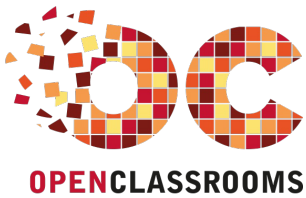
Passer du latin1 à l'unicode

Par Victor Thuillier (vyk12)



www.openclassrooms.com

Dernière mise à jour le 5/05/2011

Sommaire

Passer du latin1 à l'unicode

Par  Victor Thuillier (vyk12)

Mise à jour : 05/05/2011
Difficulté : Facile

Bienvenue à tous dans ce tutoriel.

Ici je vais vous expliquer comment migrer son site internet encodé en **latin1** (**ISO-8859-1**) à l'**unicode** (**UTF-8**), ainsi que les avantages que ça présente, mais aussi les inconvénients.

Vous trouverez en annexes les principales erreurs que vous serez susceptibles de rencontrer.
Sommaire du tutoriel :

Pourquoi passer à l'unicode ?

Voici la question que vous devez sans doute vous poser. Pourquoi passer à l'unicode ? Si vous avez suivi vos cours de xHTML / CSS sur ce site même, on vous a toujours appris (ou plutôt, M@teo21) à utiliser la norme du **latin1** (le fameux **ISO-8859-1**) que vous déclarez entre les balises `<head> </head>` :

Code : HTML

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

L'inconvénient du **latin1** est qu'il utilise un encodage de caractère basé sur 1 octet, soit 256 possibilités (1 caractère = 1 octet = 8 bits, dans chaque bit il peut y avoir soit 0 soit 1 ; 2 ^ 8 = 256, d'où les 256 possibilités). Ainsi, dans votre page, vous ne pourrez placer que 256 caractères différents. Avec cette norme, vous ne pourrez ainsi jamais placer des caractères accentués et des caractères chinois sur la même page par exemple. Si vous êtes sous Windows, vous pouvez regarder combien de caractères sont disponibles avec telle police et telle norme d'encodage (**unicode**, **Windows Occidental**, etc.). Pour cela, cliquez sur **Démarrer** > **Exécuter** et tapez **charmap**. Sélectionnez **Unicode** dans le jeu de caractère, regardez 5 secondes et passez à **Windows Occidental** (c'est plus ou moins l'équivalent de la norme **ISO 8859-1** : notez l'absence du tiret car cette norme est celle utilisée sur votre système et non sur internet). Le nombre de caractères disponibles a légèrement baissé, n'est-ce pas ?

Si vous voulez la liste des caractères de la norme **ISO-8859-1**, la voici (les mots en italiques sont des caractères de contrôles ajoutés pour son utilisation sur Internet) :

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*

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

Ainsi, pour résoudre cette contrainte, l'**unicode** a été développé. Vous pourrez donc afficher des caractères accentués et des caractères d'un autre alphabet (russe, grec etc.) ou d'autres caractères spéciaux (signes chinois, japonais etc.).

Le fait que l'**unicode** soit codé sur plusieurs octets posera certains problèmes, notamment avec les fonctions PHP gérant les chaines de caractères. En effet, certaines fonctions se basent sur les bits de cette chaine, ce qui posera problème, par exemple, pour mesurer celle-ci. J'y reviendrai plus loin dans ce tutoriel.

Préparer PHP

Avant de commencer à convertir les données de votre site web en **unicode**, nous allons préparer PHP. Par défaut, la norme qu'utilise PHP est le **latin1** (ou **iso-8859-1**). Il se peut, suivant la configuration par défaut utilisée, que PHP envoie systématiquement au client le jeu de caractère **iso-8859-1** dans les en-têtes. Il faut donc changer cela, sinon tous les efforts que vous allez faire pour suivre ce tutoriel pourraient être réduits à néant.

La première solution consiste à modifier le **php.ini** du serveur. Rajoutez cette directive, dans la section **Data Handling** :

Code : Ini

```ini
default_charset = "utf-8"
```

Une deuxième solution consiste à créer ou modifier le fichier **.htaccess**, situé à la racine du projet, afin d'y insérer une instruction :

Code : Apache

```
php_flag default_charset utf-8
```

PHP est maintenant bien configuré. Regardons maintenant du côté des fichiers et de leur encodage.

