 8859-1 n'a rien, cette norme définissant GL et GR et il faut convernir par ailleurs de la norme définissant C0 et C1) Et à propos de cette convergence parfaite dont je m'étonnais, j'ai été recherché un bouquin de 91 sur Windows, sa description du jeu ANSI diffère au moins (la fonte est mauvaise) en 2 caractères par rapport à la version actuelle. Mais ces deux caractères

HEX			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	BIN		00				01				10				11			
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
		OCT	*0*	*2*	*4*	*6*	*10*	*12*	*14*	*16*	*20*	*22*	*24*	*26*	*30*	*32*	*34*	*36*
0	0000	*0*	0	16	32 (SP)	48 0	64 @	80 P	96 `	112 p	128 €	144	160 (NBS)	176 °	192 À	208 Đ	224 à	240 ð
1	0001	*1*	1	17	33 !	49 1	65 A	81 Q	97 a	113 q	129	145 ‘	161 ¡	177 ±	193 Á	209 Ñ	225 á	241 ñ
2	0010	*2*	2	18	34 "	50 2	66 B	82 R	98 b	114 r	130 ‚	146 ’	162 ¢	178 ²	194 Â	210 Ò	226 â	242 ò
3	0011	*3*	3	19	35 #	51 3	67 C	83 S	99 c	115 s	131 *ƒ*	147 “	163 £	179 ³	195 Ã	211 Ó	227 ã	243 ó
4	0100	*4*	4	20	36 $	52 4	68 D	84 T	100 d	116 t	132 „	148 ”	164 ¤	180 ´	196 Ä	212 Ô	228 ä	244 ô
5	0101	*5*	5	21	37 %	53 5	69 E	85 U	101 e	117 u	133 …	149 •	165 ¥	181 µ	197 Å	213 Õ	229 å	245 õ
6	0110	*6*	6	22	38 &	54 6	70 F	86 V	102 f	118 v	134 †	150 –	166 ¦	182 ¶	198 Æ	214 Ö	230 æ	246 ö
7	0111	*7*	7	23	39 ’	55 7	71 G	87 W	103 g	119 w	135 ‡	151 —	167 §	183 ·	199 Ç	215 ×	231 ç	247 ÷
8	1000	*0*	8	24	40 (	56 8	72 H	88 X	104 h	120 x	136 ^	152 ~	168 ¨	184 ¸	200 È	216 Ø	232 è	248 ø
9	1001	*1*	9	25	41)	57 9	73 I	89 Y	105 i	121 y	137 ‰	153 ™	169 ©	185 ¹	201 É	217 Ù	233 é	249 ù
A	1010	*2*	10	26	42 *	58 :	74 J	90 Z	106 j	122 z	138 Š	154 š	170 ª	186 º	202 Ê	218 Ú	234 ê	250 ú
B	1011	*3*	11	27	43 +	59 ;	75 K	91 [	107 k	123 {	139 ‹	155 ›	171 «	187 »	203 Ë	219 Û	235 ë	251 û
C	1100	*4*	12	28	44 ,	60 <	76 L	92 \	108 l	124 |	140 Œ	156 œ	172 ¬	188 ¼	204 Ì	220 Ü	236 ì	252 ü
D	1101	*5*	13	29	45 -	61 =	77 M	93]	109 m	125 }	141	157	173 (SHY)	189 ½	205 Í	221 Ý	237 í	253 ý
E	1110	*6*	14	30	46 .	62 >	78 N	94 ^	110 n	126 ~	142 Ž	158	174 ®	190 ¾	206 Î	222 Þ	238 î	254 þ
F	1111	*7*	15	31	47 /	63 ?	79 O	95 _	111 o	127 (DEL)	143 ž	159	175 ¯	191 ¿	207 Ï	223 ß	239 ï	255 ÿ
		OCT	*1*	*3*	*5*	*7*	*11*	*13*	*15*	*17*	*21*	*23*	*25*	*27*	*31*	*33*	*35*	*37*

TABLE 9: Le code CP 1252 ou MS ANSI

ne sont pas définis dans la version de 85 d'ECMA 94 (version ECMA d'ISO 8859). Et il n'y a que 2 caractères dans la zone C1, contre a peu près 30 maintenant. Windows-1252 est un charset qui évolue toujours.

