

# PHP5 : La gestion avancée des dates

par Julien Pauli ([Tutoriels](#), [article et conférences PHP et développement web](#)) ([Blog](#))

Date de publication : 09/06/2008

Dernière mise à jour : 11/10/2009

Depuis PHP 5.1, la gestion des dates en PHP a profondément changé. Certaines fonctions ont été réécrites, la gestion interne des dates s'est agrandie et elle est devenue indépendante de l'OS sous-jacent.

De nouveaux objets/fonctions ont fait leur apparition qui apportent, entre autres, la gestion des fuseaux et décalages horaires (heure d'été), autant de notions intéressantes en développement web et sur lesquelles il serait dommage de faire l'impasse. Faisons un point dessus.

I - Introduction aux dates en PHP.....	3
II - Les problèmes de dates.....	3
III - Les dates en PHP <= 5.0.....	4
IV - PHP5.1 et supérieur apporte une solution.....	5
IV-A - Les objets DateTime et DateTimeZone.....	6
IV-B - La gestion des fuseaux horaires.....	8
IV-C - Les opérations sur les dates.....	9
V - Aller plus loin avec les dates.....	10
VI - Conclusion.....	11

## I - Introduction aux dates en PHP



Tout langage web doit être doté de fonctionnalités concernant les dates. C'est le cas de PHP depuis bien longtemps. Cependant sur Internet, qu'est ce qu'une date ?

La date est globalement la représentation d'un jour, d'un mois, d'une année, d'une heure, de minutes, et de secondes. Le problème vient d'Internet : l'International Network.

La Terre n'est pas plate, et n'est pas une communauté unique qui parle la même langue et utilise les mêmes représentations pour des notions comme les dates.

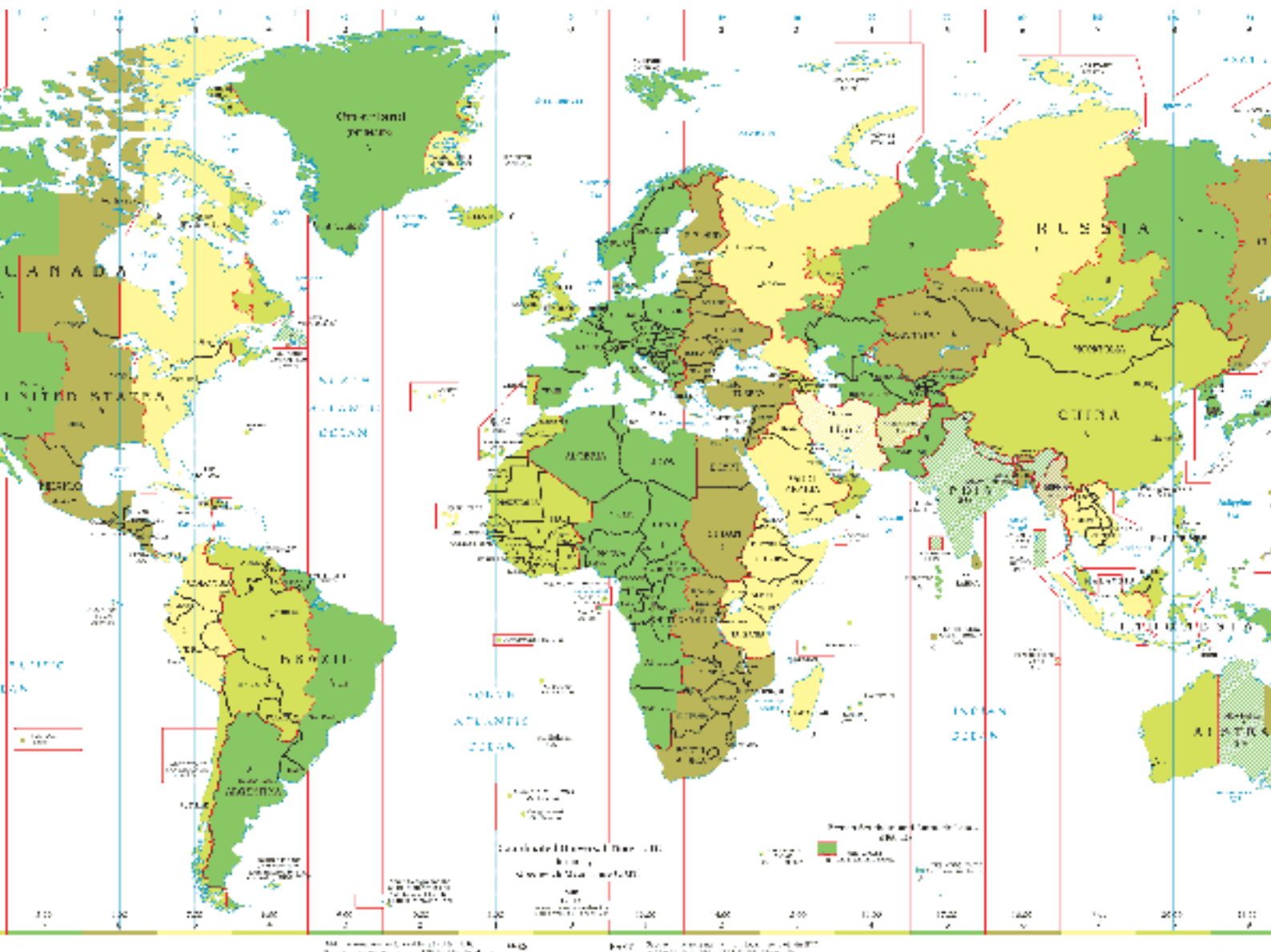
Une date doit être fixée à un point de la planète, et écrite d'une certaine manière selon ce point.

