Le texte et le binaire...

Le codage des données

S'il est assez naturel de transformer un nombre "humain" (en base 10) dans n'importe quelle autre base de calcul, y compris la base 2 (et réciproquement), c'est un peu plus compliqué de coder en binaire les symboles d'écriture.

Pourquoi ?

Parce qu'il n'y a pas réellement d'algorithme mathématique pour le faire et qu'il faudra donc travailler sur des conventions. Vous savez ce que valent les conventions, elles sont adoptées jusqu'à ce qu'elles ne le soient plus. De plus, les limites d'une convention sont bien connues :

- Une convention est attachée à un contexte. Lorsque le contexte change, la convention doit être modifiée. Un exemple simple dans le domaine qui nous intéresse ici : l'adoption par la Communauté Européenne du symbole de sa monnaie unique, l'euro. Changement de contexte, ce symbole doit être ajouté à la liste des symboles d'écriture utilisée dans tous les pays de l'UE.
- Une convention doit satisfaire toutes les parties concernées. Les dites parties cherchant chacune à faire prévaloir leur point de vue, les conventions sont généralement adoptées trop tard.

Nous allons ici essayer de passer en revue les principales conventions adoptées pour le codage des symboles d'écriture, en ayant à l'esprit que nous sommes dans un contexte mondial, avec plusieurs langues, plusieurs alphabets et, pour compliquer encore le problème, plusieurs systèmes d'information.

Plan du chapitre

L'Écriture

Pourquoi écrire ?

La question peut paraître stupide, tant l'écrit demeure un moyen primordial dans nos civilisations pour la communication. Témoin ces quelques pages.

Le problème principal vient, nous le savons, de la multitude de langues utilisées de part le monde, multitude qui utilise elle même une multitude de symboles dans sa forme écrite.

Si l'alphabet latin reste probablement le plus utilisé, notons déjà la grande quantité de symboles altérés par des accentuations et autres contractions comme le fameux "e dans l'o". L 'alphabet latin reste, même avec toutes les altérations qu'on lui connaît, largement insuffisant pour permettre l'écriture de toutes les langues telles que le grec, l'hébreu, l'arabe, le russe et sans parler encore des langues asiatiques...

Comment écrire ?

Nous parlons d'informatique ; ici, pas de crayons. Les outils qui permettent d'afficher du texte sont principalement de deux sortes :

Les imprimantes

On peut les classer en deux grandes catégories :

- Les imprimantes dont le ou les jeux de caractères sont formés mécaniquement. Même si elles n'ont plus cours aujourd'hui, elles ont été parmi les premières. Depuis les ancêtres utilisant un jeu de marteaux comme les machines à écrire mécaniques, jusqu'aux "marguerites" (une galette en matériau souple, constituée de pétales, chacun portant un caractère) en passant par les imprimantes à boule dont IBM était le champion.
  Dans tous ces cas, les symboles sont gravés sur un support mécanique et l'impression se fait par impact sur un ruban encreur intercalé entre l'outil de frappe et le papier.
- Les imprimantes dont les jeux de caractères sont formés à partir d'une matrice de points. Depuis les antiques imprimantes à aiguilles jusqu'au laser en passant par le jet d'encre, le principe consiste à dessiner les caractères par impression de points. Dans tous ces cas, l'imprimante dispose de tables qui contiennent une représentation "bitmap" de l'ensemble des caractères.

Dans tous les cas, l'imprimante reçoit un code numérique écrit sur 8 bits et déduit de ce code le caractère qu'elle doit imprimer.

Les écrans

Qu'ils soient à tube cathodique où à cristaux liquides (ou même à plasma), le principe est similaire aux imprimantes à matrices de points.

La méthode globale d'impression

S'il s'agit d'un procédé d'impression mécanique type marguerite ou boule, un code va permettre de placer l'organe mécanique à la bonne place pour imprimer le caractère souhaité. Un changement de forme de caractères implique un changement de l'organe mécanique.

S'il s'agit d'un système à matrice de points, chaque caractère est dessiné dans une table et le système n'a qu'à aller chercher le bon dessin. Bien entendu, ce système est plus souple et propose généralement plusieurs typographies.

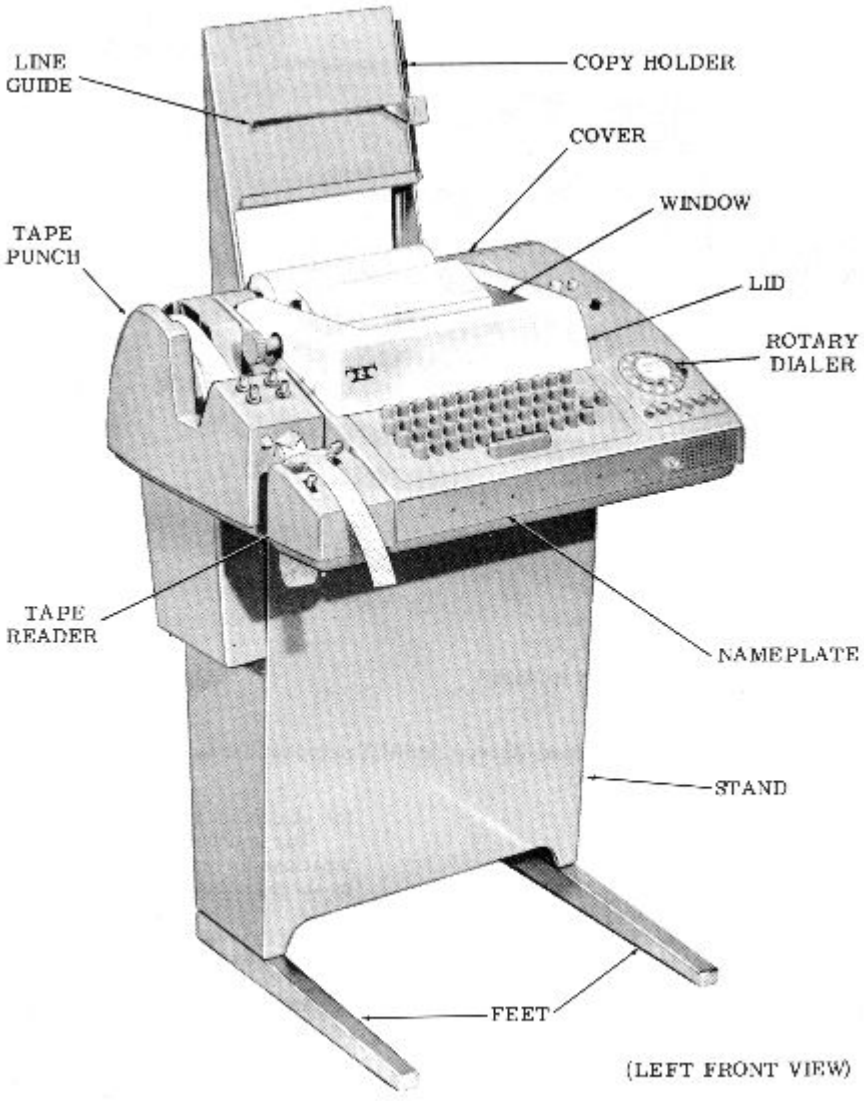
Nous n'entrerons pas trop dans les détails du pilotage d'une imprimante, mais en général, un langage particulier (PCL, PostScript) permet "d'expliquer" à l'imprimante ce qu'elle a à faire (police de caractères à utiliser, taille, format du papier à utiliser...), en plus de lui envoyer les données à imprimer.

