Normes de codage PHP

Table des matières

Introduction

Ce document recense les règles de codage à respecter dans le cadre de développements PHP. Il reprend les normes de développement des modules PEAR.

Indentation

On attachera un soin particulier à l'indentation : l'essentiel n'est pas que le programmeur se retrouve dans son propre code, mais que celui qui le maintient ou le réutilise y parvienne facilement.
Dans cette optique on veillera à ne pas utiliser le caractère « Tabulation » dans les fichiers sources. En effet, suivant l'éditeur, une tabulation équivaut à 4, 6, ou 8 caractères. L'idée est donc de paramétrer l'éditeur pour qu'il remplace les tabulations par des espaces.

Pour le code PHP, on préconise une indentation de 4 caractères

Si vous utilisez Emacs pour éditer le code PHP, vous devez fixer le paramètre indent-tabs-mode à 'nil'. Vous trouverez ici un exemple de configuration de Emacs pour respecter ces recommandations (vous devrez vous assurer qu'elle est bien appelée lorsque vous éditerez les fichiers PEAR) :

```
(defun php-mode-hook ()
  (setq tab-width 4
        c-basic-offset 4
        c-hanging-comment-ender-p nil
        indent-tabs-mode
        (not
         (and (string-match "/\\(PEAR\\|pear\\)/" (buffer-file-name))
              (string-match "\.php$" (buffer-file-name))))))
```

De la même façon, vous trouverez ici les règles pour vim :

```
set expandtab
  set shiftwidth=4
  set softtabstop=4
  set tabstop=4
```

Pour Ultraedit, le paramétrage s'effectue dans le menu « Avancé > Configuration », onglet « Edition ».

Structures de Contrôles

Les structures de contrôles incluent les 'if', 'for', 'while', 'switch', etc. Vous trouverez ici un exemple de structure 'if' qui est la plus compliquée :

```php
<?php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
?>
```

Les instructions de contrôle doivent avoir un espace entre le mot clé de l'instruction et la parenthèse ouvrante, afin de ne pas les confondre avec des appels de fonction.
Il est vivement recommandé de toujours utiliser des parenthèses, même dans les situations où elles sont techniquement optionnelles. Leur présence augmente la lisibilité du code et réduit le risque d'erreur logique lors de l'ajout de nouvelles lignes de code.
Pour l'instruction "switch" :

```php
<?php
switch (condition) {
case 1:
    action1;
    break;

case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}
?>
```

Appels de Fonction

Les fonctions doivent être appelées sans espace entre le nom de la fonction, la parenthèse ouvrante, et le premier paramètre; avec un espace entre la virgule et chaque paramètre; et aucun espace entre le dernier paramètre, la parenthèse fermante et le point virgule. Voici un exemple :

```
<?php
$var = foo($bar, $baz, $quux);
?>
```

Comme montré ci-dessus, il doit y avoir un espace de chaque côté du signe égal utilisé pour affecter la valeur de retour de la fonction à une variable. Dans le cas d'un bloc d'instructions similaire, des espaces supplémentaires peuvent être ajoutés pour améliorer la lisibilité :

```
<?php
$short         = foo($bar);
$long_variable = foo($baz);
?>
```

Définitions des Fonctions

La déclaration des fonctions respecte l'indentation classique des parenthèses :

```php
<?php
function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}
?>
```

Les arguments possédant des valeurs par défaut vont à la fin de la liste des arguments. Il faut toujours chercher à retourner une valeur ayant un sens lorsque cela est possible. Voici un exemple un peu plus long :

```php
<?php
function connect(&$dsn, $persistent = false)
{
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
?>
```

Commentaires

La documentation des classes incluse dans le code source doit suivre la convention PHPDoc, similaire à celle de Javadoc. Pour plus d'information au sujet de PHPDoc vous reporter à la documentation en ligne.
Les commentaires non inclus dans la documentation sont vivement encouragés. La règle générale est que, si en regardant une portion de code, vous pensez "Wououh, je ne veux pas me lancer dans cette explication", il faut absolument la commenter tout de suite avant que vous n'oubliez comment cela fonctionne.
Les commentaires du type C (/* */) et les commentaires standard C++ (///) sont tous les deux acceptés. Les commentaires de type Perl/shell (#) sont à éviter.