## II - Les problèmes de dates

- Chaque lieu de la planète est assimilé à un fuseau horaire amenant à un décalage positif ou négatif par rapport à  **UTC**, (anciennement  **GMT** : le fuseau zéro passant par Greenwich)
- Certains pays sont traversés par plusieurs fuseaux (4 , 5... c'est le cas des USA ou de la Russie).
- Le décalage d'un fuseau horaire peut être d'une heure, mais aussi (à certains endroits) d'une demie ou un quart d'heure
- Des pays ont des fuseaux horaires horizontaux : l'Australie, le découpage est donc sous forme de carrés
- Certains lieux changent de fuseau pendant l'année
- Certains pays utilisent une notion "d'heure d'été" (DST : Daylight Saving Time), en plus de la notion de fuseau horaire
- Ces pays ne passent pas à "l'heure d'été" tous à la même période
- Le décalage d'"heure d'été" n'est pas le même pour tous, il peut être d'une heure, mais aussi d'une demie ou un quart d'heure

Cela fait déjà beaucoup, mais ça n'est pas terminé :

- Les abréviations des noms des fuseaux horaires utilisées en informatique, ne sont pas les mêmes d'un OS à un autre
- Ces abréviations peuvent être confuses
- Les représentations des dates sont très différentes : September 16th, 2005; 20060225040821; 2003-08-11 16:12:07.11403+01; 2001-12-17T12:10:27.123-05:00...



timezones

### III - Les dates en PHP <= 5.0

En PH4 et 5.0, les fonctions de dates sont les suivantes :

- 1 **gettimeofday()** - Récupère le temps actuel (tableau PHP)
- 2 **checkdate()** Valide une date Grégorienne
- 3 **localtime()** - Récupère le temps local (tableau PHP)
- 4 **date()** / **gmdate()** - Formate une date locale/GMT
- 5 **mktime()** / **gmmktime()** - Récupère le timestamp Unix d'une date
- 6 **strftime()** / **gmstrftime()** - Formate une date/temps locale/GMT en fonction de paramètres locaux
- 7 **getdate()** - Récupère des infos de date/temps depuis un timestamp
- 8 **strtotime()** - Récupère un timestamp depuis un texte écrit en anglais (now, yesterday ...)

Toutes ces fonctions utilisent la notion de timestamp UNIX : le nombre de secondes écoulées depuis "EPOCH" : le 1er Janvier 1970 à 0h00.


Le problème est que ce chiffre ne représente rien pour un utilisateur d'un site lambda : il faut le convertir en date, dans le format que l'utilisateur a l'habitude de voir.


Et c'est bien là tout le problème : relisez la liste dans la section précédente. Pour formater une date d'un point donné (le serveur), en une date d'une autre point (le client), il faut du courage...