Pour les écrans, c'est l'interface graphique avec son "driver", mais aussi le système d'exploitation lui-même qui se chargent de ce travail. Ce qu'il est important de comprendre, c'est qu'en ce qui concerne le contenu du message à imprimer, il doit exister un code qui définisse parfaitement l'ensemble des caractères de l'alphabet d'une (ou de plusieurs) langue(s) donnée(s). Ce code, dans le cas de systèmes communicants, comme c'est le cas sur l'Internet, doit être adopté par toutes les parties qui décident de communiquer entre elles ; faute de quoi, il apparaîtra des aberrations dans les textes imprimés.

L'objectif de ce chapitre est d'essayer de clarifier autant que possible l'ensemble des procédures mises en oeuvre pour parvenir à communiquer par l'écrit de façon satisfaisante.

Au début était le texte

Nous n'avons pas le choix, nous devons adopter une convention qui associera un nombre à un symbole d'écriture, puisque nous disposons de machines qui ne savent manipuler que des nombres. Nous créerons ainsi une table d'équivalence entre des valeurs numériques et des symboles d'écriture. Toutes les parties qui communiqueront entre elles en adoptant la même convention arriveront donc, en principe, à se comprendre.



Automatic Send-Receive (ASR) Teletypewriter Set

Figurez-vous que l'informatique n'a pas toujours été aussi compliquée. Voici un exemple de terminal informatique fort courant à une certaine époque.

Cette magnifique bestiole, appelée "télétype" du nom de l'entreprise qui la fabriquait (Les moins de 35 ans ne peuvent pas connaître, la machine date de 1967), servait à dialoguer avec un ordinateur, par le truchement d'une liaison série RS232, que nous connaissons toujours, même si ses jours sont désormais comptés.

En ces temps reculés de l'informatique, le tube cathodique n'était pas un périphérique courant. On utilisait volontiers à la place une imprimante, le plus souvent à boule ou à marguerite.

Cette machine disposait par ailleurs d'un lecteur/perforateur de ruban en papier (trou/pas trou -> 1/0). Mais pour intéressantes qu'elles soient, ces considérations archéologiques nous écartent de notre sujet initial...

La liaison RS232 prévoit de transmettre en série (bit par bit) un mot de 8 bits en utilisant le bit de poids le plus fort (bit 7) comme bit de parité, pour effectuer un contrôle de validité de la donnée. Le principe est simple : dans un octet, le bit de parité est ajusté de manière à ce que le nombre de 1 soit toujours pair (ou impair, ça dépend de la convention adoptée).

Dans ce cas de figure, il n'y a que 7 bits (b0 à b6) qui sont significatifs d'une donnée, le dernier bit servant juste à ajuster la parité.

7 bits pour un caractère

"L'American Standard Code for Information Interchange" (ASCII) s'est donc ingénié à coder chaque caractère d'une machine à écrire sous la forme d'une combinaison de 7 bits. En décimal, ça nous donne des valeurs comprises entre 0 et 127.

Comme la base binaire (0 ou 1), si elle est très commode pour un calculateur électronique, l'est beaucoup moins pour le cerveau humain, nous allons utiliser une autre base qui, si elle n'est guère plus "parlante", offre tout de même l'avantage d'aboutir à une écriture beaucoup plus compacte. Cette base devra être une puissance de 2, la plus courante étant la base hexadécimale, parce que chaque "digit" hexadécimal va représenter une combinaison de 4 bits :

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

Mais on peut aussi utiliser de l'octal, sur trois bits.

Pourquoi pas la base 10 à laquelle nous sommes habitués depuis notre plus tendre enfance ? Parce que, malheureusement, 10 n'est pas une puissance de 2 et qu'un "digit" décimal ne représente donc pas toutes les combinaisons que l'on peut faire avec un groupe de n bits. 4 c'est trop (hexadécimal) et 3 c'est pas assez (octal). Plus mathématiquement, on ne peut pas trouver de valeur entière de n telle que $10=2^n$. Essayez donc de résoudre n=Log(10)/Log(2).

Certains ont mis en oeuvre un codage appelé BCD (Binary Coded Decimal). Le principe est simple : chaque "digit" décimal (de 0 à 9) est codé sur un quartet. Certaines combinaisons de bits sont donc impossibles.

- 9 va donner 1001
- 10 donnera 0001 0000 et non pas 1010

Mais revenons à notre code ASCII ; 7 bits sont-ils suffisants ? Oui et non...

D'abord, dans une machine à écrire, il n'y a pas que des caractères imprimables. Il y a aussi des "caractères de contrôle", comme le saut de ligne, le retour chariot, le saut de page, la tabulation, le retour arrière... Tous ces caractères doivent aussi être codés pour que l'ordinateur puisse efficacement piloter une imprimante.

De plus, pour transmettre convenablement un texte, il faudra quelques sémaphores pour indiquer par exemple quand commence le texte, quand il finit...

Enfin, suivant les langues, même lorsqu'elles exploitent l'alphabet latin, certaines lettres sont altérées différemment. L'anglais n'utilise pas d'accents mais la plupart des autres langues les exploitent plus ou moins parcimonieusement.