3.10 Unicode

4 Au sujet de quelques caractères

4.1 La barre oblique inversée

4.2 Les barres verticales

Toutes les autres références que j'ai sont postérieures et utilisent | plutôt que ¦ pour le caractère de code 124 ; je ne sais pas si c'est une différence de fontes – mais cet ouvrage donne des tables EBCDIC où les deux caractères sont présents – ou un changement dans la norme. [RFC20], vraisemblablement une copie de X3.4-1968, décrit le caractère comme *Vertical Line*.

4.3 Le signe monétaire universel

5 Manipulation en C et C++

En C et en C++, le type `char` est utilisé pour représenter les bytes d'un mécanisme d'encodage, qui. Ces langages imposent en plus comme contrainte qu'un byte fasse au moins 8 bits et qu'il soit la plus petite unité adressable. Si on a une machine adressable par mot (ça devient très rare), on peut soit prendre pour byte un mot complet, soit utiliser une sous-division du mot avec des pointeurs qui contiennent une information sur la sous-division à utiliser. Si on a une machine adressable par bit (je n'en connais qu'une qui ait eu une vocation d'usage générale), on ne pourra pas profiter de la chose.

Les chiffres de 0 à 9 doivent avoir des points de codage consécutif (donc '9'-'0' vaut 9). Les lettres peuvent ne pas avoir des points de codage consécutif (donc on peut utiliser de l'EBCDIC).

Tous les caractères du jeu de base doivent avoir des points de codage positifs (donc si on utilise de l'EBCDIC, char doit être non signé).

Le charset et l'encodage à utiliser pour interpréter la valeur d'un byte comme caractère fait partie de la « locale ». Je ne rentrerai pas pour le moment dans les détails sur les locales. Il y a quelques contraintes : il faut que tous les caractères du jeu de base (en gros, ceux qui servent dans la syntaxe) soit représentable par un seul byte et le même dans tous les encodages supportés et que dans tous les encodages ce byte représente toujours le caractère du jeu de base quel que soit le contexte.

Le type wchar_t est utilisé pour représenter des codes. Il faut qu'il soit assez grand pour contenir tous les codes du charset en ayant le plus. Il faut à nouveau que les caractères du jeu de base soient représentés par le même code dans tous les charsets.

- le jeu de caractère de base a pour codet des valeurs positives (donc une implémentation qui utilisant EBCDIC comme charset a un char non signé) ; ces valeurs sont

les mêmes dans tous les charsets (je ne suis pas sûr que ce soit les mêmes entre char et wchar_t ; donc char en EBCDIC + variantes et wchar_t en UCS32 est possible), et disponible dans le mode par défauts pour les charsets modaux ;

- le codet 0 désigne le caractère nul dans tous les modes ;
- les codets des chiffres sont consécutifs et dans le bon ordre ;
- (à vérifier) wchar_t est un codage ou 1 codet = 1 wchar_t (donc UTF-16 devrait être interdit pour wchar_t, mais on le rencontre).

6 Reste d'anciennes rédactions

Durant les années 60, les insuffisances d'un code sur 6 bits deviennent de moins en moins supportables. Pour les systèmes 360, IBM développe alors EBCDIC (*Extended BCDIC*), un code 8 bits ayant une structure proche du BCDIC et comme lui défini avec un encodage binaire et un encodage pour les cartes. L'encodage pour les cartes est en grosse partie compatible avec le BCDIC, l'encodage 8 bits n'est lui pas compatible avec le codage 6 bits de BCDIC, mais la transformation de l'un en autre est relativement simple à implémenter en hardware – du moins tant qu'on ne cherche pas à tenir compte de tous les caractères.

À la même époque, l'institut de normalisation américain (qui change de nom une série de fois sur la période) développe le code 7 bits ASCII (*American Standard Code for Information Interchange*). Ce code, et une série de variantes nationales remplaçant quelques caractères, sera normalisé par l'ISO sous le nom ISO 646 (généralement accompagné d'un code de deux lettres désignant la variante). Plus tard, une série de code 8 bits – dont les 128 premiers caractères reprennent ceux de l'ASCII – seront normalisés par l'ISO sous le nom ISO 8851 (généralement accompagnés d'un nombre indiquant précisément le code). Une autre norme, ISO 2022 [ECMA35], décrit une structure pour les codes – structure respectée par ISO 646 et ISO 8851 – permettant le changement d'un répertoire à un autre et l'utilisation de répertoires beaucoup plus grands, nécessaires pour des écritures idéographiques comme le Chinois, Japonais et le Coréen. Naturellement, ces écritures ont aussi des codages ne respectant pas la structure d'ISO 2022.