Ces fonctions-là répondent néanmoins à beaucoup de besoins, mais elles ne sont pas suffisantes :  
Les timestamps sont stockés sur 32bits, et vont de 1902 à 2038, mais certains OS ne traitent que les entiers positifs : la limite est donc de 1970 à 2038.  
Aussi, ces fonctions ne gèrent pas correctement (voire pas du tout pour certaines) les notions de fuseaux horaires et de décalage d'heure d'été, et beaucoup d'entre elles dépendent de l'OS sur lequel elles sont utilisées, elles renvoient donc des résultats différents en fonction de celui-ci.

## IV - PHP5.1 et supérieur apporte une solution

Heureusement, depuis PHP5.1, une solution intéressante existe grâce à l'apparition de 2 objets simplifiant la gestion des dates, et surtout proposant d'autres avantages

 Ces fonctions sont disponibles dans la distribution standard, à partir de PHP 5.2  
Il est possible d'ajouter un support expérimental dans PHP 5.1.x en utilisant le drapeau  
`CFLAGS=-DEXPERIMENTAL_DATE_SUPPORT=1` lors de la compilation.

- Calculs effectués autour d'un entier 64bits signé
- Plus de dépendances envers l'OS : embarquement d'une base timezonedb complète, mise à jour régulièrement
- Support complet des fuseaux et des décalages (plus de 550 fuseaux)
-  **Abréviations** normalisées des fuseaux : Continent/Lieu/SousLieu
- Calculs sur les dates (ajouts, retranchements)
- Formats normalisés : ISO8601, RSS, W3C, Cookie

## IV-A - Les objets DateTime et DateTimezone

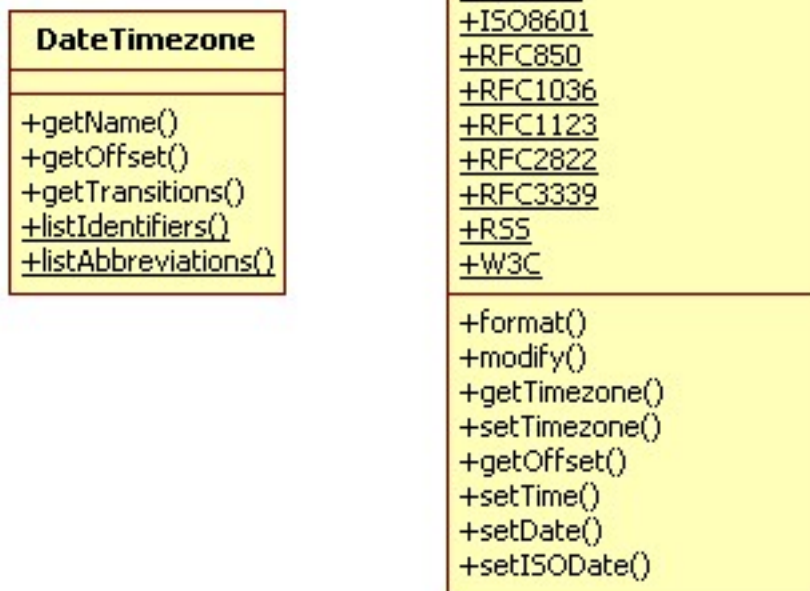


Diagramme de classes concernant la gestion des dates en PHP5.1 et supérieurs

Dans PHP5.1 et ultérieurs, avec le nouveau support des dates, il n'est plus question de manipuler des timestamps signés.

### Utilisation des fonctions classiques

```
<?php
$date = strtotime("1969-12-31 12:13:40");
echo $date;
// affiche -13390
```

Mais plutôt des objets de date, qui sont des représentations sur un entier 64bits signé, et qui peuvent être obtenus soit par instanciation de la nouvelle classe **DateTime**, soit par utilisation des nouvelles fonctions apparues, comme **date\_create()** :

### utilisation des fonctionnalités avancées

```
<?php
$date = new DateTime("1969-12-31 12:13:40");
// équivalent strictement à $date = date_create("1969-12-31 12:13:40");
echo $date->format('U'); // affiche -13390
echo $date->format(DateTime::RSS); // affiche Wed, 31 Dec 1969 12:13:40 +0100
```

L'objet **DateTime** ou **date\_create()**, prennent un paramètre tel qu'accepté par **strtotime()**, elle même calée sur


### les recommandations GNU des dates


Si l'envie vous prend d'utiliser un timestamp tout de même, veuillez à le faire précéder du symbole '@' :

```
$date = new DateTime('@'.time());
```

Notez que ceci n'est pas la procédure recommandée, mais si le besoin se présente... `new DateTime()` est valide (sans paramètres) : le défaut est 'now'.