Au final, si 7 bits suffisent généralement pour une langue donnée, éventuellement en faisant l'impasse sur certains symboles peu usités comme **[** ou **]**, nous ne pourrons pas coder l'ensemble des caractères nécessaires pour la totalité des langues utilisant l'alphabet latin.

La norme iso-646 définit un code ASCII sur 7 bits. Ce code, parfaitement adapté à l'anglais US, l'est moins pour les autres langues. Nous assistons donc à la création d'une multitude de "dialectes

ASCII", où certains caractères sont remplacés par d'autres suivant les besoins locaux. Les lecteurs les plus "anciens" se rappelleront peut-être des configurations hasardeuses de certaines imprimantes pour arriver à ce qu'elles impriment en français lisible...

Les "caractères" sur fond bleu sont les caractères non imprimables.

Pour bien lire le tableau, il faut construire le code hexadécimal en prenant d'abord le digit de la ligne, puis le digit de la colonne. Par exemple, la lettre "n" a pour code hexadécimal 6E

Comme vous le constatez, il n'y a aucune lettre accentuée dans ce codage. Ce dernier a donc été joyeusement "localisé" pour satisfaire aux exigences des divers pays utilisant l'alphabet latin. Cette situation aboutit rapidement à une impasse, les fichiers ainsi construits n'étant plus exportables dans d'autres pays. De plus, vous constaterez aisément que l'ajout de caractères supplémentaires (le "é", le "ç", le "à" etc.) implique obligatoirement la suppression d'autres caractères (le "[", le "]", le "#" etc.). Ceux qui ont quelques notions de programmation comprendront à quel point c'est facile d'écrire du code avec un jeu de caractères amputé de ces symboles. Dans la pratique, les programmeurs sont condamnés à utiliser un clavier US.

Pour un bit de plus

Avec les avancées de la technique, le huitième bit qui servait pour le contrôle de parité, contrôle rendu de plus en plus inutile, va être utilisé pour coder plus de caractères. Deux fois plus, finalement.

Ainsi, le codage "iso-latin-1", également connu sous le nom de "iso-8859-1" propose à peu près le codage suivant :

Comme vous pouvez le constater ici :

Les codes ASCII de 0 à 7F (127 en décimal) demeurent inchangés,

les codes supérieurs (ceux qui ont le bit 7 à 1) représentent quelques symboles supplémentaires, ainsi qu'une panoplie de lettres accentuées qui satisfont aux exigences des langues de l'Europe de l'Ouest.

Pourquoi "à peu près" ? Le codage ci dessus est une interprétation de la norme iso-8859-1 par notre cher Microsoft qui a un peu bricolé pour ajouter quelques symboles de plus, dont celui de l'euro... La conséquence en est qu'une fois de plus, Windows n'est compatible qu'avec lui-même. Fort heureusement, nous verrons qu'il demeure possible d'adopter un codage plus officiel avec les applications communicantes, mais avec des limites. Notez que si l'on peut reprocher à Microsoft de ne pas suivre les normes, il faut aussi reprocher aux normes d'êtres imparfaites et assez peu réactives.

Pour ajouter à la complexité, la norme iso-8859 définit pas moins de 15 versions différentes, pour satisfaire à tous les besoins mondiaux. A titre d'information, la norme iso-8859-15 devrait pouvoir être utilisée pour l'Europe de l'Ouest avec plus de "bonheur" que l'iso-8859-1.

Finalement, ce bit de plus ne fait que déplacer le problème sans toutefois l'éliminer, nous ne disposons toujours pas d'un système normalisé universel.

Les as de la confusion

Croyez-vous que la situation est suffisamment confuse comme ça ? Vous vous trompez ! D'autres choses existent, souvent venant de chez IBM.

EBCDIC

Je me contenterai de vous citer la définition issue du "jargon français[1]" :

Extended Binary Coded Decimal Interchange Code.
Jeu de caractères utilisé sur des dinosaures[2] d'IBM[3]. Il existe en 6 versions parfaitement incompatibles entre elles, et il y manque pas mal de points de ponctuation absolument nécessaires dans beaucoup de langages modernes (les caractères manquants varient de plus d'une version à l'autre...) IBM est accusé d'en avoir fait une tactique de contrôle des utilisateurs. (© Jargon File 3.0.0).

Il existe quelques "moulinettes" capables de convertir tant bien que mal des fichiers codés sous cette forme en fichiers ASCII.

Bien que l'EBCDIC soit aujourd'hui tout à fait confidentiel, puisqu'il ne concerne que les vieilles machines IBM, il faut en tenir compte pour les échanges de données inter plateformes, jusqu'à extinction totale de la race (nous ne devons plus en être très loin).

Pages de codes 437 et 850

Lorsque IBM a créé le PC (Personal Computer, faut-il le rappeler ?), des jeux de caractères ont été créés sur 8 bits, spécifiquement pour ces machines. Ci-dessous la page de code 437 (CP437). Attention, ce tableau se lit dans l'autre sens, le quartet de poids faible est celui de la ligne et le quartet de poids fort est celui de la colonne.

1 http://www.linux-france.org/prj/jargonf/
2 http://www.linux-france.org/prj/jargonf/D/dinosaure.html
3 http://www.linux-france.org/prj/jargonf/I/IBM.html

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul			0	@	P	`	p	Ç	É	á	▒	└	╨	α	≡
1			!	1	A	Q	a	q	ü	æ	í	▓	┴	╤	ß	±
2			"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3			#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4			$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5			%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6			&	6	F	V	f	v	â	û	ª	╢	╞	╓	µ	÷
7	bel		'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	bs		(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	tab		)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	∘
A	lf		*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
B	vt	esc	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	ff		.	<	L	\	l	|	î	£	¼	╝	╠	▄	∞	ⁿ
D	cr		-	=	M	]	m	}	ì	¥	¡	╜	═	▌	∅	²
E			.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	ε	■
F			/	?	O	_	o		Å	ƒ	»	┐	╧	▀	∩	

Tous les petits "grigris" à partir du code B0 étaient destinés à faire de l'art ASCII étendu. De jolies interfaces pseudo graphiques sur des terminaux en mode texte.

Si cette page de code est compatible avec l'ASCII US 7 bits (iso-646) il n'en est rien pour le reste, avec aucune iso-8859. Cette situation a été assez pénalisante, aux débuts de Windows, où l'on devait souvent jongler avec les fichiers issus d'applications DOS et Windows.

