ETHERPIN

Geste de savoir

L'encodage UTF-8 à la main

19 novembre 2022

Table des matières

Introduction

Dans cet article, on s'intéresse à UTF-8, qui permet de représenter des textes écrits dans une grande variété de langues.
C'est grâce à UTF-8 que votre navigateur peut afficher par exemple:

, dà jiā hǎo, bonjour tout le monde

Quand on y réfléchit, c'est quand même prodigieux! Ça marche sur les PC (qu'ils soient sous Linux, macOS, Windows 7 et autres OS), et aussi sur les tablettes et mobiles iOS ou Android!

UTF-8 est incontournable pour publier des textes lisibles sur une grande variété d'appareils. Autant savoir comment ça marche.

i

Prérequis : Savoir ce qu'est la table de caractères ASCII, la base hexadécimale, la base binaire.
Objectif : Comprendre comment UTF-8 peut représenter tout ce qui est écrit et sans doute tout ce qui s'écrira.
Public visé: Pour les personnes curieuses.

1. Préambule : l' Universal Character Set

Avec le développement des réseaux de données, il est devenu important de standardiser la façon de communiquer des textes. Tant qu'il s'agissait des chiffres arabes et des lettres de l'alphabet latin, ça allait encore en utilisant une base répandue: le code ASCII, sur 7 bits. Mais dès lors qu'il s'agissait d'autres signes, on a vu apparaître des codes sur 8 bits, avec des normes de fait adaptées aux marchés locaux. Le même code pouvait correspondre à plusieurs signes, d'où une certaine confusion.
Cette histoire est détaillée sur ZdS: L'épopée des encodages .

Pour s'en sortir, la norme ISO 10646 et le consortium Unicode ont développé conjointement un jeu de caractères unique, communément appelé UCS

> Le standard Unicode contient, en plus de l'UCS, des règles de présentation, par exemple pour les écritures de droite à gauche ou les écritures verticales.
> Comme les normes ISO ne sont pas libres d'accès, il est recommandé d'utiliser la documentation d'Unicode .

1.1. UCS

UCS est l'abréviation de *Universal Character Set*. Le terme *Character* n'est pas très précis. Ici, un caractère correspond à l'unité minimale ayant un sens et à laquelle correspond un graphème. Il y a d'une part des **caractères abstraits** et d'autre part leur représentation graphique (graphèmes).
L'UCS est un répertoire de caractères abstraits indépendamment de leur représentation.
L'UCS contient aussi des caractères de contrôle destinés par exemple à la mise en page ou pour générer des graphèmes composés.

1.1.1. Point de code

Chaque caractère abstrait se voit associer un nombre unique. Ce nombre est appelé point de code (*code point* en anglais),. Il est accompagné par un texte de description. Le point de code est un nombre compris entre 1 et 17×2^{16}, soit potentiellement 1114112 signes.
l'UCS est évolutif, il est régulièrement enrichi de nouveaux points de code. Un point de code est noté U+ suivi de la valeur hexadécimale du point de code.
Exemple: **U+00E9** pour le caractère **é**.

1.1.2. Quel codage pour un point de code ?

Unicode normalise trois codages: UTF-32, UTF-16 et UTF-8.

1.1.2.1. UTF-32

UTF-32 utilise 32 bits pour représenter un point de code.
Un point de code peut être codé sur 20 bits. Pour le représenter sur nos machines, on peut utiliser 4 octets, soit 32 bits.
L'UTF-32 a l'avantage d'être simple, les point de code ont tous la même taille. On peut ainsi retrouver facilement les caractères dans une chaîne de caractères. On peut aussi facilement faire des tris.
L'UTF-32 a toutefois un gros inconvénient: il n'est pas très économe, car sur les 32 bits, seuls 20, au plus, sont utiles.

1.1.2.2. UTF-16

UTF-16 est un codage sur 16 bits.
Dans les premières versions de l'UCS, les points de code étaient tous codés sur 16 bits.
Avec 16 bits, on peut de fait représenter les caractères utilisés couramment par l'humanité, par exemple les caractères dérivés de l'alphabet latin, mais aussi les caractères dérivés de l'alphabet cyrillique, les écritures du moyen orient, sans oublier les écritures asiatiques.

Pour les points de code à partir de 65536, UTF-16 utilise **deux** blocs de 16 bits.

Pour les textes utilisant des points de code inférieurs à 255 (notamment les codes ASCII), ce codage est deux fois plus volumineux que l'utilisation de codes sur 8 bits, mais il permet de représenter sans confusion la plupart des caractères usuels.

Cette représentation reste simple à gérer si on se limite aux points de code sur 16 bits, car tous les points de code ont la même taille. Ce n'est pas le cas si on veut pouvoir coder tous les points de code.

1.1.2.3. UTF-8

On y arrive enfin!

2. UTF-8

UTF-8 est un encodage utilisant de 1 à 4 octets, il est normalisé par le consortium Unicode
.

2.1. Finalité et portée

UTF-8 permet de coder sans ambiguïté tout le répertoire UCS.
Il est très répandu, car c'est aussi un des standards de l'Internet. De nos jours, la plupart des pages sont codées avec UTF-8[1].

