

# Passez à l'UTF-8 sans manquer une étape

par [Josselin Willette \(Page d'accueil\)](#)

Date de publication : 12/09/2009

Dernière mise à jour : 29/11/2010

N'avez jamais vous pesté contre des caractères s'affichant mal, carrés, points d'interrogation ou caractères étranges à la place des accents ? Et ceci dès que vous essayiez d'utiliser un encodage en UTF-8 ?

Ce tutoriel va vous expliquer comment encoder votre site intégralement en UTF-8 sans louper une étape qui pourrait faire apparaître ces caractères disgracieux.

Commentez cet article :

---

I - Introduction.....	3
II - Au niveau du document HTML.....	3
III - Au niveau du fichier.....	3
IV - Au niveau du serveur.....	3
IV-A - Méthode PHP.....	3
IV-B - Méthode Apache.....	4
V - Au niveau de la base de données.....	5
V-A - Aux champs.....	5
V-B - A la connexion.....	5
VI - Remerciements.....	6

## I - Introduction

Cet article est basé sur les technologies Apache, PHP et MySQL, donc aucun des codes suivants ne fonctionne sur un autre type d'environnement.

Selon votre environnement, le navigateur va utiliser différentes méthodes pour choisir quel encodage utiliser pour parser et afficher le document demandé. Dans le cas d'un fichier statique local, sans serveur (donc sans utiliser même en local des logiciels comme WAMP ou EasyPHP), le navigateur va utiliser la balise `<meta>` décrite plus loin, alors que dans le cas d'un serveur, le navigateur va se référer à l'en-tête renvoyé par celui-ci.

## II - Au niveau du document HTML

L'encodage au niveau d'un document HTML se définit grâce à une balise `<meta>` :

### Encodage du document

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

Cette balise doit être insérée dans l'élément `head` de votre document HTML et doit évidemment être unique.


## III - Au niveau du fichier

Chaque fichier doit impérativement être enregistré en UTF-8 sans BOM (**Byte Order Mark**).

Chaque éditeur fonctionne de manière différente pour permettre l'enregistrement des fichiers en UTF-8. Voici quelques exemples de manipulation sur certains éditeurs :

- **Notepad++** : Aller dans Format > Encoder en UTF-8 (sans BOM).
- **Dreamweaver** : Aller dans Modifier > Propriétés de la page > Titre/Codage.
- **Aptana** : Aller dans Edit > Set Encoding.
- **Bloc notes Windows** : Aller dans Fichier > Enregistrer sous... > Sélectionner UTF-8 dans la liste Codage.
- **PHPDesigner** : Aller dans Fichier > Encodage du fichier > Sélectionner UTF-8 dans la liste.
- **PHPEdit** : Dans la barre des tâches, sélectionner UTF-8 sans marque d'ordre des octets dans la liste.
- **UltraEdit** : Aller dans Fichier > Conversions > ASCII vers UTF-8.

N'hésitez pas à me faire part des manipulations sur d'autres logiciels pour compléter la liste.

 *Il ne faut pas confondre document HTML et fichier. A savoir qu'un document HTML peut être formé de plusieurs fichiers, dans le cas d'include en PHP. Donc l'ensemble des fichiers qui forment le document HTML doivent être enregistrés en UTF-8 sans BOM.*

## IV - Au niveau du serveur

### IV-A - Méthode PHP

Il existe deux méthodes en PHP permettant d'afficher du texte en UTF-8. Après avoir bien sûr encodé correctement tous les fichiers selon la manière décrite juste au-dessus. L'une est radicale au niveau du fichier, l'autre se fait au cas par cas, sur chaque texte à afficher.

La méthode radicale consiste à mettre en première ligne de chaque fichier, un header qui va préciser au serveur de renvoyer de l'UTF-8 :

### Header

#### Header

```
header( 'content-type: text/html; charset=utf-8' );
```

L'autre méthode consiste à utiliser une fonction PHP autour du texte que l'on veut afficher en UTF-8 :

#### Fonction utf8\_decode()

```
echo utf8_decode( 'Ici mon texte en UTF-8' );
```

La différence entre les deux méthodes est flagrante. La seconde est d'une part plus contraignante parce qu'il faut l'utiliser sur chaque texte et d'autre part ne précise pas au serveur de renvoyer de l'UTF-8.

Faisons un petit test pour nous en convaincre. Nous allons créer deux fichiers distincts, un nommé `test1.php` et l'autre nommé `test2.php`. Ces deux fichiers seront évidemment encodés correctement en UTF-8 (cf III).

Dans `test1.php` nous mettons ce code :

#### test1.php

```
header( 'content-type: text/html; charset=utf-8' );  
  
echo 'Texte accentué.';
```

Dans `test2.php` nous mettons ce code :

#### test2.php

```
echo utf8_decode( 'Texte accentué.' );
```

Ouvrons-les dans le navigateur. Regardons l'encodage des pages dans Affichage > Encodage des caractères. Nous pouvons constater que `test1.php` est encodé en Unicode (UTF-8) alors que `test2.php` est encodé en Occidental (ISO-8859-1).