La multiplicité des codes existants pose naturellement des problèmes. Dans les années 1990, deux projets se sont donnés pour tâche d'établir un répertoire et un code universel – l'ISO et la norme ISO 10646, et le consortium Unicode. Après quelque temps d'une certaine rivalité, ces deux projets ont fini par collaborer et avoir le même répertoire et utiliser les mêmes codets.

Références : [ITU, Jen04].

Si le code CCITT#2 provient du monde des transmission en général et du télégraphe en particulier, le code BCDIC a pour origine le traitement des données. Les codes pour cartes perforées ne sont pas l'objet de ce document (voir par exemple), mais comprendre la structure du BCDIC et donc de l'EBCDIC est impossible sans rien en connaître.

Glossaire

byte (*byte*) ([ISO2382-4] utilise *multiplet* qui est rarement employé dans l'usage courant)

1/ chaîne composée d'un certain nombre de bits traitée comme un tout et représentant habituellement un caractère ou une partie de caractère [ISO2382-4].

2/ la plus petite unité adressable par un ordinateur. En particulier quand le mot-machine n'est pas la plus petite unité adressable.

3/ chaîne de bits formée de 8 bits (voir *octet*).

caractère (*character*) Élément d'un ensemble employé pour constituer, représenter ou gérer des données [ISO2382-4].

code (*code*, *coding scheme*) ensemble de règles établissant une correspondance entre les éléments d'un premier ensemble et ceux d'un second ensemble [ISO2382-4].

codage 1/ transformation d'une représentation en une autre.

2/ règles utilisée pour effectuer cette transformation (voir *code*).

codet (*code value*, *code element*) résultat de l'application d'un code à un élément d'un jeu codé [ISO2382-4].

décodage opération inverse de l'encodage.

encodage 1/ transformation d'une représentation, considérée comme principale, en une autre.

2/ règles utilisées pour effectuer cette transformation (voir *code*).

jeu de caractères (*character set*) ensemble fini de caractères considéré comme complet pour un usage particulier.

jeu de caractères codé (*coded character set*) ensemble de caractères mis en correspondance avec un autre ensemble d'éléments selon un code [ISO2382-4].

multiplet (*byte*) voir *byte*.

octet chaîne de bits formée de 8 bits.

seizet chaîne de bits formée de 16 bits.

Lexique français-anglais

boutisme endianness

byte byte

caractère character

code code, coding scheme

codet code value, code element, code point

fonte font

gros-boutien big-endian

gros-boutiste big-endian

jeu de caractères character set

jeu de caractères codé coded character set

multiplet byte

numéro de caractère code point

page de code code page

petit-boutien little-endian

petit-boutiste little-endian

point de code code point

police font

Lexique anglais-français

big-endian gros-boutiste, gros-boutien

byte multiplet, byte

character caractère

character set jeu de caractères

code code

coded character set jeu de caractères codé

code element codet

code page page de code

code point numéro de caractère, codet, point de code

code value codet

coding scheme code

endianness boutisme

font fonte, police

little-endian petit-boutiste, petit-boutien

Voir aussi [Anda], un lexique anglais-français du vocabulaire d'Unicode.

Bibliographie

[Anda] P. Andries. Lexique unicode. [Online]. Available : http://hapax.qc.ca/dunod/Unicode_Lexique.pdf

[Andb] ——. Unicode et iso 10646 en français. [Online]. Available : http://hapax.qc.ca

[And08] ——, *Unicode 5.0 en pratique*. Dunod, 2008.

[ECMA6] *7-Bit coded Character Set*, ECMA Std. 6, 1991, technically equivalent to ISO/IEC 646. [Online]. Available : http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-006.pdf

[ECMA7] *Representation of the Standard ECMA-6 (7 bit Code) on Punched Cards*, ECMA Std. 7, 1963. [Online]. Available : http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-7,1stEdition,April1965.pdf

[ECMA35] *Character Code Structure and Extension Techniques*, ECMA Std. 35, 1994, technically equivalent to ISO/IEC 2022. [Online]. Available : http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-035.pdf

[ECMA48] *Control Functions for Coded Character Sets*, ECMA Std. 48, 1991, technically equivalent to ISO/IEC 6429. [Online]. Available : http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-048.pdf

[Har04] Y. Haralambous, *Fontes & codages*. O'Reilly, 2004.

[ISO646] *Information technology – ISO 7 bit coded character set for information interchange*, ISO Std. ISO/IEC 646 :1991, Équivalent techniquement à la norme ECMA 6, plus facile à se procurer.