Reconstruire la tour de Babel

Et si l'on construisait une table de codage sur 16 bits ? Là, on aurait de la place pour entrer dans une seule et unique table tous les symboles que l'espèce humaine a pu inventer...

Rassurez-vous, on y a déjà pensé et le projet fait même l'objet d'une normalisation, iso-10646-1[4].

Compte tenu des difficultés rencontrées pour normaliser des codes sur 8 bits, je vous laisse imaginer ce que ça risque de donner avec 16... De plus, les fichiers de texte verront subitement leur taille doubler pour dire la même chose...

La solution n'est peut-être pas là non plus. Bien que cette norme (plus connue sous le nom d'unicode ou utf-8) existe, elle n'est que peu utilisée.

Conclusion provisoire

Comme vous le voyez, nous sommes encore loin de disposer d'un système de codage efficace des

4 http://alis.isoc.org/codage/iso10646/

divers symboles utilisés dans le monde pour communiquer. La situation parait déjà assez désespérée, mais rassurez-vous, nous n'avons pas encore tout vu...

Note :

Les passionnés de la chose trouveront ici[5] beaucoup de détails sur les divers codages existants.

5 http://czyborra.com/charsets/codepages.html

Autres astuces

Code toujours, tu m'intéresses

Par une remarquable tendance à la perversité de l'esprit humain, certains protocoles "applicatifs" (SMTP par exemple) ont été conçu avec comme axiome de départ qu'il devraient transporter du texte, alors qu'il est clair qu'ils n'auraient à transporter que des valeurs numériques binaires, puisqu'un système numérique ne sait finalement faire que cela.

D'ailleurs, le premier besoin qui s'est fait sentir, c'est de pouvoir attacher aux e-mails des fichiers qui sont tout, sauf du texte pur.

Pourquoi le parti pris du texte ?

A cause des caractères de contrôle ! C'est très pratique de disposer de caractères spéciaux qui permettent, comme leur nom l'indique, de contrôler le flux de données. Avec le codage ASCII, nous avons vu qu'il en existait pas mal, même si nous ne sommes pas entrés dans le détail de leur signification.

De plus, nous n'avons pas parlé du codage des valeurs numériques. Si un nombre entier ne pose pas trop de problèmes (1, 2 octets ou plus, éventuellement, encore que reste à savoir dans quel ordre on va les passer), les nombres réels, codés sous la forme mantisse / exposant sont un réel casse-tête. Un exemple faux mais qui fait comprendre : le nombre 1 245 389 789 726 986 425 ne va pas s'utiliser ainsi, on va d'abord l'écrire, par exemple sous la forme $1245{,}389789726986425\ 10^{15}$. On s'attachera alors à stocker en mémoire ce réel sous une forme approchée en utilisant systématiquement, disons trois octets. Sa partie entière, 1245 dans l'exemple, sera codée sur deux octets et la puissance de 10, 15 dans l'exemple, sur un octet.

Cet exemple n'a pas de réalité, mais le principe est à peu près juste. Suivant les plateformes et les langages de programmation, nous aurons nos réels stockés sur 4, 6 ou 8 octets, avec plus ou moins de précision sur la mantisse, ou plus ou moins d'espace sur la puissance de 10, suivant la nature des calculs à réaliser (aviez-vous pensé que la notion d'infini ne peut être gérée par un calculateur ?). Au final, il est souvent plus simple de communiquer des valeurs numériques à un tiers sous leur forme ASCII (telles qu'on peut les afficher ou les imprimer, avec des symboles d'écriture, donc), plutôt que telles qu'elles sont stockées en mémoire.

La conséquence ?

Ces protocoles ne peuvent pas simplement transférer des données numériques, puisqu'un octet est à priori considéré comme l'image d'un caractère et non comme une donnée numérique en elle même. Ainsi, si vous voulez transférer un fichier qui contient une représentation "bitmap" d'une image, comme un fichier jpeg, png ou gif, par exemple, vous ne pouvez pas considérer que c'est du texte, puisque à priori, chaque octet peut prendre n'importe quelle valeur, y compris celle d'un caractère de contrôle. Vous connaissez beaucoup de pages Web sans aucune image dedans ? Vous n'avez jamais envoyé un e-mail avec une image en pièce jointe ?

La conclusion est qu'il a fallu trouver une astuce pour transporter des données purement numériques

sur un protocole qui n'est pas prévu pour ça.

Codage à tous les étages

Le jeu va consister maintenant à coder une donnée purement numérique sous une forme alphabétique, elle-même codée sur des valeurs numériques, pour qu'elle puisse être transportée sur un système qui ne connaît que des 0 et des 1.

Tordu, n'est-ce pas ?

Oui, mais comment faire autrement ? Les révolutions, c'est bien, mais on ne peut pas en faire tous les jours, sinon, c'est le chaos permanent. Vous allez voir que les solutions apportées sont certes parfois tordues, mais astucieuses et surtout efficaces.

Comme il est clair, à la lueur de ce que nous avons vu jusqu'ici, que la seule convention qui soit à peu près universellement acceptée et correctement transportée par les protocoles applicatifs est la norme iso-646 (US-ASCII), il faudra trouver des conventions de codage pour convertir un octet en un ensemble de caractères sur 7 bits.

C'est parti pour la grande cuisine.

Le codage "quoted printable"

Ce codage est principalement employé pour transformer un texte écrit avec un codage sur 8 bits en un texte qui ne contiendra que des caractères codables sur 7 bits. Vous allez voir comme c'est simple :

D'abord, il faut savoir sur quel codage 8 bits on va s'appuyer, en général, pour nous, iso-8859-1.

Ensuite, nous allons utiliser un "code d'échappement" (c'est une technique assez courante, nous la rencontrons souvent en informatique). Ici, le caractère d'échappement est le signe =. Ce signe signifie que les deux caractères qui vont le suivre représenteront le code hexadécimal d'un caractère et non le caractère lui-même. Bien entendu, il faudra aussi coder le caractère d'échappement.

Un petit exemple vaudra bien mieux qu'un long discours...

- le **é** dont le code 8 bits est **E9**, sera codé sur trois caractères de 7 bits de la façon suivante : **=E9**
- De la même façon, le **è** sera codé **=E8**.
- le **ç** donnera **=E7**
- le **à** donnera **=E0**
- Comme le = revêt une signification particulière : C'est le code d'échappement, il sera lui-même codé en **=3D**.