Mais attention tout de même : pour que tout cela fonctionne, il est nécessaire d'affecter un fuseau à une date. Pour que PHP ne se repose plus sur l'OS, il faut obligatoirement lui dire quel est le fuseau sur lequel il devra caler toutes ses dates (le fuseau par défaut).


Ceci est possible au moyen de la fonction  `date_default_timezone_set()` , ou alors via  la directive `date.timezone` du fichier `php.ini`


 *Tout appel a une fonction de date, alors qu'aucun fuseau (timezone) n'est défini explicitement, lèvera une erreur PHP non fatale (E\_WARNING ou E\_STRICT selon la fonction utilisée)  
Par la suite, nous supposerons la timezone par défaut comme étant 'Europe/Paris'.*

Ainsi, travailler avec les anciennes fonctions PHP sur les dates, n'a plus grand intérêt car on ne profite aucunement des nouvelles fonctionnalités. Il ne faut plus penser timestamp, mais bien objet date.

Les options de la méthode `format()` sont les mêmes que celles de la fonction PHP `date()`, à laquelle l'option 'U' a été rajoutée depuis PHP5.1 donc.

De la même manière, cette méthode `format()` sur un objet `DateTime`, est équivalente à la fonction `date_format($objetDateTime)`.

 *L'ajout des fonctionnalités relatives aux dates à partir de PHP 5.1 a été faite autant avec un style objet que procédural. Nous préférons pour la suite le style objet.  
Ce double support procédural/objet se retrouve dans d'autres fonctionnalités de PHP5 comme MySQLi*

La base de données des fuseaux et des décalages, sur laquelle est basé le désormais standard (>=PHP5.1) support des dates,  se trouve dans **PECL**. Il n'est pas spécialement nécessaire de la mettre à jour, une nouvelle version est de toute façon intégrée lors de la migration de version de PHP.

Cette base est une compilation  de la base **OLSON**.

#### formats normalisés

```
<?php
$date = new DateTime("1819-02-14 10:51");
echo $date->format('D Y-m-d H:i:s - \s\e\m\A\i\n\W'); // Sun 1819-02-14 10:51:00 - semaine 06

echo $date->format(DateTime::ATOM); // 1819-02-14T10:51:00+00:09
echo $date->format(DateTime::COOKIE); //Sunday, 14-Feb-19 10:51:00 PMT
echo $date->format(DateTime::ISO8601); //1819-02-14T10:51:00+0009
echo $date->format(DateTime::RSS); //Sun, 14 Feb 1819 10:51:00 +0009
echo $date->format(DateTime::W3C); //1819-02-14T10:51:00+00:09
echo $date->format(DateTime::RFC3339); //1819-02-14T10:51:00+00:09
```

Remarquez le +00:09. Mon fuseau est pourtant bien Europe/Paris, pourquoi n'ai-je pas +01:00 comme je pourrais m'y attendre ? Très simple : en 1819, à cette date-là précisément, le décalage de mon fuseau n'était que de 00:09 ! La base de données `timezonedb` comporte vraiment toutes ces infos.

```
<?php
$date = new DateTime("2019-01-14 10:51");
echo $date->format(DateTime::ISO8601); //2019-07-14T10:51:00+0100

$date = new DateTime("2019-07-14 10:51");
echo $date->format(DateTime::ISO8601); //2019-07-14T10:51:00+0200
```

L'heure d'été est aussi prise en compte. En juillet nous sommes à +02, alors qu'en janvier nous sommes bien à +01, ça sent bon :)

## IV-B - La gestion des fuseaux horaires

Toute date est associée à une timezone : un fuseau horaire. Celui-ci permet de mesurer le décalage par rapport à GMT (ou UTC si on préfère, l'abus de langage est courant...).

Ce décalage est la somme du décalage du fuseau (la Terre n'est pas plate) ajouté au décalage éventuel de l'"heure d'été".