[ISO2022] *Information technology – Character code structure and extension techniques*, ISO Std. ISO/IEC 2022 :1994, Équivalent techniquement à la norme ECMA 35, plus facile à se procurer.

[ISO2382-4] *Technologies de l'information – Vocabulaire, Partie 4 : Organisation des données*, ISO/IEC Std. 2382-4 :1999(E/F). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO2382-5] *Technologies de l'information – Vocabulaire, Partie 5 : Représentation des données*, ISO/IEC Std. 2382-5 :1999(E/F). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO4873] *Information technology – ISO 8-bit code for information interchange – Structure and rules for implementation*, ISO Std. ISO/IEC 4873 :1991, Équivalent techniquement à la norme ECMA 43, plus facile à se procurer.

[ISO6429] *Information technology – Control functions for coded character sets*, ISO Std. ISO/IEC 6429 :1992, Équivalent techniquement à la norme ECMA 48, plus facile à se procurer.

[ISO8859-1] *Information technology – 8-bit single-byte coded graphic character sets – Part 1 : Latin alphabet No. 1*, ISO Std. ISO/IEC 8859-1 :1998, Équivalent techniquement à une partie de la norme ECMA 94, plus facile à se procurer.

[ISO8859-2] *Information technology – 8-bit single-byte coded graphic character sets – Part 2 : Latin alphabet No. 2*, ISO Std. ISO/IEC 8859-2 :1999, Équivalent techniquement à une partie de la norme ECMA 94, plus facile à se procurer.

[ISO8859-3] *Information technology – 8-bit single-byte coded graphic character sets – Part 3 : Latin alphabet No. 3*, ISO Std. ISO/IEC 8859-3 :1999, Équivalent techniquement à une partie de la norme ECMA 94, plus facile à se procurer.

[ISO8859-4] *Information technology – 8-bit single-byte coded graphic character sets – Part 4 : Latin alphabet No. 4*, ISO Std. ISO/IEC 8859-4 :1998, Équivalent techniquement à une partie de la norme ECMA 94, plus facile à se procurer.

[ISO8859-5] *Information technology – 8-bit single-byte coded graphic character sets – Part 5 : Latin/Cyrillic alphabet*, ISO Std. ISO/IEC 8859-3 :1999.

[ISO8859-6] *Information technology – 8-bit single-byte coded graphic character sets – Part 6 : Latin/Arabic alphabet*, ISO Std. ISO/IEC 8859-6 :1999.

[ISO8859-7] *Information technology – 8-bit single-byte coded graphic character sets – Part 7 : Latin/Greek alphabet*, ISO Std. ISO/IEC 8859-7 :2003.

[ISO8859-8] *Information technology – 8-bit single-byte coded graphic character sets – Part 8 : Latin/Hebrew alphabet*, ISO Std. ISO/IEC 8859-8 :1999.

[ISO8859-9] *Information technology – 8-bit single-byte coded graphic character sets – Part 9 : Latin alphabet No. 5*, ISO Std. ISO/IEC 8859-9 :1999.

[ISO8859-10] *Information technology – 8-bit single-byte coded graphic character sets – Part 10 : Latin alphabet No. 6*, ISO Std. ISO/IEC 8859-10 :1998.

[ISO8859-11] *Information technology – 8-bit single-byte coded graphic character sets – Part 11 : Latin/Thai alphabet*, ISO Std. ISO/IEC 8859-11 :2001.

[ISO8859-13] *Information technology – 8-bit single-byte coded graphic character sets – Part 13 : Latin alphabet No. 7*, ISO Std. ISO/IEC 8859-13 :1998.

[ISO8859-14] *Information technology – 8-bit single-byte coded graphic character sets – Part 14 : Latin alphabet No. 8 (Celtic)*, ISO Std. ISO/IEC 8859-14 :1998.

[ISO8859-15] *Information technology – 8-bit single-byte coded graphic character sets – Part 15 : Latin alphabet No. 9*, ISO Std. ISO/IEC 8859-15 :1999.

[ISO8859-16] *Information technology – 8-bit single-byte coded graphic character sets – Part 15 : Latin alphabet No. 10*, ISO Std. ISO/IEC 8859-16 :2001.