Encoder les fichiers

Pour passer son site internet du **latin1** à l'**unicode**, il va falloir modifier votre environnement de travail. En effet, c'est lui qui va

encoder les caractères du fichier et les sauvegarder. La majorité des éditeurs encodent, par défaut, vos fichiers en **latin1**.

Mais ce n'est pas tout. Tous vos fichiers déjà créés sont encodés en **latin1** : il va donc falloir les modifier pour que les caractères utilisent la norme d'encodage de l'**unicode**.

On va donc commencer par l'environnement de travail, puis on verra comment encoder les fichiers après de façon automatique avec PHP.

L'environnement de travail

Il faut configurer votre outil de travail de telle sorte que lors de la création de tout nouveau fichier, l'encodage des caractères soit par défaut réglé à **UTF-8**. Si vous êtes sous Notepad++, allez dans **Paramétrage**, **Préférences**, sélectionnez l'onglet **Nouveau document / Répertoire** puis cochez **UTF-8 sans BOM**.

Petite question ... pourquoi tu as mis **sans BOM** en rouge ?

Ah oui, il serait préférable de me justifier sur ce point-là. Le BOM est le marquage d'ordre des octets du fichier, complètement inutile en **UTF-8**. Si jamais vous choisissez cette option, l'éditeur de texte insérera cet ordre d'octets **au tout début** du fichier. Ainsi, si vous voulez appeler la fonction `header` de PHP, vous ne pourrez pas. Des caractères auront déjà été envoyés au navigateur : c'est la **marque d'ordre des octets**. Pensez donc bien à ne jamais les insérer.

Encoder les fichiers déjà créés

Là, vous avez deux solutions. La première consiste à modifier les fichiers 1 à 1. Ouvrez le fichier avec votre éditeur et, si vous êtes sous Notepad++, allez dans **Format** puis cliquez sur **Convertir en UTF-8 (sans BOM)**. Enregistrez le fichier et le tour est joué.

Je me permets de ré-insister sur le fait qu'il est important de spécifier que l'on ne veut pas le BOM du fichier. Relisez juste au-dessus si vous avez oublié pourquoi ou si vous avez sauté la partie.

Je vais maintenant vous dire comment modifier tous les fichiers d'un répertoire pour les encoder en **UTF-8** grâce à PHP. On va pour cela faire appel à la fonction `utf8_encode`. Cette fonction permet d'encoder une chaine de caractères donnée (encodée en **latin1**, soit **ISO-8859-1**) en **UTF-8**.

Ce tutoriel a pour but de vous dire comment passer du **latin1** à l'**unicode** et non de vous faire faire des exercices en utilisant PHP, donc je vais vous donner le script tout fait. Placez ce fichier dans le répertoire à encoder et lancez-le. Une fois que le script a terminé de travailler, tous les fichiers du répertoire seront encodés.

Code : PHP

```php
<?php
    $dossier = opendir ('.');

    while ($fichierAEncoder = readdir ($dossier))
    {
        if (is_file ($fichierAEncoder))
        {
            $contenu = file_get_contents ($fichierAEncoder);
            $fichier = fopen ($fichierAEncoder, 'w');
            fputs ($fichier, utf8_encode ($contenu));
            fclose ($fichier);
        }
    }

    closedir ($dossier);
?>
```

Les fichiers contenus dans les sous-répertoires ne seront donc pas encodés. Si vous voulez qu'ils le soient, exécutez ce code (tous les fichiers contenus dans le même répertoire que le script ainsi que dans tous les sous-répertoires seront encodés) :

Code : PHP

```
<?php
    function encoderDossier ($dossierAEncoder)
    {
        $dossier = opendir ($dossierAEncoder);

        while ($fichierAEncoder = readdir ($dossier))
        {
            if ($fichierAEncoder != '.' AND $fichierAEncoder != '..')
            {
                if (is_file ($fichierAEncoder))
                {
                    $contenu = file_get_contents ($fichierAEncoder);
                    $fichier = fopen ($fichierAEncoder, 'w');
                    fputs ($fichier, utf8_encode ($contenu));
                    fclose ($fichier);
                }

                else
                    encoderDossier ($dossierAEncoder . $fichierAEncoder . '/');
            }
        }

        closedir ($dossier);
    }

    encoderDossier ('./');
?>
```

Et voilà, tous vos fichiers sont encodés en **UTF-8**, et votre environnement de travail est bien configuré.