L'expression **çà et là** sera donc transmise sous la forme **=E7=E0 et l=E0**

Ainsi, nous transporterons nos données uniquement sous la forme de caractères US-ASCII (7 bits), même s'ils nécessitent 8 bits pour être définis. En effet, les caractères =, **E**, et **0** ont tous des codes ASCII sur 7 bits.

Astucieux non ?

Bien entendu, il vaut mieux le savoir pour décoder correctement le message. Cette méthode est utilisée principalement pour les e-mails. En voici un exemple :

```
Return-Path: <christian.caleca@free.fr>
...
From: "Christian Caleca" <christian.caleca@free.fr>
To: <christian.caleca@free.fr>
Subject: quoted
Date: Tue, 5 Nov 2002 10:51:34 +0100
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
...
X-Mailer: Microsoft Outlook Express 6.00.2800.1106
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1106

=E7=E0 et l=E0
```

Notez que l'on parle ici de MIME, en indiquant, la nature du contenu (text/plain), le jeu de caractères utilisé (iso-8859-1) et le mode d'encodage : quoted-printable. Nous y reviendrons plus tard.

Le codage Base64

Plus généralement, ce codage permettra de passer non seulement du texte codé sur 8 bits, mais aussi tout type de données constituées d'octets. Voyons d'abord avec du texte.

Là aussi, il faudra commencer par indiquer en quel code est écrit le texte initial. Pour nous, toujours iso-8859-1.

Trois caractères de 8 bits (24 bits au total) sont découpés sous la forme de 4 paquets de 6 bits (toujours 24 bits au total). Chaque valeur sur 6 bits, comprise donc entre 0 et 3F en hexadécimal, sera symbolisée par un caractère présent, et avec le même code, dans toutes les versions de code ASCII et EBCDIC. La table d'équivalence est celle qui suit. Remarquez que les caractères choisis sont tous codés sur 7 bits en US-ASCII, mais que la valeur qu'ils représentent **n'est pas** leur code ASCII

dec	hex	car.		dec	hex	car.		dec	hex	car.		dec	hex	car.
0	0	A		16	10	Q		32	20	g		48	30	w
1	1	B		17	11	R		33	21	h		49	31	x
2	2	C		18	12	S		34	22	i		50	32	y
3	3	D		19	13	T		35	23	j		51	33	z
4	4	E		20	14	U		36	24	k		52	34	0
5	5	F		21	15	V		37	25	l		53	35	1
6	6	G		22	16	W		38	26	m		54	36	2
7	7	H		23	17	X		39	27	n		55	37	3
8	8	I		24	18	Y		40	28	o		56	38	4
9	9	J		25	19	Z		41	29	p		57	39	5
10	A	K		26	1A	a		42	2A	q		58	3A	6
11	B	L		27	1B	b		43	2B	r		59	3B	7
12	C	M		28	1C	c		44	2C	s		60	3C	8
13	D	N		29	1D	d		45	2D	t		61	3D	9
14	E	O		30	1E	e		46	2E	u		62	3E	+
15	F	P		31	1F	f		47	2F	v		63	3F	/

Comme cette explication doit paraître fumeuse à plus d'un (moi-même, plus je la relis, plus je la trouve fumeuse), là encore, prenons un exemple. Soit à coder le texte extrêmement simple : **012**

Ce texte est destiné à être écrit avec un codage iso-8859-1.

caractère initial	0	1	2
Code ASCII hexa	30	31	32
Code ASCII binaire	00110000	00110001	00110010

Bien. nous avons donc la suite de 24 bits suivante : 001100000011000100110010. Nous allons maintenant la couper en quatre morceaux de 6 bits :

les valeurs sur 6 bits	001100	000011	000100	110010
Equivalent hexadécimal	0C	03	04	32
Caractère équivalent en Base64	M	D	E	y

Et voilà. **012** donne, une fois codé en Base 64 **MDEy**. Constatez comme c'est simple. Constatez surtout que ces caractères seront transcrits en US-ASCII, donc sur 7 bits.

Pour décoder, il suffit de le faire dans l'autre sens.

Refaisons la manipulation avec un e-mail codé en Base64 :

```
Return-Path: <christian.caleca@free.fr>
...
From: "Christian Caleca" <christian.caleca@free.fr>
To: <christian.caleca@free.fr>
Subject: Base 64 (1)
Date: Tue, 5 Nov 2002 11:07:11 +0100
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: base64
...
X-Mailer: Microsoft Outlook Express 6.00.2800.1106
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1106

MDEy
```

Et voilà le travail.

Ça, c'est une démo "commerciale", c'est à dire "qui ne montre que ce qui est facile et qui marche bien". Vos messages contiennent tous un nombre de caractères qui est un exact multiple de 3 ?

Dans ce cas (nombre de caractères qui n'est pas un multiple de 3), le système de codage va "remplir le trou" avec un caractère spécial, qui ne sera pas interprété à l'arrivée. Ce caractère est le signe =

Voyons ce que ça donne si le texte initial ne contient plus que le seul caractère **0**.

- Le premier groupe de 6 octets sera toujours le même : 001100 qui donne **M**
- Le second sera : 00 (complété avec des 0, donc : 000000) qui donne **A**
- Comme il faut 24 bits quand même, on ajoutera deux fois le caractère =

Au total, on aura **MA==**

Vérification par l'e-mail :

```
Return-Path: <christian.caleca@free.fr>
...
From: "Christian Caleca" <christian.caleca@free.fr>
To: <christian.caleca@free.fr>
Subject: base 64 (3)
Date: Tue, 5 Nov 2002 11:18:06 +0100
MIME-Version: 1.0
Content-Type: text/plain;
charset="iso-8859-1"
Content-Transfer-Encoding: base64
...
X-Mailer: Microsoft Outlook Express 6.00.2800.1106
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1106

MA==
```

CQFD.

Et les autres...

uuencode

Bien entendu, d'autres conventions existent, mais n'appartiennent pas aux système MIME. Par exemple UUENCODE, assez proche de Base64, mais antérieur, utilisé sur plateformes Unix.

BinHex

Un codage propriétaire, créé dans le monde Macintosh pour les mêmes raisons...

Vous le voyez, les astuces ne manquent pas pour utiliser exclusivement de l'ASCII 7 bits dans le transport de n'importe quelle donnée.