Cartouche

```
/**
 * Classe Machin
 * -------------
 * A quoi sert cette classe, dans quel contexte, quelles sont ses
 * particularités éventuelles, qu'est-ce qu'il faut bien comprendre
 * avant de l'aborder.
 *
 * Créé le 01/01/2001 Par Armand Jean Du Plessis
 */
class Machin
{
```

Méthode

```
// }}}
// {{{ methodeBidule()

/**
 * Principe de général de la méthode, particularités éventuelles,
 * subtilités ... etc.
 *
 * @param    string     $sArg1       Description de l'argument 1
 * @param    integer    $iArg2       Description de l'argument 2
 * @param    array      $aArg3       Description de l'argument 3
 * @since    1.0
 * @access   public
 * @return   mixed      Description de la valeur retournée par la méthode
 */
function methodeBidule($sArg1, $sArg2, $sArg3 = array())
{
```

Descripteurs les plus couramment utilisés :

@author	Auteur
@deprecated	Indique que la méthode est obsolète
@param	Paramètre
@return	Valeur retournée
@see	Voir aussi
@since	Numéro de version auquel la méthode a été ajoutée
@version	Numéro de version de la méthode

Code

```php
// Instanciation de la variable $iTruc
$iTruc = 12;

/**
 * opération de multiplication par l'inverse de la variable
 * $iTruc afin de rabaisser sa valeur à hauteur de la moitié
 * de ce qu'elle était initialement
*/
$iTruc = $iTruc / 2;
```

Inclure du Code

A chaque endroit où vous voulez inclure de façon inconditionnelle un fichier de classe, utilisez **require_once()**. A chaque endroit où vous voulez inclure de façon conditionnelle un fichier de classe (par exemple des méthodes de construction), utilisez **include_once()**. Ces deux méthodes s'assurent que le fichier classe n'est inclu qu'une seule fois. Elles partagent la même liste de fichiers, il donc possible de les mélanger - un fichier inclu avec **require_once()** ne sera pas inclu une seconde fois par **include_once()**.

> **Note : include_once()** et **require_once()** sont des instructions, pas des fonctions. Vous n'avez pas besoin de parenthèses autour du nom de fichier à inclure.

Tags dans le Code PHP

Toujours utiliser `<?php ?>` pour délimiter du code PHP, et non la version abrégée `<? ?>`. Cela est obligatoire pour être conforme aux règles de PEAR et c'est aussi la méthode la plus portable pour inclure du code PHP sur différents systèmes d'exploitations et configurations.

Conventions de Nom

Variables

Les noms de variables ne doivent contenir aucun caractère spécial (ni accent, ni espace). Nous n'utiliserons pas de séparateur de mot : tout nouveau mot commencera par une majuscule.
Sachant que :

- le langage PHP n'est pas typé
- le formalisme objet induit la création de fonctions courtes (40 lignes max)

Il n'est pas nécessaire de surcharger outre mesure la définition des variables pour que le code reste lisible. On adoptera donc les conventions de préfixage suivante :

Type	Préfixe	Exemple
Numérique	**i**	$iCompteur
Chaîne de caractère	**s**	$sPrenom
Booléen	**b**	$bCondition
Tableau	**a**	$aUserIds
Objet	**o**	$oFormulaire

Classes

Les classes doivent avoir un nom parlant. Eviter les abréviations lorsque cela est possible. Les noms de classes doivent toujours commencer par une majuscule. Nous n'utiliserons pas de séparateur de mot : tout nouveau mot commencera par une majuscule :
Exemples :

Utilisateur
TableManager

Remarque : pour souligner l'architecture hiérarchique des classes PEAR, on retrouvera parfois le nom du package dans le nom de la classe, chaque niveau de la hiérarchie séparé par un trait souligné (_). (HTML_QuickForm, Net_Finger …)

Fontions et Méthodes

Les fonctions et les méthodes doivent être nommées en utilisant le style "studly caps" (aussi connus sous les noms de "bumpy case" ou "camel caps"). La première lettre du nom (après le préfixe pour une fonction) est une minuscule, et chaque premier caractère d'un nouveau "mot" doit être une majuscule.
Quelques exemples :

connect()
getSomeUsefulData()

Remarque : les fonctions PEAR seront souvent préfixées par le nom du package, pour éviter les doublons entre les packages (XML_RPC_serializeData())

Les éléments (méthodes, attributs) privés d'une classe sont précédés d'un simple souligné (_) (ces éléments sont destinés à n'être utilisés que par la classe qui les déclarent; PHP ne supportant pas encore le contrôle des noms privés).
Par exemple :

_sort()
_initTree()
$this->status

Constantes

Les constantes doivent toujours être en majuscules, commencer par « C », les mots séparés par des '_'.
Exemples :

C_DEFAULT_USER
C_IMAGE_PATH

Remarque : dans PEAR, les constantes sont parfois préfixées avec le nom en majuscule de la classe/package dans laquelle elle sont utilisée. Par exemple, les constantes utilisées par le package DB:: commencent toutes par "DB_".