La gestion des fuseaux est possible grâce à l'objet **DateTimeZone** (on a des fonctions PHP aux effets similaires). Cet objet est séparé de l'objet **DateTime**, mais les 2 restent très proches :

```
<?php
$date = new DateTime("now"); // 'now' n'est pas nécessaire, c'est la valeur par défaut
echo $date->format(DateTime::ISO8601) // 2008-05-27T20:30:46+0200

$tz = new DateTimeZone('America/New_York');
$date->setTimezone($tz);
echo $date->format(DateTime::ISO8601) // 2008-05-27T14:30:46-0400

echo $date->getOffset(); // -14400
// équivalent à $tz->getOffset($date)
```

Une date peut donc changer de fuseau, son décalage sera géré automatiquement. Lorsqu'on interroge le fuseau via **getOffset()**, nous sommes obligés de lui passer un objet **DateTime**. C'est logique : le décalage varie en fonction du fuseau certes, mais de la date que l'on cherche (heure d'été par exemple).

Ainsi, il n'y a plus besoin de calculer quoi que ce soit en additionnant ou soustrayant des timestamps, ici tout est géré, quels que soient le fuseau et la date.

### Affectation directe d'un fuseau

```
<?php
// affectation directe d'un fuseau précis (autre que celui par défaut) à une date
$date = new DateTime('now', new DateTimeZone('Europe/London'));
```

La classe **DateTimeZone** possède encore quelques méthodes, la méthode statique **listAbbreviations()** va lister toutes les abréviations des fuseaux horaires que les OS ou les programmes avaient l'habitude d'utiliser (attention, liste extrêmement longue et coûteuse). Il est déprécié d'utiliser de telles abréviations à présent, le nom d'un fuseau

doit toujours être indiqué selon  **une liste précise**

Cette liste est d'ailleurs obtenue par la méthode statique **DateTimeZone::listIdentifiers()**

Plus intéressante, la méthode **getTransitions()** va retourner un tableau indiquant tous les décalages programmés dans le temps :

### décalages programmés par fuseau

```
<?php
$tz = new DateTimeZone(date_default_timezone_get());
var_dump($tz->getTransitions());
/* affiche
[0]=>
array(5) {
  ["ts"]=>
  int(-1855958901)
  ["time"]=>
  string(24) "1911-03-10T23:51:39+0000"
  ["offset"]=>
  int(0)
  ["isdst"]=>
  bool(false)
  ["abbr"]=>
  string(3) "WET"
}
[1]=>
array(5) {
  ["ts"]=>
  int(-1689814800)
  ["time"]=>
  string(24) "1916-06-14T23:00:00+0000"
```

### décalages programmés par fuseau

```

["offset"]=>
int(3600)
["isdst"]=>
bool(true)
["abbr"]=>
string(4) "WEST"
}
il y a 182 entrées comme cela ...
*/
    
```

## IV-C - Les opérations sur les dates

Maintenant que nous connaissons bien les objets `DateTime` et `DateTimeZone`, voyons de plus près les calculs possibles :

```

<?php
$d = new DateTime();
$d->setDate(2007,10,03);
echo $d->format("d/m/Y à H\hi:s"); // 03/10/2007 à 19h39:53
$d->setTime(2,30);
echo $d->format("d/m/Y à H\hi:s"); // 03/10/2007 à 02h30:00
    
```

Si l'on ne veut changer que les minutes, ou alors les mois, il faut s'y prendre comme cela :

```

<?php
$d = new DateTime();
$d->setDate($d->format('Y'),10,$d->format('d'));
    
```

Un peu bizarre, **`setIsoDate()`**, comme son nom ne l'indique pas, permet de spécifier [année, no de semaine, jour] :


```

<?php
$d = new DateTime();
$d->setIsoDate(2007,03,05); // année 2007, et 5ème jour de sa 3ème semaine
echo $d->format(DateTime::W3C); // 2007-01-19T19:46:30+01:00
    
```

La méthode **`modify()`** va faire un calcul. Comme le constructeur de **`DateTime`**, elle prend un paramètre accepté par **`strtotime()`** :

```

<?php
$d = new DateTime("2008/03/12 15:30");
$d->modify("-3 months +1 hour");
echo $d->format(DateTime::W3C); // 2007-12-12T16:30:00+01:00
$d->modify('2 years ago'); // "il y a 2 ans"
echo $d->format(DateTime::W3C); // 2005-12-12T16:30:00+01:00
    
```

 **`modify()`** va modifier l'objet actuel (un peu logique non ?) - Si vous voulez garder une copie de l'ancien objet, clonez-le simplement.