Conclusions

En plus du codage des caractères dans des tables de 7, 8 ou même 16 bits, il faut donc ajouter des systèmes qui vont s'efforcer de représenter tout type de donnée sous forme de texte 7 bits.

Ceci nous amène naturellement à parler de MIME...

Dans le HTML

Les pieds dans la toile

Bien que pour HTTP, protocole apte à transmettre des flux d'octets sans considérer que ce sont forcément des caractères, bon nombre de problèmes sont à résoudre.

Le langage HTML (Hyper Text Markup Language), lui aussi, propose des méthodes particulières pour traiter les caractères non US-ASCII. Il règne d'ailleurs dans ce domaine la plus grande confusion.

Avec la version 3.2 du HTML, il n'y avait normalement pas d'autre possibilité que de passer par un transcodage du type "signes nommés". Depuis le version 4.0 de HTML, il est théoriquement possible de définir dans l'en-tête du document quel jeu de caractères est utilisé.

Comme HTML est probablement le lieu où les normes sont le moins respectées, il convient tout de même de rester prudent...

Les signes nommés

Le principe est simple :

- Un caractère d'échappement : Le **&**,
- un nom pour le caractère non US-ASCII
- un délimiteur de fin de codage : le **;**

Un simple exemple, juste pour illustrer. Le **é** devrait se coder dans le source HTML : **é**

Vous trouverez beaucoup plus de détails sur les signes nommés ainsi que sur beaucoup d'autres points du HTML sur le très instructif site SELFHTML[6].

Il existe une table de signes nommés définie dans HTML 3.2 . HTML 4.0 définit des ajouts à cette table, bien que, théoriquement, une balise d'en-tête du type :

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

devrait à elle seule permettre l'emploi de tous les symboles définis par iso-8859-1 (version 4.0 uniquement).

Une manipulation amusante

FrontPage 2000, l'éditeur HTML de Microsoft (celui avec lequel ces pages sont faites), ne s'embarrasse pas de considérations complexes, annonce un charset=windows-1252 (volontairement modifié manuellement dans cette page en iso-8859-1) dans ses en-têtes et ne code aucun de ces symboles de façon particulière, même pas l'euro.
Il serait pertinent de penser que cette façon cavalière de procéder va créer des problèmes sur toute plateforme autre que Windows/Internet Explorer...
Dans la pratique, ça fonctionne sans le moindre problème avec Mozilla 1.1, aussi bien sous Windows que sous Linux, ça fonctionne aussi avec Konqueror 3.0.3 sous Linux. Il ne faut tout de

6 http://fr.selfhtml.org/

même pas trop en demander, ça ne fonctionne pas pour l' euro avec Netscape 4.78 sous Linux, qui affiche à la place un **?** .

D'autres éditeurs, comme DreamWeaver (de Macromedia) ou Golive (d'Adobe) sont plus orthodoxes et, non seulement annonceront un "charset=iso-8869-1", mais encore utiliseront les signes nommés pour les caractères non US.

L'exemple qui suit est "bricolé" directement dans le source HTML. La même ligne sera codée selon diverses façons :

Caractères	codage	
é è ç ù à ê €	`é è ç ù à ê € (sans codage)`	FrontPage 2000
é è ç ù à ê €	`&eacute; &egrave; &ccedil; &ugrave; &agrave; &ecirc; &euro;`	DreamWeaver 4
é è ç ù à ê €	`&eacute; &egrave; &ccedil; &ugrave; &agrave; &ecirc; &#x20AC;`	Golive 5

Normalement, il y a de grandes chances que vous voyez tout ça correctement. Maintenant, si vous utilisez Internet Explorer, allez dans "Affichage", puis "Codage" et changez le codage par défaut. Là, vous risquez de voir les limites de FrontPage qui n'utilise pas systématiquement les signes nommés, même si en théorie, HTML 4.0 devrait le permettre. Ceux qui n'utilisent pas IE doivent avoir quelque part une fonction équivalente, pour changer l'affichage par défaut.

Notez la curieuse façon de coder le symbole de l'euro par Golive 5 : **€** . C'est tout simplement sa valeur numérique, en hexadécimal, dans la normalisation unicode (16 bits)...

Que penser de tout ça ?

Il serait possible de pousser encore plus loin les investigations, et de supprimer dans l'en-tête de chaque page la définition du "charset" , ça ne changerait très probablement rien au résultat final.

Vous le voyez, nous sommes ici dans le flou "artistique". Au bout du compte, même en HTML 4.0, il semble de bon ton d'utiliser tout de même systématiquement les signes nommés, même si l'on peut s'en passer le plus souvent

Conclusions

Si vous n'avez pas encore attrapé le vertige, vous ne l'attraperez plus. Sinon, essayons de consolider un peu nos positions.

Les faits

- Les systèmes numériques ne savent traiter que des informations binaires, donc numériques.
- Les protocoles applicatifs, le plus souvent, ont besoin de transporter du texte.
- Le texte est destiné à être lu, donc écrit, et n'est pas constitué d'une collection de valeurs numériques, mais d'une collection de symboles graphiques : Un alphabet.
- Il n'y a pas qu'un seul alphabet au monde.
- Les protocoles applicatifs peuvent avoir aussi à échanger des données qui ne sont pas du

texte (une image, du son, une vidéo...).

Les solutions

- Coder sur 7 bits un jeu de caractères minimal (US-ASCII), mais il n'y a pas que l'américain dans le monde, et 128 valeurs ne suffisent pas pour des langues riches en lettres accentuées.
- Coder sur 8 bits, mais ça ne suffit pas non plus pour toutes les langues possibles.
- Coder sur 16 bits, mais ça alourdit considérablement la taille des données. De plus, l'unité courante de transport est l'octet (8 bits), ça ne simplifie pas.

Le bricolage

Le "bricolage" le plus propre consiste à utiliser un jeu de caractères minimal et de coder les caractères supplémentaires par une combinaison identifiable des caractères de base. C'est cette solution qui est le plus souvent mise en oeuvre, et elle donne finalement les meilleurs résultats.

- Codage "quoted-printable" (e-mails, imprimantes...)
- Codage "Base64" (e-mails, fichiers pouvant contenir autre chose que du texte pur...)
- Signes nommés (HTML)

MIME

MIME. C'est quoi ?