Mais pourquoi `utf8_DEcode()` alors que l'on veut ENcoder en UTF-8 ? Ne serait-ce pas plutôt `utf8_encode()` qu'il faut utiliser ?

La réponse est non. En effet, notre fichier est **déjà** encodé en UTF-8 (cf III toujours). Donc les caractères accentués écrits dans ce fichier sont encodés en UTF-8. Le problème ? Vu que le serveur, lui, renvoie de l'ISO-8859-1 il faut donc décoder ces caractères accentués encodés en UTF-8 vers de l'ISO-8859-1.

Mais alors à quoi sert la fonction `utf8_encode()` ? A encoder des caractères en UTF-8 évidemment ! Seulement à encoder des caractères qui sont en ISO-8859-1, c'est-à-dire que le fichier est enregistré (cf III encore et toujours) non pas en UTF-8 (sans BOM) mais en ANSI et que le serveur renvoie de l'UTF-8 (ben oui, sinon aucun intérêt à encoder ces caractères en UTF-8).

Qui a dit que l'encodage était une partie de plaisir ?

## IV-B - Méthode Apache

Il existe aussi deux façons de préciser le charset au niveau Apache. Les deux méthodes reposent sur le même code qui est :

#### Encodage Apache

```
AddDefaultCharset utf-8
```

On peut mettre ce code à deux endroits différents selon que l'on travaille sur un serveur dédié sur lequel on a la main ou sur un serveur mutualisé.

Dans le cas d'un serveur dédié, on peut rajouter ce code dans le fichier de configuration Apache `httpd.conf`.

Dans le cas d'un serveur mutualisé, ce code peut être rajouté dans un fichier `.htaccess` placé à la racine du domaine. Et si par malchance votre serveur mutualisé interdit l'usage de `.htaccess`, il vous reste tout de même la **méthode PHP**.

## V - Au niveau de la base de données

### V-A - Aux champs

Avant de penser à ajouter la moindre information en base de données, il faut d'abord songer à modifier l'interclassement de tous les champs qui contiennent des données textes.

Pour cela, si vous avez phpMyAdmin, rien de plus simple, vous n'avez qu'à sélectionner un interclassement UTF-8 dans la liste déroulante de chaque champ au niveau de la structure des tables. Dans le cas contraire, il vous suffit d'exécuter une requête qui va bien pour l'ensemble de vos champs contenant des données texte :

#### Interclassement au niveau d'un champ

```
ALTER TABLE `ma_table` CHANGE `mon_champ` `mon_champ` VARCHAR( 50 ) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL;
```

Mieux encore, vous n'avez pas encore créé votre base de données et vous pouvez vous éviter cette manipulation en mettant un interclassement par défaut lors de sa création.

Sur phpMyAdmin, au moment de la créer, il suffit de sélectionner cet interclassement dans la liste déroulante. Sinon vous pouvez créer votre base de données de cette manière :

#### Interclassement au niveau de la base de données entière

```
CREATE DATABASE `ma_base` DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Devoir mettre de l'UTF-8 oui, mais lequel ? Que choisir entre `utf8_unicode_ci` et `utf8_general_ci` ?

Les différences entre ces deux interclassements sont minimes. Vous devez juste savoir que `utf8_general_ci` est plus rapide que `utf8_unicode_ci`. Mais en contrepartie, `utf8_unicode_ci` est plus précis que `utf8_general_ci`. Par exemple, lors d'une recherche, le caractère spécial allemand "ß" équivaut à "s" en `utf8_general_ci` alors qu'il vaut bien "ss" en `utf8_unicode_ci`.

En conclusion, si votre application ne gère pas le multilingue ou n'a pas besoin d'un niveau aussi précis de comparaison, il vaut mieux utiliser `utf8_general_ci` pour sa rapidité.

### V-B - A la connexion

Avoir tous les champs correctement encodés, c'est bien beau, seulement il reste à définir en quel encodage les différentes couches de notre application vont communiquer. Ca ne se fait évidemment pas automatiquement. Maintenant que l'on a notre serveur en UTF-8 et donc nos scripts PHP, que l'on a également nos champs en base de données, il va falloir que ces deux technologies communiquent entre elles en UTF-8.

Pour cela, rien de plus simple. Juste après la connexion à la base de données et la sélection d'une base en PHP, il faut appeler la fonction `mysql_set_charset()` en lui précisant l'encodage :

#### Encodage à la connexion en PHP

```
mysql_set_charset( 'utf8' );
```

Seulement cette fonction n'est disponible que pour les versions 5 et supérieures de PHP. Donc pour les versions inférieures, l'appel de la fonction est à remplacer par l'exécution de cette requête :

#### Encodage à la connexion en SQL

### Encodage à la connexion en SQL

```
SET NAMES "utf8";
```

## VI - Remerciements

Je remercie particulièrement **Macmillenium** pour sa relecture et son aide à la rédaction de cet article.  
Je remercie également **dvdby** pour m'avoir permis de compléter la liste des éditeurs.