Et si j'ai pas Notepad++ ?

- Si vous êtes sous **Eclipse**, cliquez sur **Window** > **Préférences** > dans le menu de gauche sur **General** > en bas sur **Workspace** > dans la boite **Text file encoding** cochez **Other** et sélectionnez **UTF-8** ;
- Si vous êtes sous **Dreamweaver**, cliquez sur **Edition** > **Préférences** > **Nouveau document** > **Codage par défaut** ;
- Si vous êtes sous **Zend Studio**, allez sur **Tools** > **Desktop** > **Apparence** ;
- Si vous êtes sous **gedit**, cela se fait lors de l'enregistrement du fichier. Dans la boite de dialogue ouverte après avoir cliqué sur **Fichier** > **Enregistrer sous...**, choisissez **Locale actuelle (UTF-8)** dans la liste déroulante située en bas.

Je sais qu'il y en beaucoup d'autres mais d'une part, je ne les connais pas tous, et d'autre part il serait trop long de tous les citer. Fouillez un peu dans votre éditeur ou si vous êtes vraiment bloqué, procédez à une une recherche sur Google. Au pire des cas, le forum du Site du Zér0 est là.

Encoder la base de données

Toujours et encore des données à encoder. Pour la base de données (j'utiliserai ici le SGBDR MySQL) il faut l'encoder en **utf8_general_ci**.

Avant cela, nous allons spécifier l'encodage dans lequel sont encodées les données envoyées via une requête. Si vous êtes hébergé chez un serveur mutualisé, vous ne pourrez pas faire cela de façon définitive. En effet, il vous faudra, à chaque début de script, ajouter une instruction qui dira à MySQL que les données sont encodées en **UTF-8**. Pour cela, on va regarder du côté de `mysql set charset`, que vous placerez dans vos scripts PHP juste après vous être connecté. Exemple :

Code : PHP

```php
<?php
    mysql_connect ('localhost', 'root', '');
    mysql_set_charset ('UTF8'); // notez l'absence du tiret entre UTF et 8 : c'es
?>
```

Si votre version de PHP est inférieure à la version 5, cette fonction ne sera pas disponible. Pour ce faire, il faut exécuter une simple requête grâce à `mysql_query` en utilisant `SET NAMES 'encodage'` :

Code : PHP

```php
<?php
    mysql_connect ('localhost', 'root', '');
    mysql_query ('SET NAMES \'UTF8\''); // toujours sous MySQL, donc toujours pas
?>
```

Notez que ces deux méthodes reviennent au même.

Et si j'utilise PDO, je fais quoi ?

Si vous utilisez PDO (ce que je vous encourage à faire si vous ne l'utilisez toujours pas), il vous suffit de passer un quatrième argument à son constructeur comme ceci (ça revient au même que d'exécuter plus loin un `SET NAMES` avec la méthode `exec` ou `query`) :

Code : PHP

```php
<?php
    $db = new PDO ('mysql:host=localhost;dbname=tests', 'root', '', array (PDO::
?>
```

Maintenant, si vous voulez créer une table, vous devrez impérativement choisir un interclassement commençant par **utf8_**, comme **utf8_general_ci** par exemple (champ multilingue insensible à la casse).

Par exemple, pour créer une table, voici le type de code SQL à exécuter :

Code : SQL

```sql
CREATE TABLE ma_table (
  id int(11) NOT NULL auto_increment,
  champ varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  PRIMARY KEY  (id)
) DEFAULT CHARSET=utf8;
```

Ainsi, le champ **champ** sera encodé en **UTF-8**.