UTF-8 présente l'avantage d'être un codage compact, surtout pour les textes qui contiennent une majorité de caractères ASCII. Il permet de coder les alphabets occidentaux sur deux octets. Par contre, il est moins avantageux pour les textes asiatiques, pour lesquels il nécessite 3 octets par caractère.

2.2. Compatibilité avec le standard ASCII

Rappel: le standard ASCII utilise 7 bits. En y ajoutant un bit à 0 de poids fort, on obtient un octet.

Les points de code de la table ASCII sont identiques aux 127 premiers points de code de la table Unicode. Par conséquent, un texte écrit en ASCII est également un texte codé en UTF-8 . Par exemple, le signe `@` a pour code ASCII `64` (0x40) et pour point de code Unicode `64` également, noté `U`+`0040`. Il est codé sur un octet en UTF-8, comme en ASCII.

Ce point très important fait d'UTF-8 une généralisation du standard ASCII, lui-même massivement utilisé.

Références :

Unicode Lookup permet de parcourir l'UCS. Fournis un glyphe, la description, les valeurs du point de code en octal, en décimal, en hexadécimal et le codage HTML.

Unicode Compart C'est un explorateur de la base de donnée de l'UCS. qui a de nombreuses possibilités de sélection des points code.

En savoir plus

Comprendre les encodages
Ne ratez pas la magnifique illustration `Martine Ã©crit en UTF-8`

3. Principes de codage

Pour coder un point de code en UTF-8, il faut entre 1 et 4 octets.
La table suivante permet de déterminer le nombre d'octets nécessaires en partant de la valeur du point de code.

Point de code	Codage UTF-8 en binaire	1 octet: valeurs possibles	Nb. de bits à coder
U+0000 à U+007F	0xxxxxxx (table ASCII)	00 à 7F	7
U+0080 à U+07FF	110xxxxx 10xxxxxx	C2 à DF	5+6=11
U+0D00 à U+FFFF	1110xxxx 10xxxxxx 10xxxxxx	E0 à EF	4+6+6=16
U+10000 à U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	F0 à F4	2+6+6+6=20

1. Voyez à ce sujet la RFC 3629 «UTF-8, a transformation format of ISO 10646».

3.1. Si le point de code est entre 0 et 127

Il est codé sur un octet dont le bit de poids fort est nul. Il contient un code ASCII.

3.2. Si le point de code est supérieur à 127

Dans ce cas, le point de code est codé sur 2, 3 ou 4 octets.
Le premier octet est appelé «octet d'en-tête».
Chaque octet suivant est appelé «octet de suite».
On a donc 1, 2 ou 3 octets de suite.

3.3. L'octet d'en-tête

L'octet d'en-tête a son bit de poids fort à 1.
Il a ensuite autant de 1 qu'il y a d'octet(s) de suite. Cette séquence de bit à 1 se termine par un bit à 0.
Ainsi, si le premier octet commence par `110`, il y aura deux octets, s'il commence par `1110`, 3 octets et, finalement, s'il commence par `11110`, 4 octets.

Dans l'octet d'en-tête, il reste 5, 4 ou 3 bit dans lesquels on va mettre des bits à coder.

3.4. Octet(s) de suite

Un octet de suite commence par `10`.
Du coup, il reste 6 autres bits dans lesquels on va mettre des bits à coder.

3.5. Règles importantes

On **doit** utiliser le format le plus court possible.

Il **est interdit** de coder un point de code entre `U`+`D800` et `U`+`DFFF`. Cette plage de valeurs est réservée au codage en UTF-16. Un tel point de code ne correspond à aucun caractère.

Des exemples?

Pour voir le codage UTF-8 à l'œuvre: https://mothereff.in/utf-8

i Pour information, voici la table des codages valides:

Code Point	Premier octet	Deuxième octet	Troisième octet	Quatrième octet
U+0000..U+007F	00..7F			

U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	A0..BF	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80..9F	80..BF	
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	90..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..8F	80..BF	80..BF

4. Quelques exemples (en binaire)

Voici quelques exemples d'encodage en UTF-8. Nous allons nous aider de la table définie ci-dessus pour les réaliser. Également, nous pouvons nous aider du site Unicode Compart . Pour ce faire, entrez le caractère désiré dans la barre de recherche, puis la page qui s'affiche donne tous les renseignements utiles.

Pour faire le codage à la main, il est nécessaire de convertir le point de code en binaire.