Le fuseau n'a pas bougé, il s'agit de celui par défaut, car je ne l'ai spécifié nulle part. Je rappelle que mon `php.ini` contient le fuseau par défaut de PHP : Europe/Paris.

Pour ces deux dates-là : le décalage était à ce moment-là de +1.

De la même manière, on peut comparer des dates, mais apparemment uniquement depuis PHP 5.2 (à confirmer...) :

```

<?php
$d1 = new DateTime();
$d2 = clone $d1;
$d2->modify("tomorrow");
var_dump($d2 < $d1); // false
    
```

## V - Aller plus loin avec les dates

Cette API des dates, (à partir de PHP5.1) est très pratique, surtout pour la gestion automatique des fuseaux et de leurs décalages. La possibilité de changer une date de fuseau est aussi intéressante : la date en question change bien en fonction du décalage de son fuseau. Qu'en est-il de l'intervalle de validité ?

C'est le point d'interrogation. Je peux créer une date de l'année 910 par exemple, et demander le jour de la semaine correspondant, ça fonctionne (du moins ça semble, je n'ai pas vérifié la réalité)

Sur cette même date 'exotique' (car très en arrière, largement sous 1900), le décalage avec mon fuseau est noté à +00:09, cependant, l'offset ne renvoie pas 540 secondes mais 561... ?



```
<?php
$d = new DateTime("0910/11/01"); // l'année est toujours exprimée sur 4 chiffres
echo $d->format("D"); // Sun -> il s'agirait donc d'un dimanche ?
echo $d->format(DATE_ISO8601) // 0910-11-01T00:00:00+0009
echo $d->getOffset(); // 561
```


Pour une date très supérieure à 2038, tout semble OK :

```
<?php
$d = new DateTime("8910/11/01"); // an 8910 !
echo $d->format("D"); // Sat : ce sera un Samedi
echo $d->format(DATE_ISO8601); // 8910-11-01T00:00:00+0100
echo $d->getOffset(); // 3600 : OK
```

L'intervalle semble donc virtuellement infini, mais les dates avant le 10/03/1911 retournent une information de décalage de +00:09 et un offset non pas de 540 mais 561 secondes (sur le fuseau Europe/Paris donc).

Je n'ai pas eu le courage de lire tout Wikipedia pour me renseigner, ni de me pencher plus sur la source de cette fameuse timezonedb, incluse dans PHP.

Quoiqu'il en soit, si on veut aller plus loin avec les calculs sur les dates, je conseille le composant  **Zend\_Date**, du  **Zend Framework**

Cette classe fait abstraction du fuseau (timezone) de PHP et se repose sur la bibliothèque BCMath de PHP (inclue), afin de faire des calculs très précis. Zend\_Date sait donc faire tout ce que PHP sait faire,  **et même plus...**

Zend\_Date se pilote différemment des objets dates de PHP, et gère tout dans un objet (dates, temps, fuseau, et même localisation). Ce composant permet aussi des calculs avancés, et gère l'internationalisation des dates. Voyez plutôt quelques exemples :

```
<?php
require_once 'Zend/Date.php';
$date = new Zend_Date('06/10/2004 11h47', false, 'fr_FR'); // oui carrément écrit en bon Français !

echo $date->toString("GGGG, EEEE dd MMMM yy à hh:mm"); // après Jésus-
Christ, mercredi 06 octobre 04 à 11:47

print_r (Zend_Date_Cities::City('Paris'));
/*
Array(
    [latitude] => 48.8666667
    [longitude] => 2.3333333
    [horizon] =>
)
*/

$date = new Zend_Date('12/18/1882');
echo $date->getIso(); // 1882-12-18T00:00:00+0100

$date->addHour(3); // ajoutons 3 heures
echo $date->get(Zend_Date::TIMES); // 03:00:00
```

 **L'API de Zend\_Date** vous en dira plus ;-)

## VI - Conclusion

La gestion des dates n'est pas une mince affaire. En fait elle n'est pas spécialement difficile non plus, il suffit de savoir comment elle fonctionne en PHP, et de ne pas oublier que toute date est reliée obligatoirement à un fuseau, qui va pouvoir modifier la manière dont la date est perçue, en instaurant des décalages.

Cette notion est bien gérée à partir de PHP5.1, mais a quand même des limites que tentent de repousser les Frameworks objets de plus en plus utilisés pour bâtir des applications complexes.