Si vous avez déjà créé votre table et que tous vos champs sont encodés en **latin1**, alors il va falloir les modifier de telle sorte à ce qu'ils soient encodés en **UTF-8**. Pour cela, vous avez deux solutions. La première est de les modifier un par un via phpMyAdmin. La seconde, plus rapide, est d'exécuter une simple requête sur votre table :

Code : SQL

```
ALTER TABLE `ma_table` CONVERT TO CHARACTER SET utf8
```

Plus pratique que la première méthode, il faut le reconnaitre. 

Modifier les entêtes

Encoder vos fichiers ne suffit pas à les traiter correctement par le navigateur : il faudra indiquer quelle norme d'encodage utilise la page.

Modifier l'entête envoyée par le serveur

La première chose consiste à modifier les entêtes envoyées par le serveur. Par défaut, Apache travaille avec la norme **ISO-8859-1**, il va donc falloir lui dire que l'on veut qu'il travaille avec la norme **UTF-8**. Si vous avez accès à la configuration d'Apache (en local ou avec un serveur dédié par exemple), vous pouvez modifier le fichier de configuration d'Apache, à savoir **httpd.conf**. Il vous suffit d'ajouter une des deux lignes suivantes :

Code : Apache

```
AddDefaultCharset Off
# ou alors
AddDefaultCharset UTF-8
```

La première instruction a pour objectif d'indiquer au serveur de se renseigner sur l'entête de la page HTML pour savoir quelle norme elle utilise. Le serveur sera ainsi toujours en cohérence avec la norme utilisée par la page demandée.

Si vous n'avez pas accès au fichier **httpd.conf**, vous pouvez ajouter une de ces deux lignes dans le fichier **.htaccess**.

Modifier l'entête de la page

Après la modification des entêtes du serveur, on vient à la modification des entêtes de la page. Vous avez ici deux solutions. La première consiste à passer par PHP en utilisant la fonction `header` . Vous devez lui spécifier en paramètre le type de la page ainsi que son encodage (dans notre cas, **UTF-8**).

Code : PHP

```
<?php
    header ('Content-type:text/html; charset=utf-8');
?>
```

Ainsi, le navigateur saura que cette page est encodée en **UTF-8** et pourra afficher les caractères correctement.

La seconde solution consiste à placer une balise **meta** entre les balises **<head> </head>** de vos pages.

Code : HTML

```
<meta http-equiv="Content-type" content="text/html; charset=utf-8"
```

```
/>
```

Si vous utilisez ces deux techniques sur la même page, la fonction `header` prendra le dessus. Ainsi, si vous faites un `header ('Content-type:text/html; charset=iso-8859-1');` puis que vous ajoutez la meta suivante dans l'entête de la page HTML `<meta http-equiv="Content-type" content="text/html; charset=utf-8" />` , le navigateur interprétera la page comme si elle était encodée en **latin1** !

Petite question comme ça ... tu nous dis de modifier les entêtes envoyées par le serveur puis les entêtes du fichier. Moi, j'ai juste modifié les entêtes du fichier sans modifier les entêtes envoyées par le serveur et ma page s'affiche très bien ! Donc à quoi ça sert ?

Ça sert à ce que votre page soit bien interprétée au cas où votre fichier ne spécifie aucun encodage. C'est donc celui du serveur qui sera utilisé. Si votre page est encodée en **UTF-8** et que le serveur envoie une entête avec un **ISO-8859-1** en guise de norme d'encodage, certains caractères ne s'afficheront pas comme il faut (notamment les caractères accentués).

Un problème bien ennuyeux

Comme dit dans la toute première partie de ce mini-tutoriel, le fait que l'**unicode** encode ses caractères sur un ou plusieurs octets peut poser problème. Nous allons voir ici le cas de la fonction `strlen`. Cette fonction renvoie la longueur de la chaine qu'on lui passe en paramètres, ou plus exactement, **le nombre d'octets de la chaine**. Vous voyez le problème ?