Multipurpose Internet Mail Extension. Comme son nom l'indique, c'est une suite d'extensions pour permettre, principalement aux e-mails, de transporter autre chose que du texte, à savoir, du son, des images, de la vidéo... Autant de choses pour lesquelles la messagerie n'est à priori pas faite.

Ces extensions servent également sur le web, lorsque l'on utilise HTTP pour transporter autre chose que du texte (ce qui est souvent le cas). Voyez le chapitre sur HTTP[7] à ce propos.

MIME rassemble deux choses distinctes :

- Une description normalisée d'un type de document (non texte pur).
- Le mode de codage employé pour le transporter.

MIME et SMTP

C'est ici que MIME prend toute son importance. En effet, en plus de pouvoir définir des types de documents, il peut aussi définir des types d'encodages, comme Base64 ou Quoted-Printable.

Un seul exemple significatif. Il reprendra ce que nous avons eu l'occasion de voir par ailleurs.

Le message contient le texte ::

```
juste un texte légèrement accentué...
Suivi d'une image gif.
```

codé Quoted-Printable , suivi d'une image gif en pièce jointe. Voici le message tel qu'il est reçu :

```
Return-Path: <christian.caleca@free.fr>
...
From: "Christian Caleca" <christian.caleca@free.fr>
To: <christian.caleca@free.fr>
Subject: demo MIME
Date: Sat, 9 Nov 2002 11:29:09 +0100
MIME-Version: 1.0
Content-Type: multipart/mixed;
On est averti qu'il y aura plusieurs morceaux de type différents...
boundary="----=_NextPart_000_0044_01C287E3.38B13A20"
Avec un séparateur bien défini.
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1106
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1106

This is a multi-part message in MIME format.

------=_NextPart_000_0044_01C287E3.38B13A20
Content-Type: text/plain;
charset="iso-8859-1"
```

7 http://christian.caleca.free.fr/http/le_protocole.html
ou le fichier « http.pdf »

```
Content-Transfer-Encoding: quoted-printable
La partie texte, codée "quoted-printable"...

juste un texte l=E9g=E8rement accentu=E9...
Suivi d'une image gif.
Voilà qui est fait.
------=_NextPart_000_0044_01C287E3.38B13A20
Content-Type: application/octet-stream;
name="moineau1.gif"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="moineau1.gif"
On ne peut pas être plus précis :
        type :flux d'octets
        nom : moineau1.gif
        codage : Base64...
R0lGODlhcgH8APf/AP//////zP//mf//Zv//M///AP/M///MzP/Mmf/MZv/MM//MAP+Z//+ZzP+Z
...
GZACDvqwAvWAOgEBADs=

------=_NextPart_000_0044_01C287E3.38B13A20--
```

Comme prévu, ce message contient bien deux parties :

- Du texte pur, codé en Quoted-Printable,
- une image gif, codée en Base64

L'image, dans Outlook Express, va apparaître sous le texte, séparée par un filet horizontal.

Note pour les e-mails

Selon toute logique, le codage Base64 devrait pouvoir être universellement exploité dans la messagerie, puis qu'il permet à coup sûr de transporter correctement le message codé (sur 7 bits) et définit complètement la table de codage ASCII quelque soit l'alphabet utilisé par l'auteur.

Bien entendu, la situation n'est pas aussi simple. D'abord, les dernières versions de SMTP savent transporter du texte sur 8 bits, ce qui rend le codage inutile, ensuite, nombre de clients de messagerie ne savent pas encore décoder le Base64, voire le quoted-printable. Tous les clients qui ne sont pas compatibles MIME, et il y en a encore pas mal en service. Si bien que la méthode la plus efficace (la moins hasardeuse) reste d'utiliser la police iso-8859-1 sans aucun codage et de rédiger l'objet du message sans accents ni symboles particuliers, ça fera au moins ça de lisible à coup sûr.

MIME et HTTP

Nous en avons déjà un exemple dans le chapitre HTTP, pour transporter une image gif dans une page html. Mais dans ce cas, il n'y a pas de codage (type Base64 ou quoted-printable), les octets sont brutalement transportés par le protocole. MIME sert juste à définir le type de document.

Voici juste un exemple, où HTTP va transporter un document MS Word. La manipulation est faite avec Internet Explorer 6 et Mozilla 1.1 sur une plateforme Windows disposant de MS Word. Un sniffeur regarde ce qu'il se passe au niveau HTTP.

Avec Internet Explorer 6

```
Frame 4 (387 on wire, 387 captured)
    ...
Internet Protocol, Src Addr: 192.168.0.10, Dst Addr: 192.168.0.253
    ...
Hypertext Transfer Protocol
    GET /odj.doc HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
            application/vnd.ms-powerpoint, application/vnd.ms-excel,
            application/msword, */*\r\n
    Nous le savons, IE6 accepte explicitement les fichiers au format MS Office
    si ce dernier est installé.
    Accept-Language: fr\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)\r\n
    Host: linux.maison.mrs\r\n
    Connection: Keep-Alive\r\n
    \r\n

Frame 6 (1514 on wire, 1514 captured)
    ...
Internet Protocol, Src Addr: 192.168.0.253, Dst Addr: 192.168.0.10
   ...
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Sat, 09 Nov 2002 09:32:41 GMT\r\n
    Server: Apache-AdvancedExtranetServer/1.3.26 (Mandrake Linux/6.1mdk)
            auth_ldap/1.6.0 mod_ssl/2.8.10 OpenSSL/0.9.6g PHP/4.2.3\r\n
    Last-Modified: Thu, 06 Jul 2000 15:07:29 GMT\r\n
    ETag: "57d5a-7800-3964a0b1"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 30720\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: application/msword\r\n
    \r\n
    Apache connait le type MIME msword et signale le type de contenu,
    puis, commence à envoyer les données.
    Data (1067 bytes)

0000  d0 cf 11 e0 a1 b1 1a e1 00 00 00 00 00 00 00 00   ................
0010  00 00 00 00 00 00 00 00 3e 00 03 00 fe ff 09 00   ........>.......
...
    Les octets surlignés montrent à l'évidence que HTTP transporte sur 8 bits
```

Une fois la réception terminée, Internet Explorer va afficher directement le document, en utilisant MS Word comme "plug-in".