4.0.0.1. É U+00C9 LATIN CAPITAL LETTER E WITH ACUTE

La table nous indique que ce point de code nécessite 2 octets: un octet d'en tête et un octet de suite.
Convertissons en binaire: `0x00C9` = `0000 0000 1100 1001`.
La colonne de gauche de la table nous indique comment répartir les 11 bits à coder: 5 + 6.
Les 6 derniers bits à coder sont `00 1001`.
On les met dans l'octet de suite qui est donc: `10**00 1001**`.
On met les 5 premiers bits à coder `0 0011` dans l'octet d'en-tête: `110**0 0011**`.
Résultat: `0xC3 0x89`

4.0.0.2. € U+20AC EURO SIGN

La table nous indique que ce point de code nécessite 3 octets: un octet d'en-tête et deux octets de suite.
Convertissons en binaire: `0x20AC` = `0010 0000 1010 1100`.
La colonne de gauche de la table nous indique qu'il y a 16 bits à coder à répartir ainsi: 4 + 6 + 6:
`0x20AC` = `0010` `0000 10` `10 1100`
Les 4 premiers bits `0010` vont dans l'octet d'en tête.
Les deux tranches de 6 bits: `00 00010` et `10 1100` vont dans les octets de suite.
Octet d'en tête: `1110 **0010**`.
Octet de suite n°1: `10**00 0010**`.
Octet de suite n°2: `10**10 1100**`.
Résultat: `0xE2 0x82 0xAC`.

4.0.0.3. 🍊 U+1F600 GRINNING FACE La table nous indique que ce point de code nécessite 4 octets: un octet d'en tête et trois octets de suite.
Convertissons en binaire: `0x1F600` = `0001 1111 0110 0000 0000`.
La colonne de gauche de la table nous indique qu'il y a 20 bits à coder qui se répartissent ainsi: 2 + 6 + 6 + 6.
Si on découpe comme demandé, cela donne: `0x1F600` = `00` `01 1111` `0110 00` `00 0000`.
Les octets de suite sont `1001 1111`, `1001 1000` et `1000 0000`.
Pour ce point de code, l'octet d'en-tête est `1111 0000`.
Résultat: `0xF0 0x9F 0x98 0x80`.

Et maintenant, à vous de jouer!

4.0.0.4. à U+00E0 LATIN SMALL LETTER A WITH GRAVE

> Contenu masqué n°1

Conclusion

Sur les systèmes modernes, il n'est pas nécessaire d'encoder de l'UTF-8 à la main. Il est souvent possible de gérer par exemple des fichiers texte en français avec des lettres accentuées, des `ç`, `à`, `ù` et autres spécificités de cette langue.

Pour voir l'encodage UTF-8 à l'œuvre, il suffit d'un éditeur hexadécimal et d'y ouvrir un fichier encodé en UTF-8.

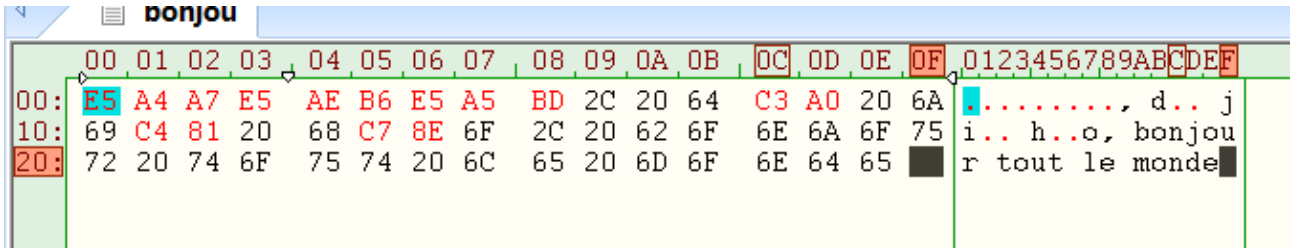

FIGURE 4.1. – Exemple d'une capture de l'éditeur Hex Edit lisant un fichier contenant le texte: «, dà jiā hǎo, bonjour tout le monde». On peut y voir le contenu en hexadécimal, par paire de deux chiffres hexadécimaux: «e5 a… 0a». Les codes des caractères ASCII sont en noirs (valeurs de 0x00 à 0x7F), les autres codes sont en rouge.

Un regard exercé distingue les trois caractères chinois, 3 séquences de trois octets commençant par `0xE5`.
De même, les séquences qui commencent par `0xC3` correspondent à un codage sur deux octets.

i **Hors sujet**

Dédécodage des séquences UTF-8.

C'est facile quand le codage respecte les règles. Avec la table des codes valides, on peut rapidement voir si une séquence est valide.
Il suffit alors de rassembler les bits codés pour obtenir la valeur du point de code en binaire.
Là où ça devient compliqué, c'est quand il y a des erreurs. On voudrait sans doute récupérer un maximum de morceaux corrects. Ce n'est pas simple.

Contenu masqué

Contenu masqué n°1

La table nous indique que ce code point nécessite 2 octets: un octet d'en tête et un octet de suite.
Convertissons en binaire: `0xE0C9` = `0000 0000 1110 0000`.
La colonne de gauche de la table nous indique comment répartir les 11 bits à coder: 5 + 6.
Les 6 derniers bits à coder sont `10 0000`.
On les met dans l'octet de suite qui est donc: `10**10 0000**`.
On met les 5 premiers bits à coder `0 0011` dans l'octet d'en-tête: `110**0 0011**`.
Résultat: `0xC3 A0`.

Retourner au texte.

Liste des abréviations

UCS Universal Character Set. 1–3

UTF Unicode Transformation Format. 2, 3