Un caractère normal (a, z, e, r, t, y etc.) est codé sur un octet, mais, par exemple, les caractères accentués sont codés sur 2 octets. Ainsi, la fonction suivante renverra 3 :

Code : PHP

```php
<?php
    echo strlen ('éa'); // é = 2 octets et a = 1 octet
?>
```

Cette norme encodant ses caractères sur plusieurs octets (4 maximum), il va falloir utiliser les fonctions sur les chaînes de caractères multi-octets. Toutes ces fonctions commencent par le préfixe **mb_** (multi bytes).

Avant toute chose, il faudra indiquer le type d'encodage que l'on souhaite utiliser avec ces fonctions en utilisant `mb_internal_encoding`. Par défaut, l'encodage utilisé est **ISO-8859-1**, il va donc falloir lui envoyer en paramètre **UTF-8**. Si vous ne lui spécifiez rien en paramètre, la fonction retourne l'encodage utilisé.

Code : PHP

```php
<?php
    echo mb_internal_encoding(); // affiche ISO-8859-1
    echo mb_strlen ('éa'); // affiche 3 car la chaine est encodée
en UTF-8 alors que les fonctions multi-octets sont par défaut
configurées pour la norme ISO-8859-1

    mb_internal_encoding ('UTF-8'); // on utilise la norme UTF-8
    echo mb_strlen ('éa'); // affiche 2
?>
```

Sachez que vous pouvez aussi spécifier en deuxième paramètre de `mb_strlen` l'encodage à utiliser si celui-ci n'est pas le même que celui actuellement utilisé. Exemple :

Code : PHP

```php
<?php
    echo mb_strlen ('éa', 'UTF-8'); // affiche 2
?>
```

Et voilà, vous pourrez désormais mesurer vos chaines en utilisant la norme **UTF-8**.

Annexes

Voici les annexes du tutoriel. Cette sous-partie répertorie les différentes erreurs que vous êtes susceptibles de rencontrer.

Caractères bizarres

Il est possible que vous rencontreriez des caractères "bizarres" qui n'ont rien à faire à cet endroit. Il y a deux sortes de caractères "bizarres" :

- "" => la chaine de caractères est encodée en **ISO-8859-1**. Il y a de fortes chances que le navigateur croit avoir affaire à de l'**UTF-8** (probablement parce que votre page dit au navigateur qu'elle utilise l'**UTF-8**). Utilisez la fonction `utf8_encode` sur la chaine de caractères posant problème si elle est issue d'une variable PHP. Ce genre de caractère peut aussi s'afficher si le caractère qui devrait s'afficher ne fait pas parti de la table de caractères de cette norme (exemple : le ÿ majuscule). Pour cela, utilisez l'entité HTML correspondante (exemple : `&Yuml;`) ;
- "Ã©", "Ã®", "Ã" etc. => la chaine de caractères est encodée en **UTF-8** mais le navigateur croit avoir affaire à de l'**ISO-8859-1** (probablement parce que votre page dit au navigateur qu'elle utilise l'**ISO-8859-1**). Utilisez la fonction `utf8_decode` sur la chaine de caractères posant problème si elle est issue d'une variable PHP.

Fonctions sur les chaines de caractères

N'utilisez jamais la fonction `htmlentities` . Cette fonction convertit la chaine en entitées HTML de la norme ISO, ce qui ne plaira pas au navigateur si vous lui dites que la page utilise la norme **UTF-8**. Si vous voulez toujours utiliser cette fonction, indiquez-lui la norme d'encodage à utiliser en 3ème paramètre :

Code : PHP

```php
<?php
    htmlentities ('azéoràèndjùd', ENT_COMPAT, 'UTF-8');
?>
```

Cependant, je vous recommande plutôt d'utiliser la fonction `htmlspecialchars` .

Les fonctions manipulant les chaines de caractères sont aussi une grosse source d'erreurs (`strlen` , `substr` , `strpos` etc.). Pour remédier aux problèmes qu'elles impliquent, utilisez les fonctions sur les chaines de caractères multi-octets.

Le dernier point important qu'il me semble utile d'évoquer est celui des expressions régulières avec les options de celles-ci. Cliquez ici pour en savoir plus.
Et voilà, ce tutoriel touche à sa fin. J'espère que cela vous aura été utile.

Partager