Avec Mozilla 1.1

```
Frame 6 (534 on wire, 534 captured)
    ...
Internet Protocol, Src Addr: 192.168.0.10, Dst Addr: 192.168.0.253
    ...
Hypertext Transfer Protocol
```

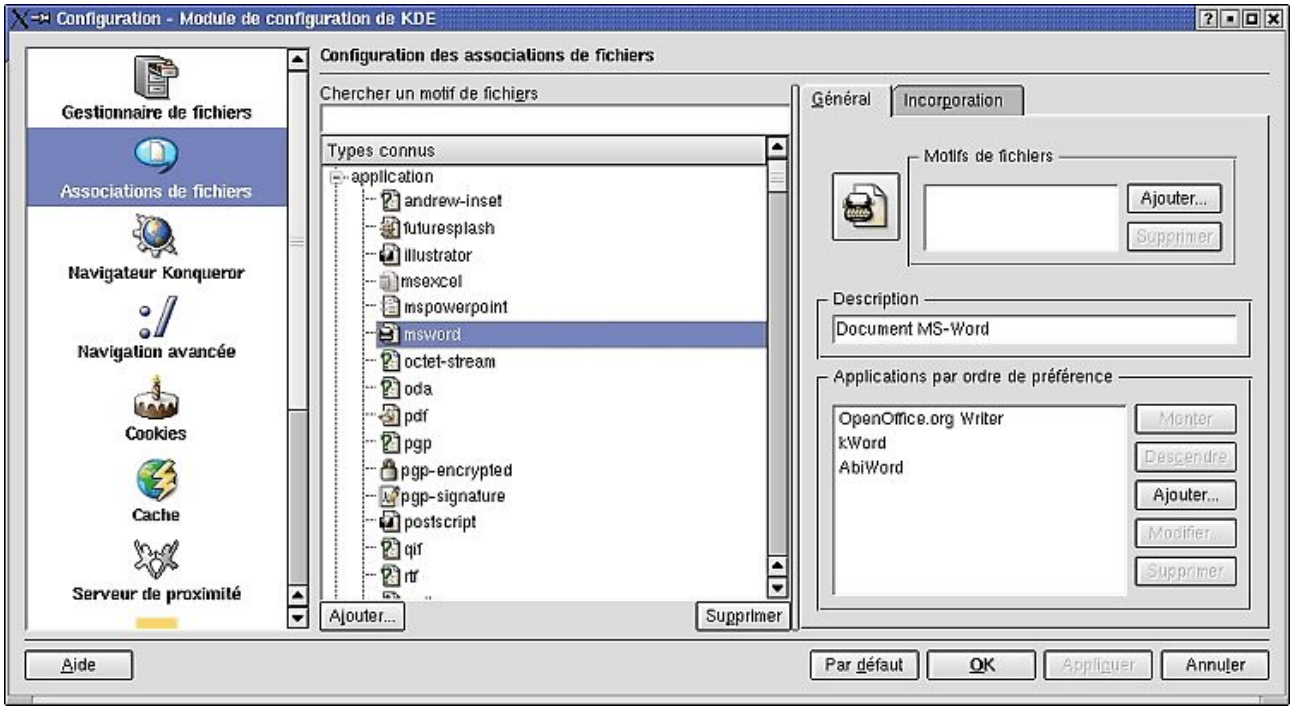
```
    GET /odj.doc HTTP/1.1\r\n
    Host: linux.maison.mrs\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.1)
                Gecko/20020826\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
            text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,
            text/css,*/*;q=0.1\r\n
    Mozilla ne connaît pas quant à lui les formats Microsoft.
    Accept-Language: fr-fr, en-us;q=0.66, en;q=0.33\r\n
    Accept-Encoding: gzip, deflate, compress;q=0.9\r\n
    Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66\r\n
    Plus respectueux de HTTP, il indique les jeux de caractères qu'il préfère
    iso-8859-1 (latin-1) d'abord, utf-8 (unicode) ensuite, * (n'importe quoi)
enfin.
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    \r\n



Frame 8 (1514 on wire, 1514 captured)
    ...
Internet Protocol, Src Addr: 192.168.0.253, Dst Addr: 192.168.0.10
    ...
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Sat, 09 Nov 2002 09:35:06 GMT\r\n
    Server: Apache-AdvancedExtranetServer/1.3.26 (Mandrake Linux/6.1mdk)
            auth_ldap/1.6.0 mod_ssl/2.8.10 OpenSSL/0.9.6g PHP/4.2.3\r\n
    Last-Modified: Thu, 06 Jul 2000 15:07:29 GMT\r\n
    ETag: "57d5a-7800-3964a0b1"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 30720\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: application/msword\r\n
    \r\n
    Data (1067 bytes)

0000  d0 cf 11 e0 a1 b1 1a e1 00 00 00 00 00 00 00 00   ................
0010  00 00 00 00 00 00 00 00 3e 00 03 00 fe ff 09 00   ........>.......
```

Rien de changé de ce côté là. Mozilla, une fois le fichier reçu proposera de l'enregistrer ou de l'afficher en démarrant MS Word, comme une application séparée.

Anecdotes diverses

Dans la lancée, la même manip, sous Linux (Mandrake 9, avec OpenOffice installé), avec le même Mozilla 1.1 et Konqueror 3.0.3. Les sniffs donnant toujours la même chose, inutile de les répéter.

- Mozilla va ouvrir le document avec Open Office Writer et, si le fichier MS Word n'est pas trop complexe, il sera lisible.
- Konqueror va proposer d'ouvrir le document avec kword. Mais kword est incapable d'ouvrir un fichier MS Word. Konqueror nous montre cependant quelque chose d'intéressant :



Konqueror montre que l'on peut facilement le configurer de manière à ce qu'il ouvre la "bonne application" en fonction du type MIME annoncé. L'illustration ci-dessus montre la correction faite pour que Konqueror ouvre désormais les types application/msword avec OpenOffice Writer. L'objet de cette manipulation est, vous l'avez compris, de montrer l'intérêt du type MIME annoncé dans une transmission HTTP.

Conclusions

Ce chapitre vous aura, je l'espère, aidé à mieux comprendre :

- Comment le Net arrive tout de même plutôt bien à se sortir élégamment du piège permanent que présente le transport de données au niveau mondial,
- certains messages que vos navigateurs web peuvent vous envoyer lorsque vous visitez des sites étrangers,
- pourquoi certains mails que vous pouvez recevoir peuvent être illisibles, et, peut-être, comment y remédier,
- les précautions qu'il faut prendre pour avoir de bonnes chances d'envoyer des e-mails lisibles par le plus grand nombre...