[ISO10646] *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC)*, ISO Std. ISO/CEI 10 646 :2003(F). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO10646a1] *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC) – Amendement 1 : Glagolitique, copte, géorgien et autres caractères*, ISO Std. ISO/CEI 10 646 :2003/Amd.1 :2005(F). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO10646a2] *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC) – Amendement 2 : N'Ko, phags-pa, phénicien et autres caractères*, ISO Std. ISOCEI 10 646 :2003/Amd.2 :2006(F). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO10646a3] *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Amendment 3 : Lepcha, Ol Chiki, Saurashtra, Vai and other characters*, ISO Std. ISO/IEC 10 646 :2003/Amd.3 :2008(E). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO10646a4] *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Amendment 4 : Cham, Game Tiles, and other characters*, ISO Std. ISO/IEC 10 646 :2003/Amd.4 :2008(E). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ISO10646a5] *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Amendment 5 : Tai Tham, Tai Viet, Avestan, Egyptian Hieroglyphs, CJK Unified Ideographs Extension C, and other characters*, ISO Std. ISO/IEC 10 646 :2003/Amd.5 :2008(E). [Online]. Available : http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ITU] International telegraph alphabet no. 2. ITU. [Online]. Available : http://www.itu.int/rec/T-REC-S.1-199303-I/en

[Jen04] T. Jennings. (2004) An annotated history of some character codes. [Online]. Available : http://wps.com/projects/codes/index.html

[Jona] D. W. Jones. Punched cards. [Online]. Available : http://www.cs.uiowa.edu/~jones/cards/

[Jonb] ——. Punched cards codes. [Online]. Available : http://www.cs.uiowa.edu/~jones/cards/codes.html

[Kat] R. H. Katz. Napoleon's secret weapon. [Online]. Available : http://bnrg.eecs.berkeley.edu/~randy/Courses/CS39C.S97/optical/optical.html

[Mac80] C. E. Mackenzie, *Coded Character Sets, History and Development*, ser. The systems programming series. Addison-Wesley, 1980.

[Mü] D. Müller. Code de Popham. [Online]. Available : http://www.apprendre-en-ligne.net/crypto/codes/popham.html

[rab] Technical information. [Online]. Available : http://rabbit.eng.miami.edu/info/index.html

[RFC20] V. Cerf. (1969) ASCII format for network interchange. RFC 20. IETF. Probably a copy of X3.4-1968. [Online]. Available : http://www.ietf.org/rfc/rfc20.txt

[RFC821] J. B. Postel. (1982) Simple mail transfer protocol. RFC 821. IETF. [Online]. Available : http://www.ietf.org/rfc/rfc821.txt

[RFC1521] N. Borenstein and N. Freed. (1993) Mime (multipurpose internet mail extensions) part one. RFC 1521. IETF. [Online]. Available : http://www.ietf.org/rfc/rfc1521.txt

[RFC2130] C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M. Crispin, and P. Svanberg. (1996) The report of the IAB character set workshop. RFC 2130. IETF. [Online]. Available : http://www.ietf.org/rfc/rfc2130.txt

[Sav08] J. Savard. (2008) Re : Vertical bar, broken bar and ASCII code 124. Usenet message on alt.folklore.computers. [Online]. Available : news:a6738911-857a-46a0-a874-f2a9229415f6@d36g2000prf.googlegroups.com

[Sea] S. J. Searle. A brief history of character codes. [Online]. Available : http://tronweb.super-nova.co.jp/characcodehist.html

[Uni] The unicode consortium home page. Unicode Consortium. [Online]. Available : http://www.unicode.org

[UTR17] K. Whistler, M. Davis, and A. Freytag, "Unicode character encoding model," Unicode Consortium, Tech. Rep. 17, 2008. [Online]. Available : http://www.unicode.org/reports/tr17

[Wika] Polybius square. Wikipedia, The Free Encyclopedia. [Online]. Available : http://en.wikipedia.org/w/index.php?title=Polybius_square&oldid=261174254

[Wikb] Sémaphore (communication). Wikipédia, l'encyclopédie libre. [Online]. Available : http://fr.wikipedia.org/w/index.php?title=S%C3%A9maphore_(communication)&oldid=35629796

[Win] D. T. Winter. Codes. [Online]. Available : http://homepages.cwi.nl/~dik/english/codes

[Wis02] C. Wissink, "Unicode and Windows XP," in *Proceedings of the Twenty-first International Unicode Conference*, 2002. [Online]. Available : http://download.microsoft.com/download/5/6/8/56803da0-e4a0-4796-a62c-ca920b73bb17/21-Unicode_WinXP.pdf